GUROBI OPTIMIZER
EXAMPLE TOUR

Version 9.0, Copyright © 2020, Gurobi Optimization, LLC
1 Introduction

2 Example tour
   2.1 A list of the Gurobi examples ........................................... 3
   2.2 Load and solve a model from a file ....................................... 6
   2.3 Build a model ................................................................. 7
   2.4 Additional modeling elements ............................................. 9
   2.5 Modify a model ............................................................... 10
   2.6 Change parameters ......................................................... 12
   2.7 Automated parameter tuning .............................................. 12
   2.8 Diagnose and cope with infeasibility .................................. 13
   2.9 MIP starts ................................................................. 14
   2.10 Model-data separation in Python ...................................... 15
   2.11 Callbacks ............................................................... 16

3 Example Source Code .......................................................... 17
   3.1 C Examples ................................................................. 17
       batchmode_.c.c .......................................................... 17
       bilinear_.c.c ......................................................... 23
       callback_.c.c ......................................................... 26
       dense_.c.c .......................................................... 31
       diet_.c.c ............................................................ 34
       facility_.c.c .......................................................... 37
       feasopt_.c.c ............................................................ 42
       fixanddive_.c.c .......................................................... 45
       gc_pwl_.c.c ............................................................. 48
       gc_pwl_func_.c.c ......................................................... 51
       genconstr_.c.c ........................................................... 56
       lp_.c.c ................................................................. 60
       lpmethod_.c.c ............................................................ 62
       lpmode_.c.c ............................................................. 63
       mip1_.c.c ............................................................... 66
       mip2_.c.c ............................................................... 68
       multiobj_.c.c ............................................................ 71
       multiscenario_.c.c ....................................................... 74
       params_.c.c ............................................................. 83
       piecewise_.c.c ............................................................ 85
       poolsearch_.c.c ............................................................ 88
       qcp_.c.c ................................................................. 91
3.2 C++ Examples

batchmode_c++.cpp ........................................ 139
bilinear_c++.cpp ........................................ 143
callback_c++.cpp ........................................ 145
dense_c++.cpp ........................................ 149
diet_c++.cpp ........................................ 151
facility_c++.cpp ........................................ 153
feasopt_c++.cpp ........................................ 156
fixanddive_c++.cpp ....................................... 158
gc_pwl_c++.cpp ........................................ 160
gc_pwl_func_c++.cpp ..................................... 162
genconstr_c++.cpp ....................................... 166
lp_c++.cpp ........................................ 169
lpmethod_c++.cpp ....................................... 170
lpmod_c++.cpp ........................................ 171
mip1_c++.cpp ........................................ 173
mip2_c++.cpp ........................................ 174
multiobj_c++.cpp ....................................... 176
multiscenario_c++.cpp ................................ 179
params_c++.cpp ........................................ 185
piecewise_c++.cpp ..................................... 186
poolsearch_c++.cpp ................................... 189
sensitivity_c++.cpp .................................. 191
qcp_c++.cpp ........................................ 194
qp_c++.cpp ........................................ 196
sos_c++.cpp ........................................ 197
sudoku_c++.cpp ......................................... 198
tsp_c++.cpp ........................................ 201
tune_c++.cpp ........................................ 205
workforce1_c++.cpp .................................. 207
workforce2_c++.cpp .................................. 209
workforce3_c++.cpp .................................. 212
workforce4_c++.cpp .................................. 215
workforce5_c++.cpp .................................. 220
### 3.3 Java Examples

<table>
<thead>
<tr>
<th>Java File</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Batchmode.java</td>
<td>224</td>
</tr>
<tr>
<td>Bilinear.java</td>
<td>228</td>
</tr>
<tr>
<td>Callback.java</td>
<td>230</td>
</tr>
<tr>
<td>Dense.java</td>
<td>233</td>
</tr>
<tr>
<td>Diet.java</td>
<td>236</td>
</tr>
<tr>
<td>Facility.java</td>
<td>238</td>
</tr>
<tr>
<td>Feasopt.java</td>
<td>241</td>
</tr>
<tr>
<td>Fixanddive.java</td>
<td>242</td>
</tr>
<tr>
<td>GCPWL.java</td>
<td>244</td>
</tr>
<tr>
<td>GCPWLFunc.java</td>
<td>246</td>
</tr>
<tr>
<td>Genconstr.java</td>
<td>249</td>
</tr>
<tr>
<td>Lp.java</td>
<td>252</td>
</tr>
<tr>
<td>Lpmethod.java</td>
<td>253</td>
</tr>
<tr>
<td>Lpmod.java</td>
<td>254</td>
</tr>
<tr>
<td>Mip1.java</td>
<td>255</td>
</tr>
<tr>
<td>Mip2.java</td>
<td>257</td>
</tr>
<tr>
<td>Multiobj.java</td>
<td>259</td>
</tr>
<tr>
<td>Multiscenario.java</td>
<td>261</td>
</tr>
<tr>
<td>Params.java</td>
<td>266</td>
</tr>
<tr>
<td>Piecewise.java</td>
<td>268</td>
</tr>
<tr>
<td>Poolsearch.java</td>
<td>270</td>
</tr>
<tr>
<td>Qcp.java</td>
<td>272</td>
</tr>
<tr>
<td>Qp.java</td>
<td>273</td>
</tr>
<tr>
<td>Sensitivity.java</td>
<td>275</td>
</tr>
<tr>
<td>Sos.java</td>
<td>278</td>
</tr>
<tr>
<td>Sudoku.java</td>
<td>279</td>
</tr>
<tr>
<td>Tsp.java</td>
<td>282</td>
</tr>
<tr>
<td>Tune.java</td>
<td>286</td>
</tr>
<tr>
<td>Workforce1.java</td>
<td>287</td>
</tr>
<tr>
<td>Workforce2.java</td>
<td>289</td>
</tr>
<tr>
<td>Workforce3.java</td>
<td>291</td>
</tr>
<tr>
<td>Workforce4.java</td>
<td>294</td>
</tr>
<tr>
<td>Workforce5.java</td>
<td>298</td>
</tr>
</tbody>
</table>

### 3.4 C# Examples

<table>
<thead>
<tr>
<th>C# File</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>batchmode_cs.cs</td>
<td>301</td>
</tr>
<tr>
<td>bilinear_cs.cs</td>
<td>305</td>
</tr>
<tr>
<td>callback_cs.cs</td>
<td>306</td>
</tr>
<tr>
<td>dense_cs.cs</td>
<td>310</td>
</tr>
<tr>
<td>diet_cs.cs</td>
<td>312</td>
</tr>
<tr>
<td>facility_cs.cs</td>
<td>314</td>
</tr>
<tr>
<td>feasopt_cs.cs</td>
<td>317</td>
</tr>
<tr>
<td>fixanddive_cs.cs</td>
<td>318</td>
</tr>
<tr>
<td>gc_pwl_cs.cs</td>
<td>320</td>
</tr>
<tr>
<td>ge_pwl_func_cs.cs</td>
<td>322</td>
</tr>
</tbody>
</table>
3.5 Visual Basic Examples

batchmode_vb.vb ........................................... 376
bilinear_vb.vb ........................................... 376
callback_vb.vb ........................................... 380
dense_vb.vb ........................................... 380
diet_vb.vb ........................................... 383
facility_vb.vb ........................................... 386
feasopt_vb.vb ........................................... 388
fixanddive_vb.vb ........................................... 390
gc_pwl_vb.vb ........................................... 392
gc_pwl_func_vb.vb ........................................... 394
genconstr_vb.vb ........................................... 396
lp_vb.vb ........................................... 398
lpmethod_vb.vb ........................................... 401
lpmod_vb.vb ........................................... 401
mip1_vb.vb ........................................... 402
mip2_vb.vb ........................................... 403
multiobj_vb.vb ........................................... 405
multiscenario_vb.vb ........................................... 406
params_vb.vb ........................................... 408
piecewise_vb.vb ........................................... 411
poolsearch_vb.vb ........................................... 416
workforce1_vb.vb ........................................... 417
workforce2_vb.vb ........................................... 419
workforce3_vb.vb ........................................... 419
workforce4_vb.vb ........................................... 419
workforce5_vb.vb ........................................... 419
3.6 Python Examples

batchmode.py .................................................. 450
bilinear.py .................................................. 450
callback.py .................................................. 453
custom.py ................................................... 454
dense.py ...................................................... 456
diet.py ......................................................... 457
diet2.py ....................................................... 459
diet3.py ....................................................... 461
diet4.py ....................................................... 462
dietmodel.py .................................................. 463
facility.py .................................................... 464
feasopt.py ..................................................... 465
fixanddive.py .................................................. 468
gc_pwl.py ..................................................... 469
gc_pwl_func.py ............................................... 470
genconstr.py .................................................. 472
lp.py ........................................................ 474
lpmethod.py ................................................... 476
lpmod.py ...................................................... 477
matrix1.py ..................................................... 478
matrix2.py ..................................................... 479
mip1.py ......................................................... 480
mip2.py ......................................................... 481
multiobj.py .................................................... 482
multiscenario.py ............................................... 483
netflow.py ..................................................... 485
params.py ..................................................... 490
piecewise.py ................................................... 491
poolsearch.py .................................................. 492
portfolio.py ................................................... 494
qcp.py ........................................................ 496
qp.py .......................................................... 499
3.7 MATLAB Examples

bilinear.m ........................................... 523
diet.m ............................................. 524
facility.m ........................................ 526
feasopt.m ......................................... 528
fixanddive.m ..................................... 529
gc_pwl.m ......................................... 530
gc_pwl_func.m .................................... 532
genconstr.m ....................................... 534
intlinprog.m ...................................... 536
linprog.m .......................................... 540
lp.m ................................................ 545
lp2.m ............................................... 545
lpmethod.m ........................................ 546
lpmad.m ........................................... 546
mip1.m ............................................. 548
mip2.m ............................................. 548
multiobj.m ......................................... 550
opttoolbox_lp.m ................................... 551
opttoolbox_mip1.m .................................. 552
params.m .......................................... 553
piecewise.m ....................................... 554
poolsearch.m ...................................... 555
qcp.m ............................................... 556
qp.m ................................................ 558
sensitivity.m ...................................... 558
sos.m ................................................ 560
sudoku.m .......................................... 560
workforce1.m ...................................... 563
workforce2.m ...................................... 565
workforce3.m ...................................... 567
workforce4.m ...................................... 569
workforce5.m ...................................... 573

3.8 R Examples ........................................... 576
The Gurobi™ distribution includes an extensive set of examples that illustrate commonly used features of the Gurobi libraries. Most examples have versions for C, C++, C#, Java, Visual Basic, and Python. A few, however, illustrate features that are specific to the Python interface.

The distribution also includes examples for our MATLAB® and R interfaces. Note, however, that our interfaces to these languages are built around the assumption that you will use the rich matrix-oriented capabilities of the underlying languages to build your optimization models. Thus, our examples for these languages don’t attempt to show you how to build models. We have instead chosen to provide a few simple examples that demonstrate how to pass matrices into our interface.

This document provides a brief tour of these examples. We won’t go through each example in detail. Instead, we’ll start with an Overview of the set of tasks that you are likely to want to perform with the Gurobi Optimizer. Later sections will then describe how specific examples accomplish each of these tasks. Alternatively, we provide a Structured List of all of our examples, which you can use to dive directly into an example of interest to you. In either case, we suggest that you browse the example source code (in a text editor, or in another browser window) while reading this document. This document includes Source Code for all of the examples, in all available languages. Source files are also available in the examples directory of the Gurobi distribution.

If you would like further details on any of the Gurobi routines used in these examples, please consult the Gurobi Reference Manual.
This document provides a quick guided tour of the Gurobi examples; we will try to highlight some of the most important features of these examples. Full source code is provided in this document, so you are free to explore the examples in full detail.

Wherever possible, we try to discuss the examples in a manner that is independent of programming languages. We will refer to each example using a brief, language independent name. You will need to map this name to the specific source file name for your language. For example, the facility example corresponds to six different implementations, one in C (facility_c.c), one in C++ (facility_cc++.cpp), one in Java (Facility.java), one in C# (facility_cs.cs), one in Visual Basic (facility_vb.vb), and one in Python (facility.py). If you would like to look at the language implementation for a particular example, please refer to the appropriate example source file.

**Topics covered in the examples**

The easiest place to start your introduction to the Gurobi examples is probably with the examples that load and solve a model from a file. These demonstrate the most basic capabilities of the Gurobi libraries. They also demonstrate the use of model attributes, which are an important concept in the Gurobi optimizer.

Once you are comfortable with these examples, you should move on to the examples that build a model from scratch. These show you how to create variables and constraints, and add them to an optimization model.

The next topic covered in this document is model modification. The Gurobi distribution includes examples that add and remove constraints, add variables, and change variable types, bounds and objective coefficients. You modify a model in much the same way that you build a model from scratch, but there are some important differences involving the use of the solution information.

Next, this document covers parameter changes. The params example shows you how to change parameters, and in particular how to use different parameter settings for different models.

On a related note, the tuning section demonstrates the use of our automated tuning tool. This tool searches for parameter settings that improve performance on a particular model.

The infeasibility section considers a few examples that cope with model infeasibility. Some use an Irreducible Inconsistent Subsystem (IIS) to handle the infeasibility, while others relax constraints.

One useful MIP feature that is worth understanding is MIP starts. A MIP start allows you to specify a known feasible solution to the MIP solver. The solution provides a bound on the objective of the best possible solution, which can help to limit the MIP search. The solution also provides a potential start point for the local search heuristics that are utilized by the Gurobi MIP solver.

It is possible to achieve model-data separation when using our Python interface, as is often done in modeling languages, but you need to make use of Python modules to do so. The model-data separation section provides an example of how this is done. It considers three versions of the diet example. All three use the same function to formulate and solve the actual optimization model, but they obtain model data from very different places.
The final topic we cover in this document is **Gurobi callbacks**. Callbacks allow the user to obtain periodic progress information related to the optimization.

### 2.1 A list of the Gurobi examples

We recommend that you begin by reading the overview of the examples (which begins in the next section). However, if you’d like to dive directly into a specific example, the following is a list of all of the examples included in the Gurobi distribution, organized by basic function. The source for the examples can be found by following the provided links, or in the **examples** directory of the Gurobi distribution.

**Read a model from a file**

- **lp** - A very simple example that reads a continuous model from a file, optimizes it, and writes the solution to a file. If the model is infeasible, it writes an Irreducible Inconsistent Subsystem (IIS) instead. C, C++, C#, Java, Python, R, VB.

- **mip2** - Reads a MIP model from a file, optimizes it, and then solves the fixed version of the MIP model. C, C++, C#, Java, Python, VB.

**Build a simple model**

- **mip1** - Builds a trivial MIP model, solves it, and prints the solution. C, C++, C#, Java, MATLAB, Python, R, VB.

- **qp** - Builds a trivial QP model, solves it, converts it to an MIQP model, and solves it again. C, C++, C#, Java, MATLAB, Python, R, VB.

- **qcp** - Builds and solves a trivial QCP model. C, C++, C#, Java, MATLAB, Python, R, VB.

- **bilinear** - Builds and solves a trivial bilinear model. C, C++, C#, Java, MATLAB, Python, R, VB.

- **sos** - Builds and solves a trivial SOS model. C, C++, C#, Java, MATLAB, Python, R, VB.

- **dense** - Solves a model stored using dense matrices. We don’t recommend using dense matrices, but this example may be helpful if your data is already in this format. C, C++, C#, Java, Python, VB.

- **genconstr** - Demonstrates the use of simple general constraints. C, C++, C#, Java, MATLAB, Python, R, VB.

- **matrix1** - Python-only example that illustrates the matrix-oriented Python interface. matrix1.py.

- **multiobj** - Demonstrates the use of multi-objective optimization. C, C++, C#, Java, MATLAB, Python, R, VB.

- **piecewise** - Demonstrates the use of piecewise-linear objective functions. C, C++, C#, Java, MATLAB, Python, R, VB.
• **gc_pwl** - Demonstrates the use of piecewise-linear constraint. C, C++, C#, Java, MATLAB, Python, R, VB.

• **gc_pwl_func** - Demonstrates the use of function constraints. C, C++, C#, Java, MATLAB, Python, R, VB.

• **poolsearch** - Demonstrates the use of solution pools. C, C++, C#, Java, MATLAB, Python, R, VB.

A few simple applications

• **diet** - Builds and solves the classic diet problem. Demonstrates model construction and simple model modification - after the initial model is solved, a constraint is added to limit the number of dairy servings. C, C++, C#, Java, MATLAB, Python, R, VB.

• **diet2, diet3, diet4, dietmodel** - Python-only variants of the diet example that illustrate model-data separation. diet2.py, diet3.py, diet4.py, dietmodel.py.

• **facility** - Simple facility location model: given a set of plants and a set of warehouses, with transportation costs between them, this example finds the least expensive set of plants to open in order to satisfy product demand. This example demonstrates the use of MIP starts -- the example computes an initial, heuristic solution and passes that solution to the MIP solver. C, C++, C#, Java, MATLAB, Python, R, VB.

• **matrix2** - Python-only example that solves the n-queens problem using the matrix-oriented Python interface. matrix2.py.

• **netflow** - A Python-only example that solves a multi-commodity network flow model. It demonstrates the use of several Python modeling constructs, including dictionaries, tuples, tupledict, and tuplelist objects. netflow.py.

• **portfolio** - A Python-only example that solves a financial portfolio optimization model, where the historical return data is stored using the pandas package and the result is plotted using the matplotlib package. It demonstrates the use of pandas, NumPy, and Matplotlib in conjunction with Gurobi. portfolio.py.

• **sudoku** - Reads a Sudoku puzzle dataset from a file, builds a MIP model to solve that model, solves it, and prints the solution. C, C++, C#, Java, MATLAB, Python, R, VB.

• **workforce1** - Formulates and solves a workforce scheduling model. If the model is infeasible, the example computes and prints an Irreducible Inconsistent Subsystem (IIS). C, C++, C#, Java, MATLAB, Python, R, VB.

• **workforce2** - An enhancement of workforce1. This example solves the same workforce scheduling model, but if the model is infeasible, it computes an IIS, removes one of the associated constraints from the model, and re-solves. This process is repeated until the model becomes feasible. Demonstrates constraint removal. C, C++, C#, Java, MATLAB, Python, R, VB.
- **workforce3** - A different enhancement of workforce1. This example solves the same workforce scheduling model, but if the model is infeasible, it adds artificial variables to each constraint and minimizes the sum of the artificial variables. This corresponds to finding the minimum total change in the right-hand side vector required in order to make the model feasible. Demonstrates variable addition. C, C++, C#, Java, MATLAB, Python, R, VB.

- **workforce4** - An enhancement of workforce3. This example solves the same workforce scheduling model, but it starts with artificial variables in each constraint. It first minimizes the sum of the artificial variables. Then, it introduces a new quadratic objective to balance the workload among the workers. Demonstrates optimization with multiple objective functions. C, C++, C#, Java, MATLAB, Python, R, VB.

- **workforce5** - An alternative enhancement of workforce3. This example solves the same workforce scheduling model, but it starts with artificial variables in each constraint. It formulates a multi-objective model where the primary objective is to minimize the sum of the artificial variables (uncovered shifts), and the secondary objective is to minimize the maximum difference in the number of shifts worked between any pair of workers. Demonstrates multi-objective optimization. C, C++, C#, Java, MATLAB, Python, R, VB.

**Illustrating specific features**

- **feasopt** - Reads a MIP model from a file, adds artificial slack variables to relax each constraint, and then minimizes the sum of the artificial variables. It then computes the same relaxation using the feasibility relaxation feature. The example demonstrates simple model modification by adding slack variables. It also demonstrates the feasibility relaxation feature. C, C++, C#, Java, MATLAB, Python, R, VB.

- **lpmethod** - Demonstrates the use of different LP algorithms. Reads a continuous model from a file and solves it using multiple algorithms, reporting which is the quickest for that model. C, C++, C#, Java, MATLAB, Python, R, VB.

- **lpmod** - Demonstrates the use of advanced starts in LP. Reads a continuous model from a file, solves it, and then modifies one variable bound. The resulting model is then solved in two different ways: starting from the solution of the original model, or restarting from scratch. C, C++, C#, Java, MATLAB, Python, R, VB.

- **params** - Demonstrates the use of Gurobi parameters. Reads a MIP model from a file, and then spends 5 seconds solving the model with each of four different values of the MIPFocus parameter. It compares the optimality gaps for the four different runs, and continues with the MIPFocus value that produced the smallest gap. C, C++, C#, Java, MATLAB, Python, R, VB.

- **sensitivity** - MIP sensitivity analysis. Reads a MIP model, solves it, and then computes the objective impact of fixing each binary variable in the model to 0 or 1. Demonstrates the multi-scenario feature. C, C++, C#, Java, MATLAB, Python, R, VB.

- **tune** - Uses the parameter tuning tool to search for improved parameter settings for a model. C, C++, C#, Java, Python, VB.
• **fixanddive** - Implements a simple MIP heuristic. It reads a MIP model from a file, relaxes the integrality conditions, and then solves the relaxation. It then chooses a set of integer variables that take integer or nearly integer values in the relaxation, fixes them to the nearest integer, and solves the relaxation again. This process is repeated until the relaxation is either integer feasible or linearly infeasible. The example demonstrates different types of model modification (relaxing integrality conditions, changing variable bounds, etc.). C, C++, C#, Java, MATLAB, Python, R, VB.

• **multiscenario** - Simple facility location model: given a set of plants and a set of warehouses, with transportation costs between them, this example finds the least expensive set of plants to open in order to satisfy product demand. Since the plant fixed costs and the warehouse demands are uncertain, multiple scenarios are created to capture different possible values. A multi-scenario model is constructed and solved, and the solutions for the different scenarios are retrieved and displayed. C, C++, C#, Java, Python, VB.

• **batchmode** - Demonstrates the use of batch optimization. C, C++, C#, Java, Python, VB.

• **workforce_batchmode** - Python-only example which formulates a workforce scheduling model. The model is solved using batch optimization. The VTag attribute is used to identify the set of variables whose solution information is needed to construct the schedule. workforce_batchmode.py.

**More advanced features**

• **tsp** - Solves a traveling salesman problem using lazy constraints. C, C++, C#, Java, Python, VB.

• **callback** - Demonstrates the use of Gurobi callbacks. C, C++, C#, Java, Python, VB.

### 2.2 Load and solve a model from a file

**Examples:** batchmode, callback, feasopt, fixanddive, lp, lpmethod, lpmod, mip2, params, sensitivity

One of the most basic tasks you can perform with the Gurobi libraries is to read a model from a file, optimize it, and report the result. The lp (lp_c.c, lp_c++.cpp, lp_cs.cs, Lp.java, lp.py, lp_vb.vb) and mip2 (mip2_c.c, mip2_c++.cpp, mip2_cs.cs, Mip2.java, mip2.m, mip2.py, mip2.R, mip2_vb.vb) examples are simple illustrations of how this is done in the various supported Gurobi languages. While the specifics vary from one language to another, the basic structure remains the same for all languages.

After initializing the Gurobi environment, the examples begin by reading the model from the specified file. In C, you call the GRBreadmodel() function:

```c
error = GRBreadmodel(masterenv, argv[1], &model);
```

In C++, this is done by constructing a GRBModel object:

```cpp
GRBModel model = GRBModel(env, argv[1]);
```

In C# and Java, this is also done by constructing a GRBModel object:
GRBModel model = new GRBModel(env, args[0]);

In Python, this is done via the `read` global function:

```python
    model = read(sys.argv[1])
```

The next step is to invoke the Gurobi optimizer on the model. In C, you call `GRBoptimize()` on the `model` variable:

```c
    error = GRBoptimize(model);
```

In C++, Java, and Python, this is accomplished by calling the `optimize` method on the `model` object:

```java
    model.optimize();
```

In C#, the first letter of the method name is capitalized:

```csharp
    model.Optimize();
```

A successful `optimize` call populates a set of solution attributes in the model. For example, once the call completes, the `X` variable attribute contains the solution value for each variable. Similarly, for continuous models, the `Pi` constraint attribute contains the dual value for each constraint.

The examples then retrieve the value of the model `Status` attribute to determine the result of the optimization. In the `lp` example, an optimal solution is written to a solution file (`model.sol`).

There are many other things you can do once you have read and solved the model. For example, `lp` checks the solution status -- which is highly recommended. If the model is found to be infeasible, this example computes an Irreducible Inconsistent Subsystem (IIS) to isolate the source of the infeasibility.

### 2.3 Build a model

**Examples:** bilinear, diet, facility, gc_pwl, gc_pwl_func, genconstr, matrix1, mip1, multiobj, multisenario, piecewise, poolsearch, qcp, qp, sensitivity, sos, sudoku, workforce1, workforce2, workforce3, workforce4, workforce5

Several of the Gurobi examples build models from scratch. We start by focusing on two: `mip1` and `sos`. Both build very simple models to illustrate the basic process.

Typically, the first step in building a model is to create an empty model. This is done using the `GRBnewmodel` function in C:

```c
    error = GRBnewmodel(env, &model, "mip1", 0, NULL, NULL, NULL, NULL);
```

You can optionally create a set of variables when you create the model, as well as specifying bounds, objective coefficients, and names for these variables. These examples add new variables separately.

In C++, C#, and Java, you create a new model using the `GRBModel` constructor. In Java, this looks like:

```java
    GRBModel model = new GRBModel(env);
```
In Python, the class is called `Model`, and its constructor is similar to the `GRBModel` constructor for C++ and Java.

Once the model has been created, the typical next step is to add variables. In C, you use the `GRBaddvars` function to add one or more variables:

```c
    error = GRBaddvars(model, 3, 0, NULL, NULL, NULL, obj, NULL, NULL, vtype, NULL);
```

In C++, Java, and Python, you use the `addVar` method on the `Model` object ( `AddVar` in C#). In Java, this looks like:

```java
    GRBVar x = model.addVar(0.0, 1.0, -1.0, GRB.BINARY, "x");
```

The new variable’s lower bound, upper bound, objective coefficient, type, and name are specified as arguments. In C++ and Python, you can omit these arguments and use default values; see the Gurobi Reference Manual for details.

The next step is to add constraints to the model. Linear constraints are added through the `GRBaddconstr` function in C:

```c
    error = GRBaddconstr(model, 3, ind, val, GRB_LESS_EQUAL, 4.0, "c0");
```

To add a linear constraint in C, you must specify a list of variable indices and coefficients for the left-hand side, a sense for the constraint (e.g., `GRB_LESS_EQUAL`), and a right-hand side constant. You can also give the constraint a name; if you omit the name, Gurobi will assign a default name for the constraint.

In C++, C#, Java, and Python, you build a linear constraint by first building linear expressions for the left- and right-hand sides. In Java, which doesn’t support operator overloading, you build an expression as follows:

```java
    GRBLinExpr expr = new GRBLinExpr();
    expr.addTerm(1.0, x); expr.addTerm(2.0, y); expr.addTerm(3.0, z);
```

You then use the `addConstr` method on the `GRBModel` object to add a constraint using these linear expressions for the left- and right-hand sides:

```c
    model.addConstr(expr, GRB_LESS_EQUAL, 4.0, "c0");
```

For C++, C#, and Python, the standard mathematical operators such as `+`, `*`, `<=` have been overloaded so that the linear expression resembles a traditional mathematical expression. In C++:

```cpp
    model.addConstr(x + 2 * y + 3 * z <= 4, "c0");
```

Once the model has been built, the typical next step is to optimize it (using `GRBoptimize` in C, `model.optimize` in C++, Java, and Python, or `model.Optimize` in C#). You can then query the `X` attribute on the variables to retrieve the solution (and the `VarName` attribute to retrieve the variable name for each variable). In C, the `X` attribute is retrieved as follows:

```c
    error = GRBgetdblattrarray(model, GRB_DBL_ATTR_X, 0, 3, sol);
```

In C++:
cout << x.get(GRB_StringAttr_VarName) << " " << x.get(GRB_DoubleAttr_X) << endl;
cout << y.get(GRB_StringAttr_VarName) << " " << y.get(GRB_DoubleAttr_X) << endl;
cout << z.get(GRB_StringAttr_VarName) << " " << z.get(GRB_DoubleAttr_X) << endl;

In Java:
System.out.println(x.get(GRB.StringAttr.VarName) + " " + x.get(GRB.DoubleAttr.X));
System.out.println(y.get(GRB.StringAttr.VarName) + " " + y.get(GRB.DoubleAttr.X));
System.out.println(z.get(GRB.StringAttr.VarName) + " " + z.get(GRB.DoubleAttr.X));

In C#:
Console.WriteLine(x.Get(GRB.StringAttr.VarName) + " " + x.Get(GRB.DoubleAttr.X));
Console.WriteLine(y.Get(GRB.StringAttr.VarName) + " " + y.Get(GRB.DoubleAttr.X));
Console.WriteLine(z.Get(GRB.StringAttr.VarName) + " " + z.Get(GRB.DoubleAttr.X));

In Python:
for v in m.getVars():
    print(v.varName, v.x)

When querying or modifying attribute values for an array of constraints or variables, it is generally more efficient to perform the action on the whole array at once. This is quite natural in the C interface, where most of the attribute routines take array arguments. In the C++, C#, Java, and Python interfaces, you can use the get and set methods on the GRBModel object to work directly with arrays of attribute values (getAttr/setAttr in Python). In the sudoku Java example, this is done as follows:

double[][][] x = model.get(GRB.DoubleAttr.X, vars);

We should point out one important subtlety in our interface. We use a lazy update approach to building and modifying a model. When you make changes, they are added to a queue. The queue is only flushed when you optimize the model (or write it to a file). In the uncommon situation where you want to query information about your model before optimizing it, you should call the update method before making your query.

2.4 Additional modeling elements

Examples: bilinear, gc_pwl, gc_pwl_func, genconstr, multiobj, multiscenario, piecewise, qcp, qp, sensitivity, sos
A mathematical programming model in its traditional form consists of a linear objective, a set of linear constraints, and a set of continuous or integer decision variables. Gurobi supports a number of additional modeling constructs. In addition to linear constraints, Gurobi also supports SOS constraints, quadratic constraints, and general constraints. In addition to a single linear objective, Gurobi also supports quadratic objectives, piecewise-linear objectives, and multiple linear objectives. Consult the corresponding examples from the Gurobi distribution for simple examples of how to use each of these modeling elements.

2.5 Modify a model

Examples: diet, feasopt, fixanddive, gc_pwl_func, lpmod, sensitivity, workforce3, workforce4, workforce5

This section considers model modification. Modification can take many forms, including adding constraints or variables, deleting constraints or variables, modifying constraint and variable attributes, changing constraint coefficients, etc. The Gurobi examples don’t cover all possible modifications, but they cover the most common types.

diet

This example builds a linear model that solves the classic diet problem: to find the minimum cost diet that satisfies a set of daily nutritional requirements. Once the model has been formulated and solved, it adds an additional constraint to limit the number of servings of dairy products and solves the model again. Let’s focus on the model modification.

Adding constraints to a model that has already been solved is no different from adding constraints when constructing an initial model. In Python, we can introduce a limit of 6 dairy servings through the following constraint:

```python
m.addConstr(buy['milk'] + buy['ice cream'] <= 6, "limit_dairy")
```

For linear models, the previously computed solution can be used as an efficient warm start for the modified model. The Gurobi solver retains the previous solution, so the next optimize call automatically starts from the previous solution.

lpmod

Changing a variable bound is also straightforward. The lpmod example changes a single variable bound, then re-solves the model in two different ways. A variable bound can be changed by modifying the UB or LB attribute of the variable. In C:

```c
error = GRBsetdblattrelement(model, GRB_DBL_ATTR_UB, var, 0.0);
```

In Python:

```python
minVar.ub = 0
```

The model is re-solved simply by calling the optimize method again. For a continuous model, this starts the optimization from the previous solution. To illustrate the difference when solving the model from an initial, unsolved state, the lpmod example calls the reset function. In C:

```c
error = GRBreset(model, 0);
```

In C++, Java, and Python:
m.reset(0)

In C#:

m.Reset(0)

When we call the optimize method after resetting the model, optimization starts from scratch. Although the difference in computation time is insignificant for this tiny example, a warm start can make a big difference for larger models.

fixanddive

The fixanddive example provides another example of bound modification. In this case, we repeatedly modify a set of variable bounds, utilizing warm starts each time. In C, variables are fixed as follows:

```c
for (j = 0; j < nfix; ++j)
{
    fixval = floor(fractional[j].X + 0.5);
    error = GRBsetdblattrelement(model, "LB", fractional[j].index, fixval);
    if (error) goto QUIT;
    error = GRBsetdblattrelement(model, "UB", fractional[j].index, fixval);
    if (error) goto QUIT;
}
```

In Python, they are fixed as follows:

```python```
for i in range(nfix):
    v = fractional[i]
    fixval = int(v.x + 0.5)
    v.lb = fixval
    v.ub = fixval
```

Again, the subsequent call to optimize starts from the previous solution.

sensitivity

The sensitivity example computes the optimal objective value associated with fixing each binary variable to 0 or 1. It first solves the given model to optimality. It then constructs a multi-scenario model, where in each scenario a binary variable is fixed to the complement of the value it took in the optimal solution. The resulting multi-scenario model is solved, giving the objective degradation associated with forcing each binary variable off of its optimal value.

feasopt

The last modification example we consider is feasopt, which adds variables to existing constraints and also changes the optimization objective. Setting the objective to zero is straightforward: simply call setObjective with a zero argument:

```c```
m.setObjective(0)
```

Adding new variables is somewhat more complex. In the example, we want to add artificial variable(s) to each constraint in order to allow the constraint to be relaxed. We use two artificial variables for equality constraints and one for inequality constraints. The Python code for adding a single artificial variable to constraint \( c \) is:
feasModel.addVar(obj=1.0, name="ArtP_" + c.Constrname, column=Column([1], [c]))

We use the `column` argument of the `addVar` method to specify the set of constraints in which the new variable participates, as well as the associated coefficients. In this example, the new variable only participates in the constraint to be relaxed. Default values are used here for all variables attributes except the objective and the variable name.

### 2.6 Change parameters

**Examples:** batchmode, callback, fixanddive, gc_pwl_func, lp, lpmethod, mip2, multiscenario, params, sensitivity, workforce_batchmode

This section illustrates the use of Gurobi parameters. Example `params` reads a MIP model from a file, then solves the model using four different values of the `MIPFocus` parameter, running for five seconds per value (`MIPFocus` chooses the high-level strategy that the MIP solver uses to solve the problem). It then chooses the parameter value that produced the smallest MIP gap, and continues solving the model until it achieves optimality.

The mechanics of setting a parameter are quite simple. To set the `MIPFocus` parameter in C, do the following:

```c
GRBsetintparam(GRBgetenv(model), GRB_INT_PAR_MIPFOCUS, i);
```

In C++:

```c++
model.setIntParam(GRB_IntParam_MIPFocus, i);
```

In Java:

```java
model.setIntParam(GRB.IntParam.MIPFocus, i);
```

In C#:

```csharp
model.Parameters.MIPFocus = 1
```
or

```csharp
model.Set(GRB.IntParam.MIPFocus, i);
```

In Python:

```python
model.Params.MIPFocus = i
```

We should add a comment on how parameter settings propagate between different models. When we set the `TimeLimit` parameter on the base model, then make a copy of that model, the parameter setting is carried over to the copy. When we set the `MIPFocus` parameter on the copy, that parameter change has no effect on the other copies, nor on the original model.

### 2.7 Automated parameter tuning

**Example:** `tune`

The next example we consider is `tune`, which demonstrates the use of our automated parameter tuning tool. This tool searches for parameter settings that improve performance on a model. While
you would typically invoke the tool through the command line, using our grbtune program, it can also be invoked from our APIs. We’ll provide only a cursory description of the tool here. We recommend that you consult the Parameter Tuning Tool section of the Gurobi Reference Manual for more precise details.

Our tuning example demonstrates a typical use of the tuning tool. You would start by invoking the tool on a model. In C:

```c
error = GRBtunemodel(model);
```

In Java:

```java
model.tune();
```

This routine solves the model multiple times, with different parameter settings, to find settings that improve performance.

Once tuning is complete, you would then use GetTuneResult to retrieve the result. In C:

```c
error = GRBgettuneresult(model, 0);
```

In Java:

```java
model.getTuneResult(0);
```

The numerical argument indicates which tuning result to retrieve (0 is the best result, 1 is the second-best, etc.). This routine loads the requested parameter set into the environment associated with the argument model.

Once the tune parameter settings have been loaded into the model, you can then call Optimize to use these parameters to solve the model, or you can call Write to write these parameters to a .prm file.

### 2.8 Diagnose and cope with infeasibility

**Examples:** feasopt, lp, multiscenario, sensitivity, workforce1, workforce2, workforce3

When solving optimization models, there are some situations where the specified constraints cannot be satisfied. When this happens, you often need to either identify and repair the root cause of the infeasibility, or alternatively find a set of constraints to relax in order to obtain a feasible model. The workforce1, workforce2, and workforce3 illustrate these different strategies.

Starting with the simplest of the three examples, workforce1 formulates a simple workforce scheduling model and solves it. If the model is infeasible, it computes an Irreducible Inconsistent Subsystem (IIS). The user can then inspect this information to understand and hopefully address the source of the infeasibility in the model.

Example workforce2 is similar, except that if the model is infeasible, the example repeatedly identifies an IIS and removes one of the associated constraints from the model until the model becomes feasible. Note that it is sufficient to remove one constraint from the IIS to address that source of infeasibility, but that one IIS may not capture all sources of infeasibility. It is therefore necessary to repeat the process until the model is feasible.

Example workforce3 takes a different approach to addressing infeasibility. Rather than identifying and removing IIS members, it allows the constraints of the model to be relaxed. Like the feasopt example, an artificial variable is added to each constraint. The example sets the objective
on the original variables to zero, and then solves a model that minimizes the total magnitude of the constraint relaxation.

The feasopt example demonstrates another approach to relaxing an infeasible model. It computes a feasibility relaxation for the infeasible model. A feasibility relaxation is a model that, when solved, minimizes the amount by which the solution violates the bounds and linear constraints of the original model. This method is invoked as follows:

In C:

```c
error = GRBfeasrelax(feasmodel, GRB_FEASRELAX_LINEAR, 1,
                      NULL, NULL, rhspen, &feasobj);
```

In C++:

```cpp
feasmodel1.feasRelax(GRB_FEASRELAX_LINEAR, true, false, true);
```

In C#:

```csharp
feasmodel1.FeasRelax(GRB.FEASRELAX_LINEAR, true, false, true);
```

In Java:

```java
feasmodel1.feasRelax(GRB.FEASRELAX_LINEAR, true, false, true);
```

In Python:

```python
feasmodel1.feasRelaxS(0, True, False, True)
```

The arguments to this method select the objective function for the relaxed model, the specific set of bounds and constraints that are allowed to be relaxed, and the penalties for relaxing specific bounds and constraints.

### 2.9 MIP starts

**Example:** facility, sensitivity

A MIP modeler often knows how to compute a feasible solution to their problem. In cases where the MIP solver is slow in finding an initial feasible solution, it can be helpful for the modeler to provide a feasible solution along with the model itself. This is done through the `Start` attribute on the variables. This is illustrated in the `facility` example.

The `facility` example solves a simple facility location problem. The model contains a set of warehouses, and a set of plants that produce the products required in the warehouses. Each plant has a maximum production capacity and a fixed operating cost. Additionally, there is a cost associated with shipping products from a plant to a warehouse. The goal is to decide which plants should satisfy the demand for the product, given the associated capacities and costs.

The example uses a simple heuristic for choosing an initial solution: it closes the plant with the highest fixed cost. The associated solution may not be optimal, but it could produce a reasonable starting solution for the MIP optimization. The MIP start is passed to the MIP solver by setting the `Start` attribute before the optimization begins. In C, we set the start attribute to open all plants using the following code:
/* First, open all plants */
for (p = 0; p < nPlants; ++p)
{
    error = GRBsetdblattrelement(model, "Start", opencol(p), 1.0);
    if (error) goto QUIT;
}

In Python:

# First open all plants
for p in plants:
    open[p].start = 1.0

When you run the example, the MIP solver reports that the start produced a feasible initial solution:

User MIP start produced solution with objective 210500 (0.01s)
Loaded user MIP start with objective 210500

This initial solution turns out to be optimal for the sample data. Although the computation difference is insignificant for this tiny example, providing a good starting solution can sometimes help for more difficult models.

Note that the MIP start in this example only specifies values for some of the variables -- the variables that determine which plants to leave open and which plants to close. The Gurobi MIP solve uses whatever start information is provided to try to construct a complete solution.

2.10 Model-data separation in Python

Examples: diet2.py, diet3.py, diet4.py

When building an optimization model in a modeling language, it is typical to separate the optimization model itself from the data used to create an instance of the model. These two model ingredients are often stored in completely different files. We show how a similar result can be achieved in our Python interface with our diet2.py, diet3.py, and diet4.py examples. These examples illustrate alternate approaches to providing data to the optimization model: diet2.py embeds the data in the source file, diet3.py reads the data from an SQL database (using the Python sqlite3 package), and diet4.py reads the data from an Excel spreadsheet (using the Python xlrd package). dietmodel.py contains the optimization model itself. The same model is used by diet2.py, diet3.py, and diet4.py.

The key construct that enables the separation of the model from the data is the Python module. A module is simply a set of functions and variables, stored in a file. You import a module into a program using the import statement. diet2.py, diet3.py, and diet4.py all populate a set of variables, and then pass them to the solve function of the dietmodel module using the following pair of statements:

import dietmodel
dietmodel.solve(categories, minNutrition, maxNutrition, foods, cost, nutritionValues)

The first statement imports the dietmodel module, which must be stored in file dietmodel.py in the current directory. The second passes the model data to the solve function in the newly imported module.
2.11 Callbacks

Example: callback

The final example we consider is callback, which demonstrates the use of Gurobi callbacks. Callbacks are used to report on the progress of the optimization or to modify the behavior of the Gurobi solver. To use a callback, the user writes a routine that implements the desired behavior. The routine is passed to the Gurobi optimizer when optimization begins, and the routine is called regularly during the optimization process. One argument of the user routine is a where value, which indicates from where in the optimization process the callback is invoked. The user callback routine can call the optimization library to query certain values. We refer the reader to the callback section of the Gurobi Reference Manual for more precise details.

Our callback example implements a simple termination scheme: the user passes a node count into the callback, and the callback asks the optimizer to terminate when that node count is reached. This is implemented in C as follows:

```c
GRBcbget(cbdata, where, GRB_CB_MIP_NODCNT, &nodecnt);
if (nodecnt > limit)
    GRBterminate(model);
```

In Python, this is implemented as:

```python
nodecnt = model.cbGet(GRB.Callback.MIP_NODCNT)
if nodecnt > model._mynodelimit:
    model.terminate()
```

To obtain the current node count, the user routine calls the cbget routine (the GRBcbget function in C, or the cbGet method on the model object in C++, C#, Java, and Python).

Our callback example also prints progress information. In C:

```c
GRBcbget(cbdata, where, GRB_CB_MIP_NODCNT, &nodecnt);
if (nodecnt - mydata->lastmsg >= 100) {
    ...  
    printf("%7.0f ...", nodecnt, ...);
}
```

In Python:

```python
nodecnt = model.cbGet(GRB.Callback.MIP_NODCNT)
if nodecnt % 100 == 0:
    print(int(nodecnt), "...")
```

Again, the user callback calls the cbGet routine to query the state of the optimization.
We have included source code for all of the distributed examples in this section. The identical example source code is included in the `examples` directory in the Gurobi distribution.

### 3.1 C Examples

This section includes source code for all of the Gurobi C examples. The same source code can be found in the `examples/c` directory of the Gurobi distribution.

**batchmode.c**

```c
/* Copyright 2020, Gurobi Optimization, LLC */
/* This example reads a MIP model from a file, solves it in batch mode, *
 * and prints the JSON solution string. */

#include <stdlib.h>
#include <stdio.h>
#include <time.h>
#include <string.h>
#if defined(WIN32) || defined(WIN64)
#include <Windows.h>
#define sleep(n) Sleep(1000*n)
#else
#include <unistd.h>
#endif
#include "gurobi_c.h"

/* setup gurobi environment */
int setupbatchconnection(GRBenv **envP)
{
    int error = 0;
    GRBenv *env = NULL;

    /* setup a batch environment */
    error = GRBemptyenv(envP);
    if (error) goto QUIT;
    env = *envP;
    error = GRBsetintparam(env, "CSBatchMode", 1);
    if (error) goto QUIT;
    error = GRBsetstrparam(env, "LogFile", "batchmode.log");
    if (error) goto QUIT;
    error = GRBsetstrparam(env, "CSManager", "http://localhost:61080");
    if (error) goto QUIT;
    error = GRBsetstrparam(env, "UserName", "gurobi");
    if (error) goto QUIT;
    error = GRBsetstrparam(env, "ServerPassword", "pass");

QUIT: return error;
}
```

if (error) goto QUIT;
error = GRBstartenv(env);
if (error) goto QUIT;

QUIT:

if (error) {
    printf("Failed to setup environment, error code %d\n", error);
} else {
    printf("Successfully created environment\n");
    return error;
}

/* display batch-error code if any */
void batcherrorinfo(GRBbatch *batch)
{
    int error = 0;
    int errorCode;
    char *errorMsg;
    char *BatchID;

    if (!batch) goto QUIT;

    /* query the last error code */
    error = GRBgetbatchintattr(batch, "BatchErrorCode", &errorCode);
    if (error || !errorCode) goto QUIT;

    /* query the last error message */
    error = GRBgetbatchstrattr(batch, "BatchErrorMessage", &errorMsg);
    if (error) goto QUIT;

    error = GRBgetbatchstrattr(batch, "BatchID", &BatchID);
    if (error) goto QUIT;

    printf("Batch ID %s Error Code %d (%s)\n", BatchID, errorCode, errorMsg);

QUIT:
    return;
}

/* create a batch request for given problem file */
int newbatchrequest(const char *filename, char *BatchID)
{
    int error = 0;
    GRBenv *env = NULL;
    GRBenv *menv = NULL;
    GRBmodel *model = NULL;
    char tag[128];
    int cols, j;

    /* setup a batch connection */
    error = setupbatchconnection(&env);
    if (error) goto QUIT;

18
/* read a model */
error = GRBreadmodel(env, filename, &model);
if (error) goto QUIT;

/* set some params */
menv = GRBgetenv(model);
error = GRBsetdblparam(menv, "MIPGap", 0.01);
if (error) goto QUIT;

/* for extra detailed information on JSON solution string */
error = GRBsetintparam(menv, "JSONSolDetail", 1);
if (error) goto QUIT;

/* setup some tags, we need tags to be able to query results later on */
error = GRBgetintattr(model, "NumVars", &cols);
if (error) goto QUIT;

if (cols > 10) cols = 10;
for (j = 0; j < cols; j++) {
    sprintf(tag, "MyUniqueVariableID%d", j);
    error = GRBsetstrattrelement(model, "VTag", j, tag);
}

/* submit batch request to the Manager */
error = GRBoptimizebatch(model, BatchID);
if (error) goto QUIT;

QUIT:

if (error) {
    printf("Failed to submit a new batch request, error code %d\n", error);
} else {
    printf("Successfully submitted new batch request %s\n", BatchID);
}
GRBfreemodel(model);
GRBfreeenv(env);
return error;

/* wait for final bstatus */
int waitforfinalstatus(const char *BatchID)
{
    int error = 0;
    GRBenv *env = NULL;
    GRBbatch *batch = NULL;
    time_t start, current;
    int bstatus;

    /* setup a batch connection */
    error = setupbatchconnection(&env);
    if (error) goto QUIT;

    /* create batch-object */
    error = GRBgetbatch(env, BatchID, &batch);
    if (error) goto QUIT;

    /* query the final status */
    error = GRBgetbstatus(batch, &bstatus);
    if (error) {
        printf("Failed to get final status of batch request %s, error code %d\n", BatchID, error);
    } else {
/* query bstatus, and wait for completed */
error = GRBgetbatchintattr(batch, "BatchStatus", &bstatus);
if (error) goto QUIT;
start = time(NULL);

while (bstatus == GRB_BATCH_SUBMITTED) {
    /* abort if taking too long */
    current = time(NULL);
    if (current - start >= 3600) {
        /* request to abort the batch */
        error = GRBabortbatch(batch);
        if (error) goto QUIT;
    }
    /* do not bombard the server */
    sleep(1u);
    /* update local attributes */
    error = GRBupdatebatch(batch);
    if (error) goto QUIT;
    /* query bstatus */
    error = GRBgetbatchintattr(batch, "BatchStatus", &bstatus);
    if (error) goto QUIT;
    /* deal with failed bstatus */
    if (bstatus == GRB_BATCH_FAILED) {
        /* retry the batch request */
        error = GRBretrybatch(batch);
        if (error) goto QUIT;
        bstatus = GRB_BATCH_SUBMITTED;
    }
}

QUIT:
if (error) {
    printf("Failed to wait for final bstatus, error code %d\n", error);
} else {
    printf("Final Batch Status %d\n", bstatus);
}
batcherrorinfo(batch);
/* release local resources */
GRBfreebatch(batch);
GRBfreeenv(env);
return error;

/* final report on batch request */
int finalreport(const char *BatchID)
{
    int error = 0;
    GRBenv *env = NULL;
    GRBbatch *batch = NULL;
```c
char *jsonsol = NULL;
int bstatus;

/* setup a batch connection */
error = setupbatchconnection(&env);
if (error) goto QUIT;

/* create batch object */
error = GRBgetbatch(env, BatchID, &batch);
if (error) goto QUIT;

/* query bstatus, and wait for completed */
error = GRBgetbatchintattr(batch, "BatchStatus", &bstatus);
if (error) goto QUIT;

/* display depending on batch bstatus */
switch (bstatus) {
    case GRB_BATCH_CREATED:
        printf("Batch is 'CREATED'\n");
        printf("maybe batch-creation process was killed?\n");
        break;
    case GRB_BATCH_SUBMITTED:
        printf("Batch is 'SUBMITTED'\n");
        printf("Some other user re-submitted this Batch object?\n");
        break;
    case GRB_BATCH_ABORTED:
        printf("Batch is 'ABORTED'\n");
        break;
    case GRB_BATCH_FAILED:
        printf("Batch is 'FAILED'\n");
        break;
    case GRB_BATCH_COMPLETED:

        /* print JSON solution into string */
        error = GRBgetbatchjsonsolution(batch, &jsonsol);
        if (error) goto QUIT;
        printf("JSON solution: %s\n", jsonsol);

        /* save solution into a file */
        error = GRBwritebatchjsonsolution(batch, "batch-sol.json.gz");
        if (error) goto QUIT;
        break;
    default:
        printf("This should not happen, probably points to a"
            " user-memory corruption problem\n");
        exit(EXIT_FAILURE);
        break;
}

QUIT:

if (error) {
    printf("Failed to perform final report, error code %d\n", error);
} else {
    printf("Reporting done\n");
}
```
batcherrorinfo(batch);

if (jsonsol)
    GRBfree(jsonsol);

GRBfreebatch(batch);
GRBfreeenv(env);
return error;
}

/* remove batch ID from manager */
int discardbatch(const char *BatchID)
{
    int error = 0;
    GRBenv *env = NULL;
    GRBbatch *batch = NULL;

    /* setup a batch connection */
    error = setupbatchconnection(&env);
    if (error) goto QUIT;

    /* create batch object */
    error = GRBgetbatch(env, BatchID, &batch);
    if (error) goto QUIT;

    /* discard the batch object in the manager */
    error = GRBdiscardbatch(batch);
    if (error) goto QUIT;

QUIT:
    batcherrorinfo(batch);
    GRBfreebatch(batch);
    GRBfreeenv(env);
    return error;
}

int main(int argc, char **argv)
{
    int error = 0;
    char BatchID[GRB_MAX_STRLEN+1];

    /* ensure enough parameters */
    if (argc < 2) {
        fprintf(stderr, "Usage: %s filename\n", argv[0]);
        goto QUIT;
    }

    /* create a new batch request */
    error = newbatchrequest(argv[1], BatchID);
    if (error) goto QUIT;

    /* wait for final bstatus */
    error = waitforfinalstatus(BatchID);

if (error) goto QUIT;

/* query final bstatus, and if completed, print JSON solution */
error = finalreport(BatchID);
if (error) goto QUIT;

/* eliminate batch from the manager */
error = discardbatch(BatchID);
if (error) goto QUIT;

QUIT:
    return error;
}

bilinear_c.c

/* Copyright 2020, Gurobi Optimization, LLC */

/* This example formulates and solves the following simple bilinear model:
   maximize x
   subject to x + y + z <= 10
          x * y <= 2 (bilinear inequality)
          x * z + y * z == 1 (bilinear equality)
       x, y, z non-negative (x integral in second version)
   */

#include <stdlib.h>
#include <stdio.h>
#include "gurobi_c.h"

int main(int argc, char *argv[])
{
    GRBenv *env = NULL;
    GRBmodel *model = NULL;
    int error = 0;
    double sol[3];
    int ind[3];
    double val[3];
    double obj[] = {1, 0, 0};
    int qrow[2];
    int qcol[2];
    double qval[2];
    int optimstatus;
    double objval;

    /* Create environment */
    error = GRBloadenv(&env, "bilinear.log");
    if (error) goto QUIT;

    /* Create an empty model */
error = GRBnewmodel(env, &model, "bilinear", 0, NULL, NULL, NULL, NULL, NULL);
if (error) goto QUIT;

/* Add variables */
error = GRBaddvars(model, 3, 0, NULL, NULL, NULL, NULL, NULL, NULL);
if (error) goto QUIT;

/* Change sense to maximization */
error = GRBsetintattr(model, GRB_INT_ATTR_MODELSENSE, GRB_MAXIMIZE);
if (error) goto QUIT;

/* Linear constraint: x + y + z <= 10 */
ind[0] = 0; ind[1] = 1; ind[2] = 2;
val[0] = 1; val[1] = 1; val[2] = 1;
error = GRBaddconstr(model, 3, ind, val, GRB_LESS_EQUAL, 10.0, "c0");
if (error) goto QUIT;

/* Bilinear inequality: x * y <= 2 */
qrow[0] = 0; qcol[0] = 1; qval[0] = 1.0;
error = GRBaddqconstr(model, 0, NULL, NULL, 1, qrow, qcol, qval,
GRB_LESS_EQUAL, 2.0, "bilinear0");
if (error) goto QUIT;

/* Bilinear equality: x * z + y * z == 1 */
qrow[0] = 0; qcol[0] = 2; qval[0] = 1.0;
qrow[1] = 1; qcol[1] = 2; qval[1] = 1.0;
error = GRBaddqconstr(model, 0, NULL, NULL, 2, qrow, qcol, qval,
GRB_EQUAL, 1, "bilinear1");
if (error) goto QUIT;

/* Optimize model - this will fail since we need to set NonConvex to 2 */
error = GRBoptimize(model);
if (!error) {
    printf("Should have failed!\n");
    goto QUIT;
}

/* Set parameter and solve again */
error = GRBsetintparam(GRBgetenv(model), GRB_INT_PAR_NONCONVEX, 2);
if (error) goto QUIT;

error = GRBoptimize(model);
if (error) goto QUIT;
/* Write model to 'bilinear.lp' */

error = GRBwrite(model, "bilinear.lp");
if (error) goto QUIT;

/* Capture solution information */

error = GRBgetintattr(model, GRB_INT_ATTR_STATUS, &optimstatus);
if (error) goto QUIT;

error = GRBgetdblattr(model, GRB_DBL_ATTR_OBJVAL, &objval);
if (error) goto QUIT;

error = GRBgetdblattrarray(model, GRB_DBL_ATTR_X, 0, 3, sol);
if (error) goto QUIT;

printf("Optimization complete\n");
if (optimstatus == GRB_OPTIMAL) {
    printf("Optimal objective: %.4e\n", objval);

    printf(" x=%.2f, y=%.2f, z=%.2f\n", sol[0], sol[1], sol[2]);
} else if (optimstatus == GRB_INF_OR_UNBD) {
    printf("Model is infeasible or unbounded\n");
} else {
    printf("Optimization was stopped early\n");
}

/* Now constrain 'x' to be integral and solve again */

error = GRBsetcharattrelement(model, GRB_CHAR_ATTR_VTYPE, 0, GRB_INTEGER);
if (error) goto QUIT;

error = GRBoptimize(model);
if (error) goto QUIT;

/* Capture solution information */

error = GRBgetintattr(model, GRB_INT_ATTR_STATUS, &optimstatus);
if (error) goto QUIT;

error = GRBgetdblattr(model, GRB_DBL_ATTR_OBJVAL, &objval);
if (error) goto QUIT;

error = GRBgetdblattrarray(model, GRB_DBL_ATTR_X, 0, 3, sol);
if (error) goto QUIT;

printf("Optimization complete\n");
if (optimstatus == GRB_OPTIMAL) {
    printf("Optimal objective: %.4e\n", objval);

    printf(" x=%.2f, y=%.2f, z=%.2f\n", sol[0], sol[1], sol[2]);
} else if (optimstatus == GRB_INF_OR_UNBD) {
    printf("Model is infeasible or unbounded\n");
} else {
    printf("Optimization was stopped early\n");
}
QUIT:

/* Error reporting */

if (error) {
    printf("ERROR: %s\n", GRBgeterrormsg(env));
    exit(1);
}

/* Free model */
GRBfreemodel(model);

/* Free environment */
GRBfreeenv(env);

return 0;
}

callback_c.c

/* Copyright 2020, Gurobi Optimization, LLC */

/* This example reads a model from a file, sets up a callback that monitors optimization progress and implements a custom termination strategy, and outputs progress information to the screen and to a log file.

The termination strategy implemented in this callback stops the optimization of a MIP model once at least one of the following two conditions have been satisfied:
   1) The optimality gap is less than 10%
   2) At least 10000 nodes have been explored, and an integer feasible solution has been found.

Note that termination is normally handled through Gurobi parameters (MIPGap, NodeLimit, etc.). You should only use a callback for termination if the available parameters don’t capture your desired termination criterion.
*/

#include <stdlib.h>
#include <stdio.h>
#include <math.h>
#include "gurobi_c.h"

/* Define structure to pass my data to the callback function */

struct callback_data {
    double lastiter;
    double lastnode;
    double *solution;
    FILE *logfile;
}
/* Define my callback function */

int __stdcall mycallback ( GRBmodel *model,
  void *cbdata,
  int where,
  void *usrdata)
{
  struct callback_data *mydata = (struct callback_data *) usrdata;

  if (where == GRB_CB_POLLING) {
    /* Ignore polling callback */
  }
  else if (where == GRB_CB_PRESOLVE) {
    /* Presolve callback */
    int cdels, rdels;
    GRBcbget(cbdata, where, GRB_CB_PRE_COLDEL, &cdels);
    GRBcbget(cbdata, where, GRB_CB_PRE_ROWDEL, &rdels);
    if (cdels || rdels) {
      printf("%7d columns and %7d rows are removed\n", cdels, rdels);
    }
  }
  else if (where == GRB_CB_SIMPLEX) {
    /* Simplex callback */
    double itcnt, obj, pinf, dinf;
    int ispert;
    char ch;
    GRBcbget(cbdata, where, GRB_CB_SPX_ITRCNT, &itcnt);
    if (itcnt - mydata->lastiter >= 100) {
      mydata->lastiter = itcnt;
      GRBcbget(cbdata, where, GRB_CB_SPX_OBJVAL, &obj);
      GRBcbget(cbdata, where, GRB_CB_SPX_ISPERT, &ispert);
      GRBcbget(cbdata, where, GRB_CB_SPX_PRIMINF, &pinf);
      GRBcbget(cbdata, where, GRB_CB_SPX_DUALINF, &dinf);
      if (ispert == 0) ch = ' ';
      else if (ispert == 1) ch = 'S';
      else ch = 'P';
      printf(" %7.0 f %14.7 e%c %13.6 e %13.6 e
", itcnt, obj, ch, pinf, dinf);
    }
  }
  else if (where == GRB_CB_MIP) {
    /* General MIP callback */
    double nodecnt, objbst, objbnd, actnodes, itcnt;
    int solcnt, cutcnt;
    GRBcbget(cbdata, where, GRB_CB_MIP_NODCNT, &nodecnt);
    GRBcbget(cbdata, where, GRB_CB_MIP_OBJBST, &objbst);
    GRBcbget(cbdata, where, GRB_CB_MIP_OBJBND, &objbnd);
    GRBcbget(cbdata, where, GRB_CB_MIP_SOLCNT, &solcnt);
    if (nodecnt - mydata->lastnode >= 100) {
      mydata->lastnode = nodecnt;
      GRBcbget(cbdata, where, GRB_CB_MIP_NODLFT, &actnodes);
      GRBcbget(cbdata, where, GRB_CB_MIP_ITRCNT, &itcnt);
      GRBcbget(cbdata, where, GRB_CB_MIP_CUTCNT, &cutcnt);
      printf("%7.0f %7.0f %8.0f %13.6e %13.6e %7d %7d\n",
        nodecnt, actnodes, itcnt, objbst, objbnd, solcnt, cutcnt);
    }
    if (fabs(objbst - objbnd) < 0.1 * (1.0 + fabs(objbst))) {
      /* Some condition here */
    }
  }
printf("Stop early - 10%% gap achieved\n");
GRBterminate(model);

} else if (where == GRB_CB_MIPSOL) {
  /* MIP solution callback */
  double nodecnt, obj;
  int solcnt;
  GRBcbget(cbdata, where, GRB_CB_MIPSOL_NODCNT, &nodecnt);
  GRBcbget(cbdata, where, GRB_CB_MIPSOL_OBJ, &obj);
  GRBcbget(cbdata, where, GRB_CB_MIPSOL_SOLCNT, &solcnt);
  GRBcbget(cbdata, where, GRB_CB_MIPSOL_SOL, mydata->solution);
  printf("**** New solution at node %.0f, obj %g, sol %d, x[0] = %.2f ****\n",
         nodecnt, obj, solcnt, mydata->solution[0]);
} else if (where == GRB_CB_MIPNODE) {
  /* MIP node callback */
  int status;
  printf("**** New node ****\n");
  GRBcbget(cbdata, where, GRB_CB_MIPNODE_STATUS, &status);
  if (status == GRB_OPTIMAL) {
    GRBcbget(cbdata, where, GRB_CB_MIPNODE_REL, mydata->solution);
    GRBcbsolution(cbdata, mydata->solution, NULL);
  }
} else if (where == GRB_CB_BARRIER) {
  /* Barrier callback */
  int itcnt;
  double primobj, dualobj, priminf, dualinf, compl;
  GRBcbget(cbdata, where, GRB_CB_BARRIER_ITRCNT, &itcnt);
  GRBcbget(cbdata, where, GRB_CB_BARRIER_PRIMOBJ, &primobj);
  GRBcbget(cbdata, where, GRB_CB_BARRIER_DUALOBJ, &dualobj);
  GRBcbget(cbdata, where, GRB_CB_BARRIER_PRIMINF, &priminf);
  GRBcbget(cbdata, where, GRB_CB_BARRIER_DUALINF, &dualinf);
  GRBcbget(cbdata, where, GRB_CB_BARRIER_COMPL, &compl);
  printf("%d %.4e %.4e %.4e %.4e %.4e\n",
         itcnt, primobj, dualobj, priminf, dualinf, compl);
} else if (where == GRB_CB_MESSAGE) {
  /* Message callback */
  char *msg;
  GRBcbget(cbdata, where, GRB_CB_MSG_STRING, &msg);
  fprintf(mydata->logfile, "%s", msg);
}
return 0;
}

int main(int argc,
        char *argv[])
{
  GRBenv *env = NULL;
  GRBmodel *model = NULL;
  int error = 0;
  int numvars, solcount, optimstatus, j;
  double objval, x;
char *varname;
struct callback_data mydata;

mydata.lastiter = -GRB_INFINITY;
mydata.lastnode = -GRB_INFINITY;
mydata.solution = NULL;
mydatalogfile = NULL;

if (argc < 2) {
    fprintf(stderr, "Usage: callback_c filename\n");
    goto QUIT;
}

/* Open log file */
mydatalogfile = fopen("cb.log", "w");
if (!mydatalogfile) {
    fprintf(stderr, "Cannot open cb.log for callback message\n");
    goto QUIT;
}

/* Create environment */
error = GRBloadenv(&env, NULL);
if (error) goto QUIT;

/* Turn off display and heuristics */
error = GRBsetintparam(env, GRB_INT_PAR_OUTPUTFLAG, 0);
if (error) goto QUIT;
error = GRBsetdblparam(env, GRB_DBL_PAR_HEURISTICS, 0.0);
if (error) goto QUIT;

/* Read model from file */
error = GRBreadmodel(env, argv[1], &model);
if (error) goto QUIT;

/* Allocate space for solution */
error = GRBgetintattr(model, GRB_INT_ATTR_NUMVARS, &numvars);
if (error) goto QUIT;

mydata.solution = malloc(numvars*sizeof(double));
if (mydata.solution == NULL) {
    fprintf(stderr, "Failed to allocate memory\n");
    exit(1);
}

/* Set callback function */
error = GRBsetcallbackfunc(model, mycallback, (void *) &mydata);
if (error) goto QUIT;

/* Solve model */
error = GRBoptimize(model);
if (error) goto QUIT;

/* Capture solution information */
printf("Optimization complete\n");

error = GRBgetintattr(model, GRB_INT_ATTR_SOLCOUNT, &solcount);
if (error) goto QUIT;

error = GRBgetintattr(model, GRB_INT_ATTR_STATUS, &optimstatus);
if (error) goto QUIT;

if (solcount == 0) {
    printf("No solution found, optimization status = %d\n", optimstatus);
    goto QUIT;
}

error = GRBgetdblattr(model, GRB_DBL_ATTR_OBJVAL, &objval);
if (error) goto QUIT;

printf("Solution found, objective = %.4e\n", objval);

for (j = 0; j < numvars; ++j) {
    error = GRBgetstrattrelement(model, GRB_STR_ATTR_VARNAME, j, &varname);
    if (error) goto QUIT;
    error = GRBgetdblattrelement(model, GRB_DBL_ATTR_X, j, &x);
    if (error) goto QUIT;
    if (x != 0.0) {
        printf("%s %f\n", varname, x);
    }
}

QUIT:

/* Error reporting */

if (error) {
    printf("ERROR: %s\n", GRBgeterrormsg(env));
    exit(1);
}

/* Close log file */

if (mydata.logfile)
    fclose(mydata.logfile);

/* Free solution */

if (mydata.solution)
    free(mydata.solution);

/* Free model */

GRBfreemodel(model);
Free environment */
GRBfreeenv(env);
return 0;
}
dense_c.c

/* Copyright 2020, Gurobi Optimization, LLC */

/* This example formulates and solves the following simple QP model:

minimize   x + y + x^2 + x*y + y^2 + y*z + z^2
subject to  x + 2 y + 3 z >= 4
            x + y      >= 1
            x, y, z non-negative

The example illustrates the use of dense matrices to store A and Q
(and dense vectors for the other relevant data). We don’t recommend
that you use dense matrices, but this example may be helpful if you
already have your data in this format.
*/
#include <stdlib.h>
#include <stdio.h>
#include "gurobi_c.h"

/*
Solve an LP/QP/MILP/MIQP represented using dense matrices. This
routine assumes that A and Q are both stored in row-major order.
It returns 1 if the optimization succeeds. When successful,
it returns the optimal objective value in 'objvalP', and the
optimal solution vector in 'solution'.
*/
static int
dense_optimize(GRBenv *env,
                int rows,
                int cols,
                double *c,  /* linear portion of objective function */
                double *Q,  /* quadratic portion of objective function */
                double *A,  /* constraint matrix */
                char *sense,  /* constraint senses */
                double *rhs,  /* RHS vector */
                double *lb,  /* variable lower bounds */
                double *ub,  /* variable upper bounds */
                char *vtype,  /* variable types (continuous, binary, etc.) */
                double *solution,
                double *objvalP)
{
    GRBmodel *model = NULL;
    int   i, j, optimstatus;
    int   error = 0;
    int   success = 0;
/ Create an empty model */
error = GRBnewmodel(env, &model, "dense", cols, c, lb, ub, vtype, NULL);
if (error) goto QUIT;

error = GRBaddconstrs(model, rows, 0, NULL, NULL, NULL, sense, rhs, NULL);
if (error) goto QUIT;

/* Populate A matrix */
for (i = 0; i < rows; i++) {
    for (j = 0; j < cols; j++) {
        if (A[i*cols+j] != 0) {
            error = GRBchgcobj(model, 1, &i, &j, &A[i*cols+j]);
            if (error) goto QUIT;
        }
    }
}

/* Populate Q matrix */
if (Q) {
    for (i = 0; i < cols; i++) {
        for (j = 0; j < cols; j++) {
            if (Q[i*cols+j] != 0) {
                error = GRBaddqpterm(model, 1, &i, &j, &Q[i*cols+j]);
                if (error) goto QUIT;
            }
        }
    }
}

/* Optimize model */
error = GRBoptimize(model);
if (error) goto QUIT;

/* Write model to 'dense.lp' */
error = GRBwrite(model, "dense.lp");
if (error) goto QUIT;

/* Capture solution information */
error = GRBgetintattr(model, GRB_INT_ATTR_STATUS, &optimstatus);
if (error) goto QUIT;
if (optimstatus == GRB_OPTIMAL) {
    error = GRBgetdblattr(model, GRB_DBL_ATTR_OBJVAL, objvalP);
    if (error) goto QUIT;

    error = GRBgetdblattrarray(model, GRB_DBL_ATTR_X, 0, cols, solution);
    if (error) goto QUIT;

    success = 1;
QUIT:

/* Error reporting */

if (error) {
    printf("ERROR: %s\n", GRBgeterrormsg(env));
    exit(1);
}

/* Free model */

GRBfreemodel(model);

return success;

}

int
main(int argc, char *argv[])
{
    GRBenv *env = NULL;
    int   error = 0;
    double c[] = {1, 1, 0};
    double Q[3][3] = {{{1, 1, 0}, {0, 1, 1}, {0, 0, 1}}};
    double A[2][3] = {{{1, 2, 3}, {1, 1, 0}}};
    char   sense[] = {'>', '>'};
    double rhs[] = {4, 1};
    double lb[] = {0, 0, 0};
    double sol[3];
    int    solved;
    double objval;

    /* Create environment */

    error = GRBloadenv(&env, "dense.log");
    if (error) goto QUIT;

    /* Solve the model */

    solved = dense_optimize(env, 2, 3, c, &Q[0][0], &A[0][0], sense, rhs, lb, NULL, NULL, sol, &objval);

    if (solved)
        printf("Solved: x=%.4f, y=%.4f, z=%.4f\n", sol[0], sol[1], sol[2]);

QUIT:

/* Free environment */

GRBfreeenv(env);

return 0;
}
/* Copyright 2020, Gurobi Optimization, LLC */

/* Solve the classic diet model, showing how to add constraints to an existing model. */

#include <stdlib.h>
#include <stdio.h>
#include <math.h>
#include "gurobi_c.h"

int printSolution(GRBmodel * model, int nCategories, int nFoods);

int main(int argc, char * argv[])
{
  GRBenv * env = NULL;
  GRBmodel * model = NULL;
  int error = 0;
  int i, j;
  int * cbeg, * cind, idx;
  double * cval, * rhs;
  char * sense;

  /* Nutrition guidelines, based on
     USDA Dietary Guidelines for Americans, 2005

  const int nCategories = 4;
  char * Categories[] =
    { "calories", "protein", "fat", "sodium" };
  double minNutrition[] = { 1800, 91, 0, 0 };
  double maxNutrition[] = { 2200, GRB_INFINITY, 65, 1779 };

  /* Set of foods */
  const int nFoods = 9;
  char * Foods[] =
    { "hamburger", "chicken", "hot dog", "fries",
      "macaroni", "pizza", "salad", "milk", "ice cream" };
  double cost[] =
    { 2.49, 2.89, 1.50, 1.89, 2.09, 1.99, 2.49, 0.89, 1.59 };

  /* Nutrition values for the foods */
  double nutritionValues[][4] =
    { { 410, 24, 26, 730 },
      { 420, 32, 10, 1190 },
      { 560, 20, 32, 1800 },
      { 380, 4, 19, 270 },
      { 320, 12, 10, 930 },
      { 320, 15, 12, 820 },
      { 320, 31, 12, 1230 },
      { 100, 8, 2.5, 125 },
      { 330, 8, 10, 180 }
/* Create environment */
error = GRBloadenv(&env, "diet.log");
if (error) goto QUIT;

/* Create initial model */
error = GRBnewmodel(env, &model, "diet", nFoods + nCategories,
         NULL, NULL, NULL, NULL, NULL);
if (error) goto QUIT;

/* Initialize decision variables for the foods to buy */
for (j = 0; j < nFoods; ++j)
{
    error = GRBsetdblattrelement(model, " Obj ", j, cost[j]);
    if (error) goto QUIT;
    error = GRBsetstrattrelement(model, " VarName ", j, Foods[j]);
    if (error) goto QUIT;
}

/* Initialize decision variables for the nutrition information,
 which we limit via bounds */
for (j = 0; j < nCategories; ++j)
{
    error = GRBsetdblattrelement(model, " LB ", j + nFoods, minNutrition[j]);
    if (error) goto QUIT;
    error = GRBsetdblattrelement(model, " UB ", j + nFoods, maxNutrition[j]);
    if (error) goto QUIT;
    error = GRBsetstrattrelement(model, " VarName ", j + nFoods, Categories[j]);
    if (error) goto QUIT;
}

/* The objective is to minimize the costs */
error = GRBsetintattr(model, " ModelSense ", GRB_MINIMIZE);
if (error) goto QUIT;

/* Nutrition constraints */
cbeg = malloc(sizeof(int) * nCategories);
if (!cbeg) goto QUIT;
cind = malloc(sizeof(int) * nCategories * (nFoods + 1));
if (!cind) goto QUIT;
cval = malloc(sizeof(double) * nCategories * (nFoods + 1));
if (!cval) goto QUIT;
rhs = malloc(sizeof(double) * nCategories);
if (!rhs) goto QUIT;
sense = malloc(sizeof(char) * nCategories);
if (!sense) goto QUIT;
idx = 0;
for (i = 0; i < nCategories; ++i)
{
    cbeg[i] = idx;
    rhs[i] = 0.0;
    sense[i] = GRB_EQUAL;
    for (j = 0; j < nFoods; ++j)
    {
        cind[idx] = j;
    }
}
cval[idx++] = nutritionValues[j][i];
} 
cind[idx] = nFoods + i;
cval[idx++] = -1.0;
}

error = GRBaddconstrs(model, nCategories, idx, cbeg, cind, cval, sense, 
rhs, Categories);
if (error) goto QUIT;
/* Solve */
error = GRBoptimize(model);
if (error) goto QUIT;
error = printSolution(model, nCategories, nFoods);
if (error) goto QUIT;

printf("Adding constraint: at most 6 servings of dairy\n");
cind[0] = 7;
cval[0] = 1.0;
cind[1] = 8;
cval[1] = 1.0;
error = GRBaddconstr(model, 2, cind, cval, GRB_LESS_EQUAL, 6.0, 
"limit_dairy");
if (error) goto QUIT;
/* Solve */
error = GRBoptimize(model);
if (error) goto QUIT;
error = printSolution(model, nCategories, nFoods);
if (error) goto QUIT;

QUIT:

/* Error reporting */
if (error)
{
    printf("ERROR: %s\n", GRBgeterrormsg(env));
    exit(1);
}
/* Free data */
free(cbeg);
free(cind);
free(cval);
free(rhs);
free(sense);
/* Free model */
GRBfreemodel(model);
/* Free environment */
GRBfreeenv(env);

    return 0;
}

int printSolution(GRBmodel* model, int nCategories, int nFoods)
{
    int error, status, i, j;
    double obj, x;
    char* vname;

    error = GRBgetintattr(model, "Status", &status);
    if (error) return error;
    if (status == GRB_OPTIMAL)
    {
        error = GRBgetdblattr(model, "ObjVal", &obj);
        if (error) return error;
        printf("\nCost : %f\n\nBuy :
", obj);
        for (j = 0; j < nFoods; ++j)
        {
            error = GRBgetdblattrelement(model, "X", j, &x);
            if (error) return error;
            if (x > 0.0001)
            {
                error = GRBgetstrattrelement(model, "VarName", j, &vname);
                if (error) return error;
                printf("%s %f\n", vname, x);
            }
        }
        printf("\nNutrition :
");
        for (i = 0; i < nCategories; ++i)
        {
            error = GRBgetdblattrelement(model, "X", i + nFoods, &x);
            if (error) return error;
            error = GRBgetstrattrelement(model, "VarName", i + nFoods, &vname);
            if (error) return error;
            printf("%s %f\n", vname, x);
        }
    }
    else
    {
        printf("No solution\n");
    }
    return 0;
}

facility_c.c

/* Copyright 2020, Gurobi Optimization, LLC */

/* Facility location: a company currently ships its product from 5 plants
   to 4 warehouses. It is considering closing some plants to reduce
   costs. What plant(s) should the company close, in order to minimize
   transportation and fixed costs? */
Based on an example from Frontline Systems:
http://www.solver.com/disfacility.htm
Used with permission.

```c
#include <stdlib.h>
#include <stdio.h>
#include <math.h>
#include "gurobi_c.h"

#define opencol(p) p
#define transportcol(w,p) nPlants*(w+1)+p
#define MAXSTR 128

int main(int argc, char *argv[])
{
  GRBenv *env = NULL;
  GRBmodel *model = NULL;
  int error = 0;
  int p, w, col;
  int *cbeg = NULL;
  int *cind = NULL;
  int idx, rowct;
  double *cval = NULL;
  double *rhs = NULL;
  char *sense = NULL;
  char vname[MAXSTR];
  int cnamect = 0;
  char **cname = NULL;
  double maxFixed = -GRB_INFINITY, sol, obj;

  /* Number of plants and warehouses */
  const int nPlants = 5;
  const int nWarehouses = 4;

  /* Warehouse demand in thousands of units */
  double Demand[] = { 15, 18, 14, 20 };

  /* Plant capacity in thousands of units */
  double Capacity[] = { 20, 22, 17, 19, 18 };

  /* Fixed costs for each plant */
  double FixedCosts[] =
  { 12000, 15000, 17000, 13000, 16000 };

  /* Transportation costs per thousand units */
  double TransCosts[4][5] =
  { 4000, 2000, 3000, 2500, 4500 },
  { 2500, 2600, 3400, 3000, 4000 },
  { 1200, 1800, 2600, 4100, 3000 },
  { 2200, 2600, 3100, 3700, 3200 }
};
```
/* Create environment */
error = GRBloadenv(&env, "facility.log");
if (error) goto QUIT;

/* Create initial model */
error = GRBnewmodel(env, &model, "facility", nPlants * (nWarehouses + 1),
    NULL, NULL, NULL, NULL, NULL);
if (error) goto QUIT;

/* Initialize decision variables for plant open variables */
for (p = 0; p < nPlants; ++p)
{
    col = opencol(p);
    error = GRBsetcharattrelement(model, "VType", col, GRB_BINARY);
    if (error) goto QUIT;
    error = GRBsetdblattrelement(model, "Obj", col, FixedCosts[p]);
    if (error) goto QUIT;
    sprintf(vname, "Open%i", p);
    error = GRBsetstrattrelement(model, "VarName", col, vname);
    if (error) goto QUIT;
}

/* Initialize decision variables for transportation decision variables:
how much to transport from a plant p to a warehouse w */
for (w = 0; w < nWarehouses; ++w)
{
    for (p = 0; p < nPlants; ++p)
    {
        col = transportcol(w, p);
        error = GRBsetdblattrelement(model, "Obj", col, TransCosts[w][p]);
        if (error) goto QUIT;
        sprintf(vname, "Trans%i.%i", p, w);
        error = GRBsetstrattrelement(model, "VarName", col, vname);
        if (error) goto QUIT;
    }
}

/* The objective is to minimize the total fixed and variable costs */
error = GRBsetintattr(model, "ModelSense", GRB_MINIMIZE);
if (error) goto QUIT;

/* Make space for constraint data */
rowct = (nPlants > nWarehouses) ? nPlants : nWarehouses;
cbeg = malloc(sizeof(int) * rowct);
if (!cbeg) goto QUIT;
cind = malloc(sizeof(int) * (nPlants * (nWarehouses + 1)));
if (!cind) goto QUIT;
cval = malloc(sizeof(double) * (nPlants * (nWarehouses + 1)));
if (!cval) goto QUIT;
rhs = malloc(sizeof(double) * rowct);
if (!rhs) goto QUIT;
sense = malloc(sizeof(char) * rowct);
if (!sense) goto QUIT;
cname = calloc(rowct, sizeof(char*));
if (!cname) goto QUIT;
/* Production constraints */

Note that the limit sets the production to zero if the plant is closed */

idx = 0;
for (p = 0; p < nPlants; ++p)
{
    cbeg[p] = idx;
    rhs[p] = 0.0;
    sense[p] = GRB_LESS_EQUAL;
    cname[p] = malloc(sizeof(char) * MAXSTR);
    if (!cname[p]) goto QUIT;
    cnamect++;
    sprintf(cname[p], "Capacity%i", p);
    for (w = 0; w < nWarehouses; ++w)
    {
        cind[idx] = transportcol(w, p);
        cval[idx++] = 1.0;
    }
    cind[idx] = opencol(p);
    cval[idx++] = -Capacity[p];
}
error = GRBaddconstrs(model, nPlants, idx, cbeg, cind, cval, sense, rhs, cname);
if (error) goto QUIT;

/* Demand constraints */

idx = 0;
for (w = 0; w < nWarehouses; ++w)
{
    cbeg[w] = idx;
    sense[w] = GRB_EQUAL;
    sprintf(cname[w], "Demand%i", w);
    for (p = 0; p < nPlants; ++p)
    {
        cind[idx] = transportcol(w, p);
        cval[idx++] = 1.0;
    }
}
error = GRBaddconstrs(model, nWarehouses, idx, cbeg, cind, cval, sense, Demand, cname);
if (error) goto QUIT;

/* Guess at the starting point: close the plant with the highest fixed costs; open all others */

/* First, open all plants */

for (p = 0; p < nPlants; ++p)
{
    error = GRBsetdblattrelement(model, "Start", opencol(p), 1.0);
    if (error) goto QUIT;
}

/* Now close the plant with the highest fixed cost */

printf("Initial guess:\n");
for (p = 0; p < nPlants; ++p)
{
{ 
    if (FixedCosts[p] > maxFixed) 
    { 
        maxFixed = FixedCosts[p];
    }
} 
for (p = 0; p < nPlants; ++p) 
{ 
    if (FixedCosts[p] == maxFixed) 
    { 
        error = GRBsetdblattrelement(model, "Start", opencol(p), 0.0);
        if (error) goto QUIT;
        printf("Closing plant %i\n\n", p);
        break;
    }
} 
/* Use barrier to solve root relaxation */
error = GRBsetintparam(GRBgetenv(model),
    GRB_INT_PAR_METHOD, 
    GRB_METHOD_BARRIER);
if (error) goto QUIT;
/* Solve */
error = GRBoptimize(model);
if (error) goto QUIT;
/* Print solution */
error = GRBgetdblattrelement(model, "X", opencol(p), &sol);
if (error) goto QUIT;
if (sol > 0.99) 
{ 
    printf("Plant %i open:\n", p);
    for (w = 0; w < nWarehouses; ++w) 
    { 
        error = GRBgetdblattrelement(model, "X", transportcol(w, p), &sol);
        if (error) goto QUIT;
        if (sol > 0.0001) 
        { 
            printf(" Transport %f units to warehouse %i\n", sol, w);
        }
    }
} else
{ 
    printf("Plant %i closed!\n", p);
} 
}
QUIT:

    /* Error reporting */
    if (error)
    {
        printf("ERROR: %s\n", GRBgeterrormsg(env));
        exit(1);
    }

    /* Free data */
    free(cbeg);
    free(cind);
    free(cval);
    free(rhs);
    free(sense);
    for (p = 0; p < cnamesct; ++p) {
        free(cname[p]);
    }
    free(cname);

    /* Free model */
    GRBfreemodel(model);

    /* Free environment */
    GRBfreeenv(env);

    return 0;
}

feasopt_c.c

/* Copyright 2020, Gurobi Optimization, LLC */

/* This example reads a MIP model from a file, adds artificial
variables to each constraint, and then minimizes the sum of the
artificial variables. A solution with objective zero corresponds
to a feasible solution to the input model.
We can also use FeasRelax feature to do it. In this example, we
use minrelax=1, i.e. optimizing the returned model finds a solution
that minimizes the original objective, but only from among those
solutions that minimize the sum of the artificial variables. */

#include <stdlib.h>
#include <stdio.h>
#include <math.h>
#include <string.h>
#include "gurobi_c.h"

int
main(int   argc,
     char *argv[])
{

42
GRBenv *env = NULL;
GRBmodel *model = NULL;
GRBmodel *feasmodel = NULL;
double *rhapsen = NULL;
int error = 0;
int i, j;
int numvars, numconstrs;
char sense;
int vind[1];
double vval[1];
double feasobj;
char *cname, *vname;

if (argc < 2)
{
    fprintf(stderr, "Usage: feasopt_c filename\n");
    exit(1);
}

error = GRBloadenv(&env, "feasopt.log");
if (error) goto QUIT;

error = GRBreadmodel(env, argv[1], &model);
if (error) goto QUIT;

/* Create a copy to use FeasRelax feature later */
feasmodel = GRBcopymodel(model);
if (error) goto QUIT;

/* clear objective */
error = GRBgetintattr(model, "NumVars", &numvars);
if (error) goto QUIT;
for (j = 0; j < numvars; ++j)
{
    error = GRBsetdblattrelement(model, "Obj", j, 0.0);
    if (error) goto QUIT;
}

/* add slack variables */
error = GRBgetintattr(model, "NumConstrs", &numconstrs);
if (error) goto QUIT;
for (i = 0; i < numconstrs; ++i)
{
    error = GRBgetcharattrelement(model, "Sense", i, &sense);
    if (error) goto QUIT;
    if (sense != '>')
    {
        error = GRBgetstrattrelement(model, "ConstrName", i, &cname);
        if (error) goto QUIT;
        vname = malloc(sizeof(char) * (6 + strlen(cname)));
        if (!vname) goto QUIT;
        strcpy(vname, "ArtN_ ");
        strcat(vname, cname);
        vind[0] = i;
        vval[0] = -1.0;
    }
error = GRBaddvar(model, 1, vind, vval, 1.0, 0.0, GRB_INFINITY, GRB_CONTINUOUS, vname);
if (error) goto QUIT;
free(vname);
}
if (sense != '<') {
    error = GRBgetstrattrelement(model, "ConstrName", i, &cname);
    if (error) goto QUIT;
    vname = malloc(sizeof(char) * (6 + strlen(cname)));
    if (!vname) goto QUIT;
    strcpy(vname, "ArtP_");
    strcat(vname, cname);
    vind[0] = i;
    vval[0] = 1.0;
    error = GRBaddvar(model, 1, vind, vval, 1.0, 0.0, GRB_INFINITY, GRB_CONTINUOUS, vname);
    if (error) goto QUIT;
    free(vname);
}
/* Optimize modified model */
error = GRBoptimize(model);
if (error) goto QUIT;
error = GRBwrite(model, "feasopt.lp");
if (error) goto QUIT;

/* Use FeasRelax feature */
rhspen = (double *) malloc(numconstrs*sizeof(double));
if (rhspen == NULL) {
    printf("ERROR: out of memory\n");
    goto QUIT;
}
/* set penalties for artificial variables */
for (i = 0; i < numconstrs; i++) rhspen[i] = 1;
/* create a FeasRelax model with the original objective recovered 
and enforcement on minimum of aretificial variables */
error = GRBfeasrelax(feasmodel, GRB_FEASRELAX_LINEAR, 1, NULL, NULL, rhspen, &feasobj);
if (error) goto QUIT;
/* optimize FeasRelax model */
error = GRBwrite(feasmodel, "feasopt1.lp");
if (error) goto QUIT;
error = GRBoptimize(feasmodel);
if (error) goto QUIT;
QUIT:
/* Error reporting */

if (error)
{
    printf("ERROR: %s\n", GRBgeterrormsg(env));
    exit(1);
}

/* Free models, env and etc. */

if (rhspen) free(rhspen);

GRBfreemodel(model);
GRBfreemodel(feasmodel);
GRBfreeenv(env);

return 0;
}

fixanddive_c.c

/* Copyright 2020, Gurobi Optimization, LLC */

/* Implement a simple MIP heuristic. Relax the model, sort variables based on fractionality, and fix the 25% of the fractional variables that are closest to integer variables. Repeat until either the relaxation is integer feasible or linearly infeasible. */

#include <stdlib.h>
#include <stdio.h>
#include <math.h>
#include "gurobi_c.h"

typedef struct
{
    int index;
    double X;
} var_t;

int vcomp(const void* v1, const void* v2);

int main(int argc,
    char *argv[])
{
    GRBenv *env = NULL, *modelenv = NULL;
    GRBmodel *model = NULL;
    int error = 0;
    int j, iter, nfix;
    int numvars, numintvars, numfractional;
    int *intvars = NULL;
int status;
char vtype, *vname;
double sol, obj, fixval;
var_t *fractional = NULL;

if (argc < 2)
{
  fprintf(stderr, "Usage: fixanddive_c filename\n");
  exit(1);
}

error = GRBloadenv(&env, "fixanddive.log");
if (error) goto QUIT;

/* Read model */
error = GRBreadmodel(env, argv[1], &model);
if (error) goto QUIT;

/* Collect integer variables and relax them */
error = GRBgetintattr(model, "NumVars", &numvars);
if (error) goto QUIT;
error = GRBgetintattr(model, "NumIntVars", &numintvars);
if (error) goto QUIT;
intvars = malloc(sizeof(int) * numintvars);
if (!intvars) goto QUIT;
fractional = malloc(sizeof(var_t) * numintvars);
if (!fractional) goto QUIT;
numfractional = 0;
for (j = 0; j < numvars; j++)
{
  error = GRBgetcharattrelement(model, "VType", j, &vtype);
  if (error) goto QUIT;
  if (vtype != GRB_CONTINUOUS)
  {
    intvars[numfractional++] = j;
    error = GRBsetcharattrelement(model, "VType", j, GRB_CONTINUOUS);
    if (error) goto QUIT;
  }
}

modelenv = GRBgetenv(model);
if (!modelenv) goto QUIT;
error = GRBsetintparam(modelenv, "OutputFlag", 0);
if (error) goto QUIT;
error = GRBoptimize(model);
if (error) goto QUIT;

/* Perform multiple iterations. In each iteration, identify the first quartile of integer variables that are closest to an integer value in the relaxation, fix them to the nearest integer, and repeat. */
for (iter = 0; iter < 1000; ++iter)
{
  /* create a list of fractional variables, sorted in order of increasing distance from the relaxation solution to the nearest
integer value */

numfractional = 0;
for (j = 0; j < numintvars; ++j)
{
    error = GRBgetdblattrelement(model, "X", intvars[j], &sol);
    if (error) goto QUIT;
    if (fabs(sol - floor(sol + 0.5)) > 1e-5)
    {
        fractional[numfractional].index = intvars[j];
        fractional[numfractional++].X = sol;
    }
}

error = GRBgetdblattr(model, "ObjVal", &obj);
if (error) goto QUIT;
printf("Iteration %i, obj %f, fractional %i\n", iter, obj, numfractional);

if (numfractional == 0)
{
    printf("Found feasible solution - objective %f\n", obj);
    break;
}

/* Fix the first quartile to the nearest integer value */
qsort(fractional, numfractional, sizeof(var_t), vcomp);
nfix = numfractional / 4;
nfix = (nfix > 1) ? nfix : 1;
for (j = 0; j < nfix; ++j)
{
    fixval = floor(fractional[j].X + 0.5);
    error = GRBsetdblattrelement(model, "LB", fractional[j].index, fixval);
    if (error) goto QUIT;
    error = GRBsetdblattrelement(model, "UB", fractional[j].index, fixval);
    if (error) goto QUIT;
    error = GRBgetstrattrelement(model, "VarName",
                                  fractional[j].index, &vname);
    if (error) goto QUIT;
    printf(" Fix %s to %f ( rel %f )\n", vname, fixval, fractional[j].X);
}

error = GRBoptimize(model);
if (error) goto QUIT;

/* Check optimization result */
error = GRBgetintattr(model, "Status", &status);
if (error) goto QUIT;
if (status != GRB_OPTIMAL)
{
    printf("Relaxation is infeasible\n");
    break;
}
}
QUIT:

/* Error reporting */

if (error)
{
    printf("ERROR: \%s\n", GRBgeterrormsg(env));
    exit(1);
}

/* Free data */

free(intvars);
free(fractional);

/* Free model */

GRBfreemodel(model);

/* Free environment */

GRBfreeenv(env);

return 0;
}

int vcomp(const void* v1, const void* v2)
{
    double sol1, sol2, frac1, frac2;
    sol1 = fabs(((var_t*)v1)->X);
    sol2 = fabs(((var_t*)v2)->X);
    frac1 = fabs(sol1 - floor(sol1 + 0.5));
    frac2 = fabs(sol2 - floor(sol2 + 0.5));
    return (frac1 < frac2) ? -1 : ((frac1 == frac2) ? 0 : 1);
}

/* Copyright 2020, Gurobi Optimization, LLC */

This example formulates and solves the following simple model with PWL constraints:

maximize
sum c[j] * x[j]

subject to
sum A[i,j] * x[j] <= 0, for i = 0, ..., m-1
sum y[j] <= 3
y[j] = pwl(x[j]), for j = 0, ..., n-1
x[j] free, y[j] >= 0, for j = 0, ..., n-1

where pwl(x) = 0, if x = 0
= 1+|x|, if x != 0

Note
1. sum $pwl(x[j]) \leq b$ is to bound $x$ vector and also to favor sparse $x$ vector. Here $b = 3$ means that at most two $x[j]$ can be nonzero and if two, then sum $x[j] \leq 1$

2. $pwl(x)$ jumps from 1 to 0 and from 0 to 1, if $x$ moves from negative 0 to 0, then to positive 0, so we need three points at $x = 0$. $x$ has infinite bounds on both sides, the piece defined with two points (-1, 2) and (0, 1) can extend $x$ to $-\infty$. Overall we can use five points (-1, 2), (0, 1), (0, 0), (0, 1) and (1, 2) to define $y = pwl(x)$

```c
#include <stdlib.h>
#include <stdio.h>
#include "gurobi_c.h"

int main(int argc, char *argv[]) {
    GRBenv *env = NULL;
    GRBmodel *model = NULL;
    int *cbeg = NULL;
    int *clen = NULL;
    int *cind = NULL;
    double *cval = NULL;
    double *rhs = NULL;
    char *sense = NULL;
    double *lb = NULL;
    double *obj = NULL;
    int status;
    double objval;
    int error = 0;

    int n = 5;
    int m = 5;
    double c[] = {0.5, 0.8, 0.5, 0.1, -1};
    double A[][5] = {{0, 0, 0, 1, -1},
                     {0, 0, 1, 1, -1},
                     {0, 1, 0, 0, -1},
                     {0, 1, 0, 0, -1},
                     {0, 0, 0, 1, -1}};

    int npts = 5;
    double xpts[] = {-1, 0, 0, 0, 1};
    double ypts[] = {2, 1, 0, 1, 2};

    /* Create environment */
    error = GRBloadenv(&env, NULL);
    if (error) goto QUIT;

    /* Allocate memory and build the model */
    nz = n; /* count nonzeros for n y variables */
    for (i = 0; i < m; i++) {
        for (j = 0; j < n; j++) {
            if (A[i][j] != 0.0) nz++;
        }
    }
}
```

49
cbeg = (int *) malloc(2*n*sizeof(int));
clen = (int *) malloc(2*n*sizeof(int));
cind = (int *) malloc(nz*sizeof(int));
cval = (double *) malloc(nz*sizeof(double));
rhs = (double *) malloc((m+1)*sizeof(double));
sense = (char *) malloc((m+1)*sizeof(char));
lb = (double *) malloc(2*n*sizeof(double));
obj = (double *) malloc(2*n*sizeof(double));

for (j = 0; j < n; j++) {
    /* for x variables */
    lb[j] = -GRB_INFINITY;
    obj[j] = c[j];
    /* for y variables */
    lb[j+n] = 0.0;
    obj[j+n] = 0.0;
}

for (i = 0; i < m; i++) {
    rhs[i] = 0.0;
    sense[i] = GRB_LESS_EQUAL;
}

sense[m] = GRB_LESS_EQUAL;
rhs[m] = 3;

nz = 0;
for (j = 0; j < n; j++) {
    cbeg[j] = nz;
    for (i = 0; i < m; i++) {
        if (A[i][j] != 0.0) {
            cind[nz] = i;
            cval[nz] = A[i][j];
            nz++;
        }
    }
    clen[j] = nz - cbeg[j];
}

for (j = 0; j < n; j++) {
    cbeg[n+j] = nz;
    clen[n+j] = 1;
    cind[nz] = m;
    cval[nz] = 1.0;
    nz++;
}

error = GRBloadmodel(env, &model, "gc_pwl_c", 2*n, m+1,
    GRB_MAXIMIZE, 0.0, obj, sense, rhs,
    cbeg, clen, cind, cval, lb, NULL,
    NULL, NULL, NULL);
if (error) goto QUIT;

/* Add piecewise constraints */
for (j = 0; j < n; j++) {
    error = GRBaddgenconstrPWL(model, NULL, j, n+j, npts, xpts, ypts);
if (error) goto QUIT;
}

/* Optimize model */
error = GRBoptimize(model);
if (error) goto QUIT;

for (j = 0; j < n; j++) {
    double x;
    error = GRBgetdblattrelement(model, "X", j, &x);
    if (error) goto QUIT;
    printf("x[%d] = %g\n", j, x);
}

/* Report the result */
error = GRBgetintattr(model, GRB_INT_ATTR_STATUS, &status);
if (error) goto QUIT;

if (status != GRB_OPTIMAL) {
    fprintf(stderr, "Error: it isn’t optimal\n");
    goto QUIT;
}

error = GRBgetdblatt(model, GRB_DBL_ATTR_OBJVAL, &objval);
if (error) goto QUIT;
printf("Obj: %g\n", objval);

QUIT:

/* Error reporting */
if (error) {
    printf("ERROR: %s\n", GRBgeterrormsg(env));
    exit(1);
}

/* Free data */
if (cbeg) free(cbeg);
if (clen) free(clen);
if (cind) free(cind);
if (cval) free(cval);
if (rhs) free(rhs);
if (sense) free(sense);
if (lb) free(lb);
if (obj) free(obj);

/* Free model */
GRBfreemodel(model);

/* Free environment */
GRBfreeenv(env);

return 0;
}
/* Copyright 2020, Gurobi Optimization, LLC

This example considers the following nonconvex nonlinear problem

maximize 2 x + y
subject to exp(x) + 4 sqrt(y) <= 9
        x, y >= 0

We show you two approaches to solve this:

1) Use a piecewise-linear approach to handle general function
   constraints (such as exp and sqrt).
   a) Add two variables
      u = exp(x)
      v = sqrt(y)
   b) Compute points (x, u) of u = exp(x) for some step length (e.g., x
      = 0, 1e-3, 2e-3, ..., xmax) and points (y, v) of v = sqrt(y) for
      some step length (e.g., y = 0, 1e-3, 2e-3, ..., ymax). We need to
      compute xmax and ymax (which is easy for this example, but this
      does not hold in general).
   c) Use the points to add two general constraints of type
      piecewise-linear.

2) Use the Gurobi's built-in general function constraints directly (EXP
   and POW). Here, we do not need to compute the points and the maximal
   possible values, which will be done internally by Gurobi. In this
   approach, we show how to "zoom in" on the optimal solution and
   tighten tolerances to improve the solution quality.

*/

#include <stdlib.h>
#include <stdio.h>
#include <math.h>
#include "gurobi_c.h"

static double f(double u) { return exp(u); }
static double g(double u) { return sqrt(u); }

static int
printsol(GRBmodel *m)
{
    double x[4];
    double vio;
    int    error = 0;

    error = GRBgetdblattrarray(m, "X", 0, 4, x);
    if (error) goto QUIT;

    printf("x = %g, u = %g\n", x[0], x[2]);
    printf("y = %g, v = %g\n", x[1], x[3]);

    /* Calculate violation of exp(x) + 4 sqrt(y) <= 9 */
    vio = f(x[0]) + 4*g(x[1]) - 9;
    if (vio < 0.0) vio = 0.0;
    printf("Vio = %g\n", vio);

QUIT:

52
```c
QUIT:

    return error;
}

int main(int argc,
    char *argv[])
{
    GRBenv *env = NULL;
    GRBmodel *model = NULL;
    int error = 0;
    double lb, ub;
    int i, len;
    double intv = 1e-3;
    double xmax, ymax, t;
    int ind[2];
    double val[2];
    double x[4];
    double *xpts = NULL;
    double *ypts = NULL;
    double *vpts = NULL;
    double *upts = NULL;

    /* Create environment */
    error = GRBloadenv(&env, NULL);
    if (error) goto QUIT;

    /* Create a new model */
    error = GRBnewmodel(env, &model, NULL, 0, NULL, NULL, NULL, NULL, NULL);
    if (error) goto QUIT;

    /* Add variables */
    lb = 0.0; ub = GRB_INFINITY;

    error = GRBaddvar(model, 0, NULL, NULL, 2.0, lb, ub, GRB_CONTINUOUS, "x");
    if (error) goto QUIT;
    error = GRBaddvar(model, 0, NULL, NULL, 1.0, lb, ub, GRB_CONTINUOUS, "y");
    if (error) goto QUIT;
    error = GRBaddvar(model, 0, NULL, NULL, 0.0, lb, ub, GRB_CONTINUOUS, "u");
    if (error) goto QUIT;
    error = GRBaddvar(model, 0, NULL, NULL, 0.0, lb, ub, GRB_CONTINUOUS, "v");
    if (error) goto QUIT;

    /* Change objective sense to maximization */
    error = GRBsetintattr(model, GRB_INT_ATTR_MODELSENSE, GRB_MAXIMIZE);
    if (error) goto QUIT;

    /* Add linear constraint: u + 4*v <= 9 */
    ind[0] = 2; ind[1] = 3;
    val[0] = 1; val[1] = 4;
```
/* Approach 1) PWL constraint approach */

xmax = log(9.0);
len = (int) ceil(xmax/intv) + 1;
xpts = (double *) malloc(len*sizeof(double));
upts = (double *) malloc(len*sizeof(double));
for (i = 0; i < len; i++) {
    xpts[i] = i*intv;
    upts[i] = f(i*intv);
}

error = GRBaddgenconstrPWL(model, "gc1", 0, 2, len, xpts, upts);
if (error) goto QUIT;

ymax = (9.0/4.0)*(9.0/4.0);
len = (int) ceil(ymax/intv) + 1;
ypts = (double *) malloc(len*sizeof(double));
vpts = (double *) malloc(len*sizeof(double));
for (i = 0; i < len; i++) {
    ypts[i] = i*intv;
    vpts[i] = g(i*intv);
}

error = GRBaddgenconstrPWL(model, "gc2", 1, 3, len, ypts, vpts);
if (error) goto QUIT;

/* Optimize the model and print solution */

error = GRBoptimize(model);
if (error) goto QUIT;

error = printsol(model);
if (error) goto QUIT;

/* Approach 2) General function constraint approach with auto PWL
 * translation by Gurobi */

/* restore unsolved state and get rid of PWL constraints */
error = GRBresetmodel(model);
if (error) goto QUIT;

ind[0] = 0; ind[1] = 1;
error = GRBdelgenconstrs(model, 2, ind);
if (error) goto QUIT;

error = GRBupdatemodel(model);
if (error) goto QUIT;

error = GRBaddgenconstrExp(model, "gcf1", 0, 2, NULL);
if (error) goto QUIT;

error = GRBaddgenconstrPow(model, "gcf2", 1, 3, 0.5, NULL);
if (error) goto QUIT;

error = GRBsetdblparam(GRBgetenv(model), "FuncPieceLength", 1e-3);
if (error) goto QUIT;

/* Optimize the model and print solution */
error = GRBoptimize(model);
if (error) goto QUIT;

error = printsol(model);
if (error) goto QUIT;

/* Zoom in, use optimal solution to reduce the ranges and use a smaller
 * pclen=1e-5 to solve it */
error = GRBgetdblattrelement(model, "X", 0, 4, x);
if (error) goto QUIT;

if (x[0] < 0.0) t = 0.0;
if (error) goto QUIT;

error = GRBsetdblattrelement(model, "UB", 0, x[0]+0.01);
if (error) goto QUIT;

error = GRBsetdblattrelement(model, "UB", 1, x[1]+0.01);
if (error) goto QUIT;

error = GRBsetdblattrelement(model, "LB", 0, t);
if (error) goto QUIT;

error = GRBsetdblattrelement(model, "LB", 1, t);
if (error) goto QUIT;

if (error) {
  printf("ERROR: %s\n", GRBgeterrormsg(env));
  QUIT:
  if (error) {
exit(1);
}
/* Free data */
if (xpts) free(xpts);
if (ypts) free(ypts);
if (upts) free(upts);
if (vpts) free(vpts);
/* Free model */
GRBfreemodel(model);
/* Free environment */
GRBfreeenv(env);
return 0;
}

genconstr_c.c

/* Copyright 2020, Gurobi Optimization, LLC */
/* In this example we show the use of general constraints for modeling
* some common expressions. We use as an example a SAT-problem where we
* want to see if it is possible to satisfy at least four (or all) clauses
* of the logical for
* * L = (x0 or ~x1 or x2) and (x1 or ~x2 or x3) and
* (x2 or ~x3 or x0) and (x3 or ~x0 or x1) and
* (-x0 or ~x1 or x2) and (-x1 or ~x2 or x3) and
* (-x2 or ~x3 or x0) and (-x3 or ~x0 or x1)
* We do this by introducing two variables for each literal (itself and its
* negated value), a variable for each clause, and then two
* variables for indicating if we can satisfy four, and another to identify
* the minimum of the clauses (so if it one, we can satisfy all clauses)
* and put these two variables in the objective.
* i.e. the Objective function will be
* * maximize Obj0 + Obj1
* *
* Obj0 = MIN(Clause1, ..., Clause8)
* Obj1 = 1 -> Clause1 + ... + Clause8 >= 4
* *
* thus, the objective value will be two if and only if we can satisfy all
* clauses; one if and only if at least four clauses can be satisfied, and
* zero otherwise.
*/
#include <stdio.h>
#include <math.h>
#include <stdlib.h>
#include <string.h>

#include "gurobi_c.h"

#define MAXSTR 128
#define NLITERALS 4
#define NCLAUSES 8
#define NOBJ 2
#define NVARS (2 * NLITERALS + NCLAUSES + NOBJ)
#define LIT(n) (n)
#define NOTLIT(n) (NLITERALS + n)
#define CLA(n) (2 * NLITERALS + n)
#define OBJ(n) (2 * NLITERALS + NCLAUSES + n)

int main(void)
{
    GRBenv *env = NULL;
    GRBmodel *model = NULL;
    int error = 0;
    int cind[NVARS];
    double cval[NVARS];
    char buffer[MAXSTR];
    int col, i, status;
    double objval;

    /* Example data */
    const int Clauses[][3] = {{ LIT(0), NOTLIT(1), LIT(2)},
                              { LIT(1), NOTLIT(2), LIT(3)},
                              { LIT(2), NOTLIT(3), LIT(0)},
                              { LIT(3), NOTLIT(0), LIT(1)},
                              { NOTLIT(0), NOTLIT(1), LIT(2)},
                              { NOTLIT(1), NOTLIT(2), LIT(3)},
                              { NOTLIT(2), NOTLIT(3), LIT(0)},
                              { NOTLIT(3), NOTLIT(0), LIT(1)}};

    /* Create environment */
    error = GRBloadenv(&env, "genconstr_c.log");
    if (error) goto QUIT;

    /* Create initial model */
    error = GRBnewmodel(env, &model, "genconstr_c", NVARS, NULL,
                         NULL, NULL, NULL, NULL);
    if (error) goto QUIT;

    /* Initialize decision variables and objective */
    for (i = 0; i < NLITERALS; i++) {
        col = LIT(i);
        sprintf(buffer, "X%d", i);
        error = GRBsetcharattrelement(model, "VType", col, GRB_BINARY);
        if (error) goto QUIT;
        error = GRBsetstrattrelement(model, "VarName", col, buffer);
        if (error) goto QUIT;
        col = NOTLIT(i);
        sprintf(buffer, "notX%d", i);
        error = GRBsetcharattrelement(model, "VType", col, GRB_BINARY);
        if (error) goto QUIT;
        error = GRBsetstrattrelement(model, "VarName", col, buffer);
        if (error) goto QUIT;
    }

    /* Solve the problem */
    error = GRBoptimize(model);
    if (error) { GRBerror现状 emancipate; }
error = GRBsetcharattrelement(model, "VType", col, GRB_BINARY);
if (error) goto QUIT;

error = GRBsetstrattrelement(model, "VarName", col, buffer);
if (error) goto QUIT;
}

for (i = 0; i < NCLAUSES; i++) {
    col = CLA(i);
    sprintf(buffer, "Clause%d", i);
    error = GRBsetcharattrelement(model, "VType", col, GRB_BINARY);
    if (error) goto QUIT;

    error = GRBsetstrattrelement(model, "VarName", col, buffer);
    if (error) goto QUIT;
}

for (i = 0; i < NOBJ; i++) {
    col = OBJ(i);
    sprintf(buffer, "Obj%d", i);
    error = GRBsetcharattrelement(model, "VType", col, GRB_BINARY);
    if (error) goto QUIT;

    error = GRBsetstrattrelement(model, "VarName", col, buffer);
    if (error) goto QUIT;

    error = GRBsetdblattrelement(model, "Obj", col, 1.0);
    if (error) goto QUIT;
}

/* Link Xi and notXi */
for (i = 0; i < NLITERALS; i++) {
    sprintf(buffer, "CNSTR_X%d", i);
    cind[0] = LIT(i);
    cind[1] = NOTLIT(i);
    cval[0] = cval[1] = 1;
    error = GRBaddconstr(model, 2, cind, cval, GRB_EQUAL, 1.0, buffer);
    if (error) goto QUIT;
}

/* Link clauses and literals */
for (i = 0; i < NCLAUSES; i++) {
    sprintf(buffer, "CNSTR_Clause%d", i);
    error = GRBaddgenconstrOr(model, buffer, CLA(i), 3, Clauses[i]);
    if (error) goto QUIT;
}

/* Link objs with clauses */
for (i = 0; i < NCLAUSES; i++) {
    cind[i] = CLA(i);
    cval[i] = 1;
}
error = GRBaddgenconstrMin(model, "CNSTR_Obj0", OBJ(0), NCLAUSES, cind, GRB_INFINITY);
if (error) goto QUIT;

/* note that passing 4 instead of 4.0 will produce undefined behavior */
error = GRBaddgenconstrIndicator(model, "CNSTR_Obj1",
OBJ(1), 1, NCLAUSES, cind, cval,
GRB_GREATER_EQUAL, 4.0);

if (error) goto QUIT;

/* Set global objective sense */
error = GRBsetintattr(model, GRB_INT_ATTR_MODELSENSE, GRB_MAXIMIZE);
if (error) goto QUIT;

/* Save problem */
error = GRBwrite(model, "genconstr_c.mps");
if (error) goto QUIT;

error = GRBwrite(model, "genconstr_c.lp");
if (error) goto QUIT;

/* Optimize */
error = GRBoptimize(model);
if (error) goto QUIT;

/* Status checking */
error = GRBgetintattr(model, "Status", & status);
if (error) goto QUIT;

if (status == GRB_INF_OR_UNBD ||
    status == GRB_INFEASIBLE ||
    status == GRB_UNBOUNDED ) {
    printf("The model cannot be solved 
        because it is infeasible or unbounded\n");
    goto QUIT;
}
if (status != GRB_OPTIMAL) {
    printf("Optimization was stopped with status %i\n", status);
    goto QUIT;
}

/* Print result */
error = GRBgetdblattr(model, GRB_DBL_ATTR_OBJVAL, &objval);
if (error) goto QUIT;

if (objval > 1.9)
    printf("Logical expression is satisfiable\n");
else if (objval > 0.9)
    printf("At least four clauses can be satisfied\n");
else
    printf("At most three clauses may be satisfied\n");
QUIT:

if (model != NULL) GRBfreemodel(model);
if (env != NULL) GRBfreeenv(env);

return error;
/* Copyright 2020, Gurobi Optimization, LLC */

/* This example reads an LP model from a file and solves it. If the model is infeasible or unbounded, the example turns off presolve and solves the model again. If the model is infeasible, the example computes an Irreducible Inconsistent Subsystem (IIS), and writes it to a file */

#include <stdlib.h>
#include <stdio.h>
#include <math.h>
#include "gurobi_c.h"

int main(int argc, char *argv[]) {
    GRBenv *masterenv = NULL;
    GRBmodel *model = NULL;
    GRBenv *modelenv = NULL;
    int error = 0;
    int optimstatus;
    double objval;

    if (argc < 2) {
        fprintf(stderr, "Usage: lp_c filename\n");
        exit(1);
    }

    /* Create environment */
    error = GRBloadenv(&masterenv, "lp.log");
    if (error) goto QUIT;

    /* Read model from file */
    error = GRBreadmodel(masterenv, argv[1], &model);
    if (error) goto QUIT;

    /* Solve model */
    error = GRBoptimize(model);
    if (error) goto QUIT;

    /* Capture solution information */
    error = GRBgetintattr(model, GRB_INT_ATTR_STATUS, &optimstatus);
    if (error) goto QUIT;

    /* If model is infeasible or unbounded, turn off presolve and resolve */
    if (optimstatus == GRB_INF_OR_UNBD) {
        modelenv = GRBgetenv(model);
        if (!modelenv) {
            goto QUIT;
        } else {
            /* Turn off presolve */
            error = GRBsetenv(model, "presolve", "0");
            if (error) goto QUIT;

            /* Solve model again */
            error = GRBoptimize(model);
            if (error) goto QUIT;
        }
    }
    /* ... */

QUIT:
    if (modelenv) GRBfreemem(modelenv);
    if (masterenv) GRBfreemem(masterenv);
    /* ... */
    return 0;
}
fprintf(stderr, "Error: could not get model environment\n");
goto QUIT;
}

/* Change parameter on model environment. The model now has
   a copy of the master environment, so changing the master will
   no longer affect the model. */

error = GRBsetintparam(modelenv, "PRESOLVE", 0);
if (error) goto QUIT;

error = GRBoptimize(model);
if (error) goto QUIT;

error = GRBgetintattr(model, GRB_INT_ATTR_STATUS, &optimstatus);
if (error) goto QUIT;

if (optimstatus == GRB_OPTIMAL) {
    error = GRBgetdblattr(model, GRB_DBL_ATTR_OBJVAL, &objval);
    if (error) goto QUIT;
    printf("Optimal objective: %.4e\n", objval);
} else if (optimstatus == GRB_INFEASIBLE) {
    printf("Model is infeasible\n");
    error = GRBcomputeIIS(model);
    if (error) goto QUIT;
    error = GRBwrite(model, "model.ilp");
    if (error) goto QUIT;
} else if (optimstatus == GRB_UNBOUNDED) {
    printf("Model is unbounded\n");
} else {
    printf("Optimization was stopped with status = %d\n", optimstatus);
}

QUIT:

/* Error reporting */

if (error) {
    printf("ERROR: %s\n", GRBgeterrormsg(masterenv));
    exit(1);
}

/* Free model */

GRBfreemodel(model);

/* Free environment */

GRBfreeenv(masterenv);

return 0;
/* Copyright 2020, Gurobi Optimization, LLC */

/* Solve a model with different values of the Method parameter; show which value gives the shortest solve time. */

#include <stdlib.h>
#include <stdio.h>
#include "gurobi_c.h"

int main(int argc, char *argv[])
{
    GRBenv *env = NULL, *menv;
    GRBmodel *m = NULL;
    int error = 0;
    int i;
    int optimstatus;
    int bestMethod = -1;
    double bestTime;

    if (argc < 2)
    {
        fprintf(stderr, "Usage: lpmethod_c filename\n");
        exit(1);
    }

    error = GRBloadenv(&env, "lpmethod.log");
    if (error) goto QUIT;

    /* Read model */
    error = GRBreadmodel(env, argv[1], &m);
    if (error) goto QUIT;
    mev = GRBgetenv(m);
    error = GRBgetdblparam(menv, "TimeLimit", &bestTime);
    if (error) goto QUIT;

    /* Solve the model with different values of Method */
    for (i = 0; i <= 2; ++i)
    {
        error = GRBreset(m, 0);
        if (error) goto QUIT;
        error = GRBsetintparam(menv, "Method", i);
        if (error) goto QUIT;
        error = GRBoptimize(m);
        if (error) goto QUIT;
        error = GRBgetintattr(m, "Status", &optimstatus);
        if (error) goto QUIT;
        if (optimstatus == GRB_OPTIMAL) {
            error = GRBgetdblattr(m, "Runtime", &bestTime);
            if (error) goto QUIT;
            bestMethod = i;
            /* Reduce the TimeLimit parameter to save time with other methods */
        }
    }
}

QUIT:

error = GRBsetdblparam(menv, "TimeLimit", bestTime);
if (error) goto QUIT;
}

/* Report which method was fastest */
if (bestMethod == -1) {
    printf("Unable to solve this model\n");
} else {
    printf("Solved in %f seconds with Method: %i\n",
           bestTime, bestMethod);
}

QUIT:

/* Error reporting */
if (error)
{
    printf("ERROR: %s\n", GRBgeterrormsg(env));
    exit(1);
}

/* Free model */
GRBfreemodel(m);

/* Free environment */
GRBfreeenv(env);

return 0;

lpmod_c.c

/* Copyright 2020, Gurobi Optimization, LLC */

/* This example reads an LP model from a file and solves it. If the model can be solved, then it finds the smallest positive variable, sets its upper bound to zero, and resolves the model two ways: first with an advanced start, then without an advanced start (i.e. 'from scratch'). */

#include <stdlib.h>
#include <stdio.h>
#include <math.h>
#include <string.h>
#include "gurobi_c.h"

int main(int argc,
        char *argv[])
{
    GRBenv *env = NULL;
    GRBmodel *model = NULL;
int error = 0;
int j;
int numvars, isMIP, status, minVar = 0;
double minVal = GRB_INFINITY, sol, lb;
char *varname;
double warmCount, warmTime, coldCount, coldTime;

if (argc < 2)
{
    fprintf(stderr, "Usage: lpmod_c filename\n");
    exit(1);
}

error = GRBloadenv(&env, "lpmod.log");
if (error) goto QUIT;

/* Read model and determine whether it is an LP */
error = GRBreadmodel(env, argv[1], &model);
if (error) goto QUIT;
error = GRBgetintattr(model, "IsMIP", &isMIP);
if (error) goto QUIT;
if (isMIP)
{
    printf("The model is not a linear program\n");
    goto QUIT;
}

error = GRBoptimize(model);
if (error) goto QUIT;

error = GRBgetintattr(model, "Status", &status);
if (error) goto QUIT;

if ((status == GRB_INF_OR_UNBD) || (status == GRB_INFEASIBLE) ||
    (status == GRB_UNBOUNDED))
{
    printf("The model cannot be solved because it is ");
    printf("infeasible or unbounded\n");
    goto QUIT;
}

if (status != GRB_OPTIMAL)
{
    printf("Optimization was stopped with status %i\n", status);
    goto QUIT;
}

/* Find the smallest variable value */
error = GRBgetintattr(model, "NumVars", &numvars);
if (error) goto QUIT;
for (j = 0; j < numvars; ++j)
{
    error = GRBgetdblattrelement(model, "X", j, &sol);
    if (error) goto QUIT;
    error = GRBgetdblattrelement(model, "LB", j, &lb);
    if (error) goto QUIT;
}
if ((sol > 0.0001) && (sol < minVal) &&
    (lb == 0.0))
{
    minVal = sol;
    minVar = j;
}
}

error = GRBgetstrattrelement(model, "VarName", minVar, &varname);
if (error) goto QUIT;
printf("\n*** Setting %s from %f to zero ***\n\n", varname, minVal);
error = GRBsetdblattrelement(model, "UB", minVar, 0.0);
if (error) goto QUIT;

/* Solve from this starting point */
error = GRBoptimize(model);
if (error) goto QUIT;

/* Save iteration & time info */
error = GRBgetdblattretr(model, "IterCount", &warmCount);
if (error) goto QUIT;
error = GRBgetdblattretr(model, "Runtime", &warmTime);
if (error) goto QUIT;

/* Reset the model and resolve */
printf("\n*** Resetting and solving ");
printf("without an advanced start ***\n\n");
error = GRBreset(model, 0);
if (error) goto QUIT;
error = GRBoptimize(model);
if (error) goto QUIT;

/* Save iteration & time info */
error = GRBgetdblattretr(model, "IterCount", &coldCount);
if (error) goto QUIT;
error = GRBgetdblattretr(model, "Runtime", &coldTime);
if (error) goto QUIT;
printf("\n*** Warm start: %f iterations, %f seconds\n", warmCount, warmTime);
printf("*** Cold start: %f iterations, %f seconds\n", coldCount, coldTime);

QUIT:

/* Error reporting */

if (error)
{
    printf("ERROR: %s\n", GRBgeterrormsg(env));
    exit(1);
}

/* Free model */
GRBfreemodel(model);
/* Free environment */
GRBfreeenv(env);

return 0;
}

mip1_c.c

/* Copyright 2020, Gurobi Optimization, LLC */

/* This example formulates and solves the following simple MIP model:

maximize x + y + 2 z
subject to x + 2 y + 3 z <= 4
       x + y >= 1
       x, y, z binary

*/

#include <stdlib.h>
#include <stdio.h>
#include "gurobi_c.h"

int main(int argc, char *argv[])
{
    GRBenv *env = NULL;
    GRBmodel *model = NULL;
    int  error = 0;
    double sol[3];
    int  ind[3];
    double val[3];
    double obj[3];
    char  vtype[3];
    int  optimstatus;
    double objval;

    /* Create environment */
    error = GRBemptyenv(&env);
    if (error) goto QUIT;

eerror = GRBsetstrparam(env, "LogFile", "mip1.log");
    if (error) goto QUIT;

    error = GRBstartenv(env);
    if (error) goto QUIT;

    /* Create an empty model */
    error = GRBnewmodel(env, &model, "mip1", 0, NULL, NULL, NULL, NULL, NULL);
    if (error) goto QUIT;

    /* Add variables */
    obj[0] = 1; obj[1] = 1; obj[2] = 2;
error = GRBaddvars(model, 3, 0, NULL, NULL, NULL, obj, NULL, NULL, vtype, NULL);
if (error) goto QUIT;

/* Change objective sense to maximization */
error = GRBsetintattr(model, GRB_INT_ATTR_MODELSENSE, GRB_MAXIMIZE);
if (error) goto QUIT;

/* First constraint: x + 2 y + 3 z <= 4 */
ind[0] = 0; ind[1] = 1; ind[2] = 2;
val[0] = 1; val[1] = 2; val[2] = 3;
error = GRBaddconstr(model, 3, ind, val, GRB_LESS_EQUAL, 4.0, "c0");
if (error) goto QUIT;

/* Second constraint: x + y >= 1 */
ind[0] = 0; ind[1] = 1;
val[0] = 1; val[1] = 1;
error = GRBaddconstr(model, 2, ind, val, GRB_GREATER_EQUAL, 1.0, "c1");
if (error) goto QUIT;

/* Optimize model */
error = GRBoptimize(model);
if (error) goto QUIT;

/* Write model to 'mip1.lp' */
error = GRBwrite(model, "mip1.lp");
if (error) goto QUIT;

/* Capture solution information */
error = GRBgetintattr(model, GRB_INT_ATTR_STATUS, &optimstatus);
if (error) goto QUIT;
error = GRBgetdblattr(model, GRB_DBL_ATTR_OBJVAL, &objval);
if (error) goto QUIT;
error = GRBgetdblattrarray(model, GRB_DBL_ATTR_X, 0, 3, sol);
if (error) goto QUIT;

printf("\nOptimization complete\n");
if (optimstatus == GRB_OPTIMAL) {
    printf("Optimal objective: %.4e\n", objval);
    printf(" x=%.0f, y=%.0f, z=%.0f\n", sol[0], sol[1], sol[2]);
} else if (optimstatus == GRB_INF_OR_UNBD) {
    printf("Model is infeasible or unbounded\n");
} else {
    printf("Optimization was stopped early\n");
}

QUIT:

/* Error reporting */
if (error) {
printf("ERROR: %s\n", GRBgeterrormsg(env));
exit(1);
}

/* Free model */
GRBfreemodel(model);

/* Free environment */
GRBfreeenv(env);

return 0;
}

mip2_c.c

/* Copyright 2020, Gurobi Optimization, LLC */

/* This example reads a MIP model from a file, solves it and
prints the objective values from all feasible solutions
generated while solving the MIP. Then it creates the fixed
model and solves that model. */

#include <stdlib.h>
#include <stdio.h>
#include <math.h>
#include "gurobi_c.h"

int main(int argc, char *argv[])
{
  GRBenv *env = NULL;
  GRBmodel *model = NULL;
  GRBmodel *fixed = NULL;
  int error = 0;
  int ismip;
  int j, k, solcount, numvars;
  double objn, objb, x;
  int optimstatus, foptimstatus;
  double objval, fobjval;
  char *varname;
  double x;

  /* To change settings for a loaded model, we need to get
     the model environment, which will be freed when the model
     is freed. */

  GRBenv *menv, *fenv;

  if (argc < 2) {
    fprintf(stderr, "Usage: mip2_c filename\n");
    exit(1);
  }

  /* Create environment */
error = GRBloadenv(&env, "mip2.log");
if (error) goto QUIT;

/* Read model from file */
error = GRBreadmodel(env, argv[1], &model);
if (error) goto QUIT;

error = GRBgetintattr(model, "IsMIP", &ismip);
if (error) goto QUIT;

if (ismip == 0) {
    printf("Model is not a MIP\n");
    goto QUIT;
}

/* Get model environment */
menv = GRBgetenv(model);
if (!menv) {
    fprintf(stderr, "Error: could not get model environment\n");
    goto QUIT;
}

/* Solve model */
error = GRBoptimize(model);
if (error) goto QUIT;

/* Capture solution information */
error = GRBgetintattr(model, GRB_INT_ATTR_STATUS, &optimstatus);
if (error) goto QUIT;

printf("\nOptimization complete\n");
if (optimstatus == GRB_OPTIMAL) {
    error = GRBgetdblattr(model, GRB_DBL_ATTR_OBJVAL, &objval);
    if (error) goto QUIT;
    printf("Optimal objective: %.4e\n", objval);
} else if (optimstatus == GRB_INF_OR_UNBD) {
    printf("Model is infeasible or unbounded\n");
    goto QUIT;
} else if (optimstatus == GRB_INFEASIBLE) {
    printf("Model is infeasible\n");
    goto QUIT;
} else if (optimstatus == GRB_UNBOUNDED) {
    printf("Model is unbounded\n");
    goto QUIT;
} else {
    printf("Optimization was stopped with status = %d\n", optimstatus);
    goto QUIT;
}

/* Iterate over the solutions and compute the objectives */
error = GRBsetintparam(menv, "OutputFlag", 0);
if (error) goto QUIT;
error = GRBgetintattr(model, "SolCount", &solcount);
if (error) goto QUIT;
error = GRBgetintattr(model, "NumVars", &numvars);
if (error) goto QUIT;

printf("\n");
for ( k = 0; k < solcount; ++k ) {
    error = GRBsetintparam(menv, "SolutionNumber", k);
    objn = 0.0;
    for ( j = 0; j < numvars; ++j ) {
        error = GRBgetdblattrelement(model, "Obj", j, &vobj);
        if (error) goto QUIT;
        error = GRBgetdblattrelement(model, "Xn", j, &xn);
        if (error) goto QUIT;
        objn += vobj * xn;
    }
    printf(" Solution %i has objective: %.f\n", k, objn);
}
printf("\n");

error = GRBsetintparam(menv, "OutputFlag", 1);
if (error) goto QUIT;

/* Create a fixed model, turn off presolve and solve */
error = GRBfixmodel(model, &fixed);
if (error || !fixed) {
    fprintf(stderr, "Error: could not create fixed model\n");
    goto QUIT;
}

fenv = GRBgetenv(fixed);
if (!fenv) {
    fprintf(stderr, "Error: could not get fixed model environment\n");
    goto QUIT;
}

error = GRBsetintparam(fenv, "PRESOLVE", 0);
if (error) goto QUIT;

error = GRBoptimize(fixed);
if (error) goto QUIT;

error = GRBgetintattr(fixed, GRB_INT_ATTR_STATUS, &foptimstatus);
if (error) goto QUIT;

if (foptimstatus != GRB_OPTIMAL) {
    fprintf(stderr, "Error: fixed model isn’t optimal\n");
    goto QUIT;
}

error = GRBgetdblattr(fixed, GRB_DBL_ATTR_OBJVAL, &fobjval);
if (error) goto QUIT;

if (fabs(fobjval - objval) > 1.0e-6 * (1.0 + fabs(objval))) {

70
fprintf(stderr, "Error: objective values are different\n");
}

/* Print values of nonzero variables */
for ( j = 0; j < numvars; ++j ) {
    error = GRBgetstrattrelement(fixed, "VarName", j, &varname);
    if (error) goto QUIT;
    error = GRBgetdblattrelement(fixed, "X", j, &x);
    if (error) goto QUIT;
    if (x != 0.0) {
        printf("%s %f\n", varname, x);
    }
}
QUIT:

/* Error reporting */
if (error) {
    printf("ERROR: %s\n", GRBgeterrormsg(env));
    exit(1);
}
/* Free models */
GRBfreemodel(model);
GRBfreemodel(fixed);
/* Free environment */
GRBfreeenv(env);
return 0;
}

multiobj_c.c

/* Copyright 2020, Gurobi Optimization, LLC */

/* Want to cover three different sets but subject to a common budget of 
* elements allowed to be used. However, the sets have different priorities to 
* be covered; and we tackle this by using multi-objective optimization. */

#include <stdlib.h>
#include <stdio.h>
#include <math.h>
#include <string.h>
#include "gurobi_c.h"

#define MAXSTR 128

int
main(void)
{
    GRBenv  *env  = NULL;
GRBEnv * menv = NULL;
GRBmodel * model = NULL;
int error = 0;
int * cind = NULL;
double * cval = NULL;
char buffer[MAXSTR];
int e, i, status, nSolutions;
double objn;

/* Sample data */
const int groundSetSize = 20;
const int nSubsets = 4;
const int Budget = 12;
double Set[][20] = 
{ { 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0 },
  { 0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1 },
  { 0, 0, 0, 1, 1, 0, 1, 1, 0, 0, 0, 0, 1, 1, 0, 1, 1, 0, 0, 0 },
  { 0, 0, 0, 1, 1, 1, 1, 0, 0, 0, 1, 1, 1, 1, 0, 1, 1, 0, 0, 0 } };
int SetObjPriority[] = {3, 2, 2, 1};
double SetObjWeight[] = {1.0, 0.25, 1.25, 1.0};

/* Create environment */
error = GRBloadenv(& env, "multiobj_c.log");
if (error) goto QUIT;

/* Create initial model */
error = GRBnewmodel(env, & model, "multiobj_c", groundSetSize, NULL,
  NULL, NULL, NULL, NULL);
if (error) goto QUIT;

/* get model environment */
menv = GRBgetenv(model);
if (!menv) {
  fprintf(stderr, "Error: could not get model environment\n");
  goto QUIT;
}

/* Initialize decision variables for ground set: */
x[e] == 1 if element e is chosen for the covering. */
for (e = 0; e < groundSetSize; e++) {
  sprintf(buffer, "El%d", e);
  error = GRBsetcharattrelement(model, "VType", e, GRB_BINARY);
  if (error) goto QUIT;
  error = GRBsetstrattrelement(model, "VarName", e, buffer);
  if (error) goto QUIT;
}

/* Make space for constraint data */
cind = malloc(sizeof(int) * groundSetSize);
if (!cind) goto QUIT;
cval = malloc(sizeof(double) * groundSetSize);
if (!cval) goto QUIT;

/* Constraint: limit total number of elements to be picked to be at most */

Budget */
for (e = 0; e < groundSetSize; e++) {
    cind[e] = e;
    cval[e] = 1.0;
}
sprintf ( buffer , "Budget");
error = GRBaddconstr (model, groundSetSize, cind, cval, GRB_LESS_EQUAL, (double)Budget, buffer);
if (error) goto QUIT;

/* Set global sense for ALL objectives */
error = GRBsetintattr(model, GRB_INT_ATTR_MODELSENSE, GRB_MAXIMIZE);
if (error) goto QUIT;

/* Limit how many solutions to collect */
error = GRBsetintparam(menv, GRB_INT_PAR_POOLSOLUTIONS, 100);
if (error) goto QUIT;

/* Set and configure i-th objective */
for (i = 0; i < nSubsets; i++) {
    sprintf(buffer, "Set%d", i+1);
    error = GRBsetobjectiven(model, i, SetObjPriority[i], SetObjWeight[i],
                                1.0 + i, 0.01, buffer, 0.0, groundSetSize,
                                cind, Set[i]);
    if (error) goto QUIT;
}

/* Save problem */
error = GRBwrite(model, "multiobj_c.lp");
if (error) goto QUIT;
error = GRBwrite(model, "multiobj_c.mps");
if (error) goto QUIT;

/* Optimize */
error = GRBoptimize(model);
if (error) goto QUIT;

/* Status checking */
error = GRBgetintattr(model, "Status", &status);
if (error) goto QUIT;

if (status == GRB_INF_OR_UNBD ||
    status == GRB_INFEASIBLE ||
    status == GRB_UNBOUNDED ) {
    printf("The model cannot be solved 
" "because it is infeasible or unbounded\n");
    goto QUIT;
}
if (status != GRB_OPTIMAL) {
    printf("Optimization was stopped with status %i\n", status);
    goto QUIT;
}

/* Print best selected set */
error = GRBgetdblattrarray(model, GRB_DBL_ATTR_X, 0, groundSetSize, cval);
if (error) goto QUIT;
printf("Selected elements in best solution:\n\t");
for (e = 0; e < groundSetSize; e++) {
    if (cval[e] < .9) continue;
    printf("El%d \t", e);
}

/* Print number of solutions stored */
error = GRBgetintattr(model, GRB_INT_ATTR_SOLCOUNT, &nSolutions);
if (error) goto QUIT;
printf("\nNumber of solutions found: %d\n", nSolutions);

/* Print objective values of solutions */
if (nSolutions > 10) nSolutions = 10;
printf("Objective values for first %d solutions: \n", nSolutions);
for (i = 0; i < nSubsets; i++) {
    error = GRBsetintparam(menv, GRB_INT_PAR_OBJNUMBER, i);
    if (error) goto QUIT;

    printf("\tSet %d: ", i);
    for (e = 0; e < nSolutions; e++) {
        error = GRBsetintparam(menv, GRB_INT_PAR_SOLUTIONNUMBER, e);
        if (error) goto QUIT;

        error = GRBgetdblattr(model, GRB_DBL_ATTR_OBJNVAL, &objn);
        if (error) goto QUIT;

        printf(" %6g", objn);
    }
    printf("\n");
}

QUIT:
if (cind != NULL) free(cind);
if (cval != NULL) free(cval);
if (model != NULL) GRBfreemodel(model);
if (env != NULL) GRBfreeenv(env);
return error;

multiscenario_c.c

/* Copyright 2020, Gurobi Optimization, LLC */

/* Facility location: a company currently ships its product from 5 plants to 4 warehouses. It is considering closing some plants to reduce costs. What plant(s) should the company close, in order to minimize transportation and fixed costs? Since the plant fixed costs and the warehouse demands are uncertain, a scenario approach is chosen. Note that this example is similar to the facility_c.c example. Here we... */
added scenarios in order to illustrate the multi-scenario feature.

Based on an example from Frontline Systems:
http://www.solver.com/disfacility.htm
Used with permission.

*/

#include <stdlib.h>
#include <stdio.h>
#include <math.h>
#include "gurobi_c.h"

#define opencol(p) p
#define transportcol(w,p) nPlants*(w+1)+p
#define demandconstr(w) nPlants+w
#define MAXSTR 128

int main(int argc, char *argv[])
{
    GRBenv * env = NULL;
    GRBenv * modelenv = NULL;
    GRBmodel * model = NULL;
    double * cval = NULL;
    double * rhs = NULL;
    int * cbeg = NULL;
    int * cind = NULL;
    char ** cname = NULL;
    char * sense = NULL;
    double maxFixed = -GRB_INFINITY;
    double minFixed = GRB_INFINITY;
    int cnamect = 0;
    int error = 0;

    int p, s, w, col;
    int idx, rowct;
    int nScenarios;
    char vname[MAXSTR];

    /* Number of plants, warehouses and scenarios */
    const int nPlants = 5;
    const int nWarehouses = 4;

    /* Warehouse demand in thousands of units */
    double Demand[] = { 15, 18, 14, 20 };

    /* Plant capacity in thousands of units */
    double Capacity[] = { 20, 22, 17, 19, 18 };

    /* Fixed costs for each plant */
    double FixedCosts[] =
    { 12000, 15000, 17000, 13000, 16000 };
/* Transportation costs per thousand units */
double TransCosts[4][5] = {
    { 4000, 2000, 3000, 2500, 4500 },
    { 2500, 2600, 3400, 3000, 4000 },
    { 1200, 1800, 2600, 4100, 3000 },
    { 2200, 2600, 3100, 3700, 3200 }
};

/* Compute minimal and maximal fixed cost */
for (p = 0; p < nPlants; p++) {
    if (FixedCosts[p] > maxFixed)
        maxFixed = FixedCosts[p];
    if (FixedCosts[p] < minFixed)
        minFixed = FixedCosts[p];
}

/* Create environment */
error = GRBloadenv(&env, "multiscenario.log");
if (error) goto QUIT;

/* Create initial model */
error = GRBnewmodel(env, &model, "multiscenario", nPlants * (nWarehouses + 1),
        NULL, NULL, NULL, NULL, NULL);
if (error) goto QUIT;

modelenv = GRBgetenv(model);

/* Initialize decision variables for plant open variables */
for (p = 0; p < nPlants; p++) {
    col = opencol(p);
    error = GRBsetcharattrelement(model, GRB_CHAR_ATTR_VTYPE,
        col, GRB_BINARY);
    if (error) goto QUIT;
    error = GRBsetdblattrelement(model, GRB_DBL_ATTR_OBJ,
        col, FixedCosts[p]);
    if (error) goto QUIT;
    sprintf(vname, "Open%i", p);
    error = GRBsetstrattrelement(model, GRB_STR_ATTR_VARNAME,
        col, vname);
    if (error) goto QUIT;
}

/* Initialize decision variables for transportation decision variables:
   how much to transport from a plant p to a warehouse w */
for (w = 0; w < nWarehouses; w++) {
    for (p = 0; p < nPlants; p++) {
        col = transportcol(w, p);
        error = GRBsetdblattrelement(model, GRB_DBL_ATTR_OBJ,
            col, TransCosts[w][p]);
        if (error) goto QUIT;
        sprintf(vname, "Trans%i.%i", p, w);
        error = GRBsetstrattrelement(model, GRB_STR_ATTR_VARNAME,
            col, vname);
        if (error) goto QUIT;
    }
}
The objective is to minimize the total fixed and variable costs:

```c
/* The objective is to minimize the total fixed and variable costs */
error = GRBsetintattr(model, GRB_INT_ATTR_MODELSENSE, GRB_MINIMIZE);
if (error) goto QUIT;

/* Make space for constraint data */
rowct = (nPlants > nWarehouses) ? nPlants : nWarehouses;
cbeg = malloc(sizeof(int) * rowct);
if (!cbeg) goto QUIT;
cind = malloc(sizeof(int) * (nPlants * (nWarehouses + 1)));
if (!cind) goto QUIT;
cval = malloc(sizeof(double) * (nPlants * (nWarehouses + 1)));
if (!cval) goto QUIT;
rhs = malloc(sizeof(double) * rowct);
if (!rhs) goto QUIT;
sense = malloc(sizeof(char) * rowct);
if (!sense) goto QUIT;
cname = calloc(rowct, sizeof(char*));
if (!cname) goto QUIT;

/* Production constraints */
Note that the limit sets the production to zero if
the plant is closed */
idx = 0;
for (p = 0; p < nPlants; p++) {
    cbeg[p] = idx;
rhs[p] = 0.0;
sense[p] = GRB_LESS_EQUAL;
cname[p] = malloc(sizeof(char) * MAXSTR);
    if (!cname[p]) goto QUIT;
cnamect ++;
sprintf(cname[p], "Capacity%i", p);
for (w = 0; w < nWarehouses; w++) {
    cind[idx] = transportcol(w, p);
cval[idx++] = 1.0;
}
cind[idx] = opencol(p);
cval[idx++] = -Capacity[p];
}
error = GRBaddconstrs(model, nPlants, idx, cbeg, cind, cval, sense,
rhs, cname);
if (error) goto QUIT;

/* Demand constraints */
idx = 0;
for (w = 0; w < nWarehouses; w++) {
    cbeg[w] = idx;
sense[w] = GRB_EQUAL;
sprintf(cname[w], "Demand%i", w);
for (p = 0; p < nPlants; p++) {
    cind[idx] = transportcol(w, p);
cval[idx++] = 1.0;
}
}
```
error = GRBaddconstrs(model, nWarehouses, idx, cbeg, cind, cval, sense,
        Demand, cname);
if (error) goto QUIT;

/* We constructed the base model, now we add 7 scenarios */

Scenario 0: Represents the base model, hence, no manipulations.
Scenario 1: Manipulate the warehouses demands slightly (constraint right
    hand sides).
Scenario 2: Double the warehouses demands (constraint right hand sides).
Scenario 3: Manipulate the plant fixed costs (objective coefficients).
Scenario 4: Manipulate the warehouses demands and fixed costs.
Scenario 5: Force the plant with the largest fixed cost to stay open
    (variable bounds).
Scenario 6: Force the plant with the smallest fixed cost to be closed
    (variable bounds). */

error = GRBsetintattr(model, GRB_INT_ATTR_NUMSCENARIOS, 7);
if (error) goto QUIT;

/* Scenario 0: Base model, hence, nothing to do except giving the
    scenario a name */
error = GRBsetintparam(modelenv, GRB_INT_PAR_SCENARIONUMBER, 0);
if (error) goto QUIT;
error = GRBsetstrattr(model, GRB_STR_ATTR_SCENNNAME, "Base model");
if (error) goto QUIT;

/* Scenario 1: Increase the warehouse demands by 10% */
error = GRBsetintparam(modelenv, GRB_INT_PAR_SCENARIONUMBER, 1);
if (error) goto QUIT;
error = GRBsetstrattr(model, GRB_STR_ATTR_SCENNNAME,
    "Increased warehouse demands");
if (error) goto QUIT;

for (w = 0; w < nWarehouses; w++) {
    error = GRBsetdblattrelement(model, GRB_DBL_ATTR_SCENNRHS,
        demandconstr(w), Demand[w] * 1.1);
    if (error) goto QUIT;
}

/* Scenario 2: Double the warehouse demands */
error = GRBsetintparam(modelenv, GRB_INT_PAR_SCENARIONUMBER, 2);
if (error) goto QUIT;
error = GRBsetstrattr(model, GRB_STR_ATTR_SCENNNAME,
    "Double the warehouse demands");
if (error) goto QUIT;

for (w = 0; w < nWarehouses; w++) {
    error = GRBsetdblattrelement(model, GRB_DBL_ATTR_SCENNRHS,
        demandconstr(w), Demand[w] * 2.0);
    if (error) goto QUIT;
}

/* Scenario 3: Decrease the plant fixed costs by 5% */
error = GRBsetintparam(modelenv, GRB_INT_PAR_SCENARIONUMBER, 3);
if (error) goto QUIT;

78
error = GRBsetstrattr(model, GRB_STR_ATTR_SCENNNAME, "Decreased plant fixed costs");
if (error) goto QUIT;

for (p = 0; p < nPlants; p++) {
    error = GRBsetdblattrelement(model, GRB_DBL_ATTR_SCENNOBJ, opencol(p), FixedCosts[p] * 0.95);
    if (error) goto QUIT;
}

/* Scenario 4: Combine scenario 1 and scenario 3 */
error = GRBsetintparam(modelenv, GRB_INT_PAR_SCENARIONUMBER, 4);
if (error) goto QUIT;
error = GRBsetstrattr(model, GRB_STR_ATTR_SCENNNAME, "Increased warehouse demands and decreased plant fixed costs");

for (w = 0; w < nWarehouses; w++) {
    error = GRBsetdblattrelement(model, GRB_DBL_ATTR_SCENNRHS, demandconstr(w), Demand[w] * 1.1);
    if (error) goto QUIT;
}
for (p = 0; p < nPlants; p++) {
    error = GRBsetdblattrelement(model, GRB_DBL_ATTR_SCENNOBJ, opencol(p), FixedCosts[p] * 0.95);
    if (error) goto QUIT;
}

/* Scenario 5: Force the plant with the largest fixed cost to stay open */
error = GRBsetintparam(modelenv, GRB_INT_PAR_SCENARIONUMBER, 5);
if (error) goto QUIT;
error = GRBsetstrattr(model, GRB_STR_ATTR_SCENNNAME, "Force plant with largest fixed cost to stay open");
if (error) goto QUIT;

for (p = 0; p < nPlants; p++) {
    if (FixedCosts[p] == maxFixed) {
        error = GRBsetdblattrelement(model, GRB_DBL_ATTR_SCENNLB, opencol(p), 1.0);
        if (error) goto QUIT;
        break;
    }
}

/* Scenario 6: Force the plant with the smallest fixed cost to be closed */
error = GRBsetintparam(modelenv, GRB_INT_PAR_SCENARIONUMBER, 6);
if (error) goto QUIT;
error = GRBsetstrattr(model, GRB_STR_ATTR_SCENNNAME, "Force plant with smallest fixed cost to be closed");
if (error) goto QUIT;

for (p = 0; p < nPlants; p++) {
    if (FixedCosts[p] == minFixed) {
        error = GRBsetdblattrelement(model, GRB_DBL_ATTR_SCENNUB, opencol(p), 0.0);
        if (error) goto QUIT;
    }
}
if (error) goto QUIT;
break;
}
}

/* Guess at the starting point: close the plant with the highest
fixed costs; open all others */

/* First, open all plants */
for (p = 0; p < nPlants; p++) {
    error = GRBsetdblattrelement(model, GRB_DBL_ATTR_START, opencol(p), 1.0);
    if (error) goto QUIT;
}

/* Now close the plant with the highest fixed cost */
printf("Initial guess:
");
for (p = 0; p < nPlants; p++) {
    if (FixedCosts[p] == maxFixed) {
        error = GRBsetdblattrelement(model, GRB_DBL_ATTR_START, opencol(p), 0.0);
        if (error) goto QUIT;
        printf("Closing plant %i\n", p);
        break;
    }
}

/* Use barrier to solve root relaxation */
error = GRBsetintparam(modelenv, GRB_INT_PAR_METHOD,
                        GRB_METHOD_BARRIER);
if (error) goto QUIT;

/* Solve multi-scenario model */
error = GRBoptimize(model);
if (error) goto QUIT;

error = GRBgetintattr(model, GRB_INT_ATTR_NUMSCENARIOS, &nScenarios);
if (error) goto QUIT;

/* Print solution for each */
for (s = 0; s < nScenarios; s++) {
    char *scenarioName;
    double scenNObjBound;
    double scenNObjVal;
    int modelSense = GRB_MINIMIZE;
    /* Set the scenario number to query the information for this
scenario */
    error = GRBsetintparam(modelenv, GRB_INT_PAR_SCENARIONUMBER, s);
    if (error) goto QUIT;
    /* Collect result for the scenario */
    error = GRBgetstrattr(model, GRB_STR_ATTR_SCENNAME, &scenarioName);
    if (error) goto QUIT;
    error = GRBgetdblattr(model, GRB_DBL_ATTR_SCENOBJBOUND, &scenNObjBound);
    if (error) goto QUIT;
    error = GRBgetdblattr(model, GRB_DBL_ATTR_SCENOBJVAL, &scenNObjVal);
if (error) goto QUIT;

printf("\n\n------ Scenario %d (%s)\n", s, scenarioName);

/* Check if we found a feasible solution for this scenario */
if (scenNObjVal >= modelSense * GRB_INFINITY)
    if (scenNObjBound >= modelSense * GRB_INFINITY)
        /* Scenario was proven to be infeasible */
        printf("\nINFEASIBLE\n");
    else
        /* We did not find any feasible solution - should not happen in this case, because we did not set any limit (like a time limit) on the optimization process */
        printf("\nNO SOLUTION\n");
else {
    printf("\nTOTAL COSTS: %g\n", scenNObjVal);
    printf("SOLUTION:\n");
    for (p = 0; p < nPlants; p++) {
        double scenNX;
        error = GRBgetdblattrelement(model, GRB_DBL_ATTR_SCENNX,
                                      opencol(p), &scenNX);
        if (error) goto QUIT;

        if (scenNX > 0.5) {
            printf("Plant %i open\n", p);
            for (w = 0; w < nWarehouses; w++) {
                error = GRBgetdblattrelement(model, GRB_DBL_ATTR_SCENNX,
                                               transportcol(w, p), &scenNX);
                if (error) goto QUIT;
                if (scenNX > 0.0001)
                    printf(" Transport %g units to warehouse %i\n",
                            scenNX, w);
            }
        } else
            printf("Plant %i closed!\n", p);
    }
}

/* Print a summary table: for each scenario we add a single summary line */
printf("\n\nSummary: Closed plants depending on scenario\n\n");
printf("%8s | %17s %13s\n", " Scenario ", " Plant ", " Costs ", " Name ");

printf("%8s |", "Scenario");
for (p = 0; p < nPlants; p++)
    printf(" %5d", p);
printf(" | %6s %s\n", "Costs", "Name");

for (s = 0; s < nScenarios; s++) {
    char *scenarioName;
    double scenNObjBound;
    double scenNObjVal;
    int modelSense = GRB_MINIMIZE;
/* Set the scenario number to query the information for this scenario */
error = GRBsetintparam(modelenv, GRB_INT_PAR_SCENARIONUMBER, s);
if (error) goto QUIT;

/* collect result for the scenario */
error = GRBgetstrattr(model, GRB_STR_ATTR_SCENName, &scenarioName);
if (error) goto QUIT;
error = GRBgetdblattr(model, GRB_DBL_ATTR_SCENNOBJBOUND, &scenNObjBound);
if (error) goto QUIT;
error = GRBgetdblattr(model, GRB_DBL_ATTR_SCENNOBJVal, &scenNObjVal);
if (error) goto QUIT;

printf("%-8d |", s);

/* Check if we found a feasible solution for this scenario */
if (scenNObjVal >= modelSense * GRB_INFINITY)
if (scenNObjBound >= modelSense * GRB_INFINITY)
   /* Scenario was proven to be infeasible */
   printf("%-30s| %6s %s
", "infeasible", "-", scenarioName);
else
   /* We did not find any feasible solution - should not happen in
this case, because we did not set any limit (like a time
limit) on the optimization process */
   printf("%-30s| %6s %s
", "no solution found", "-", scenarioName);
else {
   for (p = 0; p < nPlants; p++) {
      double scenNX;
      error = GRBgetdblattrelement(model, GRB_DBL_ATTR_SCENNX,
         opencol(p), &scenNX);
      if (scenNX > 0.5)
         printf(" %5s", " ");
      else
         printf(" %5s", "x");
   }
   printf(" | %6g %s
", scenNObjVal, scenarioName);
}

QUIT:

/* Error reporting */
if (error) {
   printf("ERROR: %s\n", GRBgeterrormsg(env));
   exit(1);
}

/* Free data */
free(cbeg);
free(cind);
free(cval);
free(rhs);
free(sense);
for (p = 0; p < cnamect; p++)
free(cname[p]);
free(cname);

/* Free model */
GRBfreemodel(model);

/* Free environment */
GRBfreeenv(env);

return 0;
}

params_c.c

/* Copyright 2020, Gurobi Optimization, LLC */

/* Use parameters that are associated with a model.

   A MIP is solved for a few seconds with different sets of parameters. The one with the smallest MIP gap is selected, and the optimization is resumed until the optimal solution is found.
*/

#include <stdlib.h>
#include <stdio.h>
#include <math.h>
#include "gurobi_c.h"

int main(int argc, char *argv[])
{
    GRBenv *env = NULL, *modelenv = NULL, *bestenv = NULL;
    GRBmodel *model = NULL, *bestmodel = NULL;
    int error = 0;
    int ismip, i, mipfocus;
    double bestgap, gap;

    if (argc < 2)
    {
        fprintf(stderr, "Usage: params_c filename\n");
        exit(1);
    }

    error = GRBloadenv(&env, "params.log");
    if (error) goto QUIT;

    /* Read model and verify that it is a MIP */
    error = GRBreadmodel(env, argv[1], &model);
    if (error) goto QUIT;
    error = GRBgetintattr(model, "IsMIP", &ismip);
    if (error) goto QUIT;
    if (ismip == 0)
    {
        printf("The model is not an integer program\n");
        exit(1);
    }
/ * Set a 2 second time limit */
modelenv = GRBgetenv(model);
if (!modelenv) {
    printf("Cannot retrieve model environment\n");
    exit(1);
}
error = GRBsetdblparam(modelenv, "TimeLimit", 2);
if (error) goto QUIT;

/* Now solve the model with different values of MIPFocus */
bestmodel = GRBcopymodel(model);
if (!bestmodel) {
    printf("Cannot copy model\n");
    exit(1);
}
error = GRBoptimize(bestmodel);
if (error) goto QUIT;
error = GRBgetdblattr(bestmodel, "MIPGap", &bestgap);
if (error) goto QUIT;
for (i = 1; i <= 3; ++i)
{
    error = GRBreset(model, 0);
    if (error) goto QUIT;
    modelenv = GRBgetenv(model);
    if (!modelenv) {
        printf("Cannot retrieve model environment\n");
        exit(1);
    }
    error = GRBsetintparam(modelenv, "MIPFocus", i);
    if (error) goto QUIT;
    error = GRBoptimize(model);
    if (error) goto QUIT;
    error = GRBgetdblattr(model, "MIPGap", &gap);
    if (error) goto QUIT;
    if (bestgap > gap)
    {
        GRBmodel *tmp = bestmodel;
        bestmodel = model;
        model = tmp;
        bestgap = gap;
    }
}

/* Finally, free the extra model, reset the time limit and 
   continue to solve the best model to optimality */
GRBfreemodel(model);
bestenv = GRBgetenv(bestmodel);
if (!bestenv) {
    printf("Cannot retrieve best model environment\n");
    exit(1);
}
error = GRBsetdblparam(bestenv, "TimeLimit", GRB_INFINITY);
if (error) goto QUIT;
error = GRBoptimize(bestmodel);
if (error) goto QUIT;
error = GRBgetintparam(bestenv, "MIPFocus", &mipfocus);
if (error) goto QUIT;

printf("Solved with MIPFocus: %i\n", mipfocus);

QUIT:

/* Error reporting */
if (error)
{
    printf("ERROR: %s\n", GRBgeterrormsg(env));
    exit(1);
}

/* Free best model */
GRBfreemodel(bestmodel);

/* Free environment */
GRBfreeenv(env);

return 0;

piecewise_c.c

/* Copyright 2020, Gurobi Optimization, LLC */

/* This example considers the following separable, convex problem:

    minimize   f(x) - y + g(z)
    subject to  x + 2 y + 3 z <= 4
                 x + y >= 1
                 x, y, z <= 1

    where f(u) = exp(-u) and g(u) = 2 u^2 - 4 u, for all real u. It
    formulates and solves a simpler LP model by approximating f and
    g with piecewise-linear functions. Then it transforms the model
    into a MIP by negating the approximation for f, which corresponds
    to a non-convex piecewise-linear function, and solves it again.

*/

#include <stdlib.h>
#include <stdio.h>
#include <math.h>
#include "gurobi_c.h"

double f(double u) { return exp(-u); }

double g(double u) { return 2 * u * u - 4 * u; }

int
main(int argc,
     char *argv[])
{
{  
  GRBenv *env = NULL;
  GRBmodel *model = NULL;
  int   error = 0;
  double lb, ub;
  int   npts, i;
  double *ptu = NULL;
  double *ptf = NULL;
  double *ptg = NULL;
  int   ind[3];
  double val[3];
  int   is mip;
  double objval;
  double sol[3];

  /* Create environment */
  error = GRBloadenv(&env, NULL);
  if (error) goto QUIT;

  /* Create a new model */
  error = GRBnewmodel(env, &model, NULL, 0, NULL, NULL, NULL, NULL, NULL);
  if (error) goto QUIT;

  /* Add variables */
  lb = 0.0; ub = 1.0;
  error = GRBaddvar(model, 0, NULL, NULL, 0.0, lb, ub, GRB_CONTINUOUS, "x");
  if (error) goto QUIT;
  error = GRBaddvar(model, 0, NULL, NULL, 0.0, lb, ub, GRB_CONTINUOUS, "y");
  if (error) goto QUIT;
  error = GRBaddvar(model, 0, NULL, NULL, 0.0, lb, ub, GRB_CONTINUOUS, "z");
  if (error) goto QUIT;

  /* Set objective for y */
  error = GRBsetdblattrelement(model, GRB_DBL_ATTR_OBJ, 1, -1.0);
  if (error) goto QUIT;

  /* Add piecewise-linear objective functions for x and z */
  npts = 101;
  ptu = (double *) malloc(npts * sizeof(double));
  ptf = (double *) malloc(npts * sizeof(double));
  ptg = (double *) malloc(npts * sizeof(double));
  for (i = 0; i < npts; i++) {
    ptu[i] = lb + (ub - lb) * i / (npts - 1);
    ptf[i] = f(ptu[i]);
    ptg[i] = g(ptu[i]);
  }
  error = GRBsetpwlobj(model, 0, npts, ptu, ptf);
  if (error) goto QUIT;

  QUIT:
  exit (1);
}
error = GRBsetpwlobj(model, 2, npts, ptu, ptg);
if (error) goto QUIT;

/* Add constraint: x + 2 y + 3 z <= 4 */
ind[0] = 0; ind[1] = 1; ind[2] = 2;
val[0] = 1; val[1] = 2; val[2] = 3;
error = GRBaddconstr(model, 3, ind, val, GRB_LESS_EQUAL, 4.0, "c0");
if (error) goto QUIT;

/* Add constraint: x + y >= 1 */
ind[0] = 0; ind[1] = 1;
val[0] = 1; val[1] = 1;
error = GRBaddconstr(model, 2, ind, val, GRB_GREATER_EQUAL, 1.0, "c1");
if (error) goto QUIT;

/* Optimize model as an LP */
error = GRBoptimize(model);
if (error) goto QUIT;

error = GRBgetintattr(model, "IsMIP", &ismip);
if (error) goto QUIT;
error = GRBgetdblattr(model, "ObjVal", &objval);
if (error) goto QUIT;
error = GRBgetdblattrarray(model, "X", 0, 3, sol);
if (error) goto QUIT;

printf(" IsMIP: %d\n", ismip);
printf("x %g\n\ny %g\n\nz %g\n", sol[0], sol[1], sol[2]);
printf("Obj: %g\n", objval);
printf("\n");

/* Negate piecewise-linear objective function for x */
for (i = 0; i < npts; i++) {
    ptf[i] = -ptf[i];
}

error = GRBsetpwlobj(model, 0, npts, ptu, ptf);
if (error) goto QUIT;

/* Optimize model as a MIP */
error = GRBoptimize(model);
if (error) goto QUIT;

error = GRBgetintattr(model, "IsMIP", &ismip);
if (error) goto QUIT;
error = GRBgetdblattr(model, "ObjVal", &objval);
if (error) goto QUIT;
error = GRBgetdblattrarray(model, "X", 0, 3, sol);
if (error) goto QUIT;
printf("IsMIP: %d\n", ismip);
printf("x %g\ny %g\nz %g\n", sol[0], sol[1], sol[2]);
printf("Obj: %g\n", objval);

QUIT:

    /* Error reporting */

    if (error) {
        printf("ERROR: %s\n", GRBgeterrormsg(env));
        exit(1);
    }

    /* Free data */

    free(ptu);
    free(ptf);
    free(ptg);

    /* Free model */

    GRBfreemodel(model);

    /* Free environment */

    GRBfreeenv(env);

    return 0;
}

poolsearch_c.c

/* Copyright 2020, Gurobi Optimization, LLC */

/* We find alternative epsilon-optimal solutions to a given knapsack problem by using PoolSearchMode */

#include <stdlib.h>
#include <stdio.h>
#include <math.h>
#include <string.h>
#include "gurobi_c.h"

#define MAXSTR 128

int main(void)
{
    GRBenv *env = NULL;
    GRBenv *menv = NULL;
    GRBmodel *model = NULL;
    int error = 0;
    char buffer[MAXSTR];
    int e, status, nSolutions, prlen;
    double objval, *cval = NULL;
int *cind = NULL;

/* Sample data */
const int groundSetSize = 10;
double objCoef[10] =
{32, 32, 15, 15, 6, 6, 1, 1, 1, 1};
double knapsackCoef[10] =
{16, 16, 8, 8, 4, 4, 2, 2, 1, 1};
double Budget = 33;

/* Create environment */
error = GRBloadenv(&env, "poolsearch_c.log");
if (error) goto QUIT;

/* Create initial model */
error = GRBnewmodel(env, &model, "poolsearch_c", groundSetSize, NULL,
NULL, NULL, NULL, NULL);
if (error) goto QUIT;

/* get model environment */
menv = GRBgetenv(model);
if (!menv) {
   fprintf(stderr, " Error: could not get model environment\n");
   goto QUIT;
}

/* set objective function */
error = GRBsetdblattrarray(model, "Obj", 0, groundSetSize, objCoef);
if (error) goto QUIT;

/* set variable types and names */
for (e = 0; e < groundSetSize; e++) {
   sprintf(buffer, "El%d", e);
   error = GRBsetcharattrelement(model, "VType", e, GRB_BINARY);
   if (error) goto QUIT;

   error = GRBsetstrattrelement(model, "VarName", e, buffer);
   if (error) goto QUIT;
}

/* Make space for constraint data */
cind = malloc(sizeof(int) * groundSetSize);
if (!cind) goto QUIT;
for (e = 0; e < groundSetSize; e++)
cind[e] = e;

/* Constraint: limit total number of elements to be picked to be at most * Budget */
sprintf(buffer, "Budget");
error = GRBaddconstr(model, groundSetSize, cind, knapsackCoef,
   GRB_LESS_EQUAL, Budget, buffer);
if (error) goto QUIT;

/* set global sense for ALL objectives */
error = GRBsetintattr(model, GRB_INT_ATTR_MODELSENSE, GRB_MAXIMIZE);
if (error) goto QUIT;
/* Limit how many solutions to collect */
error = GRBsetintparam(menv, GRB_INT_PAR_POOLSOLUTIONS, 1024);
if (error) goto QUIT;

/* Limit the search space by setting a gap for the worst possible solution that will be accepted */
error = GRBsetdblparam(menv, GRB_DBL_PAR_POOLGAP, 0.10);
if (error) goto QUIT;

/* do a systematic search for the k-best solutions */
error = GRBsetintparam(menv, GRB_INT_PAR_POOLSEARCHMODE, 2);
if (error) goto QUIT;

/* save problem */
error = GRBwrite(model, "poolsearch_c.lp");
if (error) goto QUIT;
error = GRBwrite(model, "poolsearch_c.mps");
if (error) goto QUIT;

/* Optimize */
error = GRBoptimize(model);
if (error) goto QUIT;

/* Status checking */
error = GRBgetintattr(model, "Status", &status);
if (error) goto QUIT;
if (status == GRB_INF_OR_UNBD ||
status == GRB_INFEASIBLE ||
status == GRB_UNBOUNDED)
{
printf("The model cannot be solved 
" "because it is infeasible or unbounded\n");
goto QUIT;
}
if (status != GRB_OPTIMAL) {
printf("Optimization was stopped with status %d\n", status);
goto QUIT;
}

/* make space for optimal solution */
cval = malloc(sizeof(double) * groundSetSize);
if (!cval) goto QUIT;

/* Print best selected set */
error = GRBgetdblattrarray(model, GRB_DBL_ATTR_X, 0, groundSetSize, cval);
if (error) goto QUIT;

printf("Selected elements in best solution:\n");
for (e = 0; e < groundSetSize; e++) {
if (cval[e] < .9) continue;
printf("El%d ", e);
}

/* print number of solutions stored */
error = GRBgetintattr(model, GRB_INT_ATTR_SOLCOUNT, &nSolutions);
if (error) goto QUIT;
printf("\nNumber of solutions found: %d\nValues: ", nSolutions);

/* print objective values of alternative solutions */
prlen = 0;
for (e = 0; e < nSolutions; e++) {
    error = GRBsetintparam(menv, GRB_INT_PAR_SOLUTIONNUMBER, e);
    if (error) goto QUIT;

    error = GRBgetdblattr(model, GRB_DBL_ATTR_POOLOBJVAL, &objval);
    if (error) goto QUIT;

    prlen += printf(" %g", objval);
    if (prlen >= 75 && e+1 < nSolutions) {
        prlen = printf("\n   ");
    }
}
printf("\n");

/* print fourth best set if available */
if (nSolutions >= 4) {
    error = GRBsetintparam(menv, GRB_INT_PAR_SOLUTIONNUMBER, 3);
    if (error) goto QUIT;

    /* get the solution vector */
    error = GRBgetdblattrarray(model, GRB_DBL_ATTR_XN, 0, groundSetSize, cval);
    if (error) goto QUIT;

    printf("Selected elements in fourth best solution:\n    ");
    for (e = 0; e < groundSetSize; e++) {
        if (cval[e] < .9) continue;
        printf("El%d ", e);
    }
    printf("\n");
}
QUIT:
    if (model != NULL) GRBfreemodel(model);
    if (env != NULL) GRBfreeenv(env);
    if (cind != NULL) free(cind);
    if (cval != NULL) free(cval);
    return error;
}

qcp_c.c

/* Copyright 2020, Gurobi Optimization, LLC */

/* This example formulates and solves the following simple QCP model:

maximize x
subject to x + y + z = 1
   x^2 + y^2 <= z^2 (second-order cone)
   x^2 <= yz (rotated second-order cone)
   x, y, z non-negative */
# include <stdlib.h>
# include <stdio.h>
# include "gurobi_c.h"

int main(int argc, char *argv[]) {
    GRBenv *env = NULL;
    GRBmodel *model = NULL;
    int error = 0;
    double sol[3];
    int ind[3];
    double val[3];
    double obj[] = {1, 0, 0};
    int qrow[3];
    int qcol[3];
    double qval[3];
    int optimstatus;
    double objval;

    /* Create environment */
    error = GRBloadenv(&env, "qcp.log");
    if (error) goto QUIT;

    /* Create an empty model */
    error = GRBnewmodel(env, &model, "qcp", 0, NULL, NULL, NULL, NULL, NULL);
    if (error) goto QUIT;

    /* Add variables */
    error = GRBaddvars(model, 3, 0, NULL, NULL, NULL, obj, NULL, NULL, NULL);
    if (error) goto QUIT;

    /* Change sense to maximization */
    error = GRBsetintattr(model, GRB_INT_ATTR_MODELSENSE, GRB_MAXIMIZE);
    if (error) goto QUIT;

    /* Linear constraint: x + y + z = 1 */
    ind[0] = 0; ind[1] = 1; ind[2] = 2;
    val[0] = 1; val[1] = 1; val[2] = 1;
    error = GRBaddconstr(model, 3, ind, val, GRB_EQUAL, 1.0, "c0");
    if (error) goto QUIT;

    /* Cone: x^2 + y^2 <= z^2 */
    qrow[0] = 0; qcol[0] = 0; qval[0] = 1.0;

    /* Other code goes here */
}

QUIT:
    printf("QUIT\n");

"
error = GRBaddqconstr(model, 0, NULL, NULL, 3, qrow, qcol, qval,
GRB_LESS_EQUAL, 0.0, "qc0");
if (error) goto QUIT;

/* Rotated cone: x^2 <= yz */
qrow[0] = 0; qcol[0] = 0; qval[0] = 1.0;
qrow[1] = 1; qcol[1] = 2; qval[1] = -1.0;
error = GRBaddqconstr(model, 0, NULL, NULL, 2, qrow, qcol, qval,
GRB_LESS_EQUAL, 0.0, "qc1");
if (error) goto QUIT;

/* Optimize model */
error = GRBoptimize(model);
if (error) goto QUIT;

/* Write model to 'qcp.lp' */
error = GRBwrite(model, "qcp.lp");
if (error) goto QUIT;

/* Capture solution information */
error = GRBgetintattr(model, GRB_INT_ATTR_STATUS, &optimstatus);
if (error) goto QUIT;
error = GRBgetdblattr(model, GRB_DBL_ATTR_OBJVAL, &objval);
if (error) goto QUIT;
error = GRBgetdblattrarray(model, GRB_DBL_ATTR_X, 0, 3, sol);
if (error) goto QUIT;

printf("\nOptimization complete\n");
if (optimstatus == GRB_OPTIMAL) {
    printf("Optimal objective: %.4e\n", objval);
    printf(" x=%.2f, y=%.2f, z=%.2f\n", sol[0], sol[1], sol[2]);
} else if (optimstatus == GRB_INF_OR_UNBD) {
    printf("Model is infeasible or unbounded\n");
} else {
    printf("Optimization was stopped early\n");
}

QUIT:

/* Error reporting */
if (error) {
    printf("ERROR: %s\n", GRBgeterrormsg(env));
    exit(1);
}

/* Free model */
GRBfreemodel(model);

/* Free environment */

GRBfreeenv(env);

return 0;
}

qp_c.c

/* Copyright 2020, Gurobi Optimization, LLC */

/* This example formulates and solves the following simple QP model:

minimize x^2 + x*y + y^2 + y*z + z^2 + 2 x
subject to x + 2 y + 3 z >= 4
       x + y       >= 1
       x, y, z non-negative

It solves it once as a continuous model, and once as an integer model. */

#include <stdlib.h>
#include <stdio.h>
#include "gurobi_c.h"

int main(int argc, char *argv[])
{
    GRBenv *env = NULL;
    GRBmodel *model = NULL;
    int error = 0;
    double sol[3];
    int ind[3];
    double val[3];
    int qrow[5];
    int qcol[5];
    double qval[5];
    char vtype[3];
    int optimstatus;
    double objval;

    /* Create environment */

    error = GRBloadenv(&env, "qp.log");
    if (error) goto QUIT;

    /* Create an empty model */

    error = GRBnewmodel(env, &model, "qp", 0, NULL, NULL, NULL, NULL, NULL);
    if (error) goto QUIT;

    /* Add variables */
error = GRBaddvars(model, 3, 0, NULL, NULL, NULL, NULL, NULL, NULL, NULL);
if (error) goto QUIT;

/* Quadratic objective terms */
error = GRBaddqpterms(model, 5, qrow, qcol, qval);
if (error) goto QUIT;

/* Linear objective term */
error = GRBsetdblattrelement(model, GRB_DBL_ATTR_OBJ, 0, 2.0);
if (error) goto QUIT;

/* First constraint: \(x + 2y + 3z \leq 4\) */
ind[0] = 0; ind[1] = 1; ind[2] = 2;
val[0] = 1; val[1] = 2; val[2] = 3;
error = GRBaddconstr(model, 3, ind, val, GRB_GREATER_EQUAL, 4.0, "c0");
if (error) goto QUIT;

/* Second constraint: \(x + y \geq 1\) */
ind[0] = 0; ind[1] = 1;
val[0] = 1; val[1] = 1;
error = GRBaddconstr(model, 2, ind, val, GRB_GREATER_EQUAL, 1.0, "c1");
if (error) goto QUIT;

/* Optimize model */
error = GRBoptimize(model);
if (error) goto QUIT;

/* Write model to 'qp.lp' */
error = GRBwrite(model, "qp.lp");
if (error) goto QUIT;

/* Capture solution information */
error = GRBgetintattr(model, GRB_INT_ATTR_STATUS, &optimstatus);
if (error) goto QUIT;
error = GRBgetdblattr(model, GRB_DBL_ATTR_OBJVAL, &objval);
if (error) goto QUIT;
error = GRBgetdblattarray(model, GRB_DBL_ATTR_X, 0, 3, sol);
if (error) goto QUIT;
printf("Optimization complete\n");
if (optimstatus == GRB_OPTIMAL) {
    printf("Optimal objective: %.4e\n", objval);

    printf(" x=%.4f, y=%.4f, z=%.4f\n", sol[0], sol[1], sol[2]);
} else if (optimstatus == GRB_INF_OR_UNBD) {
    printf("Model is infeasible or unbounded\n");
} else {
    printf("Optimization was stopped early\n");
}

QUIT:
/* Modify variable types */

error = GRBsetcharattrarray(model, GRB_CHAR_ATTR_VTYPE, 0, 3, vtype);
if (error) goto QUIT;

/* Optimize model */
error = GRBoptimize(model);
if (error) goto QUIT;

/* Write model to 'qp2.lp' */
error = GRBwrite(model, "qp2.lp");
if (error) goto QUIT;

/* Capture solution information */
error = GRBgetintattr(model, GRB_INT_ATTR_STATUS, &optimstatus);
if (error) goto QUIT;

error = GRBgetdblattr(model, GRB_DBL_ATTR_OBJVAL, &objval);
if (error) goto QUIT;

error = GRBgetdblattrarray(model, GRB_DBL_ATTR_X, 0, 3, sol);
if (error) goto QUIT;

printf("Optimization complete\n");
if (optimstatus == GRB_OPTIMAL) {
    printf("Optimal objective: %.4e\n", objval);

    printf(" x=%.4f, y=%.4f, z=%.4f\n", sol[0], sol[1], sol[2]);
} else if (optimstatus == GRB_INF_OR_UNBD) {
    printf("Model is infeasible or unbounded\n");
} else {
    printf("Optimization was stopped early\n");
}

QUIT:
/* Error reporting */
if (error) {

}
sensitivity_c.c

/* Copyright 2020, Gurobi Optimization, LLC */

/* A simple sensitivity analysis example which reads a MIP model from a
 * file and solves it. Then uses the scenario feature to analyze the impact
 * w.r.t. the objective function of each binary variable if it is set to
 * 1-X, where X is its value in the optimal solution.
 *
 * Usage:
 * sensitivity_c <model filename>
 */

#include <stdlib.h>
#include <stdio.h>
#include "gurobi_c.h"

#define MAXSCENARIOS 100

int main(int argc, char *argv[]){
   GRBenv *env = NULL;
   GRBenv *modelenv = NULL;
   GRBmodel *model = NULL;
   double *origx = NULL;
   double origObjVal;
   int ismip, status, numvars, i;
   int scenarios;
   int error = 0;

   if (argc < 2) {
      fprintf(stderr, "Usage: sensitivity_c filename\n");
      goto QUIT;
   }

   /* Create environment */
error = GRBloadenv(&env, "sensitivity.log");
if (error) goto QUIT;

/* Read model */
error = GRBreadmodel(env, argv[1], &model);
if (error) goto QUIT;

modelenv = GRBgetenv(model);
if (error) goto QUIT;

error = GRBgetintattr(model, GRB_INT_ATTR_IS_MIP, &ismip);
if (error) goto QUIT;
if (ismip == 0) {
    printf("Model is not a MIP\n");
    goto QUIT;
}

/* Solve model */
error = GRBoptimize(model);
if (error) goto QUIT;

error = GRBgetintattr(model, GRB_INT_ATTR_STATUS, &status);
if (error) goto QUIT;
if (status != GRB_OPTIMAL) {
    printf("Optimization ended with status %d\n", status);
    goto QUIT;
}

/* Store the optimal solution */
error = GRBgetdblattr(model, GRB_DBL_ATTR_OBJVAL, &origObjVal);
if (error) goto QUIT;
error = GRBgetintattr(model, GRB_INT_ATTR_NUMVARS, &numvars);
if (error) goto QUIT;
origx = (double *) malloc(numvars * sizeof(double));
if (origx == NULL) {
    printf("Out of memory\n");
    goto QUIT;
}
error = GRBgetdblattrarray(model, GRB_DBL_ATTR_X, 0, numvars, origx);
if (error) goto QUIT;

scenarios = 0;

/* Count number of unfixed, binary variables in model. For each we create a
 * scenario. */
for (i = 0; i < numvars; i++) {
    double lb, ub;
    char vtype;

    error = GRBgetdblattrelement(model, GRB_DBL_ATTR LB, i, &lb);
    if (error) goto QUIT;
    error = GRBgetdblattrelement(model, GRB_DBL_ATTR UB, i, &ub);
    if (error) goto QUIT;
    error = GRBgetcharattrelement(model, GRB_CHAR_ATTR VTYPE, i, &vtype);
    if (error) goto QUIT;
if (lb == 0.0 && ub == 1.0 &&
    (vtype == GRB_BINARY || vtype == GRB_INTEGER)) {
    scenarios ++;

    if (scenarios >= MAXSCENARIOS)
        break;
}

printf("### construct multi-scenario model with %d scenarios\n", scenarios);

/* Set the number of scenarios in the model */
error = GRBsetintattr(model, GRB_INT_ATTR_NUMSCENARIOS, scenarios);
if (error) goto QUIT;

scenarios = 0;

/* Create a (single) scenario model by iterating through unfixed binary
* variables in the model and create for each of these variables a
* scenario by fixing the variable to 1-X, where X is its value in the
* computed optimal solution
*/
for (i = 0; i < numvars; i++) {
    double lb, ub;
    char vtype;

    error = GRBgetdblattrelement(model, GRB_DBL_ATTR_LB, i, &lb);
    if (error) goto QUIT;
    error = GRBgetdblattrelement(model, GRB_DBL_ATTR_UB, i, &ub);
    if (error) goto QUIT;
    error = GRBgetcharattrelement(model, GRB_CHAR_ATTR_VTYPE, i, &vtype);
    if (error) goto QUIT;

    if (lb == 0.0 && ub == 1.0 &&
        (vtype == GRB_BINARY || vtype == GRB_INTEGER) &&
        scenarios < MAXSCENARIOS)
    {
        /* Set ScenarioNumber parameter to select the corresponding scenario
         * for adjustments
         */
        error = GRBsetintparam(modelenv, GRB_INT_PAR_SCENARIONUMBER, scenarios);
        if (error) goto QUIT;

        /* Set variable to 1-X, where X is its value in the optimal solution */
        if (origx[i] < 0.5) {
            error = GRBsetdblattrelement(model, GRB_DBL_ATTR_SCENNLB, i, 1.0);
            if (error) goto QUIT;
        } else {
            error = GRBsetdblattrelement(model, GRB_DBL_ATTR_SCENNUB, i, 0.0);
            if (error) goto QUIT;
        }

        scenarios ++;
    } else { 

/* Add MIP start for all other variables using the optimal solution 
 * of the base model */
error = GRBsetdblattrelement(model, GRB_DBL_ATTR_START, i, origx[i]);
if (error) goto QUIT;
}

/* Solve multi-scenario model */
error = GRBoptimize(model);
if (error) goto QUIT;

/* collect the status */
error = GRBgetintattr(model, GRB_INT_ATTR_STATUS, &status);
if (error) goto QUIT;

/* In case we solved the scenario model to optimality capture the 
 * sensitivity information */
if (status == GRB_OPTIMAL) {
  int modelSense;

  scenarios = 0;

  /* Get model sense (minimization or maximization) */
  error = GRBgetintattr(model, GRB_INT_ATTR_MODELSENSE, &modelSense);
  if (error) goto QUIT;

  for (i = 0; i < numvars; i++) {
    double lb, ub;
    char vtype;

    error = GRBgetdblattrelement(model, GRB_DBL_ATTR_LB, i, &lb);
    if (error) goto QUIT;
    error = GRBgetdblattrelement(model, GRB_DBL_ATTR_UB, i, &ub);
    if (error) goto QUIT;
    error = GRBgetcharattrelement(model, GRB_CHAR_ATTR_VTYPE, i, &vtype);
    if (error) goto QUIT;

    if (lb == 0.0 && ub == 1.0 && (vtype == GRB_BINARY || vtype == GRB_INTEGER)) {
      double scenarioObjVal;
      double scenarioObjBound;
      char *varName;

      /* Set scenario parameter to collect the objective value of the 
       * corresponding scenario */
      error = GRBsetintparam(modelenv, GRB_INT_PAR_SCENARIONUMBER, scenarios);
      if (error) goto QUIT;

      /* Collect objective value and bound for the scenario */
      error = GRBgetdblattrelement(model, GRB_DBL_ATTR_SCENNOBJVAL, &scenarioObjVal);
      if (error) goto QUIT;
      error = GRBgetdblattrelement(model, GRB_DBL_ATTR_SCENNOBJBOUND, &scenarioObjBound);
  }
}
if (error) goto QUIT;

error = GRBgetstrattrelement(model, GRB_STR_ATTR_VARNAME, i, &varName);
if (error) goto QUIT;

/* Check if we found a feasible solution for this scenario */
if (scenarioObjVal >= modelSense * GRB_INFINITY) {
    /* Check if the scenario is infeasible */
    if (scenarioObjBound >= modelSense * GRB_INFINITY)
        printf("Objective sensitivity for variable %s is infeasible\n", varName);
    else
        printf("Objective sensitivity for variable %s is unknown (no solution available)\n", varName);
} else {
    /* Scenario is feasible and a solution is available */
    printf("Objective sensitivity for variable %s is %g\n", varName, modelSense * (scenarioObjVal - origObjVal));
}

scenarios++;

if (scenarios >= MAXSCENARIOS)
    break;
}
}

QUIT:

/* Error reporting */
if (error != 0) {
    printf("ERROR: %s\n", GRBgeterrormsg(env));
    exit(1);
}

/* Free data */
free(origx);

/* Free model */
GRBfreemodel(model);

/* Free environment */
GRBfreeenv(env);

return 0;
}

SOS_C.C

/* Copyright 2020, Gurobi Optimization, LLC */

/* This example creates a very simple Special Ordered Set (SOS) model. 
The model consists of 3 continuous variables, no linear constraints, 
and a pair of SOS constraints of type 1. */
#include <stdlib.h>
#include <stdio.h>
#include "gurobi_c.h"

int
main(int argc,
     char *argv[])
{
    GRBenv *env = NULL;
    GRBmodel *model = NULL;
    int error = 0;
    double x[3];
    double obj[3];
    double ub[3];
    int sostype[2];
    int sosbeg[2];
    int sosind[4];
    double soswt[4];
    int optimstatus;
    double objval;

    /* Create environment */
    error = GRBloadenv(&env, "sos.log");
    if (error) goto QUIT;

    /* Create an empty model */
    error = GRBnewmodel(env, &model, "sos", 0, NULL, NULL, NULL, NULL, NULL);
    if (error) goto QUIT;

    /* Add variables */
    obj[0] = -2; obj[1] = -1; obj[2] = -1;
    ub[0] = 1.0; ub[1] = 1.0; ub[2] = 2.0;
    error = GRBaddvars(model, 3, 0, NULL, NULL, NULL, obj, NULL, ub, NULL, NULL);
    if (error) goto QUIT;

    /* Build first SOS1: x0=0 or x1=0 */
    sosind[0] = 0; sosind[1] = 1;
    soswt[0] = 1.0; soswt[1] = 2.0;
    sosbeg[0] = 0; sostype[0] = GRB_SOS_TYPE1;

    /* Build second SOS1: x0=0 or x2=0 */
    sosbeg[1] = 2; sostype[1] = GRB_SOS_TYPE1;

    /* Add SOSs to model */
    error = GRBaddsos(model, 2, 4, sostype, sosbeg, sosind, soswt);
    if (error) goto QUIT;

QUIT:

    return 0;
}
/ * Optimize model */

error = GRBoptimize(model);
if (error) goto QUIT;

/* Write model to 'sos.lp' */

error = GRBwrite(model, "sos.lp");
if (error) goto QUIT;

/* Capture solution information */

error = GRBgetintattr(model, GRB_INT_ATTR_STATUS, &optimstatus);
if (error) goto QUIT;

error = GRBgetdblattr(model, GRB_DBL_ATTR_OBJVAL, &objval);
if (error) goto QUIT;

error = GRBgetdblattrarray(model, GRB_DBL_ATTR_X, 0, 3, x);
if (error) goto QUIT;

printf("\nOptimization complete\n");
if (optimstatus == GRB_OPTIMAL) {
    printf("Optimal objective: %.4e\n", objval);
    printf(" x=%.4f, y=%.4f, z=%.4f\n", x[0], x[1], x[2]);
} else if (optimstatus == GRB_INF_OR_UNBD) {
    printf("Model is infeasible or unbounded\n");
} else {
    printf("Optimization was stopped early\n");
}

QUIT:

/* Error reporting */

if (error) {
    printf("ERROR: %s\n", GRBgeterrormsg(env));
    exit(1);
}

/* Free model */

GRBfreemodel(model);

/* Free environment */

GRBfreeenv(env);

return 0;

sudoku_c.c

/* Copyright 2020, Gurobi Optimization, LLC */
Sudoku example.

The Sudoku board is a 9x9 grid, which is further divided into a 3x3 grid of 3x3 grids. Each cell in the grid must take a value from 0 to 9. No two grid cells in the same row, column, or 3x3 subgrid may take the same value.

In the MIP formulation, binary variables $x[i,j,v]$ indicate whether cell $<i,j>$ takes value $'v'$. The constraints are as follows:

1. Each cell must take exactly one value ($\sum_v x[i,j,v] = 1$)
2. Each value is used exactly once per row ($\sum_i x[i,j,v] = 1$)
3. Each value is used exactly once per column ($\sum_j x[i,j,v] = 1$)
4. Each value is used exactly once per 3x3 subgrid ($\sum_{grid} x[i,j,v] = 1$)

Input datasets for this example can be found in examples/data/sudoku*.

```c
#include <stdlib.h>
#include <stdio.h>
#include <string.h>
#include "gurobi_c.h"

#define SUBDIM 3
#define DIM (SUBDIM*SUBDIM)

int main(int argc, char *argv[])
{
    FILE *fp = NULL;
    GRBenv *env = NULL;
    GRBmodel *model = NULL;
    int board[DIM][DIM];
    char inputline[100];
    int ind[DIM];
    double val[DIM];
    double lb[DIM*DIM*DIM];
    char vtype[DIM*DIM*DIM];
    char *names[DIM*DIM*DIM];
    char *namestorage[10*DIM*DIM*DIM];
    char *cursor;
    int optimstatus;
    double objval;
    int i, j, v, ig, jg, count;
    int error = 0;

    if (argc < 2) {
        fprintf(stderr, "Usage: sudoku_c datafile\n");
        exit(1);
    }

    fp = fopen(argv[1], "r");
    if (fp == NULL) {
        fprintf(stderr, "Error: unable to open input file %s\n", argv[1]);
        exit(1);
    }

    /*
     * The function reads the Sudoku board from the input file and
     * constructs a Gurobi model to solve it.
     */
    ...
for (i = 0; i < DIM; i++) {
    fgets(inputline, 100, fp);
    if (strlen(inputline) < 9) {
        fprintf(stderr, "Error: not enough board positions specified\n");
        exit(1);
    }
    for (j = 0; j < DIM; j++) {
        board[i][j] = (int) inputline[j] - (int)'1';
        if (board[i][j] < 0 || board[i][j] >= DIM)
            board[i][j] = -1;
    }
}

/* Create an empty model */
cursor = namestorage;
for (i = 0; i < DIM; i++) {
    for (j = 0; j < DIM; j++) {
        for (v = 0; v < DIM; v++) {
            if (board[i][j] == v)
                lb[i*DIM*DIM+j*DIM+v] = 1;
            else
                lb[i*DIM*DIM+j*DIM+v] = 0;
            vtype[i*DIM*DIM+j*DIM+v] = GRB_BINARY;
            names[i*DIM*DIM+j*DIM+v] = cursor;
            sprintf(names[i*DIM*DIM+j*DIM+v], "x[%d,%d,%d]", i, j, v+1);
            cursor += strlen(names[i*DIM*DIM+j*DIM+v]) + 1;
        }
    }
}

/* Create environment */
error = GRBloadenv(&env, "sudoku.log");
if (error) goto QUIT;

/* Create new model */
error = GRBnewmodel(env, &model, "sudoku", DIM*DIM*DIM, NULL, lb, NULL,
                    vtype, names);
if (error) goto QUIT;

/* Each cell gets a value */
for (i = 0; i < DIM; i++) {
    for (j = 0; j < DIM; j++) {
        for (v = 0; v < DIM; v++) {
            ind[v] = i*DIM*DIM + j*DIM + v;
            val[v] = 1.0;
        }
        error = GRBaddconstr(model, DIM, ind, val, GRB_EQUAL, 1.0, NULL);
        if (error) goto QUIT;
    }
}
/* Each value must appear once in each row */
for (v = 0; v < DIM; v++) {
    for (j = 0; j < DIM; j++) {
        for (i = 0; i < DIM; i++) {
            ind[i] = i*DIM*DIM + j*DIM + v;
            val[i] = 1.0;
        }
        error = GRBaddconstr(model, DIM, ind, val, GRB_EQUAL, 1.0, NULL);
        if (error) goto QUIT;
    }
}

/* Each value must appear once in each column */
for (v = 0; v < DIM; v++) {
    for (i = 0; i < DIM; i++) {
        for (j = 0; j < DIM; j++) {
            ind[j] = i*DIM*DIM + j*DIM + v;
            val[j] = 1.0;
        }
        error = GRBaddconstr(model, DIM, ind, val, GRB_EQUAL, 1.0, NULL);
        if (error) goto QUIT;
    }
}

/* Each value must appear once in each subgrid */
for (v = 0; v < DIM; v++) {
    for (ig = 0; ig < SUBDIM; ig++) {
        for (jg = 0; jg < SUBDIM; jg++) {
            count = 0;
            for (i = ig*SUBDIM; i < (ig+1)*SUBDIM; i++) {
                for (j = jg*SUBDIM; j < (jg+1)*SUBDIM; j++) {
                    ind[count] = i*DIM*DIM + j*DIM + v;
                    val[count] = 1.0;
                    count++;
                }
            }
            error = GRBaddconstr(model, DIM, ind, val, GRB_EQUAL, 1.0, NULL);
            if (error) goto QUIT;
        }
    }
}

/* Optimize model */
error = GRBoptimize(model);
if (error) goto QUIT;
/* Write model to 'sudoku.lp' */

error = GRBwrite(model, "sudoku.lp");
if (error) goto QUIT;

/* Capture solution information */

error = GRBgetintattr(model, GRB_INT_ATTR_STATUS, &optimstatus);
if (error) goto QUIT;

error = GRBgetdblattr(model, GRB_DBL_ATTR_OBJVAL, &objval);
if (error) goto QUIT;

printf("Optimization complete\n");
if (optimstatus == GRB_OPTIMAL)
    printf("Optimal objective: %.4e\n", objval);
else if (optimstatus == GRB_INF_OR_UNBD)
    printf("Model is infeasible or unbounded\n");
else
    printf("Optimization was stopped early\n");
printf("\n");

QUIT:

/* Error reporting */

if (error) {
    printf("ERROR: %s\n", GRBgeterrormsg(env));
    exit(1);
}

fclose(fp);

/* Free model */

GRBfreemodel(model);

/* Free environment */

GRBfreeenv(env);

return 0;
}

tsp_c.c

/* Copyright 2020, Gurobi Optimization, LLC */

/*
Solve a traveling salesman problem on a randomly generated set of points using lazy constraints. The base MIP model only includes 'degree-2' constraints, requiring each node to have exactly two incident edges. Solutions to this model may contain subtours - tours that don't visit every node. The lazy constraint callback adds new constraints to cut them off.
*/
/* Define structure to pass data to the callback function */

struct callback_data {
    int n;
};

/* Given an integer-feasible solution 'sol', find the smallest sub-tour. Result is returned in 'tour', and length is returned in 'tourlenP'. */

static void findsubtour(int n, double *sol, int *tourlenP, int *tour) {
    int i, node, len, start;
    int bestind, bestlen;
    int *seen = NULL;

    seen = (int*)malloc(n*sizeof(int));
    if (seen == NULL) {
        fprintf(stderr, "Out of memory\n");
        exit(1);
    }

    for (i = 0; i < n; i++)
        seen[i] = 0;
    start = 0;
    bestlen = n+1;
    bestind = -1;

    while (start < n) {
        for (node = 0; node < n; node++)
            if (seen[node] == 0)
                break;
        if (node == n)
            break;

        for (len = 0; len < n; len++) {
            tour[start+len] = node;
            seen[node] = 1;
            for (i = 0; i < n; i++) {
                if (sol[node*n+i] > 0.5 && !seen[i]) {
                    node = i;
                    break;
                }
            }
            if (i == n) {

```
len++;  
if (len < bestlen) {
    bestlen = len;
    bestind = start;
}  
start += len;
break;
}
}

for (i = 0; i < bestlen; i++)
    tour[i] = tour[bestind+i];
*tourlenP = bestlen;
free(seen);
}

/* Subtour elimination callback. Whenever a feasible solution is found,
find the shortest subtour, and add a subtour elimination constraint
if that tour doesn't visit every node. */

int __stdcall subtourelim(GRBmodel *model,
void *cbdata,
int where,
void *usrdata)
{
    struct callback_data *mydata = (struct callback_data *) usrdata;
    int n = mydata->n;
    int *tour = NULL;
    double *sol = NULL;
    int i, j, len, nz;
    int error = 0;

    if (where == GRB_CB_MIPSOL) {
        sol = (double *) malloc(n*n*sizeof(double));
        tour = (int *) malloc(n*sizeof(int));
        if (sol == NULL || tour == NULL) {
            fprintf(stderr, "Out of memory\n");
            exit(1);
        }
        GRBcbget(cbdata, where, GRB_CB_MIPSOL_SOL, sol);

        findsubtour(n, sol, &len, tour);

        if (len < n) {
            int *ind = NULL;
            double *val = NULL;

            ind = (int *) malloc(len*(len-1)/2*sizeof(int));
            val = (double *) malloc(len*(len-1)/2*sizeof(double));

            if (ind == NULL || val == NULL) {

            }
fprintf(stderr, "Out of memory\n");
exit(1);
}

/* Add subtour elimination constraint */

nz = 0;
for (i = 0; i < len; i++)
    for (j = i+1; j < len; j++)
        ind[nz++] = tour[i]*n+tour[j];
for (i = 0; i < nz; i++)
    val[i] = 1.0;

error = GRBcblazy(cbdata, nz, ind, val, GRB_LESS_EQUAL, len-1);

free(ind);
free(val);
}

free(sol);
free(tour);
}

return error;
}

/* Euclidean distance between points 'i' and 'j'. */

static double
distance(double *x,
        double *y,
        int i,
        int j)
{
    double dx = x[i] - x[j];
    double dy = y[i] - y[j];

    return sqrt(dx*dx + dy*dy);
}

int
main(int argc,
     char *argv[])
{
    GRBenv  *env = NULL;
    GRBmodel *model = NULL;
    int i, j, len, n, solcount;
    int error = 0;
    char name[100];
    double *x = NULL;
    double *y = NULL;
    int *ind = NULL;
    double *val = NULL;
    struct callback_data mydata;

    if (argc < 2) {
fprintf(stderr, "Usage: tsp_c size\n");
exit(1);
}

n = atoi(argv[1]);
if (n == 0) {
    fprintf(stderr, "Argument must be a positive integer.\n");
} else if (n > 100) {
    printf("It will be a challenge to solve a TSP this large.\n");
}
x = (double *) malloc(n*sizeof(double));
y = (double *) malloc(n*sizeof(double));
ind = (int *) malloc(n*sizeof(int));
val = (double *) malloc(n*sizeof(double));

if (x == NULL || y == NULL || ind == NULL || val == NULL) {
    fprintf(stderr, "Out of memory\n");
    exit(1);
}

/* Create random points */
for (i = 0; i < n; i++) {
    x[i] = ((double) rand())/RAND_MAX;
    y[i] = ((double) rand())/RAND_MAX;
}

/* Create environment */
error = GRBloadenv(&env, "tsp.log");
if (error) goto QUIT;

/* Create an empty model */
error = GRBnewmodel(env, &model, "tsp", 0, NULL, NULL, NULL, NULL, NULL);
if (error) goto QUIT;

/* Add variables - one for every pair of nodes */
/* Note: If edge from i to j is chosen, then x[i*n+j] = x[j*n+i] = 1. */
/* The cost is split between the two variables. */
for (i = 0; i < n; i++) {
    for (j = 0; j < n; j++) {
        sprintf(name, "x_%d_%d", i, j);
        error = GRBaddvar(model, 0, NULL, NULL, NULL, distance(x, y, i, j)/2,
                          0.0, 1.0, GRB_BINARY, name);
        if (error) goto QUIT;
    }
}

/* Degree-2 constraints */
for (i = 0; i < n; i++) {
    for (j = 0; j < n; j++) {
\[ \text{ind}[j] = i \times n + j; \]
\[ \text{val}[j] = 1.0; \]
\[
\text{sprintf}(\text{name}, \text{"deg2_\%d"}, i); \]
\[
\text{error} = \text{GRBaddconstr}(\text{model}, n, \text{ind}, \text{val}, \text{GRB_EQUAL}, 2, \text{name});\]
\[
\text{if} (\text{error}) \text{ goto QUIT; } \]
\[
/* \text{Forbid edge from node back to itself */} \]
\[
\text{for} (i = 0; i < n; i++) { \}
\[
\quad \text{error} = \text{GRBsetdblattrelement}(\text{model}, \text{GRB_DBL_ATTR_UB}, i \times n + i, 0); \]
\[
\quad \text{if} (\text{error}) \text{ goto QUIT; } \}
\[
/* \text{Symmetric TSP */} \]
\[
\text{for} (i = 0; i < n; i++) { \}
\[
\quad \text{for} (j = 0; j < i; j++) { \}
\[
\quad\quad \text{ind}[0] = i \times n + j; \]
\[
\quad\quad \text{ind}[1] = i + j \times n; \]
\[
\quad\quad \text{val}[0] = 1; \]
\[
\quad\quad \text{val}[1] = -1; \]
\[
\quad\quad \text{error} = \text{GRBaddconstr}(\text{model}, 2, \text{ind}, \text{val}, \text{GRB_EQUAL}, 0, \text{NULL}); \]
\[
\quad\quad \text{if} (\text{error}) \text{ goto QUIT; } \}
\]
\[
/* \text{Set callback function */} \]
\[
\text{mydata.n} = n; \]
\[
\text{error} = \text{GRBsetcallbackfunc}(\text{model}, \text{subtourelim}, (\text{void}*) &\text{mydata}); \]
\[
\text{if} (\text{error}) \text{ goto QUIT; } \]
\[
/* \text{Must set LazyConstraints parameter when using lazy constraints */} \]
\[
\text{error} = \text{GRBsetintparam}(\text{GRBgetenv(model)}, \text{GRB_INT_PAR_LAZYCONSTRAINTS}, 1); \]
\[
\text{if} (\text{error}) \text{ goto QUIT; } \]
\[
/* \text{Optimize model */} \]
\[
\text{error} = \text{GRBoptimize}(\text{model}); \]
\[
\text{if} (\text{error}) \text{ goto QUIT; } \]
\[
/* \text{Extract solution */} \]
\[
\text{error} = \text{GRBgetintattr}(\text{model}, \text{GRB_INT_ATTR_SOLCOUNT}, &\text{solcount}); \]
\[
\text{if} (\text{error}) \text{ goto QUIT; } \]
\[
\text{if} (\text{solcount} > 0) { \}
\[
\quad \text{int} *\text{tour} = \text{NULL}; \]
\[
\quad \text{double} *\text{sol} = \text{NULL}; \]
sol = (double *) malloc(n*n*sizeof(double));
tour = (int *) malloc(n*sizeof(int));
if (sol == NULL || tour == NULL) {
    fprintf(stderr, "Out of memory\n");
    exit(1);
}

error = GRBgetdblattrarray(model, GRB_DBL_ATTR_X, 0, n*n, sol);
if (error) goto QUIT;

/* Print tour */

findsubtour(n, sol, &len, tour);

printf("Tour: ");
for (i = 0; i < len; i++)
    printf("%d ", tour[i]);
printf("\n");
free(tour);
free(sol);

QUIT:

/* Free data */

free(x);
free(y);
free(ind);
free(val);

/* Error reporting */

if (error) {
    printf("ERROR: %s\n", GRBgeterrormsg(env));
    exit(1);
}

/* Free model */

GRBfreemodel(model);

/* Free environment */

GRBfreeenv(env);

return 0;

---

*tune_c.c*

/* Copyright 2020, Gurobi Optimization, LLC */

/* This example reads a model from a file and tunes it. It then writes the best parameter settings to a file */
and solves the model using these parameters. */

#include <stdlib.h>
#include <stdio.h>
#include <math.h>
#include "gurobi_c.h"

int main(int argc, char *argv[]) {
  GRBenv *env = NULL;
  GRBmodel *model = NULL;
  int tuneresultcount;
  int error = 0;

  if (argc < 2) {
    fprintf(stderr, "Usage: tune_c filename\n");
    exit(1);
  }

  /* Create environment */
  error = GRBloadenv(&env, "tune_c.log");
  if (error) goto QUIT;

  /* Read model from file */
  error = GRBreadmodel(env, argv[1], &model);
  if (error) goto QUIT;

  /* Set the TuneResults parameter to 1 */
  error = GRBsetintparam(GRBgetenv(model), GRB_INT_PAR_TUNERESULTS, 1);
  if (error) goto QUIT;

  /* Tune the model */
  error = GRBtunemodel(model);
  if (error) goto QUIT;

  /* Get the number of tuning results */
  error = GRBgetintattr(model, GRB_INT_ATTR_TUNE_RESULTCOUNT, &tuneresultcount);
  if (error) goto QUIT;

  if (tuneresultcount > 0) {
    /* Load the best tuned parameters into the model's environment */
    error = GRBgettuneresult(model, 0);
    if (error) goto QUIT;

    /* Write tuned parameters to a file */
    error = GRBwrite(model, "tune.prm");
  }
QUIT: /* cleanup */
  if (env) GRBroamenv(env);
  if (model) GRBfree(model);
  return error;
}
if (error) goto QUIT;

/* Solve the model using the tuned parameters */

error = GRBoptimize(model);
if (error) goto QUIT;
}

QUIT:

/* Error reporting */

if (error) {
    printf("ERROR: %s\n", GRBgeterrormsg(env));
    exit(1);
}

/* Free model */

GRBfreemodel(model);

/* Free environment */

GRBfreeenv(env);

return 0;
}

workforce1_c.c

/* Copyright 2020, Gurobi Optimization, LLC */

/* Assign workers to shifts; each worker may or may not be available on a particular day. If the problem cannot be solved, use IIS to find a set of conflicting constraints. Note that there may be additional conflicts besides what is reported via IIS. */

#include <stdlib.h>
#include <stdio.h>
#include <math.h>
#include "gurobi_c.h"

#define xcol(w,s) nShifts*w+s
#define MAXSTR 128

int main(int argc, char *argv[])
{
    GRBenv *env  = NULL;
    GRBmodel *model = NULL;
    int error = 0, status;
    int s, w, col;
    int *cbeg = NULL;
int *cind = NULL;
int idx;
double *cval = NULL;
char *sense = NULL;
char vname[MAXSTR];
double obj;
int i, iis, numconstrs;
char *cname;

/* Sample data */
const int nShifts = 14;
const int nWorkers = 7;

/* Sets of days and workers */
char* Shifts[] =
char* Workers[] =

/* Number of workers required for each shift */
double shiftRequirements[] =
{ 3, 2, 4, 4, 5, 6, 5, 2, 2, 3, 4, 6, 7, 5 };

/* Amount each worker is paid to work one shift */
double pay[] = { 10, 12, 10, 8, 8, 9, 11 };

/* Worker availability: 0 if the worker is unavailable for a shift */
double availability[][14] =
{ { 0, 1, 1, 0, 1, 0, 1, 1, 1, 1, 1, 1, 1 },
  { 1, 1, 0, 0, 1, 0, 1, 0, 0, 1, 0, 1, 1 },
  { 0, 0, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1 },
  { 0, 1, 1, 0, 1, 1, 0, 1, 1, 1, 1, 1, 1 },
  { 1, 1, 1, 1, 0, 1, 1, 0, 1, 0, 1, 1, 1 },
  { 1, 1, 1, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1 },
  { 1, 1, 1, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1 },
  { 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1 } };

/* Create environment */
error = GRBloadenv(&env, "workforce1.log");
if (error) goto QUIT;

/* Create initial model */
error = GRBnewmodel(env, &model, "workforce1", nWorkers * nShifts,
  NULL, NULL, NULL, NULL, NULL);
if (error) goto QUIT;

/* Initialize assignment decision variables:
x[w][s] == 1 if worker w is assigned
to shift s. Since an assignment model always produces integer
solutions, we use continuous variables and solve as an LP. */
for (w = 0; w < nWorkers; ++w)
{
  for (s = 0; s < nShifts; ++s)
  {
    col = xcol(w, s);
  
    /* Constraints */
    for (iis = 0; iis < numconstrs; ++iis)
      addconstr(model, 1, NULL, NULL, 1, 0);
  
    /* Objective */
    addvar(model, 1, NULL, NULL, 1, 0, 0, 0);
  
}
The objective is to minimize the total pay costs
/* Make space for constraint data */
/* Constraint: assign exactly shiftRequirements[s] workers to each shift s */
/* Optimize */
/* The model cannot be solved because it is unbounded
The optimal objective is %f

goto QUIT;
}
if ((status != GRB_INF_OR_UNBD) && (status != GRB_INFEASIBLE))
{
    printf("Optimization was stopped with status %i\n", status);
    goto QUIT;
}

/* do IIS */
printf("The model is infeasible; computing IIS\n");
error = GRBcomputeIIS(model);
if (error) goto QUIT;
printf("\nThe following constraint(s) cannot be satisfied:\n");
error = GRBgetintattr(model, "NumConstrs", &numconstrs);
if (error) goto QUIT;
for (i = 0; i < numconstrs; ++i)
{
    error = GRBgetintattrelement(model, "IISConstr", i, &iis);
    if (error) goto QUIT;
    if (iis)
    {
        error = GRBgetstrattrelement(model, "ConstrName", i, &cname);
        if (error) goto QUIT;
        printf("%s\n", cname);
    }
}

QUIT:

/* Error reporting */
if (error)
{
    printf("ERROR: %s\n", GRBgeterrormsg(env));
    exit(1);
}

/* Free data */
free(cbeg);
free(cind);
free(cval);
free(sense);

/* Free model */
GRBfreemodel(model);

/* Free environment */
GRBfreeenv(env);

return 0;
}
workforce2_c.c

/* Copyright 2020, Gurobi Optimization, LLC */

/* Assign workers to shifts; each worker may or may not be available on a
particular day. If the problem cannot be solved, use IIS iteratively to
find all conflicting constraints. */

#include <stdlib.h>
#include <stdio.h>
#include <math.h>
#include <string.h>
#include "gurobi_c.h"

#define xcol(w,s) nShifts*w+s
#define MAXSTR 128

int main(int argc, char *argv[])
{
    GRBenv *env = NULL;
    GRBmodel *model = NULL;
    int error = 0, status;
    int s, w, col;
    int *cbeg = NULL;
    int *cind = NULL;
    int idx;
    double *cval = NULL;
    char *sense = NULL;
    char vname[MAXSTR];
    double obj;
    int i, iis, numconstrs, numremoved = 0;
    char *cname;
    char **removed = NULL;

    /* Sample data */
    const int nShifts = 14;
    const int nWorkers = 7;

    /* Sets of days and workers */
    char* Shifts[] =
    { "Mon1", "Tue2", "Wed3", "Thu4", "Fri5", "Sat6",
      "Sun7", "Mon8", "Tue9", "Wed10", "Thu11", "Fri12", "Sat13",
      "Sun14"};
    char* Workers[] =

    /* Number of workers required for each shift */
    double shiftRequirements[] =
    { 3, 2, 4, 4, 5, 6, 5, 2, 2, 3, 4, 6, 7, 5 };

    /* Amount each worker is paid to work one shift */
    double pay[] = { 10, 12, 10, 8, 8, 9, 11 };
/* Worker availability: 0 if the worker is unavailable for a shift */
double availability[][14] =
{ { 0, 1, 1, 0, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1 },
{ 1, 1, 0, 0, 1, 1, 0, 1, 0, 0, 1, 0, 1, 0 },
{ 0, 0, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1 },
{ 0, 1, 1, 0, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1 },
{ 1, 1, 1, 1, 0, 1, 1, 1, 0, 1, 0, 1, 1, 1 },
{ 1, 1, 1, 0, 0, 1, 0, 1, 1, 0, 0, 1, 1, 1 },
{ 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1 };

/* Create environment */
error = GRBloadenv(&env, "workforce2.log");
if (error) goto QUIT;

/* Create initial model */
error = GRBnewmodel(env, &model, "workforce2", nWorkers * nShifts,
NULL, NULL, NULL, NULL, NULL);
if (error) goto QUIT;

/* Initialize assignment decision variables:
x[w][s] == 1 if worker w is assigned
to shift s. Since an assignment model always produces integer
solutions, we use continuous variables and solve as an LP. */
for (w = 0; w < nWorkers; ++w)
{
  for (s = 0; s < nShifts; ++s)
  {
    col = xcol(w, s);
    sprintf(vname, "%s.%s", Workers[w], Shifts[s]);
    error = GRBsetdblattrelement(model, "UB", col, availability[w][s]);
    if (error) goto QUIT;
    error = GRBsetdblattrelement(model, "Obj", col, pay[w]);
    if (error) goto QUIT;
    error = GRBsetstrattrelement(model, "VarName", col, vname);
    if (error) goto QUIT;
  }
}

/* The objective is to minimize the total pay costs */
error = GRBsetintattr(model, "ModelSense", GRB_MINIMIZE);
if (error) goto QUIT;

/* Make space for constraint data */
cbeg = malloc(sizeof(int) * nShifts);
if (!cbeg) goto QUIT;
cind = malloc(sizeof(int) * nShifts * nWorkers);
if (!cind) goto QUIT;
cval = malloc(sizeof(double) * nShifts * nWorkers);
if (!cval) goto QUIT;
sense = malloc(sizeof(char) * nShifts);
if (!sense) goto QUIT;

/* Constraint: assign exactly shiftRequirements[s] workers
to each shift s */
idx = 0;
for (s = 0; s < nShifts; ++s)
{
    cbeg[s] = idx;
    sense[s] = GRB_EQUAL;
    for (w = 0; w < nWorkers; ++w)
    {
        cind[idx] = xcol(w, s);
        cval[idx++] = 1.0;
    }
}
error = GRBaddconstrs(model, nShifts, idx, cbeg, cind, cval, sense,
                      shiftRequirements, Shifts);
if (error) goto QUIT;

/* Optimize */
error = GRBoptimize(model);
if (error) goto QUIT;
error = GRBgetintattr(model, "Status", &status);
if (error) goto QUIT;
if (status == GRB_UNBOUNDED)
{
    printf("The model cannot be solved because it is unbounded\n");
    goto QUIT;
}
if (status == GRB_OPTIMAL)
{
    error = GRBgetdblattr(model, "ObjVal", &obj);
    if (error) goto QUIT;
    printf("The optimal objective is %f\n", obj);
    goto QUIT;
}
if ((status != GRB_INF_OR_UNBD) && (status != GRB_INFEASIBLE))
{
    printf("Optimization was stopped with status %i\n", status);
    goto QUIT;
}

/* do IIS */
printf("The model is infeasible; computing IIS\n");

/* Loop until we reduce to a model that can be solved */
error = GRBgetintattr(model, "NumConstrs", &numconstrs);
if (error) goto QUIT;
removed = calloc(numconstrs, sizeof(char*));
if (!removed) goto QUIT;
while (1)
{
    error = GRBcomputeIIS(model);
    if (error) goto QUIT;
    printf("\nThe following constraint cannot be satisfied: \n");
    for (i = 0; i < numconstrs; ++i)
    {
        error = GRBgetintattrelement(model, "IISConstr", i, &iis);
        if (error) goto QUIT;
        if (iis)
        {
            printf("\n");
        }
    }
error = GRBgetstrattrelement(model, "ConstrName", i, &cname);
if (error) goto QUIT;
printf("%s\n", cname);
/* Remove a single constraint from the model */
removed[numremoved] = malloc(sizeof(char) * (1+strlen(cname)));
if (!removed[numremoved]) goto QUIT;
strcpy(removed[numremoved++], cname);
cind[0] = i;
error = GRBdelconstrs(model, 1, cind);
if (error) goto QUIT;
break;
}
}

printf("\n");
error = GRBoptimize(model);
if (error) goto QUIT;
error = GRBgetintattr(model, "Status", &status);
if (error) goto QUIT;
if (status == GRB_UNBOUNDED)
{
    printf("The model cannot be solved because it is unbounded\n");
    goto QUIT;
}
if (status == GRB_OPTIMAL)
{
    break;
}
if ((status != GRB_INF_OR_UNBD) && (status != GRB_INFEASIBLE))
{
    printf("Optimization was stopped with status %i\n", status);
    goto QUIT;
}

printf("\nThe following constraints were removed to get a feasible LP:\n");
for (i = 0; i < numremoved; ++i)
{
    printf("%s ", removed[i]);
}
printf("\n");

QUIT:

/* Error reporting */

if (error)
{
    printf("ERROR: %s\n", GRBgeterrormsg(env));
    exit(1);
}

/* Free data */
free(cbeg);
free(cind);
free(cval);
free(sense);
for (i=0; i<numremoved; ++i)
{
    free(removed[i]);
}
free(removed);

/* Free model */
GRBfreemodel(model);

/* Free environment */
GRBfreeenv(env);

return 0;
}

workforce3_c.c

/* Copyright 2020, Gurobi Optimization, LLC */

/* Assign workers to shifts; each worker may or may not be available on a
particular day. If the problem cannot be solved, relax the model
to determine which constraints cannot be satisfied, and how much
they need to be relaxed. */

#include "stdlib.h"
#include "stdio.h"
#include "math.h"
#include "string.h"
#include "gurobi_c.h"

#define xcol(w,s) nShifts*w+s
#define MAXSTR 128

int
main(int argc,
    char *argv[])
{
    GRBenv *env = NULL;
    GRBmodel *model = NULL;
    int    error = 0, status;
    int    s, w, col;
    int   *cbeg = NULL;
    int   *cind = NULL;
    int    idx;
    double *cval = NULL;
    char   *sense = NULL;
    char   vname[MAXSTR];
    double    obj;
int i, j, orignumvars, numvars, numconstrs;
double *rhspen = NULL;
double sol;
char *sname;

/* Sample data */
const int nShifts = 14;
const int nWorkers = 7;

/* Sets of days and workers */
char* Shifts[] =
char* Workers[] =

/* Number of workers required for each shift */
double shiftRequirements[] =
{ 3, 2, 4, 4, 5, 6, 5, 2, 2, 3, 4, 6, 7, 5 };

/* Amount each worker is paid to work one shift */
double pay[] = { 10, 12, 10, 8, 8, 9, 11 };

/* Worker availability: 0 if the worker is unavailable for a shift */
double availability[][14] =
{ { 0, 1, 1, 0, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1 },
  { 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1 },
  { 0, 0, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1 },
  { 0, 1, 1, 0, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1 },
  { 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1 },
  { 1, 1, 1, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1 },
  { 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1 } };

/* Create environment */
error = GRBloadenv(&env, "workforce3.log");
if (error) goto QUIT;

/* Create initial model */
error = GRBnewmodel(env, &model, "workforce3", nWorkers * nShifts,
  NULL, NULL, NULL, NULL, NULL);
if (error) goto QUIT;

/* Initialize assignment decision variables:
x[w][s] == 1 if worker w is assigned
to shift s. Since an assignment model always produces integer
solutions, we use continuous variables and solve as an LP. */
for (w = 0; w < nWorkers; ++w)
{
  for (s = 0; s < nShifts; ++s)
  {
    col = xcol(w, s);
    sprintf(vname, "%s.%s", Workers[w], Shifts[s]);
    error = GRBsetdblattrelement(model, "UB", col, availability[w][s]);
    if (error) goto QUIT;
    error = GRBsetdblattrelement(model, "Obj", col, pay[w]);
  }
}
if (error) goto QUIT;
error = GRBsetstrattrelement(model, "VarName", col, vname);
if (error) goto QUIT;
}
}

/* The objective is to minimize the total pay costs */
error = GRBsetintattr(model, "ModelSense", GRB_MINIMIZE);
if (error) goto QUIT;

/* Make space for constraint data */
cbeg = malloc(sizeof(int) * nShifts);
if (!cbeg) goto QUIT;
cind = malloc(sizeof(int) * nShifts * nWorkers);
if (!cind) goto QUIT;
cval = malloc(sizeof(double) * nShifts * nWorkers);
if (!cval) goto QUIT;
sense = malloc(sizeof(char) * nShifts);
if (!sense) goto QUIT;

/* Constraint: assign exactly shiftRequirements[s] workers to each shift s */
idx = 0;
for (s = 0; s < nShifts; ++s) {
  cbeg[s] = idx;
  sense[s] = GRB_EQUAL;
  for (w = 0; w < nWorkers; ++w) {
    cind[idx] = xcol(w, s);
    cval[idx++] = 1.0;
  }
}
error = GRBaddconstrs(model, nShifts, idx, cbeg, cind, cval, sense,
                      shiftRequirements, Shifts);
if (error) goto QUIT;

/* Optimize */
error = GRBoptimize(model);
if (error) goto QUIT;
error = GRBgetintattr(model, "Status", &status);
if (error) goto QUIT;
if (status == GRB_UNBOUNDED) {
  printf("The model cannot be solved because it is unbounded\n");
  goto QUIT;
}
if (status == GRB_OPTIMAL) {
  error = GRBgetdblattr(model, "ObjVal", &obj);
  if (error) goto QUIT;
  printf("The optimal objective is %f\n", obj);
  goto QUIT;
}
if ((status != GRB_INF_OR_UNBD) && (status != GRB_INFEASIBLE))
printf("Optimization was stopped with status %i\n", status);
goto QUIT;
}

/* Relax the constraints to make the model feasible */
printf("The model is infeasible; relaxing the constraints\n");

/* Determine the matrix size before relaxing the constraints */
error = GRBgetintattr(model, "NumVars", &orignumvars);
if (error) goto QUIT;
error = GRBgetintattr(model, "NumConstrs", &numconstrs);
if (error) goto QUIT;

/* Use FeasRelax feature with penalties for constraint violations */
rhspen = malloc(sizeof(double) * numconstrs);
if (!rhspen) goto QUIT;
for (i = 0; i < numconstrs; i++) rhspen[i] = 1;
error = GRBfeasrelax(model, GRB_FEASRELAX_LINEAR, 0,
    NULL, NULL, rhspen, NULL);
if (error) goto QUIT;
error = GRBoptimize(model);
if (error) goto QUIT;
error = GRBgetintattr(model, "Status", &status);
if (error) goto QUIT;
if ((status == GRB_INF_OR_UNBD) || (status == GRB_INFEASIBLE) ||
    (status == GRB_UNBOUNDED))
{
    printf("The relaxed model cannot be solved "
        "because it is infeasible or unbounded\n");
goto QUIT;
}

if (status != GRB_OPTIMAL)
{
    printf("Optimization was stopped with status %i\n", status);
    goto QUIT;
}

printf("\nSlack values:\n");
error = GRBgetintattr(model, "NumVars", &numvars);
if (error) goto QUIT;
for (j = orignumvars; j < numvars; ++j)
{
    error = GRBgetdblattrelement(model, "X", j, &sol);
    if (error) goto QUIT;
    if (sol > 1e-6)
    {
        error = GRBgetstrattrelement(model, "VarName", j, &sname);
        if (error) goto QUIT;
        printf("%s = %f\n", sname, sol);
    }
}

QUIT:

/* Error reporting */
if ( error )
{
    printf("ERROR: %s\n", GRBgeterrormsg(env));
    exit(1);
}

/* Free data */
free(cbeg);
free(cind);
free(cval);
free(sense);
free(rhspen);

/* Free model */
GRBfreemodel(model);

/* Free environment */
GRBfreeenv(env);

return 0;
}

workforce4_c.c

/* Copyright 2020, Gurobi Optimization, LLC */

/* Assign workers to shifts; each worker may or may not be available on a particular day. We use Pareto optimization to solve the model: first, we minimize the linear sum of the slacks. Then, we constrain the sum of the slacks, and we minimize a quadratic objective that tries to balance the workload among the workers. */

#include <stdlib.h>
#include <stdio.h>
#include <math.h>
#include <string.h>
#include "gurobi_c.h"

int solveAndPrint(GRBmodel* model,
    int nShifts, int nWorkers, char** Workers,
    int* status);

#define xcol(w,s) nShifts*w+s
#define slackcol(s) nShifts*nWorkers+s
#define totSlackcol nShifts*(nWorkers+1)
#define totShiftscol(w) nShifts*(nWorkers+1)+1+w
#define avgShiftscol (nShifts+1)*(nWorkers+1)
#define diffShiftscol(w) (nShifts+1)*(nWorkers+1)+1+w
#define MAXSTR 128

int
main(int argc, char *argv[])
{
    GRBenv *env = NULL;
    GRBmodel *model = NULL;
    int error = 0, status;
    int s, w, col;
    int *cbeg = NULL;
    int *cind = NULL;
    int idx;
    double *cval = NULL;
    char *sense = NULL;
    char vname[MAXSTR], cname[MAXSTR];
    double val;

    /* Sample data */
    const int nShifts = 14;
    const int nWorkers = 7;

    /* Sets of days and workers */
    char* Shifts[] =
    { "Mon1", "Tue2", "Wed3", "Thu4", "Fri5", "Sat6",
      "Sun7", "Mon8", "Tue9", "Wed10", "Thu11", "Fri12", "Sat13",
      "Sun14" };
    char* Workers[] =

    /* Number of workers required for each shift */
    double shiftRequirements[] =
    { 3, 2, 4, 4, 5, 6, 5, 2, 2, 3, 4, 6, 7, 5 };

    /* Worker availability: 0 if the worker is unavailable for a shift */
    double availability[][14] =
    { { 0, 1, 1, 0, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1 },
      { 1, 1, 0, 0, 1, 1, 0, 0, 1, 0, 1, 0, 1, 0 },
      { 0, 0, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1 },
      { 0, 1, 1, 0, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1 },
      { 1, 1, 1, 0, 0, 1, 0, 1, 1, 1, 0, 1, 0, 1 },
      { 1, 1, 1, 0, 0, 1, 0, 1, 1, 0, 1, 1, 1, 1 },
      { 1, 1, 1, 0, 1, 0, 1, 1, 0, 1, 0, 1, 1, 1 } };

    /* Create environment */
    error = GRBloadenv(&env, "workforce4.log");
    if (error) goto QUIT;

    /* Create initial model */
    error = GRBnewmodel(env, &model, "workforce4",
                         (nShifts + 1) * (nWorkers + 1),
                         NULL, NULL, NULL, NULL, NULL);
    if (error) goto QUIT;

    /* Initialize assignment decision variables:
     x[w][s] == 1 if worker w is assigned to shift s.
     This is no longer a pure assignment model, so we must
     use binary variables. */
    for (w = 0; w < nWorkers; ++w)
for (s = 0; s < nShifts; ++s)
{
    col = xcol(w, s);
    sprintf(vname, "%s.%s", Workers[w], Shifts[s]);
    error = GRBsetcharattrelement(model, "VType", col, GRB_BINARY);
    if (error) goto QUIT;
    error = GRBsetdblattrelement(model, "UB", col, availability[w][s]);
    if (error) goto QUIT;
    error = GRBsetstrattrelement(model, "VarName", col, vname);
    if (error) goto QUIT;
}

/* Initialize slack decision variables */
for (s = 0; s < nShifts; ++s)
{
    sprintf(vname, "%sSlack", Shifts[s]);
    error = GRBsetstrattrelement(model, "VarName", slackcol(s), vname);
    if (error) goto QUIT;
}

/* Initialize total slack decision variable */
error = GRBsetstrattrelement(model, "VarName", totSlackcol, "totSlack");
if (error) goto QUIT;

/* Initialize variables to count the total shifts worked by each worker */
for (w = 0; w < nWorkers; ++w)
{
    sprintf(vname, "%sTotShifts", Workers[w]);
    error = GRBsetstrattrelement(model, "VarName", totShiftscol(w), vname);
    if (error) goto QUIT;
}

/* The objective is to minimize the sum of the slacks */
error = GRBsetintattr(model, "ModelSense", GRB_MINIMIZE);
if (error) goto QUIT;
error = GRBsetdblattrelement(model, "Obj", totSlackcol, 1.0);
if (error) goto QUIT;

/* Make space for constraint data */
cbeg = malloc(sizeof(int) * nShifts);
if (!cbeg) goto QUIT;
cind = malloc(sizeof(int) * nShifts * (nWorkers + 1));
if (!cind) goto QUIT;
cval = malloc(sizeof(double) * nShifts * (nWorkers + 1));
if (!cval) goto QUIT;
sense = malloc(sizeof(char) * nShifts);
if (!sense) goto QUIT;

/* Constraint: assign exactly shiftRequirements[s] workers to each shift s, plus the slack */
idx = 0;
for (s = 0; s < nShifts; ++s)
{
    cbeg[s] = idx;
    idx += shiftRequirements[s];
}
sense[s] = GRB_EQUAL;
for (w = 0; w < nWorkers; ++w)
{
    cind[idx] = xcol(w, s);
    cval[idx++] = 1.0;
}
cind[idx] = slackcol(s);
cval[idx++] = 1.0;
}
error = GRBaddconstrs(model, nShifts, idx, cbeg, cind, cval, sense,
                      shiftRequirements, Shifts);
if (error) goto QUIT;

/* Constraint: set totSlack column equal to the total slack */
idx = 0;
for (s = 0; s < nShifts; ++s)
{
    cind[idx] = slackcol(s);
    cval[idx++] = 1.0;
}
cind[idx] = totSlackcol;
cval[idx++] = -1.0;
error = GRBaddconstr(model, idx, cind, cval, GRB_EQUAL,
                      0.0, "totSlack");
if (error) goto QUIT;

/* Constraint: compute the total number of shifts for each worker */
for (w = 0; w < nWorkers; ++w)
{
    idx = 0;
    for (s = 0; s < nShifts; ++s)
    {
        cind[idx] = xcol(w,s);
        cval[idx++] = 1.0;
    }
    sprintf(cname, "totShifts%\s", Workers[w]);
    cind[idx] = totShiftscol(w);
    cval[idx++] = -1.0;
    error = GRBaddconstr(model, idx, cind, cval, GRB_EQUAL, 0.0, cname);
    if (error) goto QUIT;
}

/* Optimize */
error = solveAndPrint(model, nShifts, nWorkers, Workers, &status);
if (error) goto QUIT;
if (status != GRB_OPTIMAL) goto QUIT;

/* Constrain the slack by setting its upper and lower bounds */
error = GRBgetdblattrelement(model, "X", totSlackcol, &val);
if (error) goto QUIT;
error = GRBsetdblattrelement(model, "UB", totSlackcol, val);
if (error) goto QUIT;
error = GRBsetdblattrelement(model, "LB", totSlackcol, val);
if (error) goto QUIT;

/* Variable to count the average number of shifts worked */
error = GRBaddvar(model, 0, NULL, NULL, 0, 0, GRB_INFINITY, GRBCONTINUOUS, "avgShifts");

if (error) goto QUIT;

/* Variables to count the difference from average for each worker; note that these variables can take negative values. */
error = GRBaddvars(model, nWorkers, 0, NULL, NULL, NULL, NULL, NULL, NULL, NULL);
if (error) goto QUIT;

for (w = 0; w < nWorkers; ++w)
{
    sprintf(vname, "%sDiff", Workers[w]);
    error = GRBsetstrattrelement(model, "VarName", diffShiftscol(w), vname);
    if (error) goto QUIT;
    error = GRBsetdblattrelement(model, "LB", diffShiftscol(w), -GRB_INFINITY);
    if (error) goto QUIT;
}

/* Constraint: compute the average number of shifts worked */
idx = 0;
for (w = 0; w < nWorkers; ++w)
{
    cind[idx] = totShiftscol(w);
    cval[idx++] = 1.0;
}

for (w = 0; w < nWorkers; ++w)
{
    cind[idx] = avgShiftscol;
    cval[idx++] = -nWorkers;
    error = GRBaddconstr(model, idx, cind, cval, GRB_EQUAL, 0.0, "avgShifts");
    if (error) goto QUIT;
}

/* Constraint: compute the difference from the average number of shifts */
for (w = 0; w < nWorkers; ++w)
{
    cind[0] = totShiftscol(w);
    cval[0] = 1.0;
    cind[1] = avgShiftscol;
    cval[1] = -1.0;
    cind[2] = diffShiftscol(w);
    cval[2] = -1.0;
    sprintf(cname, "%sDiff", Workers[w]);
    error = GRBaddconstr(model, 3, cind, cval, GRB_EQUAL, 0.0, cname);
    if (error) goto QUIT;
}

/* Objective: minimize the sum of the square of the difference from the average number of shifts worked */
error = GRBsetdblattrelement(model, "Obj", totSlackcol, 0.0);
if (error) goto QUIT;

for (w = 0; w < nWorkers; ++w)
{
    cind[w] = diffShiftscol(w);
    cval[w] = 1.0;
}
error = GRBaddqpterms(model, nWorkers, cind, cind, cval);
if (error) goto QUIT;

/* Optimize */
error = solveAndPrint(model, nShifts, nWorkers, Workers, &status);
if (error) goto QUIT;
if (status != GRB_OPTIMAL) goto QUIT;

QUIT:

/* Error reporting */
if (error)
{
    printf("ERROR: %s\n", GRBgeterrormsg(env));
    exit(1);
}

/* Free data */
free(cbeg);
free(cind);
free(cval);
free(sense);

/* Free model */
GRBfreemodel(model);

/* Free environment */
GRBfreenv(env);
return 0;
}

int solveAndPrint(GRBmodel* model,
    int nShifts, int nWorkers, char** Workers,
    int* status)
{
    int error, w;
    double val;

    error = GRBoptimize(model);
    if (error) return error;

    error = GRBgetintattr(model, "Status", status);
    if (error) return error;

    if ((*status == GRB_INF_OR_UNBD) || (*status == GRB_INFEASIBLE) ||
        (*status == GRB_UNBOUNDED))
    {
        printf("The model cannot be solved 
                because it is infeasible or unbounded\n");
        return 0;
    }
if (*status != GRB_OPTIMAL)
{
    printf("Optimization was stopped with status %i\n", *status);
    return 0;
}

/* Print total slack and the number of shifts worked for each worker */
error = GRBgetdblattrelement(model, "X", totSlackcol, &val);
if (error) return error;
printf("Total slack required: %f\n", val);
for (w = 0; w < nWorkers; ++w)
{
    error = GRBgetdblattrelement(model, "X", totShiftscol(w), &val);
    if (error) return error;
    printf("%s worked %f shifts\n", Workers[w], val);
}
printf("\n");
return 0;

workforce5_c.c

/* Copyright 2020, Gurobi Optimization, LLC */

/* Assign workers to shifts; each worker may or may not be available on a
particular day. We use multi-objective optimization to solve the model.
The highest-priority objective minimizes the sum of the slacks
(i.e., the total number of uncovered shifts). The secondary objective
minimizes the difference between the maximum and minimum number of
shifts worked among all workers. The second optimization is allowed
to degrade the first objective by up to the smaller value of 10% and 2 */

#include <stdlib.h>
#include <stdio.h>
#include <math.h>
#include <string.h>
#include "gurobi_c.h"

int solveAndPrint(GRBmodel* model,
    int nShifts, int nWorkers, char** Workers,
    int* status);

#define xcol(w,s) nShifts*w+s
#define slackcol(s) nShifts*nWorkers+s
#define totSlackcol nShifts*(nWorkers+1)
#define totShiftscol(w) nShifts*(nWorkers+1)+1+w
#define minShiftcol (nShifts+1)*(nWorkers+1)
#define maxShiftcol (nShifts+1)*(nWorkers+1)+1
#define MAXSTR 128

int main(int argc,

char *argv[])
{
GRBenv *env = NULL;
GRBenv *menv = NULL;
GRBmodel *model = NULL;
int error = 0, status;
int s, w, col;
int *cbeg = NULL;
int *cind = NULL;
int idx;
double *cval = NULL;
char *sense = NULL;
char vname[MAXSTR], cname[MAXSTR];

/* Sample data */
const int nShifts = 14;
const int nWorkers = 8;

/* Sets of days and workers */
char* Shifts[] =
{ "Mon1", "Tue2", "Wed3", "Thu4", "Fri5", "Sat6",
 "Sun7", "Mon8", "Tue9", "Wed10", "Thu11", "Fri12", "Sat13",
 "Sun14" \};
char* Workers[] =

/* Number of workers required for each shift */
double shiftRequirements[] =
{ 3, 2, 4, 4, 5, 6, 5, 2, 2, 3, 4, 6, 7, 5 \};

/* Worker availability: 0 if the worker is unavailable for a shift */
double availability[][14] =
{ { 0, 1, 1, 0, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1 },
 { 1, 1, 0, 0, 1, 1, 0, 1, 0, 1, 0, 1, 0, 1 },
 { 0, 0, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1 },
 { 0, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1 },
 { 1, 1, 1, 1, 0, 1, 1, 0, 1, 0, 1, 0, 1, 1 },
 { 1, 1, 1, 0, 0, 1, 1, 0, 1, 0, 1, 0, 1, 1 },
 { 0, 1, 1, 1, 0, 1, 1, 0, 1, 1, 1, 0, 1, 1 },
 { 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1 \};

/* Create environment */
error = GRBloadenv(&env, "workforce5.log");
if (error) goto QUIT;

/* Create initial model */
error = GRBnewmodel(env, &model, "workforce5",
 (nShifts + 1) * (nWorkers + 1) + 2,
 NULL, NULL, NULL, NULL, NULL);
if (error) goto QUIT;

/* get model environment */
menv = GRBgetenv(model);
if (!menv) {
    fprintf(stderr, "Error: could not get model environment\n");
    goto QUIT;
}
Initialize assignment decision variables:
\[ x[w][s] \] == 1 if worker w is assigned to shift s.
This is no longer a pure assignment model, so we must
use binary variables. */

for (w = 0; w < nWorkers; ++w)
{
    for (s = 0; s < nShifts; ++s)
    {
        col = xcol(w, s);
        sprintf(vname, "%s.%s", Workers[w], Shifts[s]);
        error = GRBsetcharattrelement(model, "VType", col, GRB_BINARY);
        if (error) goto QUIT;
        error = GRBsetdblattrelement(model, "UB", col, availability[w][s]);
        if (error) goto QUIT;
        error = GRBsetstrattrelement(model, "VarName", col, vname);
        if (error) goto QUIT;
    }
}

/* Initialize slack decision variables */
for (s = 0; s < nShifts; ++s)
{
    sprintf(vname, "%sSlack", Shifts[s]);
    error = GRBsetstrattrelement(model, "VarName", slackcol(s), vname);
    if (error) goto QUIT;
}

/* Initialize total slack decision variable */
error = GRBsetstrattrelement(model, "VarName", totSlackcol, "totSlack");
if (error) goto QUIT;

/* Initialize variables to count the total shifts worked by each worker */
for (w = 0; w < nWorkers; ++w)
{
    sprintf(vname, "%sTotShifts", Workers[w]);
    error = GRBsetstrattrelement(model, "VarName", totShiftscol(w), vname);
    if (error) goto QUIT;
}

/* Initialize max and min #shifts variables */
sprintf(vname, "minShifts");
error = GRBsetstrattrelement(model, "VarName", minShiftcol, vname);
sprintf(vname, "maxShifts");
error = GRBsetstrattrelement(model, "VarName", maxShiftcol, vname);

/* Make space for constraint data */
cbeg = malloc(sizeof(int) * nShifts);
if (!cbeg) goto QUIT;
cind = malloc(sizeof(int) * nShifts * (nWorkers + 1));
if (!cind) goto QUIT;
cval = malloc(sizeof(double) * nShifts * (nWorkers + 1));
if (!cval) goto QUIT;
sense = malloc(sizeof(char) * (nShifts + nWorkers));
if (!sense) goto QUIT;

/* Constraint: assign exactly shiftRequirements[s] workers to each shift s, plus the slack */
idx = 0;
for (s = 0; s < nShifts; ++s)
{
    cbeg[s] = idx;
    sense[s] = GRB_EQUAL;
    for (w = 0; w < nWorkers; ++w)
    {
        cind[idx] = xcol(w, s);
        cval[idx++] = 1.0;
    }
    cind[idx] = slackcol(s);
    cval[idx++] = 1.0;
}
error = GRBaddconstrs(model, nShifts, idx, cbeg, cind, cval, sense,
                       shiftRequirements, Shifts);
if (error) goto QUIT;

/* Constraint: set totSlack column equal to the total slack */
idx = 0;
for (s = 0; s < nShifts; ++s)
{
    cind[idx] = slackcol(s);
    cval[idx++] = 1.0;
}
idx = 0;
cind[idx] = totSlackcol;
cval[idx++] = -1.0;
error = GRBaddconstr(model, idx, cind, cval, GRB_EQUAL, 0.0, "totSlack");
if (error) goto QUIT;

/* Constraint: compute the total number of shifts for each worker */
for (w = 0; w < nWorkers; ++w)
{
    idx = 0;
    for (s = 0; s < nShifts; ++s)
    {
        cind[idx] = xcol(w, s);
        cval[idx++] = 1.0;
    }
    sprintf(cname, "totShifts%"s, Workers[w]);
    cind[idx] = totShiftscol(w);
    cval[idx++] = -1.0;
    error = GRBaddconstr(model, idx, cind, cval, GRB_EQUAL, 0.0, cname);
    if (error) goto QUIT;
}

/* Constraint: set minShift/maxShift variable to less <=/= to the * number of shifts among all workers */
for (w = 0; w < nWorkers; w++)
    cind[w] = totShiftscol(w);
error = GRBaddgenconstrMin(model, NULL, minShiftcol, nWorkers, cind, GRB_INFINITY);
if (error) goto QUIT;
error = GRBaddgenconstrMax(model, NULL, maxShiftcol, nWorkers, cind, -GRB_INFINITY);
if (error) goto QUIT;

/* Set global sense for ALL objectives */
error = GRBsetintattr(model, GRB_INT_ATTR_MODELSENSE, GRB_MINIMIZE);
if (error) goto QUIT;

/* Set primary objective */
cind[0] = totSlackcol;
cval[0] = 1.0;
error = GRBsetobjectiven(model, 0, 2, 1.0, 2.0, 0.10, "TotalSlack",
0.0, 1, cind, cval);
if (error) goto QUIT;

/* Set secondary objective */
cind[0] = maxShiftcol;
cval[0] = 1.0;
cind[1] = minShiftcol;
cval[1] = -1.0;
error = GRBsetobjectiven(model, 1, 1, 1.0, 0, 0, "Fairness",
0.0, 2, cind, cval);
if (error) goto QUIT;

/* Save problem */
error = GRBwrite(model, "workforce5.lp");
if (error) goto QUIT;
error = GRBwrite(model, "workforce5.mps");
if (error) goto QUIT;

/* Optimize */
error = solveAndPrint(model, nShifts, nWorkers, Workers, &status);
if (error) goto QUIT;
if (status != GRB_OPTIMAL) goto QUIT;

QUIT:

/* Error reporting */

if (error)
{
    printf("ERROR: %s\n", GRBgeterrormsg(env));
    exit(1);
}

/* Free data */

free(cbeg);
free(cind);
free(cval);
free(sense);

/* Free model */

GRBfreemodel(model);
/* Free environment */

GRBfreeenv(env);

return 0;
}

int solveAndPrint(GRBmodel* model,
                  int nShifts, int nWorkers, char** Workers,
                  int* status)
{
    int error, w;
    double val;

    error = GRBoptimize(model);
    if (error) return error;

    error = GRBgetintattr(model, "Status", status);
    if (error) return error;

    if (*((status == GRB_INF_OR_UNBD) || (*status == GRB_INFEASIBLE) ||
         (*status == GRB_UNBOUNDED))
    { printf("The model cannot be solved 
           because it is infeasible or unbounded\n");
      return 0;
    }
    if (*status != GRB_OPTIMAL)
    {
      printf("Optimization was stopped with status %i\n", *status);
      return 0;
    }

    /* Print total slack and the number of shifts worked for each worker */
    error = GRBgetdblattrelement(model, "X", totSlackcol, &val);
    if (error) return error;

    printf("\nTotal slack required: %f\n", val);
    for (w = 0; w < nWorkers; ++w)
    {
      error = GRBgetdblattrelement(model, "X", totShiftscol(w), &val);
      if (error) return error;
      printf("%s worked %f shifts\n", Workers[w], val);
    }
    printf("\n");
    return 0;
}

3.2 C++ Examples

This section includes source code for all of the Gurobi C++ examples. The same source code can be found in the examples/c++ directory of the Gurobi distribution.
/* Copyright 2020, Gurobi Optimization, LLC */

// This example reads a MIP model from a file, solves it in batch mode, 
// and prints the JSON solution string.
//
// You will need a Cluster Manager license for this example to work.

#include <ctime>
#if defined (WIN32) || defined(WIN64) || defined(_WIN32) || defined(_WIN64)
#include <Windows.h>
define sleep(n) Sleep(1000*n)
#else
#include <unistd.h>
#endif
#include "gurobi_c++.h"

using namespace std;

// Set-up the environment for batch mode optimization.
//
// The function configures and start an environment to be used for batch
// optimization.

void setupbatchenv(GRBEnv* env)
{
    env->set(GRB_StringParam_LogFile, "batchmode.log");
    env->set(GRB_StringParam_CSManager, "http://localhost:61080");
    env->set(GRB_StringParam_UserName, "gurobi");
    env->set(GRB_StringParam_ServerPassword, "pass");
    env->set(GRB_IntParam_CSBatchMode, 1);

    // No network communication happened up to this point. This will happen
    // now that we call the start() method.
    env->start();
}

// Print batch job error information, if any
void printbatcherrorinfo(GRBBatch &batch)
{
    if (batch.get(GRB_IntAttr_BatchErrorCode) == 0)
        return;

    cerr << "Batch ID " << batch.get(GRB_StringAttr_BatchID)
        << ": Error code " << batch.get(GRB_IntAttr_BatchErrorCode)
        << " (" << batch.get(GRB_StringAttr_BatchErrorMessage)
        << ")" << endl;
}

// Create a batch request for given problem file
string newbatchrequest(char* filename)
{
    GRBEnv* env = NULL;
    GRBModel* model = NULL;
GRBVar* v = NULL;
string batchID;

try {
    // Start environment, create Model object from file
    env = new GRBEnv(true);
    setupbatchenv(env);
    model = new GRBModel(*env, filename);

    // Set some parameters; switch on detailed JSON information
    model->set(GRB_DoubleParam_MIPGap, 0.01);
    model->set(GRB_IntParam_JSONSolDetail, 1);

    // Define tags for some variables in order to access their values later
    int numvars = model->get(GRB_IntAttr_NumVars);
    v = model->getVars();
    if (numvars > 10) numvars = 10;
    for (int j = 0; j < numvars; j++) {
        char vtag[64];
        sprintf(vtag, "Variable %d", j);
        v[j].set(GRB_StringAttr_VTag, string(vtag));
    }

    // submit batch request
    batchID = model->optimizeBatch();
}

catch (...) {
    // Free local resources
    delete[] v;
    delete model;
    delete env;
    // Let the exception propagate
    throw;
}

    // Free local resources
    delete[] v;
    delete model;
    delete env;

    return batchID;
}

// Wait for the final status of the batch.
// Initially the status of a batch is "submitted"; the status will change
// once the batch has been processed (by a compute server).

void waitforfinalstatus(string batchID)
{
    // Wait no longer than one hour
    time_t maxwaittime = 3600;
    GRBEnv* env = NULL;
    GRBBatch* batch = NULL;

    try {

// Setup and start environment, create local Batch handle object
env = new GRBEnv(true);
setupbatchenv(env);
batch = new GRBBatch(*env, batchID);
time_t starttime = time(NULL);
int BatchStatus = batch->get(GRB_IntAttr_BatchStatus);

while (BatchStatus == GRB_BATCH_SUBMITTED) {
    // Abort this batch if it is taking too long
    time_t curtime = time(NULL);
    if (curtime - starttime > maxwaittime) {
        batch->abort();
        break;
    }
    // Wait for two seconds
    sleep(2);
    // Update the resident attribute cache of the Batch object with the
    // latest values from the cluster manager.
    batch->update();
    BatchStatus = batch->get(GRB_IntAttr_BatchStatus);
    // If the batch failed, we try again
    if (BatchStatus == GRB_BATCH_FAILED)
        batch->retry();
} catch (...) {
    // Print information about error status of the job that
    // processed the batch
    printbatcherrorinfo(*batch);
    // Free local resources
    delete batch;
    delete env;
    // let the exception propagate
    throw;
}

// Free local resources
delete batch;
delete env;

void
printfinalreport(string batchID)
{
    GRBEnv* env = NULL;
    GRBBatch* batch = NULL;

    try {
        // Setup and starts environment, create local Batch handle object
        env = new GRBEnv(true);
        setupbatchenv(env);
        batch = new GRBBatch(*env, batchID);
    }
}
int BatchStatus = batch->get( GRB_IntAttr_BatchStatus);
if (BatchStatus == GRB_BATCH_CREATED)
    cout << "Batch status is 'CREATED'" << endl;
else if (BatchStatus == GRB_BATCH_SUBMITTED)
    cout << "Batch is 'SUBMITTED'" << endl;
else if (BatchStatus == GRB_BATCH_ABORTED)
    cout << "Batch is 'ABORTED'" << endl;
else if (BatchStatus == GRB_BATCH_FAILED)
    cout << "Batch is 'FAILED'" << endl;
else if (BatchStatus == GRB_BATCH_COMPLETED) {
    cout << "Batch is 'COMPLETED'" << endl;
    // Pretty printing the general solution information
    cout << "JSON solution:" << batch->getJSONSolution() << endl;
    // Write the full JSON solution string to a file
    batch->writeJSONSolution("batch-sol.json.gz");
} else {
    // Should not happen
    cout << "Batch has unknown BatchStatus" << endl;
}
} catch ( ...) {
    // Free local resources
    delete batch;
    delete env;
    // let the exception propagate
    throw;
}

// Instruct cluster manager to remove all data relating to this BatchID
void batchdiscard(string batchID)
{
    GRBEnv*   env = NULL;
    GRBBatch* batch = NULL;

    try {
    // Setup and start environment, create local Batch handle object
    env = new GRBEnv(true);
    setupbatchenv(env);
    batch = new GRBBatch(*env, batchID);

    // Remove batch request from manager
    batch->discard();
    } catch ( ...) {
    // Free local resources even
    delete batch;
    delete env;
    // let the exception propagate
    throw;
}
// Free local resources
delete batch;
delete env;
}

// Solve a given model using batch optimization
int main(int argc, char** argv)
{
    // Ensure we have an input file
    if (argc != 2) {
        cout << "Usage: " << argv[0] << " filename " << endl;
        return 0;
    }

    try {
        // Submit new batch request
        string batchID = newbatchrequest(argv[1]);

        // Wait for final status
        waitforfinalstatus(batchID);

        // Report final status info
        printfinalreport(batchID);

        // Remove batch request from manager
        batchdiscard(batchID);

        cout << "Batch optimization OK" << endl;
    }
    catch (GRBException e) {
        cout << "Error code = " << e.getErrorCode() << endl;
        cout << e.getMessage() << endl;
    }
    catch (...) {
        cout << "Exception during optimization" << endl;
    }
    return 0;
}

bilinear_c++.cpp
/* Copyright 2020, Gurobi Optimization, LLC */

/* This example formulates and solves the following simple bilinear model: */

maximize x
subject to x + y + z <= 10
x * y <= 2 (bilinear inequality)
x * z + y * z == 1 (bilinear equality)
x, y, z non-negative (x integral in second version)
*/
#include <cassert>
#include "gurobi_c++.h"
using namespace std;

int
```cpp
main(int argc, char *argv[])
{
    try {
        GRBEnv env = GRBEnv();
        GRBModel model = GRBModel(env);

        // Create variables
        GRBVar x = model.addVar(0.0, GRB_INFINITY, 0.0, GRB_CONTINUOUS, "x");
        GRBVar y = model.addVar(0.0, GRB_INFINITY, 0.0, GRB_CONTINUOUS, "y");
        GRBVar z = model.addVar(0.0, GRB_INFINITY, 0.0, GRB_CONTINUOUS, "z");

        // Set objective
        GRBLinExpr obj = x;
        model.setObjective(obj, GRB_MAXIMIZE);

        // Add linear constraint: x + y + z <= 10
        model.addConstr(x + y + z <= 10, "c0");

        // Add bilinear inequality constraint: x * y <= 2
        model.addQConstr(x*y <= 2, "bilinear0");

        // Add bilinear equality constraint: y * z == 1
        model.addQConstr(x*z + y*z == 1, "bilinear1");

        // First optimize() call will fail - need to set NonConvex to 2
        try {
            model.optimize();
            assert(0);
        } catch (GRBException e) {
            cout << "Failed (as expected)" << endl;
        }

        model.set(GRB_IntParam_NonConvex, 2);
        model.optimize();

        cout << x.get(GRB_StringAttr_VarName) << " "
             << x.get(GRB_DoubleAttr_X) << endl;
        cout << y.get(GRB_StringAttr_VarName) << " "
             << y.get(GRB_DoubleAttr_X) << endl;
        cout << z.get(GRB_StringAttr_VarName) << " "
             << z.get(GRB_DoubleAttr_X) << endl;

        // Constrain x to be integral and solve again
        x.set(GRB_CharAttr_VType, GRB_INTEGER);
        model.optimize();

        cout << x.get(GRB_StringAttr_VarName) << " "
             << x.get(GRB_DoubleAttr_X) << endl;
    }

    catch (GRBException e) {
        cout << e.getString() << endl;
    }
}
```
cout << y.get(GRB_StringAttr_VarName) << " "
<< y.get(GRB_DoubleAttr_X) << endl;
cout << z.get(GRB_StringAttr_VarName) << " "
<< z.get(GRB_DoubleAttr_X) << endl;

cout << "Obj: " << model.get(GRB_DoubleAttr_ObjVal) << endl;
} catch(GRBException e) {
    cout << "Error code = " << e.getErrorCode() << endl;
    cout << e.getMessage() << endl;
} catch(...) {
    cout << "Exception during optimization" << endl;
}

return 0;

}
logfile = xlogfile;

protected:
  void callback () {
    try {
      if (where == GRB_CB_POLLING) {
        // Ignore polling callback
      } else if (where == GRB_CB_PRESOLVE) {
        // Presolve callback
        int cdels = getIntInfo(GRB_CB_PRE_COLDEL);
        int rdels = getIntInfo(GRB_CB_PRE ROWDEL);
        if (cdels || rdels) {
          cout << cdels << " columns and " << rdels
               << " rows are removed" << endl;
        }
      } else if (where == GRB_CB_SIMPLEX) {
        // Simplex callback
        double itcnt = getDoubleInfo(GRB_CB_SPX_ITRCNT);
        if (itcnt - lastiter >= 100) {
          lastiter = itcnt;
          double obj = getDoubleInfo(GRB_CB_SPX_OBJVAL);
          int ispert = getIntInfo(GRB_CB_SPX_ISPERT);
          double pinf = getDoubleInfo(GRB_CB_SPX_PRIMINF);
          double dinf = getDoubleInfo(GRB_CB_SPX_DUALINF);
          char ch;
          if (ispert == 0) ch = ' ';
          else if (ispert == 1) ch = 'S';
          else ch = 'P';
          cout << itcnt << " " << obj << ch << " " << pinf << " " << dinf << endl;
        }
      } else if (where == GRB_CB_MIP) {
        // General MIP callback
        double nodedcnt = getDoubleInfo(GRB_CB_MIP_NODCNT);
        double objbst = getDoubleInfo(GRB_CB_MIP_OBJBST);
        double objbnd = getDoubleInfo(GRB_CB_MIP_OBJBND);
        int solcnt = getIntInfo(GRB_CB_MIP_SOLCNT);
        if (nodedcnt - lastnode >= 100) {
          lastnode = nodedcnt;
          int actnodes = (int) getDoubleInfo(GRB_CB_MIP_NODLFT);
          int itcnt = (int) getDoubleInfo(GRB_CB_MIP_ITRCNT);
          int cutcnt = getDoubleInfo(GRB_CB_MIP_CUTCNT);
          cout << nodedcnt << " " << actnodes << " " << itcnt
               << " " << objbst << " " << objbnd << " "
               << solcnt << " " << cutcnt << endl;
        }
        if (fabs(objbst - objbnd) < 0.1 * (1.0 + fabs(objbst))) {
          cout << "Stop early - 10% gap achieved" << endl;
          abort();
        }
        if (nodedcnt >= 10000 && solcnt) {
          cout << "Stop early - 10000 nodes explored" << endl;
          abort();
        }
      } else if (where == GRB_CB_MIPSOL) {
        // MIP solution callback
      }
    }
  }

146
```cpp
int nodecnt = (int) getDoubleInfo(GRB_CB_MIPSOL_NODCNT);
double obj = getDoubleInfo(GRB_CB_MIPSOL_OBJ);
int solcnt = getIntInfo(GRB_CB_MIPSOL_SOLCNT);
double* x = getSolution(vars, numvars);
cout << "**** New solution at node " << nodecnt
    << ", obj " << obj << ", sol " << solcnt
    << ", x[0] = " << x[0] << " ****" << endl;
delete[] x;
}
```

```cpp
else if (where == GRB_CB_MIPNODE) {
    // MIP node callback
    cout << "**** New node ****" << endl;
    if (getIntInfo(GRB_CB_MIPNODE_STATUS) == GRB_OPTIMAL) {
        double* x = getNodeRel(vars, numvars);
        setSolution(vars, x, numvars);
        delete[] x;
    }
}
```

```cpp
else if (where == GRB_CB_BARRIER) {
    // Barrier callback
    int itcnt = getIntInfo(GRB_CB_BARRIER_ITRCNT);
double primobj = getDoubleInfo(GRB_CB_BARRIER_PRIMOBJ);
double dualobj = getDoubleInfo(GRB_CB_BARRIER_DUALOBJ);
double priminf = getDoubleInfo(GRB_CB_BARRIER_PRIMINF);
double dualinf = getDoubleInfo(GRB_CB_BARRIER_DUALINF);
double cmpl = getDoubleInfo(GRB_CB_BARRIER_COMPL);
cout << itcnt << " " << primobj << " " << dualobj << " "
    << priminf << " " << dualinf << " " << cmpl << endl;
}
```

```cpp
else if (where == GRB_CB_MESSAGE) {
    // Message callback
    string msg = getStringInfo(GRB_CB_MSG_STRING);
    *logfile << msg;
}
```

```cpp
catch (GRBException e) {
    cout << "Error number: ": " << e.getErrorCode() << endl;
    cout << e.getMessage() << endl;
}
```

```cpp

t main(int argc,
    char *argv[])
{
    if (argc < 2) {
        cout << "Usage: callback_c++ filename" << endl;
        return 1;
    }

    // Open log file
    ofstream logfile("cb.log");
    if (!logfile.is_open()) {
        cout << "Cannot open cb.log for callback message" << endl;
        return 1;
    }
}
```
GRBEnv *env = 0;
GRBVar *vars = 0;

try {
    // Create environment
    env = new GRBEnv();

    // Read model from file
    GRBModel model = GRBModel(*env, argv[1]);

    // Turn off display and heuristics
    model.set(GRB_IntParam_OutputFlag, 0);
    model.set(GRB_DoubleParam_Heuristics, 0.0);

    // Create a callback object and associate it with the model
    int numvars = model.get(GRB_IntAttr_NumVars);
    vars = model.getVars();
    mycallback cb = mycallback(numvars, vars, &logfile);
    model.setCallback(&cb);

    // Solve model and capture solution information
    model.optimize();

    cout << endl << "Optimization complete" << endl;
    if (model.get(GRB_IntAttr_SolCount) == 0) {
        cout << "No solution found, optimization status = " <<
             model.get(GRB_IntAttr_Status) << endl;
    } else {
        cout << "Solution found, objective = " <<
             model.get(GRB_DoubleAttr_ObjVal) << endl;
        for (int j = 0; j < numvars; j++) {
            GRBVar v = vars[j];
            double x = v.get(GRB_DoubleAttr_X);
            if (x != 0.0) {
                cout << v.get(GRB_StringAttr_VarName) << " " << x << endl;
            }
        }
    }

    catch (GRBException e) {
        cout << "Error number: " << e.getErrorCode() << endl;
        cout << e.getMessage() << endl;
    }
    catch (...) {
        cout << "Error during optimization" << endl;
    }

    // Close log file
    logfile.close();
    delete[] vars;
    delete env;

    return 0;
}

148
/* Copyright 2020, Gurobi Optimization, LLC */

/* This example formulates and solves the following simple QP model:

minimize x + y + x^2 + x*y + y^2 + y*z + z^2
subject to x + 2y + 3z >= 4
x + y >= 1
x, y, z non-negative

The example illustrates the use of dense matrices to store A and Q
(and dense vectors for the other relevant data). We don't recommend
that you use dense matrices, but this example may be helpful if you
already have your data in this format. */

#include "gurobi_c++.h"
using namespace std;

static bool dense_optimize (GRBEnv * env,
int rows,
int cols,
double * c, /* linear portion of objective function */
double * Q, /* quadratic portion of objective function */
double * A, /* constraint matrix */
char * sense, /* constraint senses */
double * rhs, /* RHS vector */
char * lb, /* variable lower bounds */
char * ub, /* variable upper bounds */
char * vtype, /* variable types (continuous, binary, etc.) */
double * solution,
double * objvalP)
{
  GRBModel model = GRBModel(*env);
  int i, j;
  bool success = false;

  /* Add variables to the model */
  GRBVar * vars = model.addVars(lb, ub, NULL, vtype, NULL, cols);

  /* Populate A matrix */
  for (i = 0; i < rows; i++) {  
    GRBLinExpr lhs = 0;
    for (j = 0; j < cols; j++)
      if (A[i*cols+j] != 0)
        lhs += A[i*cols+j]*vars[j];
    model.addConstr(lhs, sense[i], rhs[i]);
  }

  GRBQuadExpr obj = 0;
  for (j = 0; j < cols; j++)
obj += c[j]*vars[j];
for (i = 0; i < cols; i++)
    for (j = 0; j < cols; j++)
        if (Q[i*cols+j] != 0)
            obj += Q[i*cols+j]*vars[i]*vars[j];

model.setObjective(obj);
model.optimize();
model.write("dense.lp");

if (model.get(GRB_IntAttr_Status) == GRB_OPTIMAL) {
    *objvalP = model.get(GRB_DoubleAttr_ObjVal);
    for (i = 0; i < cols; i++)
        solution[i] = vars[i].get(GRB_DoubleAttr_X);
    success = true;
}

delete[] vars;

return success;
}

int main(int argc, char *argv[])
{
    GRBEnv* env = 0;
    try {
        env = new GRBEnv();
        double c[] = {1, 1, 0};
        double Q[3][3] = {{1, 1, 0}, {0, 1, 1}, {0, 0, 1}};
        double A[2][3] = {{1, 2, 3}, {1, 1, 0}};
        char sense[] = {'>', '>'};
        double rhs[] = {4, 1};
        double lb[] = {0, 0, 0};
        bool success;
        double objval, sol[3];

        success = dense_optimize(env, 2, 3, c, &Q[0][0], &A[0][0], sense, rhs, lb, NULL, NULL, sol, &objval);

    }
    catch(GRBException e) {
        cout << "Error code = " << e.getErrorCode() << endl;
        cout << e.getMessage() << endl;
    }
    catch(...) {
        cout << "Exception during optimization" << endl;
    }

    delete env;
    return 0;
}
/* Copyright 2020, Gurobi Optimization, LLC */

/* Solve the classic diet model, showing how to add constraints to an existing model. */

#include "gurobi_c++.h"
using namespace std;

void printSolution(GRBModel& model, int nCategories, int nFoods, GRBVar* buy, GRBVar* nutrition);

int main(int argc, char *argv[])
{
  GRBEnv* env = 0;
  GRBVar* nutrition = 0;
  GRBVar* buy = 0;
  try
  {
    // Nutrition guidelines, based on
    // USDA Dietary Guidelines for Americans, 2005
    const int nCategories = 4;
    string Categories[] =
    { "calories", "protein", "fat", "sodium" };
    double minNutrition[] = { 1800, 91, 0, 0 };,
    double maxNutrition[] = { 2200, GRB_INFINITY, 65, 1779 };;

    // Set of foods
    const int nFoods = 9;
    string Foods[] =
    { "hamburger", "chicken", "hot dog", "fries", "macaroni", "pizza", "salad", "milk", "ice cream" };;
    double cost[] =
    { 2.49, 2.89, 1.50, 1.89, 1.99, 2.49, 0.89, 1.59, 1.59 };;

    // Nutrition values for the foods
    double nutritionValues[][nCategories] =
    { { 410, 24, 26, 730 }, // hamburger
    { 420, 32, 10, 1190 }, // chicken
    { 560, 20, 32, 1800 }, // hot dog
    { 380, 4, 19, 270 }, // fries
    { 320, 12, 10, 930 }, // macaroni
    { 320, 15, 12, 820 }, // pizza
    { 320, 31, 12, 1230 }, // salad
    { 100, 8, 2.5, 125 }, // milk
    { 330, 8, 10, 180 } // ice cream
    };

    // Model
    env = new GRBEnv();
    GRBModel model = GRBModel(*env);
model.set(GRB_StringAttr_ModelName, "diet");

// Create decision variables for the nutrition information, // which we limit via bounds
nutrition = model.addVars(minNutrition, maxNutrition, 0, 0,
    Categories, nCategories);

// Create decision variables for the foods to buy
buy = model.addVars(0, 0, cost, 0, Foods, nFoods);

// The objective is to minimize the costs
model.set(GRB_IntAttr_ModelSense, GRB_MINIMIZE);

// Nutrition constraints
for (int i = 0; i < nCategories; ++i)
{
    GRBLinExpr ntot = 0;
    for (int j = 0; j < nFoods; ++j)
    {
        ntot += nutritionValues[j][i] * buy[j];
    }
    model.addConstr(ntot == nutrition[i], Categories[i]);
}

// Solve
model.optimize();
printSolution(model, nCategories, nFoods, buy, nutrition);

cout << "Adding constraint: at most 6 servings of dairy" << endl;

// Solve
model.optimize();
printSolution(model, nCategories, nFoods, buy, nutrition);

}

void printSolution(GRBModel & model, int nCategories, int nFoods,
    GRBVar* buy, GRBVar* nutrition)
{
    if (model.get(GRB_IntAttr_Status) == GRB_OPTIMAL)
cout << "\nCost: " << model.get(GRB_DoubleAttr_ObjVal) << endl;
cout << "\nBuy:
" << endl;
for (int j = 0; j < nFoods; ++j)
{
    if (buy[j].get(GRB_DoubleAttr_X) > 0.0001)
    {
        cout << buy[j].get(GRB_StringAttr_VarName) << " " <<
             buy[j].get(GRB_DoubleAttr_X) << endl;
    }
}
cout << "\nNutrition:
" << endl;
for (int i = 0; i < nCategories; ++i)
{
    cout << nutrition[i].get(GRB_StringAttr_VarName) << " " <<
         nutrition[i].get(GRB_DoubleAttr_X) << endl;
}
else
{
    cout << "No solution" << endl;
}

facility_c++.cpp

/* Copyright 2020, Gurobi Optimization, LLC */

/* Facility location: a company currently ships its product from 5 plants
   to 4 warehouses. It is considering closing some plants to reduce
   costs. What plant(s) should the company close, in order to minimize
   transportation and fixed costs?

   Based on an example from Frontline Systems:
   http://www.solver.com/disfacility.htm
   Used with permission.
   */

#include "gurobi_c++.h"
#include <sstream>
using namespace std;

int main(int argc,
         char *argv[])
{
    GRBEnv* env = 0;
    GRBVar* open = 0;
    GRBVar** transport = 0;
    int transportCt = 0;
    try
    {
        // Number of plants and warehouses
        const int nPlants = 5;
        const int nWarehouses = 4;
// Warehouse demand in thousands of units
double Demand[] = { 15, 18, 14, 20 };

// Plant capacity in thousands of units
double Capacity[] = { 20, 22, 17, 19, 18 };

// Fixed costs for each plant
double FixedCosts[] =
{ 12000, 15000, 17000, 13000, 16000 };

// Transportation costs per thousand units
double TransCosts[][nPlants] = {
    { 4000, 2000, 3000, 2500, 4500 },
    { 2500, 2600, 3400, 3000, 4000 },
    { 1200, 1800, 2600, 4100, 3000 },
    { 2200, 2600, 3100, 3700, 3200 }
};

// Model
env = new GRBEnv();
GRBModel model = GRBModel(*env);
model.set(GRB_StringAttr_ModelName, "facility");

// Plant open decision variables: open[p] == 1 if plant p is open.
open = model.addVars(nPlants, GRB_BINARY);

int p;
for (p = 0; p < nPlants; ++p)
{
    ostringstream vname;
    vname << "Open " << p;
    open[p].set(GRB_DoubleAttr_Obj, FixedCosts[p]);
    open[p].set(GRB_StringAttr_VarName, vname.str());
}

// Transportation decision variables: how much to transport from
// a plant p to a warehouse w
transport = new GRBVar* [nWarehouses];
int w;
for (w = 0; w < nWarehouses; ++w)
{
    transport[w] = model.addVars(nPlants);
    transportCt++;

    for (p = 0; p < nPlants; ++p)
    {
        ostringstream vname;
        vname << "Trans " << p << "," << w;
        transport[w][p].set(GRB_DoubleAttr_Obj, TransCosts[w][p]);
        transport[w][p].set(GRB_StringAttr_VarName, vname.str());
    }
}

// The objective is to minimize the total fixed and variable costs
model.set(GRB_IntAttr_ModelSense, GRB_MINIMIZE);
// Production constraints
// Note that the right-hand limit sets the production to zero if
// the plant is closed
for (p = 0; p < nPlants; ++p)
{
    GRBLinExpr ptot = 0;
    for (w = 0; w < nWarehouses; ++w)
    {
        ptot += transport[w][p];
    }
    ostringstream cname;
    cname << "Capacity" << p;
    model.addConstr(ptot <= Capacity[p] * open[p], cname.str());
}

// Demand constraints
for (w = 0; w < nWarehouses; ++w)
{
    GRBLinExpr dtot = 0;
    for (p = 0; p < nPlants; ++p)
    {
        dtot += transport[w][p];
    }
    ostringstream cname;
    cname << "Demand" << w;
    model.addConstr(dtot == Demand[w], cname.str());
}

// Guess at the starting point: close the plant with the highest
// fixed costs; open all others

// First, open all plants
for (p = 0; p < nPlants; ++p)
{
    open[p].set(GRB_DoubleAttr_Start, 1.0);
}

// Now close the plant with the highest fixed cost
cout << "Initial guess:" << endl;
double maxFixed = -GRB_INFINITY;
for (p = 0; p < nPlants; ++p)
{
    if (FixedCosts[p] > maxFixed)
    {
        maxFixed = FixedCosts[p];
    }
}
for (p = 0; p < nPlants; ++p)
{
    if (FixedCosts[p] == maxFixed)
    {
        open[p].set(GRB_DoubleAttr_Start, 0.0);
        cout << "Closing plant " << p << endl << endl;
        break;
    }
Use barrier to solve root relaxation
model.set(GRB_IntParam_Method, GRB_METHOD_BARRIER);

// Solve
model.optimize();

// Print solution
cout << endl;
bARRIER COSTS: " << model.get(GRB_DoubleAttr_ObjVal) << endl;
for (p = 0; p < nPlants; ++p)
{
    if (open[p].get(GRB_DoubleAttr_X) > 0.99)
        cout << "Plant " << p << " open:" << endl;
    for (w = 0; w < nWarehouses; ++w)
    {
        if (transport[w][p].get(GRB_DoubleAttr_X) > 0.0001)
            cout << " Transport " << transport[w][p].get(GRB_DoubleAttr_X) << " units to warehouse " << w << endl;
    }
}
else
{
    cout << "Plant " << p << " closed!" << endl;
}

catch (GRBException e)
{
    cout << "Error code = " << e.getErrorCode() << endl;
    cout << e.getMessage() << endl;
}
catch (...)
{
    cout << "Exception during optimization" << endl;
}

delete[] open;
for (int i = 0; i < transportCt; ++i) {
    delete[] transport[i];
}
delete[] transport;
delete env;
return 0;

feasopt_c++.cpp

/* Copyright 2020, Gurobi Optimization, LLC */
/* This example reads a MIP model from a file, adds artificial variables to each constraint, and then minimizes the sum of the artificial variables. A solution with objective zero corresponds to a feasible solution to the input model. We can also use FeasRelax feature to do it. In this example, we use minrelax=1, i.e. optimizing the returned model finds a solution that minimizes the original objective, but only from among those solutions that minimize the sum of the artificial variables. */

#include "gurobi_c++.h"
using namespace std;

int main(int argc, char *argv[])
{
  if (argc < 2)
  {
    cout << "Usage: feasopt_c++ filename" << endl;
    return 1;
  }

  GRBEnv* env = 0;
  GRBConstr* c = 0;
  try
  {
    env = new GRBEnv();
    GRBModel feasmodel = GRBModel(*env, argv[1]);

    // Create a copy to use FeasRelax feature later *
    GRBModel feasmodel1 = GRBModel(feasmodel);

    // clear objective
    feasmodel.setObjective(GRBLinExpr(0.0));

    // add slack variables
    c = feasmodel.getConstrs();
    for (int i = 0; i < feasmodel.get(GRB_IntAttr_NumConstrs); ++i)
    {
      char sense = c[i].get(GRB_CharAttr_Sense);
      if (sense != '>)
      {
        double coef = -1.0;
        feasmodel.addVar(0.0, GRB_INFINITY, 1.0, GRB_CONTINUOUS, 1,
                          &c[i], &coef, "ArtN_" +
                          c[i].get(GRB_StringAttr_ConstrName));
      }
      if (sense != '<')
      {
        double coef = 1.0;
        feasmodel.addVar(0.0, GRB_INFINITY, 1.0, GRB_CONTINUOUS, 1,
                         &c[i], &coef, "ArtP_" +
                         c[i].get(GRB_StringAttr_ConstrName));
      }
    }
  }
}
// optimize modified model
feasmodel.optimize();
feasmodel.write("feasopt.lp");

// use FeasRelax feature */
feasmodel1.feasRelax(GRB_FEASRELAX_LINEAR, true, false, true);
feasmodel1.write("feasopt1.lp");
feasmodel1.optimize();
}
catch (GRBException e)
{
    cout << "Error code = " << e.getErrorCode() << endl;
    cout << e.getMessage() << endl;
}
catch (...)
{
    cout << "Error during optimization" << endl;
}

delete[] c;
delete env;
return 0;
}

fixanddive_c++.cpp
/* Copyright 2020, Gurobi Optimization, LLC */

/* Implement a simple MIP heuristic. Relax the model,
sort variables based on fractionality, and fix the 25% of
the fractional variables that are closest to integer variables.
Repeat until either the relaxation is integer feasible or
linearly infeasible. */

#include "gurobi_c++.h"
#include <algorithm>
#include <cmath>
#include <deque>
using namespace std;

bool vcomp(GRBVar*, GRBVar*);

int main(int argc, 
    char *argv[])
{
    if (argc < 2)
    {
        cout << "Usage: fixanddive_c++ filename" << endl;
        return 1;
    }

    GRBEnv* env = 0;
    GRBVar* x = 0;
    try
    {

    )
// Read model
env = new GRBEnv();
GRBModel model = GRBModel(*env, argv[1]);

// Collect integer variables and relax them
// Note that we use GRBVar* to copy variables
deque<GRBVar*> intvars;
x = model.getVars();
for (int j = 0; j < model.get(GRB_IntAttr_NumVars); ++j)
{
    if (x[j].get(GRB_CharAttr_VType) != GRB_CONTINUOUS)
    {
        intvars.push_back(&x[j]);
        x[j].set(GRB_CharAttr_VType, GRB_CONTINUOUS);
    }
}

model.set(GRB_IntParam_OutputFlag, 0);
model.optimize();

// Perform multiple iterations. In each iteration, identify the first
// quartile of integer variables that are closest to an integer value
// in the relaxation, fix them to the nearest integer, and repeat.
for (int iter = 0; iter < 1000; ++iter)
{
    // create a list of fractional variables, sorted in order of
    // increasing distance from the relaxation solution to the nearest
    // integer value
    deque<GRBVar*> fractional;
    for (size_t j = 0; j < intvars.size(); ++j)
    {
        double sol = fabs(intvars[j]->get(GRB_DoubleAttr_X));
        if (fabs(sol - floor(sol + 0.5)) > 1e-5)
        {
            fractional.push_back(intvars[j]);
        }
    }

    cout << "Iteration " << iter << ", obj " << model.get(GRB_DoubleAttr_ObjVal) << "", fractional.size() << fractional.size() << endl;
    if (fractional.size() == 0)
    {
        cout << "Found feasible solution - objective " << model.get(GRB_DoubleAttr_ObjVal) << endl;
        break;
    }

    // Fix the first quartile to the nearest integer value
    sort(fractional.begin(), fractional.end(), vcomp);
    int nfix = (int) fractional.size() / 4;
    nfix = (nfix > 1) ? nfix : 1;
for (int i = 0; i < nfix; ++i) {
    GRBVar* v = fractional[i];
    double fixval = floor(v->get(GRB_DoubleAttr_X) + 0.5);
    v->set(GRB_DoubleAttr_LB, fixval);
    v->set(GRB_DoubleAttr_UB, fixval);
    cout << " Fix " << v->get(GRB_StringAttr_VarName) << " to " <<
         fixval << " ( rel " << v->get(GRB_DoubleAttr_X) << " )" << endl;
}

model.optimize();

// Check optimization result
if (model.get(GRB_IntAttr_Status) != GRB_OPTIMAL) {
    cout << "Relaxation is infeasible" << endl;
    break;
}
}

bool vcomp(GRBVar* v1,
            GRBVar* v2)
{
    double sol1 = fabs(v1->get(GRB_DoubleAttr_X));
    double sol2 = fabs(v2->get(GRB_DoubleAttr_X));
    double frac1 = fabs(sol1 - floor(sol1 + 0.5));
    double frac2 = fabs(sol2 - floor(sol2 + 0.5));
    return (frac1 < frac2);
}

gc_pwl_c++.cpp

/* Copyright 2020, Gurobi Optimization, LLC

This example formulates and solves the following simple model with PWL constraints:

    minimize
    subject to
     x1 + 2 x2 + x3 = 10
     x1 + x2 + 3 x3 <= 15
     x1 - x2 + x3 >= 5
     x1, x2, x3 >= 0
 */
maximize 
\[ \sum c[j] \cdot x[j] \]
subject to 
\[ \sum A[i,j] \cdot x[j] \leq 0, \text{ for } i = 0, \ldots, m-1 \]
\[ \sum y[j] \leq 3 \]
\[ y[j] = \text{pwl}(x[j]), \text{ for } j = 0, \ldots, n-1 \]
\[ x[j] \text{ free, } y[j] \geq 0, \text{ for } j = 0, \ldots, n-1 \]
where \text{pwl}(x) = 0, \text{ if } x = 0 
= 1+|x|, \text{ if } x \neq 0

Note
1. \[ \sum \text{pwl}(x[j]) \leq b \] is to bound \( x \) vector and also to favor sparse \( x \) vector.
   Here \( b = 3 \) means that at most two \( x[j] \) can be nonzero and if two, then
   \[ \sum x[j] \leq 1 \]
2. \text{pwl}(x) jumps from 1 to 0 and from 0 to 1, if \( x \) moves from negative 0 to 0,
   then to positive 0, so we need three points at \( x = 0 \). \( x \) has infinite bounds
   on both sides, the piece defined with two points (-1, 2) and (0, 1) can
   extend \( x \) to -infinite. Overall we can use five points (-1, 2), (0, 1),
   (0, 0), (0, 1) and (1, 2) to define \( y = \text{pwl}(x) \)
*/

#include "gurobi_c++.h"
#include <sstream>
using namespace std;

int main(int argc, char *argv[])
{
    int n = 5;
    int m = 5;
    double c[] = { 0.5, 0.8, 0.5, 0.1, -1 };
    double A[][5] = { {0, 0, 0, 1, -1},
                     {0, 0, 1, 1, -1},
                     {1, 1, 0, 0, -1},
                     {1, 0, 1, 0, -1},
                     {1, 0, 0, 1, -1} };
    int npts = 5;
    double xpts[] = {-1, 0, 0, 0, 1};
    double ypts[] = {2, 1, 0, 1, 2};
    GRBEnv* env = 0;
    GRBVar* x = 0;
    GRBVar* y = 0;

    try {
        // Env and model
        env = new GRBEnv();
        GRBModel model = GRBModel(*env);
        model.set(GRB_StringAttr_ModelName, "gc_pwl_c++");

        // Add variables, set bounds and obj coefficients
        x = model.addVars(n);
        for (int i = 0; i < n; i++) {
            x[i].set(GRB_DoubleAttr_LB, -GRB_INFINITY);
            x[i].set(GRB_DoubleAttr_Obj, c[i]);
        }
    }
}
y = model.addVars(n);

// Set objective to maximize
model.set(GRB_IntAttr_ModelSense, GRB_MAXIMIZE);

// Add linear constraints
for (int i = 0; i < m; i++) {
    GRBLinExpr le = 0;
    for (int j = 0; j < n; j++) {
        le += A[i][j] * x[j];
    }
    model.addConstr(le <= 0);
}

GRBLinExpr le1 = 0;
for (int j = 0; j < n; j++) {
    le1 += y[j];
}
model.addConstr(le1 <= 3);

// Add piecewise constraints
for (int j = 0; j < n; j++) {
    model.addGenConstrPWL(x[j], y[j], npts, xpts, ypts);
}

// Optimize model
model.optimize();

for (int j = 0; j < n; j++) {
    cout << "x[" << j << "] = " << x[j].get(GRB_DoubleAttr_X) << endl;
}

cout << "Obj: " << model.get(GRB_DoubleAttr_ObjVal) << endl;

} catch (GRBException e) {
    cout << "Error code = " << e.getErrorCode() << endl;
    cout << e.getMessage() << endl;
} catch (...) {
    cout << "Exception during optimization" << endl;
}

delete[] x;
delete[] y;
delete env;
return 0;

/* Copyright 2020, Gurobi Optimization, LLC

This example considers the following nonconvex nonlinear problem

162 */
maximize \quad 2x + y \\
subject to \quad \exp(x) + 4 \sqrt{y} \leq 9 \\
\quad \quad \quad \quad x, y \geq 0 \\

We show you two approaches to solve this:

1) Use a piecewise-linear approach to handle general function constraints (such as \exp \text{ and } \sqrt{\cdot}).
   a) Add two variables
      \quad u = \exp(x) \\
      \quad v = \sqrt{y} \\
   b) Compute points \((x, u)\) of \(u = \exp(x)\) for some step length (e.g., \(x = 0, 1e^{-3}, 2e^{-3}, \ldots, \text{xmax}\)) and points \((y, v)\) of \(v = \sqrt{y}\) for some step length (e.g., \(y = 0, 1e^{-3}, 2e^{-3}, \ldots, \text{ymax}\)). We need to compute \(\text{xmax}\) and \(\text{ymax}\) (which is easy for this example, but this does not hold in general).
   c) Use the points to add two general constraints of type piecewise-linear.

2) Use the Gurobi's built-in general function constraints directly (\text{EXP} and \text{POW}). Here, we do not need to compute the points and the maximal possible values, which will be done internally by Gurobi. In this approach, we show how to "zoom in" on the optimal solution and tighten tolerances to improve the solution quality.

```cpp
/*
#define defined (WIN32) || defined (WIN64)
#include <Windows.h>
#endif
#include "gurobi_c++.h"
#include <cmath>
using namespace std;

static double f(double u) { return exp(u); }
static double g(double u) { return sqrt(u); }

static void printsol(GRBModel& m, GRBVar & x, GRBVar & y, GRBVar & u, GRBVar & v)
{
    cout << "x = " << x.get(GRB_DoubleAttr_X) << ", u = " << u.get(GRB_DoubleAttr_X) << endl;
    cout << "y = " << y.get(GRB_DoubleAttr_X) << ", v = " << v.get(GRB_DoubleAttr_X) << endl;
    cout << "Obj = " << m.get(GRB_DoubleAttr_ObjVal) << endl;

    // Calculate violation of \(\exp(x) + 4 \sqrt{y} \leq 9\)
    double vio = f(x.get(GRB_DoubleAttr_X)) + 4 * g(y.get(GRB_DoubleAttr_X)) - 9;
    if (vio < 0.0) vio = 0.0;
    cout << "Vio = " << vio << endl;
}

int main(int argc, char* argv[])
{
    double* xpts = NULL;
    double* ypts = NULL;
```
```cpp
double* vpts = NULL;
double* upts = NULL;

try {
    // Create environment
    GRBEnv env = GRBEnv();
    // Create a new model
    GRBModel m = GRBModel(env);
    // Create variables
    double lb = 0.0, ub = GRB_INFINITY;
    GRBVar x = m.addVar(lb, ub, 0.0, GRB_CONTINUOUS, "x");
    GRBVar y = m.addVar(lb, ub, 0.0, GRB_CONTINUOUS, "y");
    GRBVar u = m.addVar(lb, ub, 0.0, GRB_CONTINUOUS, "u");
    GRBVar v = m.addVar(lb, ub, 0.0, GRB_CONTINUOUS, "v");
    // Set objective
    m.setObjective(2*x + y, GRB_MAXIMIZE);
    // Add linear constraint
    m.addConstr(u + 4*v <= 9, "l1");
    // Approach 1) PWL constraint approach
    double intv = 1e-3;
    double xmax = log(9.0);
    int len = (int) ceil(xmax/intv) + 1;
    xpts = new double[len];
    upts = new double[len];
    for (int i = 0; i < len; i++) {
        xpts[i] = i*intv;
        upts[i] = f(i*intv);
    }
    GRBGenConstr gc1 = m.addGenConstrPWL(x, u, len, xpts, upts, "gc1");
    double ymax = (9.0/4.0)*(9.0/4.0);
    len = (int) ceil(ymax/intv) + 1;
    ypts = new double[len];
    vpts = new double[len];
    for (int i = 0; i < len; i++) {
        ypts[i] = i*intv;
        vpts[i] = g(i*intv);
    }
    GRBGenConstr gc2 = m.addGenConstrPWL(y, v, len, ypts, vpts, "gc2");
    // Optimize the model and print solution
    m.optimize();
}
```

printsol(m, x, y, u, v);

// Approach 2) General function constraint approach with auto PWL
// translation by Gurobi

// restore unsolved state and get rid of PWL constraints
m.reset();
m.remove(gc1);
m.remove(gc2);
m.update();

GRBGenConstr gcf1 = m.addGenConstrExp(x, u, "gcf1");
GRBGenConstr gcf2 = m.addGenConstrPow(y, v, 0.5, "gcf2");
m.set(GRB_DoubleParam_FuncPieceLength, 1e-3);

// Optimize the model and print solution
m.optimize();
printsol(m, x, y, u, v);

// Zoom in, use optimal solution to reduce the ranges and use a smaller
// pclen=1e-5 to solve it

double xval = x.get(GRB_DoubleAttr_X);
double yval = y.get(GRB_DoubleAttr_X);

x.set(GRB_DoubleAttr_LB, max(x.get(GRB_DoubleAttr_LB), xval - 0.01));
x.set(GRB_DoubleAttr_UB, min(x.get(GRB_DoubleAttr_UB), xval + 0.01));
y.set(GRB_DoubleAttr_LB, max(y.get(GRB_DoubleAttr_LB), yval - 0.01));
y.set(GRB_DoubleAttr_UB, min(y.get(GRB_DoubleAttr_UB), yval + 0.01));
m.update();
m.reset();

m.set(GRB_DoubleParam_FuncPieceLength, 1e-5);

// Optimize the model and print solution
m.optimize();
printsol(m, x, y, u, v);

} catch(GRBException e) {
cout << "Error code = " << e.getErrorCode() << endl;
cout << e.getMessage() << endl;
} catch(...) {
cout << "Exception during optimization" << endl;
}

if (xpts) delete[] xpts;
if (ypts) delete[] ypts;
if (upts) delete[] upts;
if (vpts) delete[] vpts;

return 0;
}
/* Copyright 2020, Gurobi Optimization, LLC */

/* In this example we show the use of general constraints for modeling
some common expressions. We use as an example a SAT-problem where we
want to see if it is possible to satisfy at least four (or all) clauses
of the logical for

L = (x0 or -x1 or x2) and (x1 or -x2 or x3) and
(x2 or -x3 or x0) and (x3 or -x0 or x1) and
(-x0 or -x1 or x2) and (-x1 or -x2 or x3) and
(-x2 or -x3 or x0) and (-x3 or -x0 or x1)

We do this by introducing two variables for each literal (itself and its
negated value), a variable for each clause, and then two
variables for indicating if we can satisfy four, and another to identify
the minimum of the clauses (so if it one, we can satisfy all clauses)
and put these two variables in the objective.
 i.e. the Objective function will be

maximize Obj0 + Obj1

Obj0 = MIN(Clause1, ... , Clause8)
Obj1 = 1 -> Clause1 + ... + Clause8 >= 4

thus, the objective value will be two if and only if we can satisfy all
clauses; one if and only if at least four clauses can be satisfied, and
zero otherwise.
*/

#include "gurobi_c++.h"
#include <sstream>
#include <iomanip>
using namespace std;

#define n 4
#define NLITERALS 4 // same as n
#define NCLAUSES 8
#define NOBJ 2

int main(void)
{
    GRBEnv *env = 0;
    try{
        // Example data
        // e.g. {0, n+1, 2} means clause (x0 or -x1 or x2)
        const int Clauses[][3] = {
            {0, n+1, 2}, {1, n+2, 3},
            {2, n+3, 0}, {3, n+0, 1},
            {n+0, n+1, 2}, {n+1, n+2, 3},
            {n+2, n+3, 0}, {n+3, n+0, 1}};

        int i, status;
// Create environment
env = new GRBEnv("genconstr_c++.log");

// Create initial model
GRBModel model = GRBModel(*env);
model.set(GRB_StringAttr_ModelName, "genconstr_c++");

// Initialize decision variables and objective

GRBVar Lit[NLITERALS];
GRBVar NotLit[NLITERALS];
for (i = 0; i < NLITERALS; i++) {
    stringstream vname;
    vname << "X" << i;
    Lit[i] = model.addVar(0.0, 1.0, 0.0, GRB_BINARY, vname.str());
    vname.str("" );
    vname << "notX" << i;
    NotLit[i] = model.addVar(0.0, 1.0, 0.0, GRB_BINARY, vname.str());
}

GRBVar Cla[NCLAUSES];
for (i = 0; i < NCLAUSES; i++) {
    stringstream vname;
    vname << "Clause" << i;
    Cla[i] = model.addVar(0.0, 1.0, 0.0, GRB_BINARY, vname.str());
}

GRBVar Obj[NOBJ];
for (i = 0; i < NOBJ; i++) {
    stringstream vname;
    vname << "Obj" << i;
    Obj[i] = model.addVar(0.0, 1.0, 1.0, GRB_BINARY, vname.str());
}

// Link Xi and notXi
GRBLinExpr lhs;
for (i = 0; i < NLITERALS; i++) {
    stringstream cname;
    cname << "CNSTR_X" << i;
    lhs = 0;
    lhs += Lit[i];
    lhs += NotLit[i];
    model.addConstr(lhs == 1.0, cname.str());
}

// Link clauses and literals
GRBVar clause[3];
for (int i = 0; i < NCLAUSES; i++) {
    for (int j = 0; j < 3; j++) {
        if (Clauses[i][j] >= n) clause[j] = NotLit[Clauses[i][j]-n];
        else clause[j] = Lit[Clauses[i][j]];
    }
    stringstream cname;
    cname << "CNSTR_Clause" << i;
}
model.addGenConstrOr(Cla[i], clause, 3, cname.str());
}

// Link objs with clauses
model.addGenConstrMin(Obj[0], Cla, NCLAUSES, GRB_INFINITY, "CNSTR_Obj0");

lhs = 0;
for (i = 0; i < NCLAUSES; i++) {
    lhs += Cla[i];
}
model.addGenConstrIndicator(Obj[1], 1, lhs >= 4.0, "CNSTR_Obj1");

// Set global objective sense
model.set(GRB_IntAttr_ModelSense, GRB_MAXIMIZE);

// Save problem
model.write("genconstr_c++.mps");
model.write("genconstr_c++.lp");

// Optimize
model.optimize();

// Status checking
status = model.get(GRB_IntAttr_Status);

if (status == GRB_INF_OR_UNBD ||
    status == GRB_INFEASIBLE ||
    status == GRB_UNBOUNDED) {
    cout << "The model cannot be solved " <<
         "because it is infeasible or unbounded" << endl;
    return 1;
}
if (status != GRB_OPTIMAL) {
    cout << "Optimization was stopped with status " << status << endl;
    return 1;
}

// Print result
double objval = model.get(GRB_DoubleAttr_ObjVal);

if (objval > 1.9) 
    cout << "Logical expression is satisfiable" << endl;
else if (objval > 0.9) 
    cout << "At least four clauses can be satisfied" << endl;
else 
    cout << "Not even three clauses can be satisfied" << endl;

} catch (GRBException e) {
    cout << "Error code = " << e.getErrorCode() << endl;
    cout << e.getMessage() << endl;
}

} catch (...) {
    cout << "Exception during optimization" << endl;
}

// Free environment
delete env;
return 0;
}

lp_c++.cpp

/* Copyright 2020, Gurobi Optimization, LLC */

/* This example reads an LP model from a file and solves it. If the model is infeasible or unbounded, the example turns off presolve and solves the model again. If the model is infeasible, the example computes an Irreducible Inconsistent Subsystem (IIS), and writes it to a file */

#include "gurobi_c++.h"
using namespace std;

int main(int argc, char *argv[])
{
  if (argc < 2) {
    cout << " Usage : lp_c++ filename" << endl;
    return 1;
  }

  try {
    GRBEnv env = GRBEnv();
    GRBModel model = GRBModel(env, argv[1]);
    model.optimize();

    int optimstatus = model.get(GRB_IntAttr_Status);

    if (optimstatus == GRB_INF_OR_UNBD) {
      model.set(GRB_IntParam_Presolve, 0);
      model.optimize();
      optimstatus = model.get(GRB_IntAttr_Status);
    }

    if (optimstatus == GRB_OPTIMAL) {
      double objval = model.get(GRB_DoubleAttr_ObjVal);
      cout << "Optimal objective: " << objval << endl;
    } else if (optimstatus == GRB_INFEASIBLE) {
      cout << "Model is infeasible" << endl;

      // compute and write out IIS
      model.computeIIS();
      model.write("model.ilp");
    } else if (optimstatus == GRB_UNBOUNDED) {
      cout << "Model is unbounded" << endl;
    } else {
      cout << "Optimization was stopped with status = "
           << optimstatus << endl;
  }
  return 0;
}
lpmethod_c++.cpp

/* Copyright 2020, Gurobi Optimization, LLC */

/* Solve a model with different values of the Method parameter; show which value gives the shortest solve time. */

#include "gurobi_c++.h"

using namespace std;

int main(int argc, char *argv[])
{
    if (argc < 2)
    {
        cout << "Usage: lpmethod_c++ filename" << endl;
        return 1;
    }

    try {
        // Read model
        GRBEnv env = GRBEnv();
        GRBModel m = GRBModel(env, argv[1]);

        // Solve the model with different values of Method
        int bestMethod = -1;
        double bestTime = m.get(GRB_DoubleParam_TimeLimit);
        for (int i = 0; i <= 2; ++i) {
            m.reset();
            m.set(GRB_IntParam_Method, i);
            m.optimize();
            if (m.get(GRB_IntAttr_Status) == GRB_OPTIMAL) {
                bestTime = m.get(GRB_DoubleAttr_Runtime);
                bestMethod = i;
                // Reduce the TimeLimit parameter to save time
                // with other methods
                m.set(GRB_DoubleParam_TimeLimit, bestTime);
            }
        }

        // Report which method was fastest
        if (bestMethod == -1) {

```
cout << "Unable to solve this model" << endl;
} else {
    cout << "Solved in " << bestTime
    << " seconds with Method: " << bestMethod << endl;
}
}
} catch(GRBException e) {
    cout << "Error code = " << e.getErrorCode() << endl;
    cout << e.getMessage() << endl;
} catch(...) {
    cout << "Exception during optimization" << endl;
}

return 0;

lpmod_c++.cpp

/* Copyright 2020, Gurobi Optimization, LLC */

/* This example reads an LP model from a file and solves it.
   If the model can be solved, then it finds the smallest positive variable,
   sets its upper bound to zero, and resolves the model two ways:
   first with an advanced start, then without an advanced start
   (i.e. 'from scratch'). */

#include "gurobi_c++.h"
using namespace std;

int main(int argc, char *argv[])
{
    if (argc < 2)
    {
        cout << "Usage: lpmod_c++ filename" << endl;
        return 1;
    }

    GRBEnv* env = 0;
    GRBVar* v = 0;
    try
    {
        // Read model and determine whether it is an LP
        env = new GRBEnv();
        GRBModel model = GRBModel(*env, argv[1]);
        if (model.get(GRB_IntAttr_IsMIP) != 0)
        {
            cout << "The model is not a linear program" << endl;
            return 1;
        }

        model.optimize();

        int status = model.get(GRB_IntAttr_Status);
        if ((status == GRB_INF_OR_UNBD) || (status == GRB_INFEASIBLE) ||
(status == GRB_UNBOUNDED))
{
  cout << "The model cannot be solved because it is "
  << "infeasible or unbounded" << endl;
  return 1;
}

if (status != GRB_OPTIMAL)
{
  cout << "Optimization was stopped with status " << status << endl;
  return 0;
}

// Find the smallest variable value
double minVal = GRB_INFINITY;
int minVar = 0;

v = model.getVars();
for (int j = 0; j < model.get(GRB_IntAttr_NumVars); ++j)
{
  double sol = v[j].get(GRB_DoubleAttr_X);
  if ((sol > 0.0001) && (sol < minVal) &&
      (v[j].get(GRB_DoubleAttr_LB) == 0.0))
  {
    minVal = sol;
    minVar = j;
  }
}

cout << "
*** Setting " << v[minVar].get(GRB_StringAttr_VarName) << " from " << minVal << " to zero ***" << endl;

v[minVar].set(GRB_DoubleAttr_UB, 0.0);

// Solve from this starting point
model.optimize();

// Save iteration & time info
double warmCount = model.get(GRB_DoubleAttr_IterCount);
double warmTime = model.get(GRB_DoubleAttr_Runtime);

// Reset the model and resolve
cout << "
*** Resetting and solving " << endl;
model.reset();
model.optimize();

// Save iteration & time info
double coldCount = model.get(GRB_DoubleAttr_IterCount);
double coldTime = model.get(GRB_DoubleAttr_Runtime);

cout << "
*** Warm start: " << warmCount << " iterations, " << warmTime << " seconds" << endl;
cout << "*** Cold start: " << coldCount << " iterations, " << coldTime << " seconds" << endl;
}
catch (GRBException e)
```cpp
// Create an environment
GRBEnv env = GRBEnv(true);
env.set("LogFile", "mip1.log");
env.start();

// Create an empty model
GRBModel model = GRBModel(env);

// Create variables
GRBVar x = model.addVar(0.0, 1.0, 0.0, GRB_BINARY, "x");
GRBVar y = model.addVar(0.0, 1.0, 0.0, GRB_BINARY, "y");
GRBVar z = model.addVar(0.0, 1.0, 0.0, GRB_BINARY, "z");

// Set objective: maximize x + y + 2 z
model.setObjective(x + y + 2 * z, GRB_MAXIMIZE);

// Add constraint: x + 2 y + 3 z <= 4
model.addConstr(x + 2 * y + 3 * z <= 4, "c0");

// Add constraint: x + y >= 1
model.addConstr(x + y >= 1, "c1");
```

/* Copyright 2020, Gurobi Optimization, LLC */

/* This example formulates and solves the following simple MIP model:

maximize x + y + 2 z
subject to x + 2 y + 3 z <= 4
       x + y >= 1
       x, y, z binary */

#include "gurobi_c++.h"
using namespace std;

int main(int argc, char *argv[])
{
    try {

        // Create an environment
        GRBEnv env = GRBEnv(true);
        env.set("LogFile", "mip1.log");
        env.start();

        // Create an empty model
        GRBModel model = GRBModel(env);

        // Create variables
        GRBVar x = model.addVar(0.0, 1.0, 0.0, GRB_BINARY, "x");
        GRBVar y = model.addVar(0.0, 1.0, 0.0, GRB_BINARY, "y");
        GRBVar z = model.addVar(0.0, 1.0, 0.0, GRB_BINARY, "z");

        // Set objective: maximize x + y + 2 z
        model.setObjective(x + y + 2 * z, GRB_MAXIMIZE);

        // Add constraint: x + 2 y + 3 z <= 4
        model.addConstr(x + 2 * y + 3 * z <= 4, "c0");

        // Add constraint: x + y >= 1
        model.addConstr(x + y >= 1, "c1");
    }

    delete[] v;
    delete env;
    return 0;
}
```
// Optimize model
model.optimize();

cout << x.get(GRB_StringAttr_VarName) << " " << x.get(GRB_DoubleAttr_X) << endl;
cout << y.get(GRB_StringAttr_VarName) << " " << y.get(GRB_DoubleAttr_X) << endl;
cout << z.get(GRB_StringAttr_VarName) << " " << z.get(GRB_DoubleAttr_X) << endl;
cout << "Obj: " << model.get(GRB_DoubleAttr_ObjVal) << endl;
} catch(GRBException e) {
cout << "Error code = " << e.getErrorCode() << endl;
cout << e.getMessage() << endl;
} catch(...) {
cout << "Exception during optimization" << endl;
}

return 0;
}

mip2_c++.cpp

/* Copyright 2020, Gurobi Optimization, LLC */

/* This example reads a MIP model from a file, solves it and prints the objective values from all feasible solutions generated while solving the MIP. Then it creates the fixed model and solves that model. */

#include "gurobi_c++.h"
#include <cmath>
using namespace std;

int main(int argc, char *argv[])
{
  if (argc < 2) {
    cout << "Usage: mip2_c++ filename" << endl;
    return 1;
  }

  GRBEnv *env = 0;
  GRBVar *vars = 0, *fvars = 0;
  try {
    env = new GRBEnv();
    GRBModel model = GRBModel(*env, argv[1]);

    if (model.get(GRB_IntAttr_IsMIP) != 0) {
      throw GRBException("Model is not a MIP");
    }
    model.optimize();
  }
  catch(GRBException e) {
    cout << "Error code = " << e.getErrorCode() << endl;
    cout << e.getMessage() << endl;
  }
  catch(...) {
    cout << "Exception during optimization" << endl;
  }
  return 0;
}
int optimstatus = model.get(GRB_IntAttr_Status);

cout << "Optimization complete" << endl;

double objval = 0;
if (optimstatus == GRB_OPTIMAL) {
    objval = model.get(GRB_DoubleAttr_ObjVal);
    cout << "Optimal objective: " << objval << endl;
} else if (optimstatus == GRB_INF_OR_UNBD) {
    cout << "Model is infeasible or unbounded" << endl;
    return 0;
} else if (optimstatus == GRB_INFEASIBLE) {
    cout << "Model is infeasible" << endl;
    return 0;
} else if (optimstatus == GRB_UNBOUNDED) {
    cout << "Model is unbounded" << endl;
    return 0;
} else {
    cout << "Optimization was stopped with status = " << optimstatus << endl;
    return 0;
}

/* Iterate over the solutions and compute the objectives */

int numvars = model.get(GRB_IntAttr_NumVars);
vars = model.getVars();
model.set(GRB_IntParam_OutputFlag, 0);

cout << endl;
for (int k = 0; k < model.get(GRB_IntAttr_SolCount); ++k) {
    model.set(GRB_IntParam_SolutionNumber, k);
    double objn = 0.0;

    for (int j = 0; j < numvars; j++) {
        GRBVar v = vars[j];
        objn += v.get(GRB_DoubleAttr_Obj) * v.get(GRB_DoubleAttr_Xn);
    }

    cout << "Solution " << k << " has objective: " << objn << endl;
}
cout << endl;
model.set(GRB_IntParam_OutputFlag, 1);

/* Create a fixed model, turn off presolve and solve */

GRBModel fixed = model.fixedModel();
fixed.set(GRB_IntParam_Presolve, 0);
fixed.optimize();

int foptimstatus = fixed.get(GRB_IntAttr_Status);

if (foptimstatus != GRB_OPTIMAL) {
    cerr << "Error: fixed model isn’t optimal" << endl;
}
return 0;

double fobjval = fixed.getObjectiveValue();

if (fabs(fobjval - objval) > 1.0e-6 * (1.0 + fabs(objval))) {
    cerr << "Error: objective values are different" << endl;
    return 0;
}

/* Print values of nonzero variables */
for (int j = 0; j < numVars; j++) {
    GRBVar v = fvars[j];
    if (v.get(GRB_DoubleAttr_X) != 0.0) {
        cout << v.get(GRB_StringAttr_VarName) << " "
             << v.get(GRB_DoubleAttr_X) << endl;
    }
}

catch(GRBException e) {
    cout << "Error code = " << e.getErrorCode() << endl;
    cout << e.getMessage() << endl;
}
catch(...) {
    cout << "Error during optimization" << endl;
}

delete[] fvars;
delete[] vars;
delete env;
return 0;
}

multiobj_c++.cpp

/* Copyright 2020, Gurobi Optimization, LLC */

/* Want to cover three different sets but subject to a common budget of
* elements allowed to be used. However, the sets have different priorities to
* be covered; and we tackle this by using multi-objective optimization. */

#include "gurobi_c++.h"
#include <sstream>
#include <iomanip>
using namespace std;

int main(void)
{
    GRBEnv *env = 0;
    GRBVar *Elem = 0;
    int e, i, status, nSolutions;

    try{
        // Sample data
        const int groundSetSize = 20;
const int nSubsets = 4;
const int Budget = 12;

double Set[][20] =
{ { 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0 },
  { 0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1 },
  { 0, 0, 0, 1, 1, 0, 1, 1, 0, 0, 0, 0, 1, 1, 0, 1, 1, 0, 0, 0 },
  { 0, 0, 0, 1, 1, 1, 0, 0, 0, 1, 1, 1, 0, 0, 0, 1, 1, 1, 0, 0 } };

int SetObjPriority[] = {3, 2, 2, 1};
double SetObjWeight[] = {1.0, 0.25, 1.25, 1.0};

// Create environment
env = new GRBEnv("multiobj_c++.log");

// Create initial model
GRBModel model = GRBModel(*env);
model.set(GRB_StringAttr_ModelName, "multiobj_c++");

// Initialize decision variables for ground set:
// x[e] == 1 if element e is chosen for the covering.
Elem = model.addVars(groundsetSize, GRB_BINARY);
for (e = 0; e < groundsetSize; e++) {
  ostringstream vname;
  vname << "El" << e;
  Elem[e].set(GRB_StringAttr_VarName, vname.str());
}

// Constraint: limit total number of elements to be picked to be at most
// Budget
GRBLinExpr lhs;
lhs = 0;
for (e = 0; e < groundsetSize; e++) {
  lhs += Elem[e];
}
model.addConstr(lhs <= Budget, "Budget");

// Set global sense for ALL objectives
model.set(GRB_IntAttr_ModelSense, GRB_MAXIMIZE);

// Limit how many solutions to collect
model.set(GRB_IntParam_PoolSolutions, 100);

// Set and configure i-th objective
for (i = 0; i < nSubsets; i++) {
  GRBLinExpr objn = 0;
  for (e = 0; e < groundsetSize; e++)
    objn += Set[i][e]*Elem[e];
  ostringstream vname;
  vname << "Set" << i;
  model.setObjectiveN(objn, i, SetObjPriority[i], SetObjWeight[i],
                      1.0 + i, 0.01, vname.str());
}

// Save problem
model.write("multiobj_c++.lp");
// Optimize
model.optimize();

// Status checking
status = model.get(GRB_IntAttr_Status);

if (status == GRB_INF_OR_UNBD ||
    status == GRB_INFEASIBLE ||
    status == GRB_UNBOUNDED ) {
    cout << "The model cannot be solved " <<
    "because it is infeasible or unbounded" << endl;
    return 1;
}
if (status != GRB_OPTIMAL) {
    cout << "Optimization was stopped with status " << status << endl;
    return 1;
}

// Print best selected set
for (e = 0; e < groundSetSize; e++) {
    if (Elem[e].get(GRB_DoubleAttr_X) < .9) continue;
    cout << " El" << e;
}

// Print number of solutions stored
nSolutions = model.get(GRB_IntAttr_SolCount);

// Print objective values of solutions
if (nSolutions > 10) nSolutions = 10;
for (i = 0; i < nSubsets; i++) {
    model.set(GRB_IntParam_ObjNumber, i);
    cout << "\tSet" << i;
    for (e = 0; e < nSolutions; e++) {
        cout << " ";
        model.set(GRB_IntParam_SolutionNumber, e);
        double val = model.get(GRB_DoubleAttr_ObjNVal);
        cout << std::setw(6) << val;
    }
    cout << endl;
}

catch (GRBException e) {
    cout << "Error code = " << e.getErrorCode() << endl;
    cout << e.getMessage() << endl;
}
catch (...){
    cout << "Exception during optimization" << endl;
}
// Free environment/vars
delete[] Elem;
delete env;
return 0;
}

multiscenario_c++.cpp

// Copyright 2020, Gurobi Optimization, LLC

// Facility location: a company currently ships its product from 5 plants
// to 4 warehouses. It is considering closing some plants to reduce
// costs. What plant(s) should the company close, in order to minimize
// transportation and fixed costs?
//
// Since the plant fixed costs and the warehouse demands are uncertain, a
// scenario approach is chosen.
//
// Note that this example is similar to the facility_c++.cpp example. Here
// we added scenarios in order to illustrate the multi-scenario feature.
//
// Based on an example from Frontline Systems:
// http://www.solver.com/disfacility.htm
// Used with permission.

#include "gurobi_c++.h"
#include <sstream>
#include <iomanip>
using namespace std;

int main(int argc, char *argv[])
{
    GRBEnv *env = 0;
    GRBVar *open = 0;
    GRBVar **transport = 0;
    GRBConstr *demandConstr = 0;

    int transportCt = 0;

    try {
        // Number of plants and warehouses
        const int nPlants = 5;
        const int nWarehouses = 4;

        // Warehouse demand in thousands of units
        double Demand[] = { 15, 18, 14, 20 };

        // Plant capacity in thousands of units
        double Capacity[] = { 20, 22, 17, 19, 18 };

        // Fixed costs for each plant
        double FixedCosts[] =
            { 12000, 15000, 17000, 13000, 16000 };
// Transportation costs per thousand units
double TransCosts[][nPlants] = {
    { 4000, 2000, 3000, 2500, 4500 },
    { 2500, 2600, 3400, 3000, 4000 },
    { 1200, 1800, 2600, 4100, 3000 },
    { 2200, 2600, 3100, 3700, 3200 }
};

double maxFixed = -GRB_INFINITY;
double minFixed = GRB_INFINITY;

int p;
for (p = 0; p < nPlants; p++) {
    if (FixedCosts[p] > maxFixed)
        maxFixed = FixedCosts[p];

    if (FixedCosts[p] < minFixed)
        minFixed = FixedCosts[p];
}

// Model
env = new GRBEnv();
GRBModel model = GRBModel(*env);
model.set(GRB_StringAttr_ModelName, "multiscenario");

// Plant open decision variables: open[p] == 1 if plant p is open.
on = model.addVars(nPlants, GRB_BINARY);

    for (p = 0; p < nPlants; p++) {
        ostringstream vname;
        vname << "Open" << p;
        open[p].set(GRB_DoubleAttr_Obj, FixedCosts[p]);
        open[p].set(GRB_StringAttr_VarName, vname.str());
    }

// Transportation decision variables: how much to transport from
// a plant p to a warehouse w
transport = new GRBVar* [nWarehouses];

    for (w = 0; w < nWarehouses; w++) {
        transport[w] = model.addVars(nPlants);
        transportCt++;

        for (p = 0; p < nPlants; p++) {
            ostringstream vname;
            vname << "Trans" << p << "." << w;
            transport[w][p].set(GRB_DoubleAttr_Obj, TransCosts[w][p]);
            transport[w][p].set(GRB_StringAttr_VarName, vname.str());
        }
    }

    // The objective is to minimize the total fixed and variable costs
    model.set(GRB_IntAttr_ModelSense, GRB_MINIMIZE);
    // Production constraints
    // Note that the right-hand limit sets the production to zero if
// the plant is closed
for (p = 0; p < nPlants; p++) {
    GRBLinExpr ptot = 0;
    for (w = 0; w < nWarehouses; w++) {
        ptot += transport[w][p];
    }
    ostringstream cname;
    cname << "Capacity " << p;
    model.addConstr(ptot <= Capacity[p] * open[p], cname.str());
}

// Demand constraints
demandConstr = new GRBConstr[nWarehouses];
for (w = 0; w < nWarehouses; w++) {
    GRBLinExpr dtot = 0;
    for (p = 0; p < nPlants; p++)
        dtot += transport[w][p];
    ostringstream cname;
    cname << "Demand " << w;
    demandConstr[w] = model.addConstr(dtot == Demand[w], cname.str());
}

// We constructed the base model, now we add 7 scenarios
//
// Scenario 0: Represents the base model, hence, no manipulations.
// Scenario 1: Manipulate the warehouses demands slightly (constraint right
// hand sides).
// Scenario 2: Double the warehouses demands (constraint right hand sides).
// Scenario 3: Manipulate the plant fixed costs (objective coefficients).
// Scenario 4: Manipulate the warehouses demands and fixed costs.
// Scenario 5: Force the plant with the largest fixed cost to stay open
// (variable bounds).
// Scenario 6: Force the plant with the smallest fixed cost to be closed
// (variable bounds).
model.set(GRB_IntAttr_NumScenarios, 7);

// Scenario 0: Base model, hence, nothing to do except giving the
// scenario a name
model.set(GRB_IntParam_ScenarioNumber, 0);
model.set(GRB_StringAttr_ScenNName, "Base model");

// Scenario 1: Increase the warehouse demands by 10%
model.set(GRB_IntParam_ScenarioNumber, 1);
model.set(GRB_StringAttr_ScenNName, "Increased warehouse demands");
for (w = 0; w < nWarehouses; w++)
    demandConstr[w].set(GRB_DoubleAttr_ScenNRHS, Demand[w] * 1.1);

// Scenario 2: Double the warehouse demands
model.set(GRB_IntParam_ScenarioNumber, 2);
model.set(GRB_StringAttr_ScenNName, "Double the warehouse demands");
for (w = 0; w < nWarehouses; w++)

181
demandConstr[w].set(GRB_DoubleAttr_ScenNRHS, Demand[w] * 2.0);
}

// Scenario 3: Decrease the plant fixed costs by 5%
model.set(GRB_IntParam_ScenarioNumber, 3);
model.set(GRB_StringAttr_ScenNName, "Decreased plant fixed costs");

for (p = 0; p < nPlants; p++) {
    open[p].set(GRB_DoubleAttr_ScenNObj, FixedCosts[p] * 0.95);
}

// Scenario 4: Combine scenario 1 and scenario 3 */
model.set(GRB_IntParam_ScenarioNumber, 4);
model.set(GRB_StringAttr_ScenNName, "Increased warehouse demands and decreased plant fixed costs");

for (w = 0; w < nWarehouses; w++) {
    demandConstr[w].set(GRB_DoubleAttr_ScenNRHS, Demand[w] * 1.1);
}
for (p = 0; p < nPlants; p++) {
    open[p].set(GRB_DoubleAttr_ScenNObj, FixedCosts[p] * 0.95);
}

// Scenario 5: Force the plant with the largest fixed cost to stay open
//
model.set(GRB_IntParam_ScenarioNumber, 5);
model.set(GRB_StringAttr_ScenNName, "Force plant with largest fixed cost to stay open");

for (p = 0; p < nPlants; p++) {
    if (FixedCosts[p] == maxFixed) {
        open[p].set(GRB_DoubleAttr_ScenNLB, 1.0);
        break;
    }
}

// Scenario 6: Force the plant with the smallest fixed cost to be closed
//
model.set(GRB_IntParam_ScenarioNumber, 6);
model.set(GRB_StringAttr_ScenNName, "Force plant with smallest fixed cost to be closed");

for (p = 0; p < nPlants; p++) {
    if (FixedCosts[p] == minFixed) {
        open[p].set(GRB_DoubleAttr_ScenNUB, 0.0);
        break;
    }
}

// Guess at the starting point: close the plant with the highest fixed costs; open all others

// First, open all plants
for (p = 0; p < nPlants; p++)
    open[p].set(GRB_DoubleAttr_Start, 1.0);

// Now close the plant with the highest fixed cost
cout << "Initial guess:"
for (p = 0; p < nPlants; p++) {
if (FixedCosts[p] == maxFixed) {
  open[p].set(GRB_DoubleAttr_Start, 0.0);
  cout << "Closing plant " << p << endl << endl;
  break;
}
}

// Use barrier to solve root relaxation
model.set(GRB_IntParam_Method, GRB_METHOD_BARRIER);

// Solve multi-scenario model
model.optimize();

int nScenarios = model.get(GRB_IntAttr_NumScenarios);

// Print solution for each */
for (int s = 0; s < nScenarios; s++) {
  int modelSense = GRB_MINIMIZE;

  // Set the scenario number to query the information for this scenario
  model.set(GRB_IntParam_ScenarioNumber, s);

  // collect result for the scenario
  double scenNObjBound = model.get(GRB_DoubleAttr_ScenNObjBound);
  double scenNObjVal = model.get(GRB_DoubleAttr_ScenNObjVal);
  cout << endl << endl << "------ Scenario " << s << " (" << model.get(GRB_StringAttr_ScenNName) << ")" << endl;

  // Check if we found a feasible solution for this scenario
  if (scenNObjVal >= modelSense * GRB_INFINITY)
    if (scenNObjBound >= modelSense * GRB_INFINITY)
      // Scenario was proven to be infeasible
      cout << endl << "INFEASIBLE" << endl;
    else
      // We did not find any feasible solution - should not happen in
      // this case, because we did not set any limit (like a time
      // limit) on the optimization process
      cout << endl << "NO SOLUTION" << endl;
  else {
    cout << endl << "TOTAL COSTS: " << scenNObjVal << endl;
    cout << "SOLUTION: " << endl;
    for (p = 0; p < nPlants; p++) {
      double scenNX = open[p].get(GRB_DoubleAttr_ScenNX);
      if (scenNX > 0.5) {
        cout << "Plant " << p << " open" << endl;
        for (w = 0; w < nWarehouses; w++) {
          scenNX = transport[w][p].get(GRB_DoubleAttr_ScenNX);
          if (scenNX > 0.0001)
            cout << " Transport " << scenNX
                  << " units to warehouse " << w << endl;
        }
      } else
        cout << "Plant " << p << " closed!" << endl;
  }
// Print a summary table: for each scenario we add a single summary
// line
cout << endl << endl << "Summary: Closed plants depending on scenario" << endl << endl;
cout << setw(8) << " | " << setw(17) << " Plant " << setw(14) << "|" << endl;
cout << setw(8) << " Scenario " << "|";
for (p = 0; p < nPlants; p++)
  cout << " | " << setw(5) << p;
cout << " | " << setw(6) << " Costs " << " Name " << endl;
for (int s = 0; s < nScenarios; s++) {
  int modelSense = GRB_MINIMIZE;

  // Set the scenario number to query the information for this scenario
  model.set(GRB_IntParam_ScenarioNumber, s);

  // Collect result for the scenario
  double scenNObjBound = model.get(GRB_DoubleAttr_ScenNObjBound);
  double scenNObjVal = model.get(GRB_DoubleAttr_ScenNObjVal);
  cout << left << setw(8) << s << right << " |";
  // Check if we found a feasible solution for this scenario
  if (scenNObjVal >= modelSense * GRB_INFINITY) {
    if (scenNObjBound >= modelSense * GRB_INFINITY)
      // Scenario was proven to be infeasible
      cout << " | " << left << setw(30) << " infeasible " << right;
    else
      // We did not find any feasible solution - should not happen in
      // this case, because we did not set any limit (like a time
      // limit) on the optimization process
      cout << " | " << left << setw(30) << " no solution found " << right;
  } else {
    for (p = 0; p < nPlants; p++) {
      double scenNX = open[p].get(GRB_DoubleAttr_ScenNX);
      if (scenNX > 0.5)
        cout << setw(6) << "|" << scenNObjVal
             << " | " << model.get(GRB_StringAttr_ScenNName)
             << endl;
    }
  }
}
catch (GRBException e) {
cout << "Error code = " << e.getErrorCode() << endl;
cout << e.getMessage() << endl;
}
catch (...) {
cout << "Exception during optimization" << endl;
}
delete[] open;
for (int i = 0; i < transportCt; ++i) {
delete[] transport[i];
}
delete[] transport;
delete[] demandConstr;
delete env;
return 0;
}

params_c++.cpp

/* Copyright 2020, Gurobi Optimization, LLC */

/* Use parameters that are associated with a model.

A MIP is solved for a few seconds with different sets of parameters. 
The one with the smallest MIP gap is selected, and the optimization is resumed until the optimal solution is found.
*/

#include "gurobi_c++.h"
using namespace std;

int main(int argc, char *argv[])
{
if (argc < 2)
{
cout << "Usage: params_c++ filename" << endl;
return 1;
}

GRBEnv* env = 0;
GRBModel *bestModel = 0, *m = 0;
try
{
// Read model and verify that it is a MIP
env = new GRBEnv();
m = new GRBModel(*env, argv[1]);
if (m->get(GRB_IntAttr_IsMIP) == 0)
{
cout << "The model is not an integer program" << endl;
return 1;
}

// Set a 2 second time limit
m->set(GRB_DoubleParam_TimeLimit, 2);
// Now solve the model with different values of MIPFocus
bestModel = new GRBModel(*m);
bestModel->optimize();
for (int i = 1; i <= 3; ++i)
{
    m->reset();
    m->set(GRB_IntParam_MIPFocus, i);
    m->optimize();
    if (bestModel->get(GRB_DoubleAttr_MIPGap) >
        m->get(GRB_DoubleAttr_MIPGap))
    {
        swap(bestModel, m);
    }
}

// Finally, delete the extra model, reset the time limit and
// continue to solve the best model to optimality
delete m;
bestModel->set(GRB_DoubleParam_TimeLimit, GRB_INFINITY);
bestModel->optimize();
cout << "Solved with MIPFocus: " << bestModel->get(GRB_IntParam_MIPFocus) << endl;

} catch (GRBException e)
{
    cout << "Error code = " << e.getErrorCode() << endl;
    cout << e.getMessage() << endl;
} catch (...)
{
    cout << "Error during optimization" << endl;
}

delete bestModel;
delete m;
delete env;
return 0;

}
into a MIP by negating the approximation for \( f \), which corresponds to a non-convex piecewise-linear function, and solves it again.

```c++
#include "gurobi_c++.h"
#include <cmath>
using namespace std;

double f(double u) { return exp(-u); }
double g(double u) { return 2 * u * u - 4 * u; }

int main(int argc, char *argv[])
{
    double *ptu = NULL;
    double *ptf = NULL;
    double *ptg = NULL;

    try {

        // Create environment
        GRBEnv env = GRBEnv();

        // Create a new model
        GRBModel model = GRBModel(env);

        // Create variables
        double lb = 0.0, ub = 1.0;
        GRBVar x = model.addVar(lb, ub, 0.0, GRB_CONTINUOUS, "x");
        GRBVar y = model.addVar(lb, ub, 0.0, GRB_CONTINUOUS, "y");
        GRBVar z = model.addVar(lb, ub, 0.0, GRB_CONTINUOUS, "z");

        // Set objective for y
        model.setObjective(-y);

        // Add piecewise-linear objective functions for x and z
        int npts = 101;
        ptu = new double[npts];
        ptf = new double[npts];
        ptg = new double[npts];

        for (int i = 0; i < npts; i++) {
            ptu[i] = lb + (ub - lb) * i / (npts - 1);
            ptf[i] = f(ptu[i]);
            ptg[i] = g(ptu[i]);
        }

        model.setPWLObj(x, npts, ptu, ptf);
        model.setPWLObj(z, npts, ptu, ptg);
    }
```
// Add constraint: x + 2 y + 3 z <= 4
model.addConstr(x + 2 * y + 3 * z <= 4, "c0");

// Add constraint: x + y >= 1
model.addConstr(x + y >= 1, "c1");

// Optimize model as an LP
model.optimize();

cout << " IsMIP: " << model.get(GRB_IntAttr_IsMIP) << endl;

cout << x.get(GRB_StringAttr_VarName) << " "
   << x.get(GRB_DoubleAttr_X) << endl;
cout << y.get(GRB_StringAttr_VarName) << " "
   << y.get(GRB_DoubleAttr_X) << endl;
cout << z.get(GRB_StringAttr_VarName) << " "
   << z.get(GRB_DoubleAttr_X) << endl;

cout << " Obj: " << model.get(GRB_DoubleAttr_ObjVal) << endl;

cout << endl;

// Negate piecewise-linear objective function for x
for (int i = 0; i < npts; i++) {
    ptf[i] = -ptf[i];
}
model.setPWLObj(x, npts, ptu, ptf);

// Optimize model as a MIP
model.optimize();

cout << " IsMIP: " << model.get(GRB_IntAttr_IsMIP) << endl;

cout << x.get(GRB_StringAttr_VarName) << " "
   << x.get(GRB_DoubleAttr_X) << endl;
cout << y.get(GRB_StringAttr_VarName) << " "
   << y.get(GRB_DoubleAttr_X) << endl;
cout << z.get(GRB_StringAttr_VarName) << " "
   << z.get(GRB_DoubleAttr_X) << endl;

cout << " Obj: " << model.get(GRB_DoubleAttr_ObjVal) << endl;
}
catch(GRBException e) {
    cout << " Error code = " << e.getErrorCode() << endl;
    cout << e.getMessage() << endl;
}
catch(...) {  
    cout << " Exception during optimization " << endl;
}
/* Copyright 2020, Gurobi Optimization, LLC */

/* We find alternative epsilon-optimal solutions to a given knapsack
 * problem by using PoolSearchMode */

#include "gurobi_c++.h"
#include <sstream>
#include <iomanip>
using namespace std;

int main(void) {
    GRBEnv *env = 0;
    GRBVar *Elem = 0;
    int e, status, nSolutions;

    try {
        // Sample data
        const int groundSetSize = 10;
        double objCoef[10] = {32, 32, 15, 15, 6, 6, 1, 1, 1, 1};
        double knapsackCoef[10] = {16, 16, 8, 8, 4, 4, 2, 2, 1, 1};
        double Budget = 33;

        // Create environment
        env = new GRBEnv("poolsearch_c++.log");

        // Create initial model
        GRBModel model = GRBModel(*env);
        model.set(GRB_StringAttr_ModelName, "poolsearch_c++");

        // Initialize decision variables for ground set:
        // x[e] == 1 if element e is chosen
        Elem = model.addVars(groundSetSize, GRB_BINARY);
        model.set(GRB_DoubleAttr_Obj, Elem, objCoef, groundSetSize);

        for (e = 0; e < groundSetSize; e++) {
            ostringstream vname;
            vname << "El" << e;
            Elem[e].set(GRB_StringAttr_VarName, vname.str());
        }

        // Constraint: limit total number of elements to be picked to be at most
        // Budget
        GRBLinExpr lhs;
lhs = 0;
for (e = 0; e < groundSetSize; e++) {
    lhs += Elem[e] * knapsackCoef[e];
}  
model.addConstr(lhs <= Budget, "Budget");

// set global sense for ALL objectives
model.set(GRB_IntAttr_ModelSense, GRB_MAXIMIZE);

// Limit how many solutions to collect
model.set(GRB_IntParam_PoolSolutions, 1024);

// Limit the search space by setting a gap for the worst possible solution that will be accepted
model.set(GRB_DoubleParam_PoolGap, 0.10);

// do a systematic search for the k-best solutions
model.set(GRB_IntParam_PoolSearchMode, 2);

// save problem
model.write("poolsearch_c++.lp");

// Optimize
model.optimize();

// Status checking
status = model.get(GRB_IntAttr_Status);

if (status == GRB_INF_OR_UNBD ||
    status == GRB_INFEASIBLE ||
    status == GRB_UNBOUNDED) {
    cout << "The model cannot be solved " <<
         "because it is infeasible or unbounded" << endl;
    return 1;
}
if (status != GRB_OPTIMAL) {
    cout << "Optimization was stopped with status " << status << endl;
    return 1;
}

// Print best selected set
cout << "Selected elements in best solution:" << endl << "\t";
for (e = 0; e < groundSetSize; e++) {
    if (Elem[e].get(GRB_DoubleAttr_X) < .9) continue;
    cout << " El" << e;
}  
cout << endl;

// Print number of solutions stored
nSolutions = model.get(GRB_IntAttr_SolCount);
cout << "Number of solutions found: " << nSolutions << endl;

// Print objective values of solutions
for (e = 0; e < nSolutions; e++) {
    model.set(GRB_IntParam_SolutionNumber, e);
    cout << model.get(GRB_DoubleAttr_PoolObjVal) << " ";
    if (e%15 == 14) cout << endl;
// print fourth best set if available
if (nSolutions >= 4) {
    model.set(GRB_IntParam_SolutionNumber, 3);

    cout << "Selected elements in fourth best solution: " << endl << "\t";
    for (e = 0; e < groundSetSize; e++) {
        if (Elem[e].get(GRB_DoubleAttr_Xn) < .9) continue;
        cout << " El" << e;
    }
    cout << endl;
}

} catch (GRBException e) {
    cout << "Error code = " << e.getErrorCode() << endl;
    cout << e.getMessage() << endl;
} catch (...) {
    cout << "Exception during optimization" << endl;
}

// Free environment/vars
delete[] Elem;
delete env;
return 0;

sensitivity_c++.cpp

// Copyright 2020, Gurobi Optimization, LLC

// A simple sensitivity analysis example which reads a MIP model from a
// file and solves it. Then uses the scenario feature to analyze the impact
// w.r.t. the objective function of each binary variable if it is set to
// 1-X, where X is its value in the optimal solution.
//
// Usage:
// sensitivity_c++ <model filename>

#include "gurobi_c++.h"
using namespace std;

// Maximum number of scenarios to be considered
#define MAXSCENARIOS 100

int main(int argc,
    char *argv[])
{
    if (argc < 2) {
        cout << "Usage: sensitivity_c++ filename" << endl;
        return 1;
    }
GRBVar *vars = NULL;
double *origX = NULL;

try {
  // Create environment
  GRBEnv env = GRBEnv();

  // Read model
  GRBModel model = GRBModel(env, argv[1]);

  int scenarios;
  if (model.get(GRB_IntAttr_IsMIP) == 0) {
    cout << "Model is not a MIP" << endl;
    return 1;
  }

  // Solve model
  model.optimize();

  if (model.get(GRB_IntAttr_Status) != GRB_OPTIMAL) {
    cout << "Optimization ended with status "
    << model.get(GRB_IntAttr_Status) << endl;
    return 1;
  }

  // Store the optimal solution
  double origObjVal = model.get(GRB_DoubleAttr_ObjVal);
  vars = model.getVars();
  int numVars = model.get(GRB_IntAttr_NumVars);
  origX = model.get(GRB_DoubleAttr_X, vars, numVars);

  scenarios = 0;

  // Count number of unfixed, binary variables in model. For each we
  // create a scenario.
  for (int i = 0; i < numVars; i++) {
    GRBVar v = vars[i];
    char vType = v.get(GRB_CharAttr_VType);

    if (v.get(GRB_DoubleAttr_LB) == 0.0)
      if (v.get(GRB_DoubleAttr_UB) == 1.0)
        if (vType == GRB_BINARY || vType == GRB_INTEGER)
          scenarios++;

    if (scenarios >= MAXSCENARIOS)
      break;
  }

  cout << "### construct multi-scenario model with "
  << scenarios << " scenarios" << endl;

  // Set the number of scenarios in the model */
  model.set(GRB_IntAttr_NumScenarios, scenarios);
scenarios = 0;

// Create a (single) scenario model by iterating through unfixed binary
// variables in the model and create for each of these variables a
// scenario by fixing the variable to 1-X, where X is its value in the
// computed optimal solution
for (int i = 0; i < numVars; i++) {
    GRBVar v = vars[i];
    char vType = v.get(GRB_CharAttr_VType);

    if (v.get(GRB_DoubleAttr_LB) == 0.0 &&
        v.get(GRB_DoubleAttr_UB) == 1.0 &&
        (vType == GRB_BINARY || vType == GRB_INTEGER) &&
        scenarios < MAXSCENARIOS) {
        // Set ScenarioNumber parameter to select the corresponding
        // scenario for adjustments
        model.set(GRB_IntParam_ScenarioNumber, scenarios);

        // Set variable to 1-X, where X is its value in the optimal solution */
        if (origX[i] < 0.5)
            v.set(GRB_DoubleAttr_ScenNLB, 1.0);
        else
            v.set(GRB_DoubleAttr_ScenNUB, 0.0);

        scenarios++;
    } else {
        // Add MIP start for all other variables using the optimal solution
        // of the base model
        v.set(GRB_DoubleAttr_Start, origX[i]);
    }
}

// Solve multi-scenario model
model.optimize();

// In case we solved the scenario model to optimality capture the
// sensitivity information
if (model.get(GRB_IntAttr_Status) == GRB_OPTIMAL) {

    // get the model sense (minimization or maximization)
    int modelSense = model.get(GRB_IntAttr_ModelSense);

    scenarios = 0;

    for (int i = 0; i < numVars; i++) {
        GRBVar v = vars[i];
        char vType = v.get(GRB_CharAttr_VType);

        if (v.get(GRB_DoubleAttr_LB) == 0.0 &&
            v.get(GRB_DoubleAttr_UB) == 1.0 &&
            (vType == GRB_BINARY || vType == GRB_INTEGER) ) {
            // Set scenario parameter to collect the objective value of the
            // corresponding scenario
        }
    }
}
model.set(GRB_IntParam_ScenarioNumber, scenarios);

// Collect objective value and bound for the scenario
double scenarioObjVal = model.get(GRB_DoubleAttr_ScenNObjVal);
double scenarioObjBound = model.get(GRB_DoubleAttr_ScenNObjBound);

cout << "Objective sensitivity for variable "
<< v.get(GRB_StringAttr_VarName)
<< " is ";

// Check if we found a feasible solution for this scenario
if (scenarioObjVal >= modelSense * GRB_INFINITY) {
    // Check if the scenario is infeasible
    if (scenarioObjBound >= modelSense * GRB_INFINITY)
        cout << "infeasible" << endl;
    else
        cout << "unknown (no solution available)" << endl;
} else {
    // Scenario is feasible and a solution is available
    cout << modelSense * (scenarioObjVal - origObjVal) << endl;
}

scenarios ++;

if (scenarios >= MAXSCENARIOS)
    break;

} catch (GRBException e) {
    cout << "Error code = " << e.getErrorCode() << endl;
    cout << e.getMessage() << endl;
} catch (...) {
    cout << "Error during optimization" << endl;
}

delete[] vars;
delete[] origX;

return 0;

qcp_c++.cpp

/* Copyright 2020, Gurobi Optimization, LLC */

/* This example formulates and solves the following simple QCP model:

maximize x
subject to x + y + z = 1
    x^2 + y^2 <= z^2 (second-order cone)
    x^2 <= yz (rotated second-order cone)
    x, y, z non-negative
*/

#include "gurobi_c++.h"
using namespace std;

int main(int argc, char *argv[]) {
    try {
        GRBEnv env = GRBEnv();
        GRBModel model = GRBModel(env);

        // Create variables
        GRBVar x = model.addVar(0.0, GRB_INFINITY, 0.0, GRB_CONTINUOUS, "x");
        GRBVar y = model.addVar(0.0, GRB_INFINITY, 0.0, GRB_CONTINUOUS, "y");
        GRBVar z = model.addVar(0.0, GRB_INFINITY, 0.0, GRB_CONTINUOUS, "z");

        // Set objective
        GRBLinExpr obj = x;
        model.setObjective(obj, GRB_MAXIMIZE);

        // Add linear constraint: x + y + z = 1
        model.addConstr(x + y + z == 1, "c0");

        // Add second-order cone: x^2 + y^2 <= z^2
        model.addQConstr(x*x + y*y <= z*z, "qc0");

        // Add rotated cone: x^2 <= yz
        model.addQConstr(x*x <= y*z, "qc1");

        // Optimize model
        model.optimize();

        cout << x.get(GRB_StringAttr_VarName) << " " << x.get(GRB_DoubleAttr_X) << endl;
        cout << y.get(GRB_StringAttr_VarName) << " " << y.get(GRB_DoubleAttr_X) << endl;
        cout << z.get(GRB_StringAttr_VarName) << " " << z.get(GRB_DoubleAttr_X) << endl;

        cout << "Obj: " << model.get(GRB_DoubleAttr_ObjVal) << endl;
    } catch(GRBException e) {
        cout << "Error code = " << e.getErrorCode() << endl;
        cout << e.getMessage() << endl;
    } catch(...) {
        cout << "Exception during optimization" << endl;
    }

    return 0;
}
qp_c++.cpp

/* Copyright 2020, Gurobi Optimization, LLC */

/* This example formulates and solves the following simple QP model:

minimize x^2 + x*y + y^2 + y*z + z^2 + 2 x
subject to  x + 2 y + 3 z >= 4
x + y >= 1
x, y, z non-negative

It solves it once as a continuous model, and once as an integer model. */

#include "gurobi_c++.h"
using namespace std;

int main(int argc, char *argv[])
{
try {
GRBEnv env = GRBEnv();

GRBModel model = GRBModel(env);

// Create variables
GRBVar x = model.addVar(0.0, 1.0, 0.0, GRB_CONTINUOUS, "x");
GRBVar y = model.addVar(0.0, 1.0, 0.0, GRB_CONTINUOUS, "y");
GRBVar z = model.addVar(0.0, 1.0, 0.0, GRB_CONTINUOUS, "z");

// Set objective
GRBQuadExpr obj = x*x + x*y + y*y + y*z + z*z + 2*x;
model.setObjective(obj);

// Add constraint: x + 2 y + 3 z >= 4
model.addConstr(x + 2 * y + 3 * z >= 4, "c0");

// Add constraint: x + y >= 1
model.addConstr(x + y >= 1, "c1");

// Optimize model
model.optimize();

cout << x.get(GRB_StringAttr_VarName) << " "
<< x.get(GRB_DoubleAttr_X) << endl;
cout << y.get(GRB_StringAttr_VarName) << " "
<< y.get(GRB_DoubleAttr_X) << endl;
cout << z.get(GRB_StringAttr_VarName) << " "
<< z.get(GRB_DoubleAttr_X) << endl;
}

return 0;
}
cout << "Obj: " << model.get(GRB_DoubleAttr_ObjVal) << endl;

// Change variable types to integer
x.set(GRB_CharAttr_VType, GRB_INTEGER);
y.set(GRB_CharAttr_VType, GRB_INTEGER);
z.set(GRB_CharAttr_VType, GRB_INTEGER);

// Optimize model
model.optimize();

cout << x.get(GRB_StringAttr_VarName) << " "
<< x.get(GRB_DoubleAttr_X) << endl;
cout << y.get(GRB_StringAttr_VarName) << " "
<< y.get(GRB_DoubleAttr_X) << endl;
cout << z.get(GRB_StringAttr_VarName) << " "
<< z.get(GRB_DoubleAttr_X) << endl;
cout << "Obj: " << model.get(GRB_DoubleAttr_ObjVal) << endl;
}

} catch (GRBException e) {
    cout << "Error code = " << e.getErrorCode() << endl;
    cout << e.getMessage() << endl;
} catch (...) {
    cout << "Exception during optimization" << endl;
}

return 0;

} }
```cpp
double obj[] = {-2, -1, -1};
string names[] = {"x0", "x1", "x2");

x = model.addVars(NULL, ub, obj, NULL, names, 3);

// Add first SOS1: x0=0 or x1=0
GRBVar sosv1[] = {x[0], x[1]};
double soswt1[] = {1, 2};
model.addSOS(sosv1, soswt1, 2, GRB_SOS_TYPE1);

// Add second SOS1: x0=0 or x2=0 */
GRBVar sosv2[] = {x[0], x[2]};
double soswt2[] = {1, 2};
model.addSOS(sosv2, soswt2, 2, GRB_SOS_TYPE1);

// Optimize model
model.optimize();

for (int i = 0; i < 3; i++)
  cout << x[i].get(GRB_StringAttr_VarName) << " 
  << x[i].get(GRB_DoubleAttr_X) << endl;

cout << "Obj: " << model.get(GRB_DoubleAttr_ObjVal) << endl;
}
catch(GRBException e) {
  cout << "Error code = " << e.getErrorCode() << endl;
  cout << e.getMessage() << endl;
}
catch(...) {
  cout << "Exception during optimization" << endl;
}

delete[] x;
delete env;
return 0;
```

The Sudoku board is a 9x9 grid, which is further divided into a 3x3 grid of 3x3 grids. Each cell in the grid must take a value from 0 to 9. No two grid cells in the same row, column, or 3x3 subgrid may take the same value.

In the MIP formulation, binary variables x[i,j,v] indicate whether cell <i,j> takes value 'v'. The constraints are as follows:
1. Each cell must take exactly one value (sum_v x[i,j,v] = 1)
2. Each value is used exactly once per row (sum_i x[i,j,v] = 1)
3. Each value is used exactly once per column ($\sum_j x[i,j,v] = 1$)
4. Each value is used exactly once per 3x3 subgrid ($\sum_{\text{grid}} x[i,j,v] = 1$)

Input datasets for this example can be found in examples/data/sudoku*. *

```c++
#include "gurobi_c++.h"
#include <sstream>
using namespace std;

#define sd 3
#define n (sd*sd)

string itos(int i) { stringstream s; s << i; return s.str(); }

int main(int argc, char *argv[]) {
   try {
      GRBEnv env = GRBEnv();
      GRBModel model = GRBModel(env);
      GRBVar vars[n][n][n];
      int i, j, v;

      // Create 3-D array of model variables
      for (i = 0; i < n; i++) {
         for (j = 0; j < n; j++) {
            for (v = 0; v < n; v++) {
               string s = "G_" + itos(i) + "_" + itos(j) + "_" + itos(v);
               vars[i][j][v] = model.addVar(0.0, 1.0, 0.0, GRB_BINARY, s);
            }
         }
      }

      // Add constraints
      // Each cell must take one value
      for (i = 0; i < n; i++) {
         for (j = 0; j < n; j++) {
            for (v = 0; v < n; v++) {
               GRBLinExpr expr = 0;
               expr += vars[i][j][v];
               string s = "V_" + itos(i) + "_" + itos(j);
               model.addConstr(expr, GRB_EQUAL, 1.0, s);
            }
         }
      }

      // Each value appears once per row
      for (i = 0; i < n; i++) {
         for (v = 0; v < n; v++) {
            GRBLinExpr expr = 0;
            for (j = 0; j < n; j++) {
               expr += vars[i][j][v];
               string s = "R_" + itos(i) + "_" + itos(j);
               model.addConstr(expr, GRB_EQUAL, 1.0, s);
            }
         }
      }

      // Each value appears once per column
      for (j = 0; j < n; j++) {
         for (v = 0; v < n; v++) {
            GRBLinExpr expr = 0;
            for (i = 0; i < n; i++) {
               expr += vars[i][j][v];
               string s = "C_" + itos(i) + "_" + itos(j);
               model.addConstr(expr, GRB_EQUAL, 1.0, s);
            }
         }
      }

      // Each value appears once per 3x3 subgrid
      for (i = 0; i < n; i += sd) {
         for (j = 0; j < n; j += sd) {
            for (v = 0; v < n; v += sd) {
               GRBLinExpr expr = 0;
               for (k = 0; k < sd; k++) {
                  expr += vars[i][j][v];
               }
               string s = "B_" + itos(i) + "_" + itos(j) + "_" + itos(v);
               model.addConstr(expr, GRB_EQUAL, 1.0, s);
            }
         }
      }

      // Solve model
      model.optimize();

      // Print solution
      string s = "Solution:
      for (i = 0; i < n; i++) {
         for (j = 0; j < n; j++) {
            for (v = 0; v < n; v++) {
               if (vars[i][j][v].get(GRB_Double_ATTR_X) == 1.0) {
                  string s = "G_" + itos(i) + "_" + itos(j) + "_" + itos(v);
                  cout << s << " = 1.0\n";
               }
            }
         }
      }

      return 0;
   } catch (GRBException e) { cout << "Error code = \"" << e.get(GRB_Error) << "\n"; }
}
```
for (j = 0; j < n; j++)
    expr += vars[i][j][v];
string s = "R_" + itos(i) + ",_" + itos(v);
model.addConstr(expr == 1.0, s);
}
}

// Each value appears once per column
for (j = 0; j < n; j++) {
    for (v = 0; v < n; v++) {
        GRBLinExpr expr = 0;
        for (i = 0; i < n; i++)
            expr += vars[i][j][v];
        string s = "C_" + itos(j) + ",_" + itos(v);
        model.addConstr(expr == 1.0, s);
    }
}

// Each value appears once per sub-grid
for (v = 0; v < n; v++) {
    for (int i0 = 0; i0 < sd; i0++) {
        for (int j0 = 0; j0 < sd; j0++) {
            GRBLinExpr expr = 0;
            for (int i1 = 0; i1 < sd; i1++) {
                for (int j1 = 0; j1 < sd; j1++) {
                    expr += vars[i0*sd+i1][j0*sd+j1][v];
                }
            }
            string s = "Sub_" + itos(v) + ",_" + itos(i0) + ",_" + itos(j0);
            model.addConstr(expr == 1.0, s);
        }
    }
}

// Fix variables associated with pre-specified cells
char input[10];
for (i = 0; i < n; i++) {
    cin >> input;
    for (j = 0; j < n; j++) {
        int val = (int) input[j] - 48 - 1; // 0-based
        if (val >= 0)
            vars[i][j][val].set(GRB_DoubleAttr_LB, 1.0);
    }
}

// Optimize model
model.optimize();

// Write model to file
model.write("sudoku.lp");

cout << endl;
for (i = 0; i < n; i++) {
   for (j = 0; j < n; j++) {
      for (v = 0; v < n; v++) {
         if (vars[i][j][v].get(GRB_DoubleAttr_X) > 0.5)
            cout << v+1;
      }
   }
   cout << endl;
}
cout << endl;
}

} catch(GRBException e) {
   cout << "Error code = " << e.getErrorCode() << endl;
   cout << e.getMessage() << endl;
} catch(...) {
   cout << "Error during optimization" << endl;
}

return 0;
}

tsp_c++.cpp

/* Copyright 2020, Gurobi Optimization, LLC */

/* Solve a traveling salesman problem on a randomly generated set of points using lazy constraints. The base MIP model only includes 'degree-2' constraints, requiring each node to have exactly two incident edges. Solutions to this model may contain subtours that don't visit every node. The lazy constraint callback adds new constraints to cut them off. */

#include "gurobi_c++.h"
#include <cassert>
#include <cstdlib>
#include <cmath>
#include <sstream>
using namespace std;

string itos(int i) {stringstream s; s << i; return s.str(); }
double distance(double* x, double* y, int i, int j);
void findsubtour(int n, double* sol, int* tourlenP, int* tour);

// Subtour elimination callback. Whenever a feasible solution is found, // find the smallest subtour, and add a subtour elimination constraint // if the tour doesn't visit every node.

class subtourelim: public GRBCallback {
   public:
      GRBVar** vars;
      int n;
      subtourelim(GRBVar** xvars, int xn) {
         vars = xvars;
\[ n = xn; \]

```java
protected:
    void callback() {
        try {
            if (where == GRB_CB_MIPSOL) {
                // Found an integer feasible solution - does it visit every node?
                double **x = new double*[n];
                int *tour = new int[n];
                int i, j, len;
                for (i = 0; i < n; i++)
                    x[i] = getSolution(vars[i], n);

                findsubtour(n, x, &len, tour);

                if (len < n) {
                    // Add subtour elimination constraint
                    GRBLinExpr expr = 0;
                    for (i = 0; i < len; i++)
                        for (j = i + 1; j < len; j++)
                            expr += vars[tour[i]][tour[j]];
                    addLazy(expr <= len - 1);
                }

                for (i = 0; i < n; i++)
                    delete[] x[i];
                delete[] x;
                delete[] tour;
            }
        } catch (GRBException e) {
            cout << "Error number: " << e.getErrorCode() << endl;
            cout << e.getMessage() << endl;
        } catch (...) {
            cout << "Error during callback" << endl;
        }
    }
};
```

```java
// Given an integer-feasible solution 'sol', find the smallest
// sub-tour. Result is returned in 'tour', and length is
// returned in 'tourlenP'.

void
findsubtour(int n,
            double **sol,
            int* tourlenP,
            int* tour)
{
    bool *seen = new bool[n];
    int bestind, bestlen;
    int i, node, len, start;

    for (i = 0; i < n; i++)
        seen[i] = false;

    start = 0;
```
bestlen = n + 1;
bestind = -1;
node = 0;
while (start < n) {
  for (node = 0; node < n; node++)
    if (!seen[node])
      break;
  if (node == n)
    break;
  for (len = 0; len < n; len++) {
    tour[start+len] = node;
    seen[node] = true;
    for (i = 0; i < n; i++)
      if (sol[node][i] > 0.5 && !seen[i]) {
        node = i;
        break;
      }
    if (i == n) {
      len++;
      if (len < bestlen) {
        bestlen = len;
        bestind = start;
      }
      start += len;
      break;
    }
  }
}
for (i = 0; i < bestlen; i++)
  tour[i] = tour[bestind+i];
*tourlenP = bestlen;
delete[] seen;

// Euclidean distance between points 'i' and 'j'.

double
distance(double* x,
         double* y,
         int    i,
         int    j)
{
  double dx = x[i]-x[j];
  double dy = y[i]-y[j];
  return sqrt(dx*dx+dy*dy);
}

int
main(int    argc,
     char  *argv[1])
{
  if (argc < 2) {
cout << "Usage: tsp_c++ size" << endl;
return 1;
}

int n = atoi(argv[1]);
double* x = new double[n];
double* y = new double[n];

int i;
for (i = 0; i < n; i++) {
    x[i] = ((double) rand()) / RAND_MAX;
    y[i] = ((double) rand()) / RAND_MAX;
}

GRBEnv *env = NULL;
GRBVar **vars = NULL;

vars = new GRBVar*[n];
for (i = 0; i < n; i++)
    vars[i] = new GRBVar[n];

try {
    int j;

    env = new GRBEnv();
    GRBModel model = GRBModel(*env);

    // Must set LazyConstraints parameter when using lazy constraints
    model.set(GRB_IntParam_LazyConstraints, 1);

    // Create binary decision variables
    for (i = 0; i < n; i++) {
        for (j = 0; j <= i; j++) {
            vars[i][j] = model.addVar(0.0, 1.0, distance(x, y, i, j),
                                       GRB_BINARY, "x_" + itos(i) + "_" + itos(j));
            vars[j][i] = vars[i][j];
        }
    }

    // Degree-2 constraints
    for (i = 0; i < n; i++) {
        GRBLinExpr expr = 0;
        for (j = 0; j < n; j++)
            expr += vars[i][j];
        model.addConstr(expr == 2, "deg2_" + itos(i));
    }

    // Forbid edge from node back to itself
    for (i = 0; i < n; i++)
        vars[i][i].set(GRB_DoubleAttr_UB, 0);

    // Set callback function

```cpp
subtourelim cb = subtourelim(vars, n);
model.setCallback(&cb);

// Optimize model
model.optimize();

// Extract solution
if (model.get(GRB_IntAttr_SolCount) > 0) {
    double **sol = new double*[n];
    for (i = 0; i < n; i++)
        sol[i] = model.get(GRB_DoubleAttr_X, vars[i], n);

    int* tour = new int[n];
    int len;

    findsuctour(n, sol, &len, tour);
    assert(len == n);

    cout << "Tour: ";
    for (i = 0; i < len; i++)
        cout << tour[i] << " ";
    cout << endl;

    for (i = 0; i < n; i++)
        delete[] sol[i];
    delete[] sol;
    delete[] tour;
}

} catch (GRBException e) {
    cout << "Error number: " << e.getErrorCode() << endl;
    cout << e.getMessage() << endl;
} catch (...) {
    cout << "Error during optimization" << endl;
}

for (i = 0; i < n; i++)
    delete[] vars[i];
delete[] vars;
delete[] x;
delete[] y;
delete env;
return 0;
```

tune_c++.cpp

/* Copyright 2020, Gurobi Optimization, LLC */

/* This example reads a model from a file and tunes it.
   It then writes the best parameter settings to a file
   and solves the model using these parameters. */
```cpp
#include "gurobi_c++.h"
#include <cmath>
using namespace std;

int main(int argc,  
char *argv[])
{
    if (argc < 2) {
        cout << "Usage: tune_c++ filename" << endl;
        return 1;
    }

    GRBEnv *env = 0;
    try {
        env = new GRBEnv();

        // Read model from file
        GRBModel model = GRBModel(*env, argv[1]);

        // Set the TuneResults parameter to 1
        model.set(GRB_IntParam_TuneResults, 1);

        // Tune the model
        model.tune();

        // Get the number of tuning results
        int resultcount = model.get(GRB_IntAttr_TuneResultCount);
        if (resultcount > 0) {
            // Load the tuned parameters into the model’s environment
            model.getTuneResult(0);

            // Write tuned parameters to a file
            model.write("tune.prm");

            // Solve the model using the tuned parameters
            model.optimize();
        }
    } catch (GRBException e) {
        cout << "Error code = " << e.getErrorCode() << endl;
        cout << e.getMessage() << endl;
    } catch (...) {
        cout << "Error during tuning" << endl;
    }

    delete env;
    return 0;
}
```
/* Copyright 2020, Gurobi Optimization, LLC */

/* Assign workers to shifts; each worker may or may not be available on a
   particular day. If the problem cannot be solved, use IIS to find a set of
   conflicting constraints. Note that there may be additional conflicts
   besides what is reported via IIS. */

#include "gurobi_c++.h"
#include <sstream>
using namespace std;

int main(int argc, char *argv[])
{
    GRBEnv* env = 0;
    GRBConstr* c = 0;
    GRBVar** x = 0;
    int xCt = 0;
    try
    {
        // Sample data
        const int nShifts = 14;
        const int nWorkers = 7;

        // Sets of days and workers
        string Shifts[] = {
            "Mon1", "Tue2", "Wed3", "Thu4", "Fri5", "Sat6",
            "Sun7", "Mon8", "Tue9", "Wed10", "Thu11", "Fri12", "Sat13",
            "Sun14" }
        ;
        string Workers[] = {
            "Amy", "Bob", "Cathy", "Dan", "Ed", "Fred", "Gu" }
        ;

        // Number of workers required for each shift
        double shiftRequirements[] = {
            3, 2, 4, 4, 5, 6, 5, 2, 2, 3, 4, 6, 7, 5 }
        ;

        // Amount each worker is paid to work one shift
        double pay[] = { 10, 12, 10, 8, 8, 9, 11 }
        ;

        // Worker availability: 0 if the worker is unavailable for a shift
        double availability[][nShifts] =
            { { 0, 1, 1, 0, 1, 0, 1, 1, 1, 1, 1, 1 },
              { 1, 1, 0, 0, 1, 0, 0, 1, 0, 1, 0, 1 },
              { 0, 0, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1 },
              { 0, 1, 1, 0, 1, 1, 0, 1, 1, 1, 1, 1 },
              { 1, 1, 1, 0, 1, 0, 1, 1, 1, 1, 1, 1 },
              { 1, 1, 1, 0, 0, 1, 0, 1, 0, 1, 1, 1 },
              { 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1 } }
        ;

        // Model
env = new GRBEnv();
GRBModel model = GRBModel(*env);
model.set(GRB_StringAttr_ModelName, "assignment");

// Assignment variables: x[w][s] == 1 if worker w is assigned to shift s. Since an assignment model always produces integer solutions, we use continuous variables and solve as an LP.
x = new GRBVar*[nWorkers];
for (int w = 0; w < nWorkers; ++w)
{
    x[w] = model.addVars(nShifts);
    xCt++;
    for (int s = 0; s < nShifts; ++s)
    {
        stringstream vname;
        vname << Workers[w] << "." << Shifts[s];
        x[w][s].set(GRB_DoubleAttr_UB, availability[w][s]);
        x[w][s].set(GRB_DoubleAttr_Obj, pay[w]);
        x[w][s].set(GRB_StringAttr_VarName, vname.str());
    }
}

// The objective is to minimize the total pay costs
model.set(GRB_IntAttr_ModelSense, GRB_MINIMIZE);

// Constraint: assign exactly shiftRequirements[s] workers to each shift s
for (int s = 0; s < nShifts; ++s)
{
    GRBLinExpr lhs = 0;
    for (int w = 0; w < nWorkers; ++w)
    {
        lhs += x[w][s];
    }
    model.addConstr(lhs == shiftRequirements[s], Shifts[s]);
}

// Optimize
model.optimize();
int status = model.get(GRB_IntAttr_Status);
if (status == GRB_UNBOUNDED)
{
    cout << "The model cannot be solved "
    << "because it is unbounded" << endl;
    return 1;
}
if (status == GRB_OPTIMAL)
{
    cout << "The optimal objective is " <<
    model.get(GRB_DoubleAttr_ObjVal) << endl;
    return 0;
}
if ((status != GRB_INF_OR_UNBD) && (status != GRB_INFEASIBLE))
{
    cout << "Optimization was stopped with status " << status << endl;
    return 1;
}
// do IIS
cout << "The model is infeasible; computing IIS" << endl;
model.computeIIS();
cout << "\nThe following constraint(s) " << "cannot be satisfied:" << endl;
c = model.getConstrs();
for (int i = 0; i < model.get(GRB_IntAttr_NumConstrs); ++i)
{
    if (c[i].get(GRB_IntAttr_IISConstr) == 1)
    {
        cout << c[i].get(GRB_StringAttr_ConstrName) << endl;
    }
}

catch (GRBException e)
{
    cout << "Error code = " << e.getErrorCode() << endl;
    cout << e.getMessage() << endl;
} catch (...) {
    cout << "Exception during optimization" << endl;
}

delete[] c;
for (int i = 0; i < xCt; ++i) {
    delete[] x[i];
}
delete[] x;
delete env;
return 0;

workforce2_c++.cpp

/* Copyright 2020, Gurobi Optimization, LLC */

/* Assign workers to shifts; each worker may or may not be available on a 
particular day. If the problem cannot be solved, use IIS iteratively to 
find all conflicting constraints. */

#include "gurobi_c++.h"
#include <sstream>
#include <deque>
using namespace std;

int main(int argc, 
    char *argv[])
{
    GRBEnv* env = 0;
    GRBConstr* c = 0;
    GRBVar** x = 0;
int xCt = 0;
try
{

    // Sample data
    const int nShifts = 14;
    const int nWorkers = 7;

    // Sets of days and workers
    string Shifts[] = 
        { "Mon1", "Tue2", "Wed3", "Thu4", "Fri5", "Sat6", 
          "Sun7", "Mon8", "Tue9", "Wed10", "Thu11", "Fri12", "Sat13", 
          "Sun14" }; 
    string Workers[] = 

    // Number of workers required for each shift
    double shiftRequirements[] = 
        { 3, 2, 4, 4, 5, 6, 5, 2, 2, 3, 4, 6, 7, 5 }; 

    // Amount each worker is paid to work one shift
    double pay[] = { 10, 12, 10, 8, 8, 9, 11 }; 

    // Worker availability: 0 if the worker is unavailable for a shift
    double availability[][nShifts] = 
        { { 0, 1, 1, 0, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1 }, 
          { 1, 1, 0, 0, 1, 1, 0, 0, 1, 0, 1, 0, 1, 0 }, 
          { 0, 0, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1 }, 
          { 0, 1, 1, 0, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1 }, 
          { 1, 1, 1, 1, 0, 1, 1, 0, 1, 0, 1, 0, 1, 1 }, 
          { 1, 1, 1, 0, 0, 1, 0, 1, 1, 0, 0, 1, 1, 1 }, 
          { 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1 } }; 

    // Model
    env = new GRBEnv();
    GRBModel model = GRBModel(*env);
    model.set(GRB_PairAttr_ModelName, "assignment");

    // Assignment variables: x[w][s] == 1 if worker w is assigned 
    // to shift s. Since an assignment model always produces integer 
    // solutions, we use continuous variables and solve as an LP. 
    x = new GRBVar*[nWorkers];
    for (int w = 0; w < nWorkers; ++w)
    {
        x[w] = model.addVars(nShifts);
        xCt++;
        for (int s = 0; s < nShifts; ++s)
        {
            stringstream vname;
            vname << Workers[w] << "." << Shifts[s];
            x[w][s].set(GRB_DoubleAttr_UB, availability[w][s]);
            x[w][s].set(GRB_DoubleAttr_Obj, pay[w]);
            x[w][s].set(GRB_PairAttr_VarName, vname.str());
        }
    }
}
// The objective is to minimize the total pay costs
model.set(GRB_IntAttr_ModelSense, GRB_MINIMIZE);

// Constraint: assign exactly shiftRequirements[s] workers to each shift s
for (int s = 0; s < nShifts; ++s)
{
    GRBLinExpr lhs = 0;
    for (int w = 0; w < nWorkers; ++w)
    {
        lhs += x[w][s];
    }
    model.addConstr(lhs == shiftRequirements[s], Shifts[s]);
}

// Optimize
model.optimize();
int status = model.get(GRB_IntAttr_Status);
if (status == GRB_UNBOUNDED)
{
    cout << "The model cannot be solved because it is unbounded" << endl;
    return 1;
}
if (status == GRB_OPTIMAL)
{
    cout << "The optimal objective is " << model.get(GRB_DoubleAttr_ObjVal) << endl;
    return 0;
}
if ((status != GRB_INF_OR_UNBD) && (status != GRB_INFEASIBLE))
{
    cout << "Optimization was stopped with status " << status << endl;
    return 1;
}

// do IIS
cout << "The model is infeasible; computing IIS" << endl;
deleque<string> removed;

// Loop until we reduce to a model that can be solved
while (1)
{
    model.computeIIS();
    cout << "The following constraint cannot be satisfied:" << endl;
    c = model.getConstrs();
    for (int i = 0; i < model.get(GRB_IntAttr_NumConstrs); ++i)
    {
        if (c[i].get(GRB_IntAttr_IISConstr) == 1)
        {
            cout << c[i].get(GRB_StringAttr_ConstrName) << endl;
            // Remove a single constraint from the model
            removed.push_back(c[i].get(GRB_StringAttr_ConstrName));
            model.remove(c[i]);
            break;
        }
    }
}
cout << endl;
model.optimize();
status = model.get(GRB_IntAttr_Status);

if (status == GRB_UNBOUNDED)
{
    cout << "The model cannot be solved because it is unbounded" << endl;
    return 0;
}
if (status == GRB_OPTIMAL)
{
    break;
}
if ((status != GRB_INF_OR_UNBD) && (status != GRB_INFEASIBLE))
{
    cout << "Optimization was stopped with status " << status << endl;
    return 1;
}
}
cout << 
\nThe following constraints were removed "
<< "to get a feasible LP:" << endl;

for (deque<string>::iterator r = removed.begin();
    r != removed.end();
++r)
{
    cout << *r << " ";
}
cout << endl;

}
catch (GRBException e)
{
    cout << "Error code = " << e.getErrorCode() << endl;
    cout << e.getMessage() << endl;
}
catch (...)
{
    cout << "Exception during optimization" << endl;
}

delete[] c;
for (int i = 0; i < xCt; ++i) {
    delete[] x[i];
}
delete[] x;
delete env;
return 0;

workforce3_c++.cpp
/* Copyright 2020, Gurobi Optimization, LLC */

/* Assign workers to shifts; each worker may or may not be available on a
particular day. If the problem cannot be solved, relax the model
to determine which constraints cannot be satisfied, and how much
they need to be relaxed. */

#include "gurobi_c++.h"
#include <sstream>
using namespace std;

int main(int argc, char *argv[])
{
    GRBEnv* env = 0;
    GRBConstr* c = 0;
    GRBVar** x = 0;
    GRBVar* vars = 0;
    int xCt = 0;
    try
    {

        // Sample data
        const int nShifts = 14;
        const int nWorkers = 7;

        // Sets of days and workers
        string Shifts[] =
            { "Mon1", "Tue2", "Wed3", "Thu4", "Fri5", "Sat6",
              "Sun7", "Mon8", "Tue9", "Wed10", "Thu11", "Fri12", "Sat13",
              "Sun14" };
        string Workers[] =

        // Number of workers required for each shift
        double shiftRequirements[] =
            { 3, 2, 4, 4, 5, 6, 5, 2, 2, 3, 4, 6, 7, 5 };

        // Amount each worker is paid to work one shift
        double pay[] = { 10, 12, 10, 8, 8, 9, 11 };

        // Worker availability: 0 if the worker is unavailable for a shift
        double availability[][nShifts] =
            { { 0, 1, 1, 0, 1, 0, 1, 1, 1, 1, 1, 0, 1 },
              { 1, 1, 0, 0, 1, 1, 0, 0, 1, 0, 1, 0, 0 },
              { 0, 0, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1 },
              { 0, 1, 1, 0, 1, 1, 0, 1, 1, 1, 1, 1, 1 },
              { 1, 1, 1, 1, 0, 1, 1, 0, 1, 1, 1, 0, 1 },
              { 1, 1, 1, 0, 0, 1, 1, 0, 0, 1, 1, 1, 1 };

        // Model
        env = new GRBEnv();
        GRBModel model = GRBModel(*env);
        model.set(GRB_StringAttr_ModelName, "assignment");
Assignment variables: \( x[w][s] = 1 \) if worker \( w \) is assigned to shift \( s \). Since an assignment model always produces integer solutions, we use continuous variables and solve as an LP.

```cpp
x = new GRBVar*[nWorkers];
for (int w = 0; w < nWorkers; ++w)
{
    x[w] = model.addVars(nShifts);
    xCt++;
    for (int s = 0; s < nShifts; ++s)
    {
        stringstream vname;
        vname << Workers[w] << "." << Shifts[s];
        x[w][s].set(GRB_DoubleAttr_UB, availability[w][s]);
        x[w][s].set(GRB_DoubleAttr_Obj, pay[w]);
        x[w][s].set(GRB_StringAttr_VarName, vname.str());
    }
}
```

The objective is to minimize the total pay costs:

```cpp
model.set(GRB_IntAttr_ModelSense, GRB_MINIMIZE);
```

Constraint: assign exactly \( shiftRequirements[s] \) workers to each shift \( s \):

```cpp
for (int s = 0; s < nShifts; ++s)
{
    GRBLinExpr lhs = 0;
    for (int w = 0; w < nWorkers; ++w)
    {
        lhs += x[w][s];
    }
    model.addConstr(lhs == shiftRequirements[s], Shifts[s]);
}
```

Optimize:

```cpp
model.optimize();
int status = model.get(GRB_IntAttr_Status);
if (status == GRB_UNBOUNDED)
{
    cout << "The model cannot be solved 
because it is unbounded" << endl;
    return 1;
}
if (status == GRB_OPTIMAL)
{
    cout << "The optimal objective is " <<
    model.get(GRB_DoubleAttr_ObjVal) << endl;
    return 0;
}
if ((status != GRB_INF_OR_UNBD) && (status != GRB_INFEASIBLE))
{
    cout << "Optimization was stopped with status " << status << endl;
    return 1;
}
```

Relax the constraints to make the model feasible.
cout << "The model is infeasible; relaxing the constraints" << endl;
int orignumvars = model.getIntAttr(Grb_IntAttr_NumVars);
model.feasRelax(0, false, false, true);
model.optimize();
status = model.getIntAttr(Grb_IntAttr_Status);
if ((status == GRB_INF_OR_UNBD) || (status == GRB_INFEASIBLE) ||
    (status == GRB_UNBOUNDED))
{
    cout << "The relaxed model cannot be solved " <<
        "because it is infeasible or unbounded" << endl;
    return 1;
}
if (status != GRB_OPTIMAL)
{
    cout << "Optimization was stopped with status " << status << endl;
    return 1;
}
cout << "\nSlack values:" << endl;
vars = model.getVars();
for (int i = orignumvars; i < model.getIntAttr(Grb_IntAttr_NumVars); ++i)
{
    GRBVar sv = vars[i];
    if (sv.getDoubleAttr(Grb_DoubleAttr_X) > 1e-6)
    {
        cout << sv.getDoubleAttr(Grb_StringAttr_VarName) << " = " <<
            sv.getDoubleAttr(Grb_DoubleAttr_X) << endl;
    }
}
}

} catch (GRBException e)
{
    cout << "Error code = " << e.getErrorCode() << endl;
    cout << e.getMessage() << endl;
} catch (...
{
    cout << "Exception during optimization" << endl;
}
delete[] c;
for (int i = 0; i < xCt; ++i) {
    delete[] x[i];
}
delete[] x;
delete[] vars;
delete env;
return 0;
}

workforce4_c++.cpp

/* Copyright 2020, Gurobi Optimization, LLC */

/* Assign workers to shifts; each worker may or may not be available on a
We use Pareto optimization to solve the model: first, we minimize the linear sum of the slacks. Then, we constrain the sum of the slacks, and we minimize a quadratic objective that tries to balance the workload among the workers. */

```cpp
#include "gurobi_c++.h"
#include <sstream>
using namespace std;

int solveAndPrint(GRBModel & model, GRBVar & totSlack, int nWorkers, string* Workers, GRBVar* totShifts);

int main(int argc, char *argv[])
{
    GRBEnv* env = 0;
    GRBVar** x = 0;
    GRBVar * slacks = 0;
    GRBVar * totShifts = 0;
    GRBVar * diffShifts = 0;
    int xCt = 0;

    try
    {
        // Sample data
        const int nShifts = 14;
        const int nWorkers = 7;

        // Sets of days and workers
        string Shifts[] = {
            "Mon1", "Tue2", "Wed3", "Thu4", "Fri5", "Sat6",
            "Sun7", "Mon8", "Tue9", "Wed10", "Thu11", "Fri12", "Sat13",
            "Sun14" 
        };
        string Workers[] = {
            "Amy", "Bob", "Cathy", "Dan", "Ed", "Fred", "Gu" 
        };

        // Number of workers required for each shift
        double shiftRequirements[] =
            { 3, 2, 4, 4, 5, 6, 5, 2, 2, 3, 4, 6, 7, 5 };

        // Worker availability: 0 if the worker is unavailable for a shift
        double availability[][nShifts] =
            { { 0, 1, 1, 0, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1 },
              { 1, 1, 0, 0, 1, 1, 0, 1, 0, 1, 0, 1, 0, 1 },
              { 0, 0, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1 },
              { 0, 1, 1, 0, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1 },
              { 1, 1, 1, 1, 0, 1, 1, 1, 0, 1, 0, 1, 1, 1 },
              { 1, 1, 1, 0, 0, 1, 1, 0, 1, 0, 1, 0, 1, 1 },
              { 1, 1, 1, 0, 0, 1, 1, 0, 1, 0, 1, 0, 1, 1 },
              { 1, 1, 1, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1 } 
        };

        // Model
        env = new GRBEnv();
        GRBModel model = GRBModel(*env);
        model.set(GRB_StringAttr_ModelName, "assignment");
```
Assignment variables: \( x[w][s] == 1 \) if worker \( w \) is assigned to shift \( s \). This is no longer a pure assignment model, so we must use binary variables.

```c
x = new GRBVar*[nWorkers];
int s, w;

for (w = 0; w < nWorkers; ++w) {
    x[w] = model.addVars(nShifts);
    xCt++;

    for (s = 0; s < nShifts; ++s) {
        stringstream vname;
        vname << Workers[w] << "." << Shifts[s];
        x[w][s].set(GRB_DoubleAttr_UB, availability[w][s]);
        x[w][s].set(GRB_CharAttr_VType, GRB_BINARY);
        x[w][s].set(GRB_StringAttr_VarName, vname.str());
    }
}
```

Slack variables for each shift constraint so that the shifts can be satisfied

```c
slacks = model.addVars(nShifts);
for (s = 0; s < nShifts; ++s) {
    stringstream vname;
    vname << Shifts[s] << "Slack";
    slacks[s].set(GRB_StringAttr_VarName, vname.str());
}
```

Variable to represent the total slack

```
GRBVar totSlack = model.addVar(0, GRB_INFINITY, 0, GRB_CONTINUOUS, "totSlack");
```

Variables to count the total shifts worked by each worker

```c
totShifts = model.addVars(nWorkers);
for (w = 0; w < nWorkers; ++w) {
    stringstream vname;
    vname << Workers[w] << "TotShifts";
    totShifts[w].set(GRB_StringAttr_VarName, vname.str());
}
```

GRBLinExpr lhs;

Constraint: assign exactly \( \text{shiftRequirements}[s] \) workers // to each shift \( s \)

```
for (s = 0; s < nShifts; ++s) {
    lhs = 0;
    lhs += slacks[s];

    for (w = 0; w < nWorkers; ++w) {
        lhs += x[w][s];
    }
```
model.addConstr(lhs == shiftRequirements[s], Shifts[s]);
}

// Constraint: set totSlack equal to the total slack
lhs = 0;
for (s = 0; s < nShifts; ++s)
{
    lhs += slacks[s];
}
model.addConstr(lhs == totSlack, "totSlack");

// Constraint: compute the total number of shifts for each worker
for (w = 0; w < nWorkers; ++w) {
    lhs = 0;
    for (s = 0; s < nShifts; ++s) {
        lhs += x[w][s];
    }
    ostringstream vname;
    vname << "totShifts" << Workers[w];
    model.addConstr(lhs == totShifts[w], vname.str());
}

// Objective: minimize the total slack
GRBLinExpr obj = 0;
obj += totSlack;
model.setObjective(obj);

// Optimize
int status = solveAndPrint(model, totSlack, nWorkers, Workers, totShifts);
if (status != GRB_OPTIMAL)
    return 1;

// Constrain the slack by setting its upper and lower bounds
totSlack.set(GRB_DoubleAttr_UB, totSlack.get(GRB_DoubleAttr_X));
totSlack.set(GRB_DoubleAttr_LB, totSlack.get(GRB_DoubleAttr_X));

// Variable to count the average number of shifts worked
GRBVar avgShifts =
model.addVar(0, GRB_INFINITY, 0, GRB_CONTINUOUS, "avgShifts");

// Variables to count the difference from average for each worker;
// note that these variables can take negative values.
diffShifts = model.addVars(nWorkers);
for (w = 0; w < nWorkers; ++w) {
    ostringstream vname;
    vname << Workers[w] << "Diff";
    diffShifts[w].set(GRB_StringAttr_VarName, vname.str());
    diffShifts[w].set(GRB_DoubleAttr_LB, -GRB_INFINITY);
}

// Constraint: compute the average number of shifts worked
lhs = 0;
for (w = 0; w < nWorkers; ++w) {
    lhs += totShifts[w];
}
model.addConstr(lhs == nWorkers * avgShifts, "avgShifts");
// Constraint: compute the difference from the average number of shifts
for (w = 0; w < nWorkers; ++w) {
    lhs = 0;
    lhs += totShifts[w];
    lhs -= avgShifts;
    ostringstream vname;
    vname << Workers[w] << "Diff";
    model.addConstr(lhs == diffShifts[w], vname.str());
}

// Objective: minimize the sum of the square of the difference from the
// average number of shifts worked
GRBQuadExpr qobj;
for (w = 0; w < nWorkers; ++w) {
    qobj += diffShifts[w] * diffShifts[w];
} model.setObjective(qobj);

// Optimize
status = solveAndPrint(model, totSlack, nWorkers, Workers, totShifts);
if (status != GRB_OPTIMAL)
    return 1;

catch (GRBException e) {
    cout << "Error code = " << e.getErrorCode() << endl;
    cout << e.getMessage() << endl;
}
catch (...) {
    cout << "Exception during optimization" << endl;
}

for (int i = 0; i < xCt; ++i) {
    delete[] x[i];
}
delete[] x;
delete[] slacks;
delete[] totShifts;
delete[] diffShifts;
delete env;

return 0;
}

int solveAndPrint(GRBModel& model,
                    GRBVar& totSlack,
                    int nWorkers,
                    string* Workers,
                    GRBVar* totShifts)
{
    model.optimize();
    int status = model.get(GRB_IntAttr_Status);

    if ((status == GRB_INF_OR_UNBD) ||
        (status == GRB_INFEASIBLE) ||
        (status == GRB_UNBOUNDED) ) {
cout << "The model cannot be solved " <<
"because it is infeasible or unbounded" << endl;
return status;
}
if (status != GRB_OPTIMAL) {
    cout << "Optimization was stopped with status " << status << endl;
    return status;
}

// Print total slack and the number of shifts worked for each worker
cout << endl << "Total slack required: " <<
totSlack.get(GRB_DoubleAttr_X) << endl;
for (int w = 0; w < nWorkers; ++w) {
    cout << Workers[w] << " worked " <<
totShifts[w].get(GRB_DoubleAttr_X) << " shifts" << endl;
}
cout << endl;
return status;
}

workforce5_c++.cpp

/* Copyright 2020, Gurobi Optimization, LLC */

/* Assign workers to shifts; each worker may or may not be available on a
particular day. We use multi-objective optimization to solve the model.
The highest-priority objective minimizes the sum of the slacks
(i.e., the total number of uncovered shifts). The secondary objective
minimizes the difference between the maximum and minimum number of
shifts worked among all workers. The second optimization is allowed
to degrade the first objective by up to the smaller value of 10% and 2 */

#include "gurobi_c++.h"
#include <sstream>
using namespace std;

int solveAndPrint(GRBModel& model, GRBVar & totSlack,
int nWorkers, string * Workers,
GRBVar* totShifts);

int main(int argc, char * argv[])
{
    GRBEnv * env = 0;
    GRBVar ** x = 0;
    GRBVar * slacks = 0;
    GRBVar * totShifts = 0;
    int xCt = 0;
    int s, w;

    try {
        // Sample data
        const int nShifts = 14;
        const int nWorkers = 8;

        // Code for solving the model
    }
}
// Sets of days and workers
string Shifts[] =
string Workers[] =

// Number of workers required for each shift
double shiftRequirements[] =
{ 3, 2, 4, 4, 5, 6, 5, 2, 2, 3, 4, 6, 7, 5 };

// Worker availability: 0 if the worker is unavailable for a shift
double availability[][14] =
{ { 0, 1, 1, 0, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1 },
{ 1, 1, 0, 0, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1 },
{ 0, 0, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1 },
{ 0, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1 },
{ 1, 1, 1, 1, 0, 1, 1, 0, 1, 0, 1, 0, 1, 1 },
{ 1, 1, 0, 0, 1, 0, 1, 1, 0, 0, 1, 0, 1, 1 },
{ 0, 1, 1, 1, 0, 1, 1, 0, 1, 1, 0, 1, 0, 1 },
{ 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1 } };

// Create environment
GRBEnv env = new GRBEnv("workforce5_c++ . log ");

// Create initial model
GRBModel model = GRBModel(*env);
model.set(GRB_StringAttr_ModelName, "workforce5_c++");

// Initialize assignment decision variables:
// x[w][s] == 1 if worker w is assigned to shift s.
// This is no longer a pure assignment model, so we must
// use binary variables.
x = new GRBVar*[nWorkers];
for (w = 0; w < nWorkers; w++) {
x[w] = model.addVars(nShifts, GRB_BINARY);
xCt++;
for (s = 0; s < nShifts; s++) {
    ostringstream vname;
    vname << Workers[w] << "." << Shifts[s];
x[w][s].set(GRB_DoubleAttr_UB, availability[w][s]);
x[w][s].set(GRB_StringAttr_VarName, vname.str());
}
}

// Initialize slack decision variables
slacks = model.addVars(nShifts);
for (s = 0; s < nShifts; s++) {
    ostringstream vname;
    vname << Shifts[s] << "Slack";
    slacks[s].set(GRB_StringAttr_VarName, vname.str());
}
// Variable to represent the total slack
GRBVar totSlack = model.addVar(0, GRB_INFINITY, 0, GRB_CONTINUOUS, "totSlack");

// Initialize variables to count the total shifts worked by each worker
totShifts = model.addVars(nWorkers);
for (w = 0; w < nWorkers; w++) {
    ostringstream vname;
    vname << Workers[w] << " TotShifts";
    totShifts[w].set(GRB_StringAttr_VarName, vname.str());
}

GRBLinExpr lhs;

// Constraint: assign exactly shiftRequirements[s] workers to each shift s, plus the slack
for (s = 0; s < nShifts; s++) {
    lhs = 0;
    lhs += slacks[s];
    for (w = 0; w < nWorkers; w++) {
        lhs += x[w][s];
    }
    model.addConstr(lhs == shiftRequirements[s], Shifts[s]);
}

// Constraint: set totSlack column equal to the total slack
lhs = 0;
for (s = 0; s < nShifts; s++) {
    lhs += slacks[s];
}
model.addConstr(lhs == totSlack, "totSlack");

// Constraint: compute the total number of shifts for each worker
for (w = 0; w < nWorkers; w++) {
    lhs = 0;
    for (s = 0; s < nShifts; s++) {
        lhs += x[w][s];
    }
    ostringstream vname;
    vname << "totShifts" << Workers[w];
    model.addConstr(lhs == totShifts[w], vname.str());
}

// Constraint: set minShift/maxShift variable to less <=/>= to the number of shifts among all workers
GRBVar minShift = model.addVar(0, GRB_INFINITY, 0, GRB_CONTINUOUS, "minShift");
GRBVar maxShift = model.addVar(0, GRB_INFINITY, 0, GRB_CONTINUOUS, "maxShift");
model.addGenConstrMin(minShift, totShifts, nWorkers, GRB_INFINITY, "minShift");
model.addGenConstrMax(maxShift, totShifts, nWorkers, -GRB_INFINITY, "maxShift");
// Set global sense for ALL objectives
model.set(GRB_IntAttr_ModelSense, GRB_MINIMIZE);

// Set primary objective
model.setObjectiveN(totSlack, 0, 2, 1.0, 2.0, 0.1, "TotalSlack");

// Set secondary objective
model.setObjectiveN(maxShift - minShift, 1, 1, 1.0, 0, 0, "Fairness");

// Save problem
model.write("workforce5_c++.lp");

// Optimize
int status = solveAndPrint(model, totSlack, nWorkers, Workers, totShifts);

// Delete local variables
if (status != GRB_OPTIMAL)
    return 1;

} catch (GRBException e){
    cout << "Error code = " << e.getErrorCode() << endl;
    cout << e.getMessage() << endl;
} catch (...) {
    cout << "Exception during optimization" << endl;
}

for (s = 0; s < xCt; s++)
    delete[] x[s];
delete[] x;
delete[] slacks;
delete[] totShifts;
delete env;
return 0;
}

int solveAndPrint(GRBModel& model, 
    GRBVar& totSlack, 
    int nWorkers, 
    string* Workers, 
    GRBVar* totShifts)
{
    model.optimize();
    int status = model.get(GRB_IntAttr_Status);

    if ((status == GRB_INF_OR_UNBD) || 
        (status == GRB_INFEASIBLE) || 
        (status == GRB_UNBOUNDED)) {
        cout << "The model cannot be solved " << 
                "because it is infeasible or unbounded" << endl;
        return status;
    }
    if (status != GRB_OPTIMAL) {
        cout << "Optimization was stopped with status " << status << endl;
        return status;
    }
// Print total slack and the number of shifts worked for each worker
cout << endl << "Total slack required: " <<
totSlack.get(GRB_DoubleAttr_X) << endl;
for (int w = 0; w < nWorkers; ++w) {
cout << Workers[w] << " worked " <<
totShifts[w].get(GRB_DoubleAttr_X) << " shifts" << endl;
}
cout << endl;
return status;
}

3.3 Java Examples

This section includes source code for all of the Gurobi Java examples. The same source code can be found in the examples/java directory of the Gurobi distribution.

Batchmode.java

/* Copyright 2020, Gurobi Optimization, LLC */

/* This example reads a model from a file, solves it in batch mode
* and prints the JSON solution string. */

import gurobi.*;

public class Batchmode {

    // Set-up a batch-mode environment
    private static GRBEnv setupbatchconnection() throws GRBException {
        GRBEnv env = new GRBEnv(true);
        env.set(GRB.IntParam.CSBatchMode, 1);
        env.set(GRB.StringParamLogFile, "batchmode.log");
        env.set(GRB.StringParamCServer, "http://localhost:61080");
        env.set(GRB.StringParamUserName, "gurobi");
        env.set(GRB.StringParamPassword, "pass");
        env.start();
        return env;
    }

    // Display batch-error if any
    private static void batcherrorinfo(GRBBatch batch) throws GRBException {
        // Get last error code
        int error = batch.get(GRB.IntAttrBatchErrorCode);
        if (error == 0) return;

        // Query last error message
        String errMsg = batch.get(GRB.StringAttrBatchErrorMessage);

        // Query batchID
        String batchID = batch.get(GRB.StringAttrBatchID);

        System.out.println("Batch ID " + batchID + " Error Code " +
                           error + "(" + errMsg + ")");
private static String newbatchrequest(String filename) throws GRBException {
    // Setup a batch connection
    GRBEnv env = setupbatchconnection();

    // Read a model
    GRBModel model = new GRBModel(env, filename);

    // Set some parameters
    model.set(GRB.DoubleParam.MIPGap, 0.01);
    model.set(GRB.IntParam.JSONSolDetail, 1);

    // Set-up some tags, we need tags to be able to query results
    int count = 0;
    for (GRBVar v: model.getVars()) {
        v.set(GRB.StringAttr.VTag, "UniqueVariableIdentifier" + count);
        count += 1;
        if (count >= 10) break;
    }

    // Batch-mode optimization
    String batchid = model.optimizeBatch();

    // no need to keep the model around
    model.dispose();

    // no need to keep environment
    env.dispose();

    return batchid;
}

private static void waitforfinalstatus(String batchid) throws Exception {
    // Setup a batch connection
    GRBEnv env = setupbatchconnection();

    // Create Batch-object
    GRBBatch batch = new GRBBatch(env, batchid);

    try {
        // Query status, and wait for completed
        int status = batch.get(GRB.IntAttr.BatchStatus);
        long timestart = System.currentTimeMillis();

        while(status == GRB.BatchStatus.SUBMITTED) {

            // Abort if taking too long
            long curtime = System.currentTimeMillis();
            if (curtime - timestart > 3600 * 1000) {
                // Request to abort the batch
                batch.abort();
                break;
            }
        }
    }
// Do not bombard the server
Thread.sleep(2000);

// Update local attributes
batch.update();

// Query current status
status = batch.get(GRB.IntAttr.BatchStatus);

// Deal with failed status
if (status == GRB.BatchStatus.FAILED ||
    status == GRB.BatchStatus.ABORTED ) {
    // Retry the batch job
    batch.retry();
}

} catch (Exception e) {
    // Display batch-error if any
    batcherrorinfo(batch);
    throw e;
} finally {
    // Dispose resources
    batch.dispose();
    env.dispose();
}

// Final report on batch request
private static void finalreport(String batchid) throws GRBException {

    // Setup a batch connection
    GRBEnv env = setupbatchconnection();

    // Create batch object
    GRBBatch batch = new GRBBatch(env, batchid);

    try {
        int status = batch.get(GRB.IntAttr.BatchStatus);
        // Display depending on batch status
        switch (status) {
            case GRB.BatchStatus.CREATION:
                System.out.println("Batch is 'CREATED'");
                System.out.println("maybe batch-creation process was killed?");
                break;
            case GRB.BatchStatus.SUBMITTED:
                System.out.println("Batch is 'SUBMITTED'");
                System.out.println("Some other user re-submitted this Batch object?");
                break;
            case GRB.BatchStatus.ABORTED:
                System.out.println("Batch is 'ABORTED'");
                break;
            case GRB.BatchStatus.FAILED:
                System.out.println("Batch is 'FAILED'");
                break;
        }
    } finally {
        // Dispose resources
        batch.dispose();
        env.dispose();
    }
}
case GRB.BatchStatus.COMPLETED:

    // print JSON solution into string
    System.out.println("JSON solution:" + batch.getJSONSolution());

    // save solution into a file
    batch.writeJSONSolution("batch-sol.json.gz");
    break;
    default:
    System.out.println("This should not happen, probably points to a user-memory corruption problem");
    System.exit(1);
    break;

} catch (GRBException e) {
    // Display batch-error if any
    batcherrorinfo(batch);
    throw e;
} finally {
    // Dispose resources
    batch.dispose();
    env.dispose();
}

// Discard batch data from the Cluster Manager
private static void discardbatch(String batchid) throws GRBException {
    // Setup a batch connection
    GRBEnv env = setupbatchconnection();

    // Create batch object
    GRBBatch batch = new GRBBatch(env, batchid);

    try {
        // Request to erase input and output data related to this batch
        batch.discard();
    } catch (GRBException e) {
        // Display batch-error if any
        batcherrorinfo(batch);
        throw e;
    } finally {
        // Dispose resources
        batch.dispose();
        env.dispose();
    }
}

// Main public function
public static void main(String[] args) {

    // Ensure enough parameters
    if (args.length < 1) {
        System.out.println("Usage: java Batch filename");
        System.exit(1);
    }
try {

    // Create a new batch request
    String batchid = newbatchrequest(args[0]);

    // Wait for final status
    waitforfinalstatus(batchid);

    // Query final status, and if completed, print JSON solution
    finalreport(batchid);

    // once the user is done, discard all remote information
    discardbatch(batchid);

    // Signal success
    System.out.println("OK");
} catch (GRBException e) {
    System.out.println("Error code: "+ e.getErrorCode()+". " 
    + e.getMessage());
} catch (Exception e) {
    System.out.println("Error");
}

Bilinear.java

/* Copyright 2020, Gurobi Optimization, LLC */

/* This example formulates and solves the following simple bilinear model:

maximize     x
subject to   x + y + z <= 10
             x * y <= 2        (bilinear inequality)
             x * z + y * z == 1 (bilinear equality)
             x, y, z non-negative (x integral in second version)
*/

import gurobi.*;

public class Bilinear {
    public static void main(String[] args) {
        try {
            GRBEnv    env    = new GRBEnv("bilinear.log");
            GRBModel  model  = new GRBModel(env);

            // Create variables
            GRBVar    x = model.addVar(0.0, GRB.INFINITY, 0.0, GRB.CONTINUOUS, "x");
            GRBVar    y = model.addVar(0.0, GRB.INFINITY, 0.0, GRB.CONTINUOUS, "y");
            GRBVar    z = model.addVar(0.0, GRB.INFINITY, 0.0, GRB.CONTINUOUS, "z");

            // Set objective
            GRBLinExpr obj = new GRBLinExpr();

        } catch (GRBException e) {
            System.out.println("Error code: "+ e.getErrorCode()+". " 
            + e.getMessage());
        } catch (Exception e) {
            System.out.println("Error");
        }
    }
}
obj.addTerm(1.0, x);
model.setObjective(obj, GRB.MAXIMIZE);

// Add linear constraint: x + y + z <= 10
GRBLinExpr expr = new GRBLinExpr();
expr.addTerm(1.0, x); expr.addTerm(1.0, y); expr.addTerm(1.0, z);
model.addConstr(expr, GRB.LESS_EQUAL, 10.0, "c0");

// Add bilinear inequality: x * y <= 2
GRBQuadExpr qexpr = new GRBQuadExpr();
qexpr.addTerm(1.0, x, y);
model.addQConstr(qexpr, GRB.LESS_EQUAL, 2.0, "bilinear0");

// Add bilinear equality: x * z + y * z == 1
qexpr = new GRBQuadExpr();
qexpr.addTerm(1.0, x, z);
qexpr.addTerm(1.0, y, z);
model.addQConstr(qexpr, GRB.EQUAL, 1.0, "bilinear1");

// First optimize() call will fail - need to set NonConvex to 2
try {
    model.optimize();
    assert false;
} catch (GRBException e) {
    System.out.println("Failed (as expected)");
}

// Change parameter and optimize again
model.set(GRB.IntParam.NonConvex, 2);
model.optimize();

System.out.println(x.get(GRB.StringAttr.VarName) + " " + x.get(GRB.DoubleAttr.X));
System.out.println(y.get(GRB.StringAttr.VarName) + " " + y.get(GRB.DoubleAttr.X));
System.out.println(z.get(GRB.StringAttr.VarName) + " " + z.get(GRB.DoubleAttr.X));
System.out.println("Obj: " + model.get(GRB.DoubleAttr.ObjVal) + " " + obj.getValue());
System.out.println();

// Constrain x to be integral and solve again
x.set(GRB.CharAttr.VType, GRB.INTEGER);
model.optimize();

System.out.println(x.get(GRB.StringAttr.VarName) + " " + x.get(GRB.DoubleAttr.X));
System.out.println(y.get(GRB.StringAttr.VarName) + " " + y.get(GRB.DoubleAttr.X));
System.out.println(z.get(GRB.StringAttr.VarName) + " " + z.get(GRB.DoubleAttr.X));
System.out.println("Obj: " + model.get(GRB.DoubleAttr.ObjVal) + " " + obj.getValue());
System.out.println();
// Dispose of model and environment
model.dispose();
env.dispose();

} catch (GRBException e) {
System.out.println("Error code: " + e.getErrorCode() + ". " + e.getMessage());
}
}

**Callback.java**

/* Copyright 2020, Gurobi Optimization, LLC */

/*
This example reads a model from a file, sets up a callback that monitors optimization progress and implements a custom termination strategy, and outputs progress information to the screen and to a log file.

The termination strategy implemented in this callback stops the optimization of a MIP model once at least one of the following two conditions have been satisfied:
  1) The optimality gap is less than 10%
  2) At least 10000 nodes have been explored, and an integer feasible solution has been found.

Note that termination is normally handled through Gurobi parameters (MIPGap, NodeLimit, etc.). You should only use a callback for termination if the available parameters don't capture your desired termination criterion.
*/

import gurobi.*;
import java.io.FileWriter;
import java.io.IOException;

public class Callback extends GRBCallback {
    private double lastiter;
    private double lastnode;
    private GRBVar[] vars;
    private FileWriter logfile;

    public Callback(GRBVar[] xvars, FileWriter xlogfile) {
        lastiter = lastnode = -GRB.INFINITY;
        vars = xvars;
        logfile = xlogfile;
    }
}
protected void callback() {
    try {
        if (where == GRB.CB_POLLING) {
            // Ignore polling callback
        } else if (where == GRB.CB_PRESOLVE) {
            // Presolve callback
            int cdels = getIntInfo(GRB.CB_PRE_COLDEL);
            int rdels = getIntInfo(GRB.CB_PRE_ROWDEL);
            if (cdels != 0 || rdels != 0) {
                System.out.println(cdels + " columns and " + rdels + " rows are removed");
            }
        } else if (where == GRB.CB_SIMPLEX) {
            // Simplex callback
            double itcnt = getDoubleInfo(GRB.CB_SPX_ITRCNT);
            if (itcnt - lastiter >= 100) {
                lastiter = itcnt;
                double obj = getDoubleInfo(GRB.CB_SPX_OBJVAL);
                int ispert = getIntInfo(GRB.CB_SPX_ISPERT);
                double pinf = getDoubleInfo(GRB.CB_SPX_PRIMINF);
                double dinf = getDoubleInfo(GRB.CB_SPX_DUALINF);
                char ch;
                if (ispert == 0) ch = ' ';
                else if (ispert == 1) ch = 'S';
                else ch = 'P';
                System.out.println(itcnt + " " + obj + ch + " " + pinf + " " + dinf);
            }
        } else if (where == GRB.CB_MIP) {
            // General MIP callback
            double nodecnt = getDoubleInfo(GRB.CB_MIP_NODCNT);
            double objbst = getDoubleInfo(GRB.CB_MIP_OBJBST);
            double objbnd = getDoubleInfo(GRB.CB_MIP_OBJBND);
            int solcnt = getIntInfo(GRB.CB_MIP_SOLCNT);
            if (nodecnt - lastnode >= 100) {
                lastnode = nodecnt;
                int actnodes = (int) getDoubleInfo(GRB.CB_MIP_NODLFT);
                int itcnt = (int) getDoubleInfo(GRB.CB_MIP_ITRCNT);
                int cutcnt = getDoubleInfo(GRB.CB_MIP_CUTCNT);
                System.out.println(nodecnt + " " + actnodes + " " + itcnt + " " + objbst + " " + objbnd + " " + solcnt + " " + cutcnt);
            }
            if (Math.abs(objbst - objbnd) < 0.1 * (1.0 + Math.abs(objbst))) {
                System.out.println("Stop early - 10% gap achieved");
                abort();
            }
            if (nodecnt >= 10000 && solcnt > 0) {
                System.out.println("Stop early - 10000 nodes explored");
                abort();
            }
        } else if (where == GRB.CB_MIPSOL) {
            // MIP solution callback
            int nodecnt = (int) getDoubleInfo(GRB.CB_MIPSOL_NODCNT);
            double obj = getDoubleInfo(GRB.CB_MIPSOL_OBJ);
```java
int solcnt = getIntInfo(GRB.CB_MIPSOL_SOLCNT);
double[] x = getSolution(vars);
System.out.println("**** New solution at node " + nodecnt + ", obj " + obj + ", sol " + solcnt + ", x[0] = " + x[0] + " ****");
}
else if (where == GRB.CB_MIPNODE) {
    // MIP node callback
    System.out.println("**** New node ****");
    if (getIntInfo(GRB.CB_MIPNODE_STATUS) == GRB.OPTIMAL) {
        double[] x = getNodeRel(vars);
        setSolution(vars, x);
    }
}
else if (where == GRB.CB_BARRIER) {
    // Barrier callback
    int itcnt = getIntInfo(GRB.CB_BARRIER_ITRCNT);
double primobj = getDoubleInfo(GRB.CB_BARRIER_PRIMOBJ);
double dualobj = getDoubleInfo(GRB.CB_BARRIER_DUALOBJ);
double priminf = getDoubleInfo(GRB.CB_BARRIER_PRIMINF);
double dualinf = getDoubleInfo(GRB.CB_BARRIER_DUALINF);
double cmpl = getDoubleInfo(GRB.CB_BARRIER_COMPL);
    System.out.println(itcnt + " " + primobj + " " + dualobj + " " +
    priminf + " " + dualinf + " " + cmpl);
}
else if (where == GRB.CB_MESSAGE) {
    // Message callback
    String msg = getStringInfo(GRB.CB_MSG_STRING);
    if (msg != null) logfile.write(msg);
}

catch (GRBException e) {
    System.out.println("Error code: " + e.getErrorCode());
    System.out.println(e.getMessage());
e.printStackTrace();
}
catch (Exception e) {
    System.out.println("Error during callback");
e.printStackTrace();
}
}

public static void main(String[] args) {
if (args.length < 1) {
    System.out.println("Usage: java Callback filename");
    System.exit(1);
}

FileWriter logfile = null;

try {
    // Create environment
    GRBEnv env = new GRBEnv();

    // Read model from file
    GRBModel model = new GRBModel(env, args[0]);

    // Turn off display and heuristics
    model.set(GRB.IntParam.OutputFlag, 0);
    model.set(GRB.DoubleParam.Heuristics, 0.0);
}
```
// Open log file
logfile = new FileWriter("cb.log");

// Create a callback object and associate it with the model
GRBVar[] vars = model.getVars();
Callback cb = new Callback(vars, logfile);
model.setCallback(cb);

// Solve model and capture solution information
model.optimize();

System.out.println("\n");
System.out.println("Optimization complete");
if (model.get(GRB.IntAttr.SolCount) == 0) {
    System.out.println("No solution found, optimization status = "
        + model.get(GRB.IntAttr.Status));
} else {
    System.out.println("Solution found, objective = "
        + model.get(GRB.DoubleAttr.ObjVal));

    String[] vnames = model.get(GRB.StringAttr.VarName, vars);
    double[] x = model.get(GRB.DoubleAttr.X, vars);

    for (int j = 0; j < vars.length; j++) {
        if (x[j] != 0.0) System.out.println(vnames[j] + " = " + x[j]);
    }
}

// Dispose of model and environment
model.dispose();
env.dispose();

} catch (GRBException e) {
    System.out.println("Error code: " + e.getErrorCode());
    System.out.println(e.getMessage());
    e.printStackTrace();
} catch (Exception e) {
    System.out.println("Error during optimization");
    e.printStackTrace();
} finally {
    // Close log file
    if (logfile != null) {
        try { logfile.close(); } catch (IOException e) {}
    }
}

Dense.java

/* Copyright 2020, Gurobi Optimization, LLC */

/* This example formulates and solves the following simple QP model:
minimize \quad x + y + x^2 + x*y + y^2 + y*z + z^2
subject to \quad x + 2 y + 3 z \geq 4
\quad x + y \geq 1
\quad x, y, z \text{ non-negative}

The example illustrates the use of dense matrices to store $A$ and $Q$ (and dense vectors for the other relevant data). We don’t recommend that you use dense matrices, but this example may be helpful if you already have your data in this format.

```java
import gurobi.*;

public class Dense {

    protected static boolean dense_optimize(GRBEnv env, int rows, int cols, double[] c, // linear portion of objective function
                                            double[][] Q, // quadratic portion of objective function
                                            double[][] A, // constraint matrix
                                            char[] sense, // constraint senses
                                            double[] rhs, // RHS vector
                                            double[] lb, // variable lower bounds
                                            double[] ub, // variable upper bounds
                                            char[] vtype, // variable types (continuous, binary, etc.)
                                            double[] solution) {
        boolean success = false;

        try {
            GRBModel model = new GRBModel(env);

            // Add variables to the model
            GRBVar[] vars = model.addVars(lb, ub, null, vtype, null);

            // Populate A matrix
            for (int i = 0; i < rows; i++) {
                GRBLinExpr expr = new GRBLinExpr();
                for (int j = 0; j < cols; j++)
                    if (A[i][j] != 0)
                        expr.addTerm(A[i][j], vars[j]);
                model.addConstr(expr, sense[i], rhs[i], "");
            }

            // Populate objective
            GRBQuadExpr obj = new GRBQuadExpr();
            if (Q != null) {
                for (int i = 0; i < cols; i++)
                    for (int j = 0; j < cols; j++)
                        if (Q[i][j] != 0)
                            obj.addTerm(Q[i][j], vars[i], vars[j]);
            }

            model.setObjective(obj, GRB.MINIMIZE);
            model.optimize();
            success = model.get(GRB.IntAttr.Status) == GRB.OPTIMAL;
        } catch (GRBException e) {
            System.out.println(e.getMessage());
        }

        return success;
    }
}
```
for (int j = 0; j < cols; j++)
    if (c[j] != 0)
        obj.addTerm(c[j], vars[j]);
model.setObjective(obj);

// Solve model
model.optimize();

// Extract solution
if (model.get(GRB.IntAttr.Status) == GRB.Status.OPTIMAL) {
    success = true;
    for (int j = 0; j < cols; j++)
        solution[j] = vars[j].get(GRB.DoubleAttr.X);
}
model.dispose();

} catch (GRBException e) {
    System.out.println("Error code: " + e.getErrorCode() + ". " +
        e.getMessage());
    e.printStackTrace();
}
return success;

public static void main(String[] args) {
    try {
        GRBEnv env = new GRBEnv();
        double c[] = new double[] {1, 1, 0};
        double Q[][] = new double[3][3] {{1, 1, 0}, {0, 1, 1}, {0, 0, 1}};
        double A[][] = new double[2][3] {{1, 2, 3}, {1, 1, 0}};
        char sense[] = new char[] {'>', '>'};
        double rhs[] = new double[] {4, 1};
        double lb[] = new double[] {0, 0, 0};
        boolean success;
        double sol[] = new double[3];

        success = dense_optimize(env, 2, 3, c, Q, A, sense, rhs,
            lb, null, null, null, sol);

        if (success) {
        }

        // Dispose of environment
        env.dispose();
    } catch (GRBException e) {
        System.out.println("Error code: " + e.getErrorCode() + ". " +
            e.getMessage());
    }

}
Diet.java

/* Copyright 2020, Gurobi Optimization, LLC */
/* Solve the classic diet model, showing how to add constraints
to an existing model. */

import gurobi.*;

public class Diet {

    public static void main(String[] args) {
        try {

            // Nutrition guidelines, based on
            // USDA Dietary Guidelines for Americans, 2005
            String Categories[] =
                new String[] { "calories", "protein", "fat", "sodium" };
            int nCategories = Categories.length;
            double minNutrition[] = new double[] { 1800, 91, 0, 0 };
            double maxNutrition[] = new double[] { 2200, GRB.INFINITY, 65, 1779 };

            // Set of foods
            String Foods[] =
                new String[] { "hamburger", "chicken", "hot dog", "fries",
                               "macaroni", "pizza", "salad", "milk", "ice cream" };
            int nFoods = Foods.length;
            double cost[] =
                new double[] { 2.49, 2.89, 1.50, 1.89, 2.09, 1.99, 2.49, 0.89, 1.59 };

            // Nutrition values for the foods
            double nutritionValues[][] = new double[][4] {
                { 410, 24, 26, 730 }, // hamburger
                { 420, 32, 10, 1190 }, // chicken
                { 560, 20, 32, 1800 }, // hot dog
                { 380, 4, 19, 270 }, // fries
                { 320, 12, 10, 930 }, // macaroni
                { 320, 15, 12, 820 }, // pizza
                { 320, 31, 12, 1230 }, // salad
                { 100, 8, 2.5, 125 }, // milk
                { 330, 8, 10, 180 } // ice cream
            };

            // Model
            GRBEnv env = new GRBEnv();
            GRBModel model = new GRBModel(env);
            model.set(GRB.StringAttr.ModelName, "diet");
// Create decision variables for the nutrition information,  
// which we limit via bounds  
GRBVar[] nutrition = new GRBVar[nCategories];  
for (int i = 0; i < nCategories; ++i) {  
    nutrition[i] =  
        model.addVar(minNutrition[i], maxNutrition[i], 0, GRB.CONTINUOUS,  
                      Categories[i]);  
}  

// Create decision variables for the foods to buy  
GRBVar[] buy = new GRBVar[nFoods];  
for (int j = 0; j < nFoods; ++j) {  
    buy[j] =  
        model.addVar(0, GRB.INFINITY, cost[j], GRB.CONTINUOUS, Foods[j]);  
}  

// The objective is to minimize the costs  
model.set(GRB.IntAttr.ModelSense, GRB.MINIMIZE);  

// Nutrition constraints  
for (int i = 0; i < nCategories; ++i) {  
    GRBLinExpr ntot = new GRBLinExpr();  
    for (int j = 0; j < nFoods; ++j) {  
        ntot.addTerm(nutritionValues[j][i], buy[j]);  
    }  
    model.addConstr(ntot, GRB.EQUAL, nutrition[i], Categories[i]);  
}  

// Solve  
model.optimize();  
printSolution(model, buy, nutrition);  
System.out.println(" JSON solution:" + model.getJSONSolution());  

System.out.println(" Adding constraint: at most 6 servings of dairy");  
GRBLinExpr lhs = new GRBLinExpr();  
lhs.addTerm(1.0, buy[7]);  
lhs.addTerm(1.0, buy[8]);  
model.addConstr(lhs, GRB.LESS_EQUAL, 6.0, "limit_dairy");  

// Solve  
model.optimize();  
printSolution(model, buy, nutrition);  
System.out.println(" JSON solution:" + model.getJSONSolution());  

// Dispose of model and environment  
model.dispose();  
env.dispose();  
}  

private static void printSolution(GRBModel model, GRBVar[] buy,  
        GRBVar[] nutrition) throws GRBException {

}
if (model.get(GRB.IntAttr.Status) == GRB.Status.OPTIMAL) {
    System.out.println("\nCost: " + model.get(GRB.DoubleAttr.ObjVal));
    System.out.println("\nBuy:");
    for (int j = 0; j < buy.length; ++j) {
        if (buy[j].get(GRB.DoubleAttr.X) > 0.0001) {
            System.out.println(buy[j].get(GRB.StringAttr.VarName) + " " +
                buy[j].get(GRB.DoubleAttr.X));
        }
    }
    System.out.println("\nNutrition:");
    for (int i = 0; i < nutrition.length; ++i) {
        System.out.println(nutrition[i].get(GRB.StringAttr.VarName) + " " +
            nutrition[i].get(GRB.DoubleAttr.X));
    }
} else {
    System.out.println("No solution");
}

Facility.java

/* Copyright 2020, Gurobi Optimization, LLC */

/* Facility location: a company currently ships its product from 5 plants
to 4 warehouses. It is considering closing some plants to reduce
costs. What plant(s) should the company close, in order to minimize
transportation and fixed costs?

Based on an example from Frontline Systems:
http://www.solver.com/disfacility.htm
Used with permission.
*/

import gurobi.*;

class Facility {

    public static void main(String[] args) {
        try {
            // Warehouse demand in thousands of units
            double Demand[] = new double[] { 15, 18, 14, 20 };

            // Plant capacity in thousands of units
            double Capacity[] = new double[] { 20, 22, 17, 19, 18 };

            // Fixed costs for each plant
            double FixedCosts[] =
                new double[] { 12000, 15000, 17000, 13000, 16000 };

            // Transportation costs per thousand units
            double TransCosts[][] =
                new double[][] {
                    { 4000, 2000, 3000, 2500, 4500 },
                    { 2500, 2600, 3400, 3000, 4000 },
                    { 1200, 1800, 2600, 4100, 3000 },
                };

            Facility model =
                new Facility(Demand, Capacity, FixedCosts, TransCosts);

        } catch (Exception e) {
            System.out.println(e.getMessage());
        }
    }
}

import gurobi.*;

class Facility {

    public static void main(String[] args) {
        try {
            // Warehouse demand in thousands of units
            double Demand[] = new double[] { 15, 18, 14, 20 };

            // Plant capacity in thousands of units
            double Capacity[] = new double[] { 20, 22, 17, 19, 18 };

            // Fixed costs for each plant
            double FixedCosts[] =
                new double[] { 12000, 15000, 17000, 13000, 16000 };

            // Transportation costs per thousand units
            double TransCosts[][] =
                new double[][] {
                    { 4000, 2000, 3000, 2500, 4500 },
                    { 2500, 2600, 3400, 3000, 4000 },
                    { 1200, 1800, 2600, 4100, 3000 },
                };

            Facility model =
                new Facility(Demand, Capacity, FixedCosts, TransCosts);

        } catch (Exception e) {
            System.out.println(e.getMessage());
        }
    }
}

import gurobi.*;

class Facility {

    public static void main(String[] args) {
        try {
            // Warehouse demand in thousands of units
            double Demand[] = new double[] { 15, 18, 14, 20 };

            // Plant capacity in thousands of units
            double Capacity[] = new double[] { 20, 22, 17, 19, 18 };

            // Fixed costs for each plant
            double FixedCosts[] =
                new double[] { 12000, 15000, 17000, 13000, 16000 };

            // Transportation costs per thousand units
            double TransCosts[][] =
                new double[][] {
                    { 4000, 2000, 3000, 2500, 4500 },
                    { 2500, 2600, 3400, 3000, 4000 },
                    { 1200, 1800, 2600, 4100, 3000 },
                };

            Facility model =
                new Facility(Demand, Capacity, FixedCosts, TransCosts);

        } catch (Exception e) {
            System.out.println(e.getMessage());
        }
    }
}
// Number of plants and warehouses
int nPlants = Capacity.length;
int nWarehouses = Demand.length;

// Model
GRBEnv env = new GRBEnv();
GRBModel model = new GRBModel(env);
model.set(GRB.StringAttr.ModelName, "facility");

// Plant open decision variables: open[p] == 1 if plant p is open.
GRBVar[] open = new GRBVar[nPlants];
for (int p = 0; p < nPlants; ++p) {
    open[p] = model.addVar(0, 1, FixedCosts[p], GRB.BINARY, "Open" + p);
}

// Transportation decision variables: how much to transport from
// a plant p to a warehouse w
GRBVar[][] transport = new GRBVar[nWarehouses][nPlants];
for (int w = 0; w < nWarehouses; ++w) {
    for (int p = 0; p < nPlants; ++p) {
        transport[w][p] =
            model.addVar(0, GRB.INFINITY, TransCosts[w][p], GRB.CONTINUOUS,
                         "Trans" + p + "." + w);
    }
}

// The objective is to minimize the total fixed and variable costs
model.set(GRB.IntAttr.ModelSense, GRB.MINIMIZE);

// Production constraints
// Note that the right-hand limit sets the production to zero if
// the plant is closed
for (int p = 0; p < nPlants; ++p) {
    GRBLinExpr ptot = new GRBLinExpr();
    for (int w = 0; w < nWarehouses; ++w) {
        ptot.addTerm(1.0, transport[w][p]);
    }
    GRBLinExpr limit = new GRBLinExpr();
    limit.addTerm(Capacity[p], open[p]);
    model.addConstr(ptot, GRB.LESS_EQUAL, limit, "Capacity" + p);
}

// Demand constraints
for (int w = 0; w < nWarehouses; ++w) {
    GRBLinExpr dtot = new GRBLinExpr();
    for (int p = 0; p < nPlants; ++p) {
        dtot.addTerm(1.0, transport[w][p]);
    }
    model.addConstr(dtot, GRB.EQUAL, Demand[w], "Demand" + w);
}

// Guess at the starting point: close the plant with the highest
// fixed costs; open all others
// First, open all plants
for (int p = 0; p < nPlants; ++p) {
    open[p].set(GRB.DoubleAttr.Start, 1.0);
}

// Now close the plant with the highest fixed cost
System.out.println("Initial guess:");
double maxFixed = -GRB.INFINITY;
for (int p = 0; p < nPlants; ++p) {
    if (FixedCosts[p] > maxFixed) {
        maxFixed = FixedCosts[p];
    }
}
for (int p = 0; p < nPlants; ++p) {
    if (FixedCosts[p] == maxFixed) {
        open[p].set(GRB.DoubleAttr.Start, 0.0);
        System.out.println("Closing plant " + p + "");
        break;
    }
}

// Use barrier to solve root relaxation
model.set(GRB.IntParam.Method, GRB.METHOD_BARRIER);

// Solve
model.optimize();

// Print solution
System.out.println("TOTAL COSTS: " + model.get(GRB.DoubleAttr.ObjVal));
System.out.println("SOLUTION:");
for (int p = 0; p < nPlants; ++p) {
    if (open[p].get(GRB.DoubleAttr.X) > 0.99) {
        System.out.println("Plant " + p + " open:"");
        for (int w = 0; w < nWarehouses; ++w) {
            if (transport[w][p].get(GRB.DoubleAttr.X) > 0.0001) {
                System.out.println(" Transport " + transport[w][p].get(GRB.DoubleAttr.X) + " units to warehouse " + w);
            }
        }
    } else {
        System.out.println("Plant " + p + " closed!");
    }
}

// Dispose of model and environment
model.dispose();
env.dispose();

} catch (GRBException e) {
    System.out.println("Error code: " + e.getErrorCode() + ". " + e.getMessage());
}
}
import gurobi.*;

public class Feasopt {
    public static void main(String[] args) {
        if (args.length < 1) {
            System.out.println("Usage: java Feasopt filename");
            System.exit(1);
        }

        try {
            GRBEnv env = new GRBEnv();
            GRBModel feasmodel = new GRBModel(env, args[0]);

            // Create a copy to use FeasRelax feature later */
            GRBModel feasmodel1 = new GRBModel(feasmodel);

            // Clear objective
            feasmodel.setObjective(new GRBLinExpr());

            // Add slack variables
            GRBConstr[] c = feasmodel.getConstrs();
            for (int i = 0; i < c.length; ++i) {
                char sense = c[i].get(GRB.CharAttr.Sense);
                if (sense != '>') {
                    GRBConstr[] constrs = new GRBConstr[] { c[i] };
                    double[] coeffs = new double[] { -1 };
                    feasmodel.addVar(0.0, GRB.INFINITY, 1.0, GRB.CONTINUOUS, constrs,
                                     coeffs, "ArtN_" +
                                     c[i].get(GRB.StringAttr.ConstrName));
                }
                if (sense != '<') {
                    GRBConstr[] constrs = new GRBConstr[] { c[i] };
                    double[] coeffs = new double[] { 1 };
                    feasmodel.addVar(0.0, GRB.INFINITY, 1.0, GRB.CONTINUOUS, constrs,
                                     coeffs, "ArtP_" +
                                     c[i].get(GRB.StringAttr.ConstrName));
                }
            }

            // Optimize modified model
            feasmodel.optimize();
            feasmodel.write("feasopt.lp");
        }
    }
}
// use FeasRelax feature */
feasmodel1.feasRelax(GRB.FEASRELAX_LINEAR, true, false, true);
feasmodel1.write("feasopt1.lp");
feasmodel1.optimize();

// Dispose of model and environment
feasmodel1.dispose();
feasmmodel.dispose();
env.dispose();

} catch (GRBException e) {
    System.out.println("Error code: " + e.getErrorCode() + ". " + 
e.getMessage());
}

Fixanddive.java

/* Copyright 2020, Gurobi Optimization, LLC */

/* Implement a simple MIP heuristic. Relax the model,
sort variables based on fractionality, and fix the 25% of
the fractional variables that are closest to integer variables.
Repeat until either the relaxation is integer feasible or
linearly infeasible. */

import gurobi.*;
import java.util.*;

public class Fixanddive {
    public static void main(String[] args) {

        // Comparison class used to sort variable list based on relaxation
        // fractionality

        class FractionalCompare implements Comparator<GRBVar> {
            public int compare(GRBVar v1, GRBVar v2) {
                try {
                    double sol1 = Math.abs(v1.get(GRB.DoubleAttr.X));
                    double sol2 = Math.abs(v2.get(GRB.DoubleAttr.X));
                    double frac1 = Math.abs(sol1 - Math.floor(sol1 + 0.5));
                    double frac2 = Math.abs(sol2 - Math.floor(sol2 + 0.5));
                    if (frac1 < frac2) {
                        return -1;
                    } else if (frac1 == frac2) {
                        return 0;
                    } else {
                        return 1;
                    }
                } catch (GRBException e) {
                    System.out.println("Error code: " + e.getCode() + ". " + 
e.getMessage());
                }
                return 0;
            }
        }
if (args.length < 1) {
    System.out.println("Usage: java Fixanddive filename");
    System.exit(1);
}

try {
    // Read model
    GRBEnv env = new GRBEnv();
    GRBModel model = new GRBModel(env, args[0]);

    // Collect integer variables and relax them
    ArrayList<GRBVar> intvars = new ArrayList<GRBVar>();
    for (GRBVar v : model.getVars()) {
        if (v.get(GRB.CharAttr.VType) != GRB.CONTINUOUS) {
            intvars.add(v);
            v.set(GRB.CharAttr.VType, GRB.CONTINUOUS);
        }
    }
    model.setIntParam(GRB.IntParam.OutputFlag, 0);
    model.optimize();

    // Perform multiple iterations. In each iteration, identify the first
    // quartile of integer variables that are closest to an integer value
    // in the relaxation, fix them to the nearest integer, and repeat.
    for (int iter = 0; iter < 1000; ++iter) {
        // create a list of fractional variables, sorted in order of
        // increasing distance from the relaxation solution to the nearest
        // integer value
        ArrayList<GRBVar> fractional = new ArrayList<GRBVar>();
        for (GRBVar v : intvars) {
            double sol = Math.abs(v.get(GRB.DoubleAttr.X));
            if (Math.abs(sol - Math.floor(sol + 0.5)) > 1e-5) {
                fractional.add(v);
            }
        }
        System.out.println("Iteration ", obj +
            model.get(GRB.DoubleAttr.ObjVal) + ", fractional " +
            fractional.size());

        if (fractional.size() == 0) {
            System.out.println("Found feasible solution - objective " +
                model.get(GRB.DoubleAttr.ObjVal));
            break;
        }

        // Fix the first quartile to the nearest integer value
        Collections.sort(fractional, new FractionalCompare());
    }
}
```java
int nfix = Math.max(fractional.size() / 4, 1);
for (int i = 0; i < nfix; ++i) {
    GRBVar v = fractional.get(i);
    double fixval = Math.floor(v.get(GRB.DoubleAttr.X) + 0.5);
    v.set(GRB.DoubleAttr.LB, fixval);
    v.set(GRB.DoubleAttr.UB, fixval);
    System.out.println(" Fix " + v.get(GRB.StringAttr.VarName) + " to " + fixval + " ( rel " + v.get(GRB.DoubleAttr.X) + " )");
}
model.optimize();

// Check optimization result
if (model.get(GRB.IntAttr.Status) != GRB.Status.OPTIMAL) {
    System.out.println("Relaxation is infeasible");
    break;
}

// Dispose of model and environment
model.dispose();
env.dispose();
}
}
}
```

GCPWL.java

/* Copyright 2020, Gurobi Optimization, LLC
This example formulates and solves the following simple model
with PWL constraints:
maximize
sum c[j] * x[j]
subject to
sum A[i,j] * x[j] <= 0, for i = 0, ..., m-1
sum y[j] <= 3
y[j] = pwl(x[j]), for j = 0, ..., n-1
x[j] free, y[j] >= 0, for j = 0, ..., n-1
where pwl(x) = 0, if x = 0
      = 1+| x|, if x != 0
Note
1. sum pwl(x[j]) <= b is to bound x vector and also to favor sparse x vector. Here b = 3 means that at most two x[j] can be nonzero and if two, then sum x[j] <= 1
2. pwl(x) jumps from 1 to 0 and from 0 to 1, if x moves from negative 0 to 0, then to positive 0, so we need three points at x = 0. x has infinite bounds on both sides, the piece defined with two points (-1, 2) and (0, 1) can

244
extend x to -infinite. Overall we can use five points (-1, 2), (0, 1), (0, 0), (0, 1) and (1, 2) to define y = pwl(x)

*/

import gurobi.*;
import java.util.*;

public class GCPWL {

    public static void main(String[] args) {
        try {
            int n = 5;
            int m = 5;
            double c[] = { 0.5 , 0.8 , 0.5 , 0.1 , -1 };
            double A[][] = { {0 , 0 , 0 , 1 , -1},
                             {0 , 0 , 1 , 1 , -1},
                             {1 , 1 , 0 , 0 , -1},
                             {1 , 0 , 1 , 0 , -1},
                             {1 , 0 , 0 , 1 , -1} };
            double xpts[] = {-1, 0, 0, 0, 1};
            double ypts[] = {2 , 1, 0, 1, 2};

            // Env and model
            GRBEnv env = new GRBEnv();
            GRBModel model = new GRBModel(env);
            model.set(GRB.StringAttr.ModelName, "GCPWL");

            // Add variables, set bounds and obj coefficients
            GRBVar[] x = model.addVars(n, GRB.CONTINUOUS);
            for (int i = 0; i < n; i++) {
                x[i].set(GRB.DoubleAttr.LB, -GRB.INFINITY);
                x[i].set(GRB.DoubleAttr.Obj, c[i]);
            }
            GRBVar[] y = model.addVars(n, GRB.CONTINUOUS);

            // Set objective to maximize
            model.set(GRB.IntAttr.ModelSense, GRB.MAXIMIZE);

            // Add linear constraints
            for (int i = 0; i < m; i++) {
                GRBLinExpr le = new GRBLinExpr();
                for (int j = 0; j < n; j++) {
                    le.addTerm(A[i][j], x[j]);
                }
                model.addConstr(le, GRB.LESS_EQUAL, 0, "cx" + i);
            }
            GRBLinExpr le1 = new GRBLinExpr();
            for (int j = 0; j < n; j++) {
                le1.addTerm(1.0, y[j]);
            }
            model.addConstr(le1, GRB.LESS_EQUAL, 3, "cy");

            // Add piecewise constraints
            for (int j = 0; j < n; j++) {
            }
        }
    }
}
```java
model.addGenConstrPWL(x[j], y[j], xpts, ypts, "pwl" + j);
}

// Optimize model
model.optimize();

for (int j = 0; j < n; j++) {
    System.out.println("x[" + j + "] = " + x[j].get(GRB.DoubleAttr.X));
}
System.out.println("Obj: " + model.get(GRB.DoubleAttr.ObjVal));

// Dispose of model and environment
model.dispose();
env.dispose();
}
}
```
private static double f(double u) { return Math.exp(u); }
private static double g(double u) { return Math.sqrt(u); }

private static void printsol(GRBModel m, GRBVar x, GRBVar y, GRBVar u, GRBVar v)
throws GRBException {
assert(m.get(GRB.IntAttr.Status) == GRB.OPTIMAL);
System.out.println("x = " + x.get(GRB.DoubleAttr.X) + ", u = " + u.get(GRB.DoubleAttr.X));
System.out.println("y = " + y.get(GRB.DoubleAttr.X) + ", v = " + v.get(GRB.DoubleAttr.X));
System.out.println("Obj = " + m.get(GRB.DoubleAttr.ObjVal));

// Calculate violation of exp(x) + 4 sqrt(y) <= 9
double vio = f(x.get(GRB.DoubleAttr.X)) + 4 * g(y.get(GRB.DoubleAttr.X)) - 9;
if (vio < 0.0) vio = 0.0;
System.out.println("Vio = " + vio);
}

public static void main(String[] args) {
try {

    // Create environment
    GRBEnv env = new GRBEnv();
    // Create a new m
    GRBModel m = new GRBModel(env);
    double lb = 0.0, ub = GRB.INFINITY;
    GRBVar x = m.addVar(lb, ub, 0.0, GRB.CONTINUOUS, "x");
    GRBVar y = m.addVar(lb, ub, 0.0, GRB.CONTINUOUS, "y");
    GRBVar u = m.addVar(lb, ub, 0.0, GRB.CONTINUOUS, "u");
    GRBVar v = m.addVar(lb, ub, 0.0, GRB.CONTINUOUS, "v");

    // Set objective
    GRBLinExpr obj = new GRBLinExpr();
    obj.addTerm(2.0, x); obj.addTerm(1.0, y);
    m.setObjective(obj, GRB.MAXIMIZE);

    // Add linear constraint
    GRBLinExpr expr = new GRBLinExpr();
    expr.addTerm(1.0, u); expr.addTerm(4.0, v);
    m.addConstr(expr, GRB.LESS_EQUAL, 9.0, "l1");

    // Approach 1) PWL constraint approach
    double intv = 1e-3;
    double xmax = Math.log(9.0);
    int len = (int) Math.ceil(xmax/intv) + 1;
    double[] xpts = new double[len];
    double[] upts = new double[len];
    for (int i = 0; i < len; i++) {
xpts[i] = i*intv;
upts[i] = f(i*intv);
}
GRBGenConstr gc1 = m.addGenConstrPWL(x, u, xpts, upts, "gc1");

double ymax = (9.0/4.0)*(9.0/4.0);
len = (int)Math.ceil(ymax/intv) + 1;
double[] ypts = new double[len];
double[] vpts = new double[len];
for (int i = 0; i < len; i++) {
ypts[i] = i*intv;
vpts[i] = g(i*intv);
}
GRBGenConstr gc2 = m.addGenConstrPWL(y, v, ypts, vpts, "gc2");

// Optimize the model and print solution
m.optimize();
printsol(m, x, y, u, v);

// Approach 2) General function constraint approach with auto PWL
// translation by Gurobi

// restore unsolved state and get rid of PWL constraints
m.reset();
m.remove(gc1);
m.remove(gc2);
m.update();

GRBGenConstr gcf1 = m.addGenConstrExp(x, u, "gcf1", null);
GRBGenConstr gcf2 = m.addGenConstrPow(y, v, 0.5, "gcf2", "");
m.set(GRB.DoubleParam.FuncPieceLength, 1e-3);

// Optimize the model and print solution
m.optimize();
printsol(m, x, y, u, v);

// Zoom in, use optimal solution to reduce the ranges and use a smaller
// pclen=1e-5 to solve it

double xval = x.get(GRB.DoubleAttr.X);
double yval = y.get(GRB.DoubleAttr.X);
x.set(GRB.DoubleAttr.LB, Math.max(x.get(GRB.DoubleAttr.LB), xval-0.01));
x.set(GRB.DoubleAttr.UB, Math.min(x.get(GRB.DoubleAttr.UB), xval+0.01));
y.set(GRB.DoubleAttr.LB, Math.max(y.get(GRB.DoubleAttr.LB), yval-0.01));
y.set(GRB.DoubleAttr.UB, Math.min(y.get(GRB.DoubleAttr.UB), yval+0.01));
m.update();
m.reset();
m.set(GRB.DoubleParam.FuncPieceLength, 1e-5);

// Optimize the model and print solution
m. optimize();
printsol(m, x, y, u, v);

// Dispose of model and environment
m.dispose();
env.dispose();

} catch (GRBException e) {
    System.out.println("Error code: " + e.getErrorCode() + ". " + e.getMessage());
}

} Genconstr.java

/*******************************************************************************/

import gurobi.*;

public class Genconstr {

    public static final int n = 4;
    public static final int NLITERALS = 4; // same as n
    public static final int NCLAUSES = 8;
    public static final int NOBJ = 2;

    /* Copyright 2020, Gurobi Optimization, LLC */
    /* In this example we show the use of general constraints for modeling
      some common expressions. We use as an example a SAT-problem where we
      want to see if it is possible to satisfy at least four (or all) clauses
      of the logical for

      L = (x0 or ~x1 or x2) and (x1 or ~x2 or x3) and
          (x2 or ~x3 or x0) and (x3 or ~x0 or x1) and
          (~x0 or ~x1 or x2) and (~x1 or ~x2 or x3) and
          (~x2 or ~x3 or x0) and (~x3 or ~x0 or x1)

      We do this by introducing two variables for each literal (itself and its
      negated value), a variable for each clause, and then two
      variables for indicating if we can satisfy four, and another to identify
      the minimum of the clauses (so if it one, we can satisfy all clauses)
      and put these two variables in the objective.
    i.e. the Objective function will be

    maximize Obj0 + Obj1

    Obj0 = MIN(Clause1, ... , Clause8)
    Obj1 = 1 -> Clause1 + ... + Clause8 >= 4

    thus, the objective value will be two if and only if we can satisfy all
    clauses; one if and only if at least four clauses can be satisfied, and
    zero otherwise.
    */
public static void main(String[] args) {

    try {
        // Example data:
        // e.g. {0, n+1, 2} means clause (x0 or -x1 or x2)
        int Clauses[][] = new int[][]
        {{ 0, n+1, 2}, { 1, n+2, 3},
         { 2, n+3, 0}, { 3, n+0, 1},
         {n+0, n+1, 2}, {n+1, n+2, 3},
         {n+2, n+3, 0}, {n+3, n+0, 1}};

        int i, status, nSolutions;

        // Create environment
        GRBEnv env = new GRBEnv("Genconstr.log");

        // Create initial model
        GRBModel model = new GRBModel(env);
        model.set(GRB.StringAttr.ModelName, "Genconstr");

        // Initialize decision variables and objective
        GRBVar[] Lit = new GRBVar[NLITERALS];
        GRBVar[] NotLit = new GRBVar[NLITERALS];
        for (i = 0; i < NLITERALS; i++) {
            Lit[i] = model.addVar(0.0, 1.0, 0.0, GRB.BINARY, "X" + String.valueOf(i));
            NotLit[i] = model.addVar(0.0, 1.0, 0.0, GRB.BINARY, "notX" + String.valueOf(i));
        }

        GRBVar[] Cla = new GRBVar[NCLAUSES];
        for (i = 0; i < NCLAUSES; i++) {
            Cla[i] = model.addVar(0.0, 1.0, 0.0, GRB.BINARY, "Clause" + String.valueOf(i));
        }

        GRBVar[] Obj = new GRBVar[NOBJ];
        for (i = 0; i < NOBJ; i++) {
            Obj[i] = model.addVar(0.0, 1.0, 1.0, GRB.BINARY, "Obj" + String.valueOf(i));
        }

        // Link Xi and notXi
        GRBLinExpr lhs;
        for (i = 0; i < NLITERALS; i++) {
            lhs = new GRBLinExpr();
            lhs.addTerm(1.0, Lit[i]);
            lhs.addTerm(1.0, NotLit[i]);
            model.addConstr(lhs, GRB.EQUAL, 1.0, "CNSTR_X" + String.valueOf(i));
        }

        // Link clauses and literals
        for (i = 0; i < NCLAUSES; i++) {
            GRBVar[] clause = new GRBVar[3];
            for (int j = 0; j < 3; j++) {
                if (Clauses[i][j] >= n) clause[j] = NotLit[Clauses[i][j]-n];
                else clause[j] = Lit[Clauses[i][j]];
            }
            model.addGenConstrOr(Cla[i], clause, "CNSTR_Clause" + String.valueOf(i));
        }
    }
// Link objs with clauses
model.addGenConstrMin(Obj[0], Cla, GRB.INFINITY, "CNSTR_Obj0");
lhs = new GRBLinExpr();
for (i = 0; i < NCLAUSES; i++) {
    lhs.addTerm(1.0, Cla[i]);
}model.addGenConstrIndicator(Obj[1], 1, lhs, GRB.GREATER_EQUAL, 4.0, "CNSTR_Obj1");

// Set global objective sense
model.set(GRB.IntAttr.ModelSense, GRB.MAXIMIZE);

// Save problem
model.write("Genconstr.mps");
model.write("Genconstr.lp");

// Optimize
model.optimize();

// Status checking
status = model.get(GRB.IntAttr.Status);
if (status == GRB.INF_OR_UNBD ||
    status == GRB.INFEASIBLE ||
    status == GRB.UNBOUNDED)
    System.out.println("The model cannot be solved " +
    "because it is infeasible or unbounded");
    System.exit(1);
if (status != GRB.OPTIMAL)
    System.out.println("Optimization was stopped with status " + status);
    System.exit(1);

// Print result
double objval = model.get(GRB.DoubleAttr.ObjVal);
if (objval > 1.9)
    System.out.println("Logical expression is satisfiable");
else if (objval > 0.9)
    System.out.println("At least four clauses can be satisfied");
else
    System.out.println("Not even three clauses can be satisfied");

// Dispose of model and environment
model.dispose();
env.dispose();

} catch (GRBException e) {
    System.out.println("Error code: " + e.getErrorCode() + ", " +
        e.getMessage());
}
}
/* Copyright 2020, Gurobi Optimization, LLC */

/* This example reads an LP model from a file and solves it.
   If the model is infeasible or unbounded, the example turns off
   presolve and solves the model again. If the model is infeasible,
   the example computes an Irreducible Inconsistent Subsystem (IIS),
   and writes it to a file */

import gurobi.*;

public class Lp {
    public static void main(String[] args) {
        if (args.length < 1) {
            System.out.println("Usage: java Lp filename");
            System.exit(1);
        }

        try {
            GRBEnv env = new GRBEnv();
            GRBModel model = new GRBModel(env, args[0]);

            model.optimize();

            int optimstatus = model.get(GRB.IntAttr.Status);

            if (optimstatus == GRB.Status.INF_OR_UNBD) {
                model.set(GRB.IntParam.Presolve, 0);
                model.optimize();
                optimstatus = model.get(GRB.IntAttr.Status);
            }

            if (optimstatus == GRB.Status.OPTIMAL) {
                double objval = model.get(GRB.DoubleAttr.ObjVal);
                System.out.println("Optimal objective: " + objval);
            } else if (optimstatus == GRB.Status.INFEASIBLE) {
                System.out.println("Model is infeasible");
                // Compute and write out IIS
                model.computeIIS();
                model.write("model.ilp");
            } else if (optimstatus == GRB.Status.UNBOUNDED) {
                System.out.println("Model is unbounded");
            } else {
                System.out.println("Optimization was stopped with status = " + optimstatus);
            }

            // Dispose of model and environment
            model.dispose();
            env.dispose();
        }
        catch (GRBException e) {
            System.out.println("Error code: " + e.getErrorCode() + ". " + e.getMessage());
        }
    }
}
Lpmethod.java

/* Copyright 2020, Gurobi Optimization, LLC */

/* Solve a model with different values of the Method parameter; 
   show which value gives the shortest solve time. */

import gurobi.*;

public class Lpmethod {

    public static void main(String[] args) {

        if (args.length < 1) {
            System.out.println("Usage: java Lpmethod filename");
            System.exit(1);
        }

        try {
            // Read model
            GRBEnv env = new GRBEnv();
            GRBModel model = new GRBModel(env, args[0]);

            // Solve the model with different values of Method
            int bestMethod = -1;
            double bestTime = model.get(GRB.DoubleParam.TimeLimit);
            for (int i = 0; i <= 2; ++i) {
                model.reset();
                model.set(GRB.IntParam.Method, i);
                model.optimize();
                if (model.get(GRB.IntAttr.Status) == GRB.Status.OPTIMAL) {
                    bestTime = model.get(GRB.DoubleAttr.Runtime);
                    bestMethod = i;
                    // Reduce the TimeLimit parameter to save time
                    // with other methods
                    model.set(GRB.DoubleParam.TimeLimit, bestTime);
                }
            }

            // Report which method was fastest
            if (bestMethod == -1) {
                System.out.println("Unable to solve this model");
            } else {
                System.out.println("Solved in " + bestTime + " seconds with Method: " + bestMethod);
            }

            // Dispose of model and environment
            model.dispose();
            env.dispose();
        }
    }
}

253
try {
    // Read model and determine whether it is an LP
    GRBEnv env = new GRBEnv();
    GRBModel model = new GRBModel(env, args[0]);
    if (model.get(GRB.IntAttr.IsMIP) != 0) {
        System.out.println("The model is not a linear program");
        System.exit(1);
    }
    model.optimize();

    int status = model.get(GRB.IntAttr.Status);
    if (status == GRB.Status.INF_OR_UNBD ||
        status == GRB.Status.UNFEASIBLE ||
        status == GRB.Status.UNBOUNDED) {
        System.out.println("The model cannot be solved because it is "+ "infeasible or unbounded");
        System.exit(1);
    }
    if (status != GRB.Status.OPTIMAL) {
        System.out.println("Optimization was stopped with status "+ status);
        System.exit(0);
    }

    // Find the smallest variable value
    double minVal = GRB.INFINITY;
    GRBVar minVar = null;
}}}
for (GRBVar v : model.getVars()) {
    double sol = v.get(GRB.DoubleAttr.X);
    if ((sol > 0.0001) && (sol < minVal) &&
        (v.get(GRB.DoubleAttr.LB) == 0.0)) {
        minVal = sol;
        minVar = v;
    }
}

System.out.println("\n*** Setting " +
        minVar.get(GRB.StringAttr.VarName) + " from " + minVal +
        " to zero ***\n");
minVar.set(GRB.DoubleAttr.UB, 0.0);

// Solve from this starting point
model.optimize();

// Save iteration & time info
double warmCount = model.get(GRB.DoubleAttr.IterCount);
double warmTime = model.get(GRB.DoubleAttr.Runtime);

// Reset the model and resolve
System.out.println("\n*** Resetting and solving "
        + "without an advanced start ***\n");
model.reset();
model.optimize();

double coldCount = model.get(GRB.DoubleAttr.IterCount);
double coldTime = model.get(GRB.DoubleAttr.Runtime);

System.out.println("\n*** Warm start: " + warmCount + " iterations, " +
        warmTime + " seconds\n");
System.out.println("*** Cold start: " + coldCount + " iterations, " +
        coldTime + " seconds\n");

// Dispose of model and environment
model.dispose();
env.dispose();

} catch (GRBException e) {
    System.out.println("Error code: " + e.getErrorCode() + ". " +
        e.getMessage());
}
}
import gurobi.*;

public class Mip1 {
    public static void main(String[] args) {
        try {
            // Create empty environment, set options, and start
            GRBEnv env = new GRBEnv(true);
            env.set("logFile", "mip1.log");
            env.start();

            // Create empty model
            GRBModel model = new GRBModel(env);

            // Create variables
            GRBVar x = model.addVar(0.0, 1.0, 0.0, GRB.BINARY, "x");
            GRBVar y = model.addVar(0.0, 1.0, 0.0, GRB.BINARY, "y");
            GRBVar z = model.addVar(0.0, 1.0, 0.0, GRB.BINARY, "z");

            // Set objective: maximize x + y + 2 z
            GRBLinExpr expr = new GRBLinExpr();
            expr.addTerm(1.0, x); expr.addTerm(1.0, y); expr.addTerm(2.0, z);
            model.setObjective(expr, GRB.MAXIMIZE);

            // Add constraint: x + 2 y + 3 z <= 4
            expr = new GRBLinExpr();
            expr.addTerm(1.0, x); expr.addTerm(2.0, y); expr.addTerm(3.0, z);
            model.addConstr(expr, GRB.LESS_EQUAL, 4.0, "c0");

            // Add constraint: x + y >= 1
            expr = new GRBLinExpr();
            expr.addTerm(1.0, x); expr.addTerm(1.0, y);
            model.addConstr(expr, GRB.GREATER_EQUAL, 1.0, "c1");

            // Optimize model
            model.optimize();

            System.out.println(x.get(GRB.StringAttr.VarName) + " " + x.get(GRB.DoubleAttr.X));
            System.out.println(y.get(GRB.StringAttr.VarName) + " " + y.get(GRB.DoubleAttr.X));
            System.out.println(z.get(GRB.StringAttr.VarName) + " " + z.get(GRB.DoubleAttr.X));

            System.out.println("Obj: " + model.get(GRB.DoubleAttr.ObjVal));

            // Dispose of model and environment
            model.dispose();
            env.dispose();
        }
        catch (GRBException e) {
            System.out.println("Error code: " + e.getErrorCode() + ". " + e.getMessage());
        }
    }
}
Mip2.java

/* Copyright 2020, Gurobi Optimization, LLC */

/* This example reads a MIP model from a file, solves it and 
prints the objective values from all feasible solutions 
generated while solving the MIP. Then it creates the fixed 
model and solves that model. */

import gurobi.*;

public class Mip2 {
    public static void main(String[] args) {
        if (args.length < 1) {
            System.out.println("Usage: java Mip2 filename");
            System.exit(1);
        }

        try {
            GRBEnv env = new GRBEnv();
            GRBModel model = new GRBModel(env, args[0]);
            if (model.get(GRB.IntAttr.IsMIP) == 0) {
                System.out.println("Model is not a MIP");
                System.exit(1);
            }

            model.optimize();

            int optimstatus = model.get(GRB.IntAttr.Status);
            double objval = 0;
            if (optimstatus == GRB.Status.OPTIMAL) {
                objval = model.get(GRB.DoubleAttr.ObjVal);
                System.out.println("Optimal objective: " + objval);
            } else if (optimstatus == GRB.Status.INF_OR_UNBD) {
                System.out.println("Model is infeasible or unbounded");
                return;
            } else if (optimstatus == GRB.Status.INFEASIBLE) {
                System.out.println("Model is infeasible");
                return;
            } else if (optimstatus == GRB.Status.UNBOUNDED) {
                System.out.println("Model is unbounded");
                return;
            } else {
                System.out.println("Optimization was stopped with status = " + optimstatus);
                return;
            }

            /* Iterate over the solutions and compute the objectives */
            GRBVar[] vars = model.getVars();
            model.set(GRB.IntParam.OutputFlag, 0);
System.out.println();
for (int k = 0; k < model.get(GRB.IntAttr.SolCount); ++k) {
    model.set(GRB.IntParam.SolutionNumber, k);
    double objn = 0.0;

    for (int j = 0; j < vars.length; j++) {
        objn += vars[j].get(GRB.DoubleAttr.Obj) * vars[j].get(GRB.DoubleAttr.Xn);
    }

    System.out.println("Solution "+k+" has objective: "+objn);
}
System.out.println();
model.set(GRB.IntParam.OutputFlag, 1);
/* Create a fixed model, turn off presolve and solve */
GRBModel fixed = model.fixedModel();
fixed.set(GRB.IntParam.Presolve, 0);
fixed.optimize();
int foptimstatus = fixed.get(GRB.IntAttr.Status);
if (foptimstatus != GRB.Status.OPTIMAL) {
    System.err.println("Error: fixed model isn't optimal");
    return;
}
double fobjval = fixed.get(GRB.DoubleAttr.ObjVal);
if (Math.abs(fobjval - objval) > 1.0e-6 * (1.0 + Math.abs(objval))) {
    System.err.println("Error: objective values are different");
    return;
}
GRBVar[] fvars = fixed.getVars();
double[] x = fixed.get(GRB.DoubleAttr.X, fvars);
String[] vnames = fixed.get(GRB.StringAttr.VarName, fvars);
for (int j = 0; j < fvars.length; j++) {
    if (x[j] != 0.0) {
        System.out.println(vnames[j] + " "+x[j]);
    }
}
// Dispose of models and environment
fixed.dispose();
model.dispose();
env.dispose();
}
catch (GRBException e) {
    System.out.println("Error code: "+e.getErrorCode()+ ". " + e.getMessage());
}
Multiobj.java

/* Copyright 2020, Gurobi Optimization, LLC */

/* Want to cover three different sets but subject to a common budget of
elements allowed to be used. However, the sets have different priorities to
be covered; and we tackle this by using multi-objective optimization. */

import gurobi.*;

public class Multiobj {
    public static void main(String[] args) {
        try {
            // Sample data
            int groundSetSize = 20;
            int nSubsets = 4;
            int Budget = 12;
            double Set[][] = {
                { 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0 },
                { 0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1 },
                { 0, 0, 0, 1, 1, 0, 0, 0, 0, 0, 0, 0, 1, 1, 0, 1, 1, 1, 1, 1 },
                { 0, 0, 0, 1, 1, 1, 0, 0, 0, 1, 1, 1, 0, 0, 0, 0, 1, 1, 1, 1, 1 };
            int SetObjPriority[] = {3, 2, 2, 1};
            double SetObjWeight[] = {1.0, 0.25, 1.25, 1.0};
            int e, i, status, nSolutions;

            // Create environment
            GRBEnv env = new GRBEnv("Multiobj.log");

            // Create initial model
            GRBModel model = new GRBModel(env);
            model.set(GRB.StringAttr.ModelName, "Multiobj");

            // Initialize decision variables for ground set:
            // x[e] == 1 if element e is chosen for the covering.
            GRBVar[] Elem = model.addVars(groundSetSize, GRB.BINARY);
            for (e = 0; e < groundSetSize; e++) {
                String vname = "El" + String.valueOf(e);
                Elem[e].set(GRB.StringAttr.VarName, vname);
            }

            // Constraint: limit total number of elements to be picked to be at most
            // Budget
            GRBLinExpr lhs = new GRBLinExpr();
            for (e = 0; e < groundSetSize; e++) {
                lhs.addTerm(1.0, Elem[e]);
            }
            model.addConstr(lhs, GRB.LESS_EQUAL, Budget, "Budget");

            // Set global sense for ALL objectives
            model.set(GRB.IntAttr.ModelSense, GRB.MAXIMIZE);
        }
    }
}
// Limit how many solutions to collect
model.set(GRB.IntParam.PoolSolutions, 100);

// Set and configure i-th objective
for (i = 0; i < nSubsets; i++) {
  GRBLinExpr objn = new GRBLinExpr();
  String vname = "Set" + String.valueOf(i);
  for (e = 0; e < groundSetSize; e++)
    objn.addTerm(Set[i][e], Elem[e]);

  model.setObjectiveN(objn, i, SetObjPriority[i], SetObjWeight[i],
                    1.0 + i, 0.01, vname);
}

// Save problem
model.write("Multiobj.lp");

// Optimize
model.optimize();

// Status checking
status = model.get(GRB.IntAttr.Status);
if (status == GRB.INF_OR_UNBD ||
    status == GRB.INFEASIBLE ||
    status == GRB.UNBOUNDED)
  { System.out.println("The model cannot be solved " +
                  "because it is infeasible or unbounded");
    System.exit(i);
  }
if (status != GRB.OPTIMAL) {
  System.out.println("Optimization was stopped with status " + status);
  System.exit(i);
}

// Print best selected set
System.out.println("Selected elements in best solution:");
System.out.println("\t");
for (e = 0; e < groundSetSize; e++) {
  if (Elem[e].get(GRB.DoubleAttr.X) < .9) continue;
  if (Elem[e].get(GRB.DoubleAttr.X) < .9) continue;
  System.out.print(" El" + e);
}
System.out.println();

// Print number of solutions stored
nSolutions = model.get(GRB.IntAttr.SolCount);
System.out.println("Number of solutions found: " + nSolutions);

// Print objective values of solutions
if (nSolutions > 10) nSolutions = 10;
System.out.println("Objective values for first " + nSolutions);
System.out.println(" solutions:");
for (i = 0; i < nSubsets; i++) {
  model.set(GRB.IntParam.ObjNumber, i);
System.out.print("\tSet" + i);
for (e = 0; e < nSolutions; e++) {
    System.out.print(" ");
    model.set(GRB.IntParam.SolutionNumber, e);
    double val = model.get(GRB.DoubleAttr.ObjNVal);
    System.out.print(" " + val);
}
System.out.println();
model.dispose();
env.dispose();
} catch (GRBException e) {
    System.out.println("Error code = " + e.getErrorCode());
    System.out.println(e.getMessage());
}

Multiscenario.java

// Copyright 2020, Gurobi Optimization, LLC

// Facility location: a company currently ships its product from 5 plants
// to 4 warehouses. It is considering closing some plants to reduce
// costs. What plant(s) should the company close, in order to minimize
// transportation and fixed costs?
//
// Since the plant fixed costs and the warehouse demands are uncertain, a
// scenario approach is chosen.
//
// Note that this example is similar to the Facility.java example. Here we
// added scenarios in order to illustrate the multi-scenario feature.
//
// Based on an example from Frontline Systems:
// http://www.solver.com/disfacility.htm
// Used with permission.

import gurobi.*;

public class Multiscenario {

    public static void main(String[] args) {
        try {

            // Warehouse demand in thousands of units
            double Demand[] = new double[] { 15, 18, 14, 20 };

            // Plant capacity in thousands of units
            double Capacity[] = new double[] { 20, 22, 17, 19, 18 };

            // Fixed costs for each plant
            double FixedCosts[] =
                new double[] { 12000, 15000, 17000, 13000, 16000 };

            // Transportation costs per thousand units
            double TransCosts[][] =

        } catch (GRBException e) {
            System.out.println("Error code = " + e.getErrorCode());
            System.out.println(e.getMessage());
        }
new double[][] { { 4000, 2000, 3000, 2500, 4500 },
                { 2500, 2600, 3400, 3000, 4000 },
                { 1200, 1800, 2600, 4100, 3000 },
                { 2200, 2600, 3100, 3700, 3200 } };  

// Number of plants and warehouses
int nPlants = Capacity.length;
int nWarehouses = Demand.length;

double maxFixed = -GRB.INFINITY;
double minFixed = GRB.INFINITY;
for (int p = 0; p < nPlants; ++p) {
    if (FixedCosts[p] > maxFixed)
        maxFixed = FixedCosts[p];
    if (FixedCosts[p] < minFixed)
        minFixed = FixedCosts[p];
}

// Model
GRBEnv env = new GRBEnv();
GRBModel model = new GRBModel(env);
model.set(GRB.StringAttr.ModelName, "multiscenario");

// Plant open decision variables: open[p] == 1 if plant p is open.
GRBVar[] open = new GRBVar[nPlants];
for (int p = 0; p < nPlants; ++p) {
    open[p] = model.addVar(0, 1, FixedCosts[p], GRB.BINARY, "Open" + p);
}

// Transportation decision variables: how much to transport from
// a plant p to a warehouse w
GRBVar[][] transport = new GRBVar[nWarehouses][nPlants];
for (int w = 0; w < nWarehouses; ++w) {
    for (int p = 0; p < nPlants; ++p) {
        transport[w][p] = model.addVar(0, GRB.INFINITY, TransCosts[w][p],
                                        GRB.CONTINUOUS, "Trans" + p + "," + w);
    }
}

// The objective is to minimize the total fixed and variable costs
model.set(GRB.IntAttr.ModelSense, GRB.MINIMIZE);

// Production constraints
// Note that the right-hand limit sets the production to zero if
// the plant is closed
for (int p = 0; p < nPlants; ++p) {
    GRBLinExpr ptot = new GRBLinExpr();
    for (int w = 0; w < nWarehouses; ++w) {
        ptot.addTerm(1.0, transport[w][p]);
    }
    GRBLinExpr limit = new GRBLinExpr();
    limit.addTerm(Capacity[p], open[p]);
    model.addConstr(ptot, GRB.LESS_EQUAL, limit, "Capacity" + p);
}
// Demand constraints
GRBConstr[] demandConstr = new GRBConstr[nWarehouses];
for (int w = 0; w < nWarehouses; ++w) {
    GRBLinExpr dtot = new GRBLinExpr();
    for (int p = 0; p < nPlants; ++p) {
        dtot.addTerm(1.0, transport[w][p]);
    }
    demandConstr[w] = model.addConstr(dtot, GRB.EQUAL, Demand[w], "Demand" + w);
}

// We constructed the base model, now we add 7 scenarios

// Scenario 0: Represents the base model, hence, no manipulations.
// Scenario 1: Manipulate the warehouses demands slightly (constraint right hand sides).
// Scenario 2: Double the warehouses demands (constraint right hand sides).
// Scenario 3: Manipulate the plant fixed costs (objective coefficients).
// Scenario 4: Manipulate the warehouses demands and fixed costs.
// Scenario 5: Force the plant with the largest fixed cost to stay open (variable bounds).
// Scenario 6: Force the plant with the smallest fixed cost to be closed (variable bounds).

model.set(GRB.IntAttr.NumScenarios, 7);

// Scenario 0: Base model, hence, nothing to do except giving the scenario a name
model.set(GRB.IntParam.ScenarioNumber, 0);
model.set(GRB.StringAttr.ScenNName, "Base model");

// Scenario 1: Increase the warehouse demands by 10%
model.set(GRB.IntParam.ScenarioNumber, 1);
model.set(GRB.StringAttr.ScenNName, "Increased warehouse demands");
for (int w = 0; w < nWarehouses; w++) {
    demandConstr[w].set(GRB.DoubleAttr.ScenNRHS, Demand[w] * 1.1);
}

// Scenario 2: Double the warehouse demands
model.set(GRB.IntParam.ScenarioNumber, 2);
model.set(GRB.StringAttr.ScenNName, "Double the warehouse demands");
for (int w = 0; w < nWarehouses; w++) {
    demandConstr[w].set(GRB.DoubleAttr.ScenNRHS, Demand[w] * 2.0);
}

// Scenario 3: Decrease the plant fixed costs by 5%
model.set(GRB.IntParam.ScenarioNumber, 3);
model.set(GRB.StringAttr.ScenNName, "Decreased plant fixed costs");
for (int p = 0; p < nPlants; p++) {
    open[p].set(GRB.DoubleAttr.ScenNObj, FixedCosts[p] * 0.95);
}

// Scenario 4: Combine scenario 1 and scenario 3 */
model.set(GRB.IntParam.ScenarioNumber, 4);
model.set(GRB.StringAttr.ScenNName, "Increased warehouse demands and decreased plant fixed costs");

for (int w = 0; w < nWarehouses; w++) {
    demandConstr[w].set(GRB.DoubleAttr.ScenNRHS, Demand[w] * 1.1);
}

for (int p = 0; p < nPlants; p++) {
    open[p].set(GRB.DoubleAttr.ScenNObj, FixedCosts[p] * 0.95);
}

// Scenario 5: Force the plant with the largest fixed cost to stay open
model.set(GRB.IntParam.ScenarioNumber, 5);
model.set(GRB.StringAttr.ScenNName, "Force plant with largest fixed cost to stay open");

for (int p = 0; p < nPlants; p++) {
    if (FixedCosts[p] == maxFixed) {
        open[p].set(GRB.DoubleAttr.ScenNLB, 1.0);
        break;
    }
}

// Scenario 6: Force the plant with the smallest fixed cost to be closed
model.set(GRB.IntParam.ScenarioNumber, 6);
model.set(GRB.StringAttr.ScenNName, "Force plant with smallest fixed cost to be closed");

for (int p = 0; p < nPlants; p++) {
    if (FixedCosts[p] == minFixed) {
        open[p].set(GRB.DoubleAttr.ScenNUB, 0.0);
        break;
    }
}

// Guess at the starting point: close the plant with the highest fixed costs; open all others

// First, open all plants
for (int p = 0; p < nPlants; ++p) {
    open[p].set(GRB.DoubleAttr.Start, 1.0);
}

// Now close the plant with the highest fixed cost
System.out.println("Initial guess:");
for (int p = 0; p < nPlants; ++p) {
    if (FixedCosts[p] == maxFixed) {
        open[p].set(GRB.DoubleAttr.Start, 0.0);
        System.out.println("Closing plant " + p + "\n");
        break;
    }
}

// Use barrier to solve root relaxation
model.set(GRB.IntParam.Method, GRB.METHOD_BARRIER);

// Solve multi-scenario model
model.optimize();
int nScenarios = model.get(GRB.IntAttr.NumScenarios);

// Print solution for each */
for (int s = 0; s < nScenarios; s++) {
    int modelSense = GRB.MINIMIZE;

    // Set the scenario number to query the information for this scenario
    model.set(GRB.IntParam.ScenarioNumber, s);

    // collect result for the scenario
    double scenNObjBound = model.get(GRB.DoubleAttr.ScenNObjBound);
    double scenNObjVal = model.get(GRB.DoubleAttr.ScenNObjVal);

    System.out.println("\n\n----- Scenario " + s + " (" + model.get(GRB.StringAttr.ScenNName) + ")");

    // Check if we found a feasible solution for this scenario
    if (scenNObjVal >= modelSense * GRB.INFINITY)
        if (scenNObjBound >= modelSense * GRB.INFINITY)
            // Scenario was proven to be infeasible
            System.out.println("\nINFEASIBLE");
        else
            // We did not find any feasible solution - should not happen in
            // this case, because we did not set any limit (like a time
            // limit) on the optimization process
            System.out.println("\nNO SOLUTION");
    else {
        System.out.println("\nTOTAL COSTS: "+ scenNObjVal);
        System.out.println("SOLUTION: ");
        for (int p = 0; p < nPlants; p++) {
            double scenNX = open[p].get(GRB.DoubleAttr.ScenNX);
            if (scenNX > 0.5) {
                System.out.println("Plant " + p + " open");
                for (int w = 0; w < nWarehouses; w++) {
                    scenNX = transport[w][p].get(GRB.DoubleAttr.ScenNX);
                    if (scenNX > 0.0001)
                        System.out.println(" Transport " + scenNX +
                            " units to warehouse " + w);
                }
            } else
                System.out.println("Plant " + p + " closed!");
        }
    }
}

// Print a summary table: for each scenario we add a single summary
// line
System.out.println("\n\nSummary: Closed plants depending on scenario\n");
System.out.format("%8s | %17s %13s
", "", "Plant", "");

System.out.format("%8s |", "Scenario");
for (int p = 0; p < nPlants; p++)
    System.out.format(" %5d", p);
System.out.format(" | %-6s %s
", "Costs", "Name");

for (int s = 0; s < nScenarios; s++) {
    int modelSense = GRB.MINIMIZE;

    // Set the scenario number to query the information for this scenario
    model.set(GRB.IntParam.ScenarioNumber, s);

    // Collect result for the scenario
    double scenNObjBound = model.get(GRB.DoubleAttr.ScenNObjBound);
    double scenNObjVal = model.get(GRB.DoubleAttr.ScenNObjVal);

    System.out.format("%d |", s);

    // Check if we found a feasible solution for this scenario
    if (scenNObjVal >= modelSense * GRB.INFINITY) {
        if (scenNObjBound >= modelSense * GRB.INFINITY)
            // Scenario was proven to be infeasible
            System.out.format(" %30s| %6s %s
", "infeasible", "-", model.get(GRB.StringAttr.ScenNName));
        else
            // We did not find any feasible solution - should not happen in
            // this case, because we did not set any limit (like a time
            // limit) on the optimization process
            System.out.format("%d | %6s %s
", "no solution found", "-", model.get(GRB.StringAttr.ScenNName));
    } else {
        for (int p = 0; p < nPlants; p++) {
            double scenNX = open[p].get(GRB.DoubleAttr.ScenNX);
            if (scenNX > 0.5)
                System.out.format("%6s", " ");
            else
                System.out.format("%6s", "x");
        }

        System.out.format(" | %6g %s
", scenNObjVal, model.get(GRB.StringAttr.ScenNName));
    }
}

// Dispose of model and environment
model.dispose();
env.dispose();
} catch (GRBException e) {
    System.out.println("Error code: " + e.getErrorCode() + ". " +
                      e.getMessage());
}
}

Params.java

/* Copyright 2020, Gurobi Optimization, LLC */

/* Use parameters that are associated with a model.

A MIP is solved for a few seconds with different sets of parameters.*/
The one with the smallest MIP gap is selected, and the optimization is resumed until the optimal solution is found.

*/

import gurobi.*;

public class Params {

    public static void main(String[] args) {

        if (args.length < 1) {
            System.out.println("Usage: java Params filename");
            System.exit(1);
        }

        try {

            // Read model and verify that it is a MIP
            GRBEnv env = new GRBEnv();
            GRBModel m = new GRBModel(env, args[0]);
            if (m.get(GRB.IntAttr.IsMIP) == 0) {
                System.out.println("The model is not an integer program");
                System.exit(1);
            }

            // Set a 2 second time limit
            m.set(GRB.DoubleParam.TimeLimit, 2);

            // Now solve the model with different values of MIPFocus
            GRBModel bestModel = new GRBModel(m);
            bestModel.optimize();
            for (int i = 1; i <= 3; ++i) {
                m.reset();
                m.set(GRB.IntParam.MIPFocus, i);
                m.optimize();
                if (bestModel.get(GRB.DoubleAttr.MIPGap) >
                    m.get(GRB.DoubleAttr.MIPGap)) {
                    GRBModel swap = bestModel;
                    bestModel = m;
                    m = swap;
                }
            }

            // Finally, delete the extra model, reset the time limit and continue to solve the best model to optimality
            m.dispose();
            bestModel.set(GRB.DoubleParam.TimeLimit, GRB.INFINITY);
            bestModel.optimize();
            System.out.println("Solved with MIPFocus: " +
                bestModel.get(GRB.IntParam.MIPFocus));
        }

        catch (GRBException e) {
            System.out.println("Error code: " + e.getErrorCode() + ". " +
                e.getMessage());
        }
    }
}

267
Piecewise.java

/* Copyright 2020, Gurobi Optimization, LLC */

/* This example considers the following separable, convex problem:

minimize f(x) - y + g(z)
subject to x + 2 y + 3 z <= 4
      x + y >= 1
      x, y, z <= 1

where f(u) = exp(-u) and g(u) = 2 u^2 - 4 u, for all real u. It
formulates and solves a simpler LP model by approximating f and
g with piecewise-linear functions. Then it transforms the model
into a MIP by negating the approximation for f, which corresponds
to a non-convex piecewise-linear function, and solves it again. */

import gurobi.*;

public class Piecewise {

  private static double f(double u) { return Math.exp(-u); }
  private static double g(double u) { return 2 * u * u - 4 * u; }

  public static void main(String[] args) {
    try {

      // Create environment
      GRBEnv env = new GRBEnv();

      // Create a new model
      GRBModel model = new GRBModel(env);

      // Create variables
      double lb = 0.0, ub = 1.0;
      GRBVar x = model.addVar(lb, ub, 0.0, GRB.CONTINUOUS, "x");
      GRBVar y = model.addVar(lb, ub, 0.0, GRB.CONTINUOUS, "y");
      GRBVar z = model.addVar(lb, ub, 0.0, GRB.CONTINUOUS, "z");

      // Set objective for y
      GRBLinExpr obj = new GRBLinExpr();
      obj.addTerm(-1.0, y);
      model.setObjective(obj);

      // Add piecewise-linear objective functions for x and z
      int npts = 101;
      double[] ptu = new double[npts];
      double[] ptf = new double[npts];
      double[] ptg = new double[npts];
for (int i = 0; i < npts; i++) {
    ptu[i] = lb + (ub - lb) * i / (npts - 1);
    ptf[i] = f(ptu[i]);
    ptg[i] = g(ptu[i]);
}

model.setPWLObj(x, ptu, ptf);
model.setPWLObj(z, ptu, ptg);

// Add constraint: x + 2 y + 3 z <= 4
GRBLinExpr expr = new GRBLinExpr();
expr.addTerm(1.0, x); expr.addTerm(2.0, y); expr.addTerm(3.0, z);
model.addConstr(expr, GRB.LESS_EQUAL, 4.0, "c0");

// Add constraint: x + y >= 1
expr = new GRBLinExpr();
expr.addTerm(1.0, x); expr.addTerm(1.0, y);
model.addConstr(expr, GRB.GREATER_EQUAL, 1.0, "c1");

// Optimize model as an LP
model.optimize();
System.out.println("IsMIP: " + model.get(GRB.IntAttr.IsMIP));
System.out.println(x.get(GRB.StringAttr.VarName) + " " + x.get(GRB.DoubleAttr.X));
System.out.println(y.get(GRB.StringAttr.VarName) + " " + y.get(GRB.DoubleAttr.X));
System.out.println(z.get(GRB.StringAttr.VarName) + " " + z.get(GRB.DoubleAttr.X));
System.out.println("Obj: " + model.get(GRB.DoubleAttr.ObjVal));
System.out.println();

// Negate piecewise-linear objective function for x
for (int i = 0; i < npts; i++) {
    ptf[i] = -ptf[i];
}

model.setPWLObj(x, ptu, ptf);

// Optimize model as a MIP
model.optimize();
System.out.println("IsMIP: " + model.get(GRB.IntAttr.IsMIP));
System.out.println(x.get(GRB.StringAttr.VarName) + " " + x.get(GRB.DoubleAttr.X));
System.out.println(y.get(GRB.StringAttr.VarName) + " " + y.get(GRB.DoubleAttr.X));
Poolsearch.java

/* Copyright 2020, Gurobi Optimization, LLC */

/* We find alternative epsilon-optimal solutions to a given knapsack problem by using PoolSearchMode */

import gurobi.*;

public class Poolsearch {

    public static void main(String[] args) {

        try{
            // Sample data
            int groundSetSize = 10;
            double objCoef[] = new double[]{32, 32, 15, 15, 6, 6, 1, 1, 1, 1};
            double knapsackCoef[] = new double[]{16, 16, 8, 8, 4, 4, 2, 2, 1, 1};
            double Budget = 33;
            int e, status, nSolutions;

            // Create environment
            GRBEnv env = new GRBEnv("Poolsearch.log");

            // Create initial model
            GRBModel model = new GRBModel(env);
            model.set(GRB.StringAttr.ModelName, "Poolsearch");

            // Initialize decision variables for ground set:
            // x[e] == 1 if element e is chosen
            GRBVar[] Elem = model.addVars(groundSetSize, GRB.BINARY);
            model.set(GRB.DoubleAttr.Obj, Elem, objCoef, 0, groundSetSize);

            for (e = 0; e < groundSetSize; e++) {
                Elem[e].set(GRB.StringAttr.VarName, "El" + String.valueOf(e));
            }

            // Constraint: limit total number of elements to be picked to be at most 270
        }
    }
}
// Budget
GRBLinExpr lhs = new GRBLinExpr();
for (e = 0; e < groundSetSize; e++) {
    lhs.addTerm(knapsackCoef[e], Elem[e]);
}
model.addConstr(lhs, GRB.LESS_EQUAL, Budget, "Budget");

// set global sense for ALL objectives
model.set(GRB.IntAttr.ModelSense, GRB.MAXIMIZE);

// Limit how many solutions to collect
model.set(GRB.IntParam.PoolSolutions, 1024);

// Limit the search space by setting a gap for the worst possible solution that will be accepted
model.set(GRB.DoubleParam.PoolGap, 0.10);

// do a systematic search for the k-best solutions
model.set(GRB.IntParam.PoolSearchMode, 2);

// save problem
model.write("Poolsearch.lp");

// Optimize
model.optimize();

// Status checking
status = model.get(GRB.IntAttr.Status);
if (status == GRB.INF_OR_UNBD ||
    status == GRB.INFEASIBLE ||
    status == GRB.UNBOUNDED ) {
    System.out.println("The model cannot be solved " +
        "because it is infeasible or unbounded");
    System.exit(1);
}
if (status != GRB.OPTIMAL) {
    System.out.println("Optimization was stopped with status " + status);
    System.exit(1);
}

// Print best selected set
System.out.println("Selected elements in best solution:");
System.out.println("\t");
for (e = 0; e < groundSetSize; e++) {
    if (Elem[e].get(GRB.DoubleAttr.X) < .9) continue;
    System.out.print(" El" + e);
}
System.out.println();

// Print number of solutions stored
nSolutions = model.get(GRB.IntAttr.SolCount);
System.out.println("Number of solutions found: " + nSolutions);

// Print objective values of solutions
for (e = 0; e < nSolutions; e++) {
    model.set(GRB.IntParam.SolutionNumber, e);
System.out.println(model.get(GRB.DoubleAttr.PoolObjVal) + " ");
if (e%15 == 14) System.out.println();
}
System.out.println();

// print fourth best set if available
if (nSolutions >= 4) {
    model.set(GRB.IntParam.SolutionNumber, 3);
    System.out.println("Selected elements in fourth best solution:");
    System.out.print("\n");
    for (e = 0; e < groundSetSize; e++) {
        if (Elem[e].get(GRB.DoubleAttr.Xn) < .9) continue;
        System.out.print(" El" + e);
    }
    System.out.println();
}
model.dispose();
env.dispose();
}
catch (GRBException e) {
    System.out.println("Error code: " + e.getErrorCode() + ". " + e.getMessage());
}
}

Qcp.java

/* Copyright 2020, Gurobi Optimization, LLC */

/* This example formulates and solves the following simple QCP model:

maximize x
subject to x + y + z = 1
    x^2 + y^2 <= z^2 (second-order cone)
    x^2 <= yz (rotated second-order cone)
    x, y, z non-negative
*/

import gurobi.*;

public class Qcp {
    public static void main(String[] args) {
        try {
            GRBEnv    env = new GRBEnv("qcp.log");
            GRBModel  model = new GRBModel(env);

            // Create variables
            GRBVar x = model.addVar(0.0, GRB.INFINITY, 0.0, GRB.CONTINUOUS, "x");
            GRBVar y = model.addVar(0.0, GRB.INFINITY, 0.0, GRB.CONTINUOUS, "y");
            GRBVar z = model.addVar(0.0, GRB.INFINITY, 0.0, GRB.CONTINUOUS, "z");

            // Set objective
            model.setObjective(x, GRB.MAXIMIZE);

            model.addConstr(x + y + z == 1.0);
            model.addQConstr(x*x + y*y <= z*z);
            model.addQConstr(x*x <= y*z);

            model.optimize();
        }
    }
}
GRBLinExpr obj = new GRBLinExpr();
obj.addTerm(1.0, x);
model.setObjective(obj, GRB.MAXIMIZE);

// Add linear constraint: x + y + z = 1
GRBLinExpr expr = new GRBLinExpr();
expr.addTerm(1.0, x); expr.addTerm(1.0, y); expr.addTerm(1.0, z);
model.addConstr(expr, GRB.EQUAL, 1.0, "c0");

// Add second-order cone: x^2 + y^2 <= z^2
GRBQuadExpr qexpr = new GRBQuadExpr();
qexpr.addTerm(1.0, x, x);
qexpr.addTerm(1.0, y, y);
qexpr.addTerm(-1.0, z, z);
model.addQConstr(qexpr, GRB.LESS_EQUAL, 0.0, "qc0");

// Add rotated cone: x^2 <= yz
qexpr = new GRBQuadExpr();
qexpr.addTerm(1.0, x, x);
qexpr.addTerm(-1.0, y, z);
model.addQConstr(qexpr, GRB.LESS_EQUAL, 0.0, "qc1");

// Optimize model
model.optimize();

System.out.println(x.get(GRB.StringAttr.VarName) + " " + x.get(GRB.DoubleAttr.X));
System.out.println(y.get(GRB.StringAttr.VarName) + " " + y.get(GRB.DoubleAttr.X));
System.out.println(z.get(GRB.StringAttr.VarName) + " " + z.get(GRB.DoubleAttr.X));

System.out.println("Obj: " + model.get(GRB.DoubleAttr.ObjVal) + " " + obj.getValue());
System.out.println();

// Dispose of model and environment
model.dispose();
env.dispose();

} catch (GRBException e) {
    System.out.println("Error code: " + e.getErrorCode() + ". " + e.getMessage());
}

Qp.java

/* Copyright 2020, Gurobi Optimization, LLC */
/* This example formulates and solves the following simple QP model:

minimize \[ x^2 + xy + y^2 + yz + z^2 + 2x \]
subject to
\[ x + 2y + 3z \geq 4 \]
\[ x + y \geq 1 \]
\[ x, y, z \text{ non-negative} \]

It solves it once as a continuous model, and once as an integer model.
*/

import gurobi.*;

public class Qp {
    public static void main(String[] args) {
        try {
            GRBEnv env = new GRBEnv("qp.log");
            GRBModel model = new GRBModel(env);

            // Create variables
            GRBVar x = model.addVar(0.0, 1.0, 0.0, GRB.CONTINUOUS, "x");
            GRBVar y = model.addVar(0.0, 1.0, 0.0, GRB.CONTINUOUS, "y");
            GRBVar z = model.addVar(0.0, 1.0, 0.0, GRB.CONTINUOUS, "z");

            // Set objective
            GRBQuadExpr obj = new GRBQuadExpr();
            obj.addTerm(1.0, x, x);
            obj.addTerm(1.0, x, y);
            obj.addTerm(1.0, y, y);
            obj.addTerm(1.0, y, z);
            obj.addTerm(1.0, z, z);
            obj.addTerm(2.0, x);
            model.setObjective(obj);

            // Add constraint: \[ x + 2y + 3z \geq 4 \]
            GRBLinExpr expr = new GRBLinExpr();
            expr.addTerm(1.0, x); expr.addTerm(2.0, y); expr.addTerm(3.0, z);
            model.addConstr(expr, GRB.GREATER_EQUAL, 4.0, "c0");

            // Add constraint: \[ x + y \geq 1 \]
            expr = new GRBLinExpr();
            expr.addTerm(1.0, x); expr.addTerm(1.0, y);
            model.addConstr(expr, GRB.GREATER_EQUAL, 1.0, "c1");

            // Optimize model
            model.optimize();

            System.out.println(x.get(GRB.StringAttr.VarName) + " " + x.get(GRB.DoubleAttr.X));
            System.out.println(y.get(GRB.StringAttr.VarName) + " " + y.get(GRB.DoubleAttr.X));
        }
    }
}
System.out.println(z.get(GRB.StringAttr.VarName) + " " + z.get(GRB.DoubleAttr.X));

System.out.println("Obj: " + model.get(GRB.DoubleAttr.ObjVal) + " " + obj.getValue());
System.out.println();

// Change variable types to integer
x.set(GRB.CharAttr.VType, GRB.INTEGER);
y.set(GRB.CharAttr.VType, GRB.INTEGER);
z.set(GRB.CharAttr.VType, GRB.INTEGER);

// Optimize again
model.optimize();

System.out.println(x.get(GRB.StringAttr.VarName) + " " + x.get(GRB.DoubleAttr.X));
System.out.println(y.get(GRB.StringAttr.VarName) + " " + y.get(GRB.DoubleAttr.X));
System.out.println(z.get(GRB.StringAttr.VarName) + " " + z.get(GRB.DoubleAttr.X));

System.out.println("Obj: " + model.get(GRB.DoubleAttr.ObjVal) + " " + obj.getValue());

// Dispose of model and environment
model.dispose();
env.dispose();

} catch (GRBException e) {
    System.out.println("Error code: " + e.getErrorCode() + ". " + e.getMessage());
}

Sensitivity.java

// Copyright 2020, Gurobi Optimization, LLC

// A simple sensitivity analysis example which reads a MIP model from a
// file and solves it. Then uses the scenario feature to analyze the impact
// w.r.t. the objective function of each binary variable if it is set to
// 1-X, where X is its value in the optimal solution.

// Usage:
// java Sensitivity <model filename>

import gurobi.*;

public class Sensitivity {


// Maximum number of scenarios to be considered
private static final int MAXSCENARIOS = 100;

public static void main(String[] args) {
    if (args.length < 1) {
        System.out.println("Usage: java Sensitivity filename");
        System.exit(1);
    }
    try {
        // Create environment
        GRBEnv env = new GRBEnv();
        // Read model
        GRBModel model = new GRBModel(env, args[0]);
        int scenarios;
        if (model.get(GRB.IntAttr.IsMIP) == 0) {
            System.out.println("Model is not a MIP");
            System.exit(1);
        }
        // Solve model
        model.optimize();
        if (model.get(GRB.IntAttr.Status) != GRB.OPTIMAL) {
            System.out.println("Optimization ended with status " + model.get(GRB.IntAttr.Status));
            System.exit(1);
        }
        // Store the optimal solution
        double origObjVal = model.get(GRB.DoubleAttr.ObjVal);
        GRBVar[] vars = model.getVars();
        double[] origX = model.get(GRB.DoubleAttr.X, vars);
        scenarios = 0;
        // Count number of unfixed, binary variables in model. For each we
        // create a scenario.
        for (int i = 0; i < vars.length; i++) {
            GRBVar v = vars[i];
            char vType = v.get(GRB.CharAttr.VType);
            if (v.get(GRB.DoubleAttr.LB) == 0 && v.get(GRB.DoubleAttr.UB) == 1 &&
                (vType == GRB.BINARY || vType == GRB.INTEGER)) {
                scenarios++;
                if (scenarios >= MAXSCENARIOS) break;
            }
        }
    }
}
System.out.println("### construct multi-scenario model with "+ scenarios + " scenarios");

// Set the number of scenarios in the model */
model.set(GRB.IntAttr.NumScenarios, scenarios);

scenarios = 0;

// Create a (single) scenario model by iterating through unfixed
// binary variables in the model and create for each of these
// variables a scenario by fixing the variable to 1-X, where X is its
// value in the computed optimal solution
for (int i = 0; i < vars.length; i++) {
    GRBVar v = vars[i];
    char vType = v.get(GRB.CharAttr.VType);

    if (v.get(GRB.DoubleAttr.LB) == 0 &&
        v.get(GRB.DoubleAttr.UB) == 1 &&
        (vType == GRB.BINARY || vType == GRB.INTEGER) &&
        scenarios < MAXSCENARIOS) {
        // Set ScenarioNumber parameter to select the corresponding
        // scenario for adjustments
        model.set(GRB.IntParam.ScenarioNumber, scenarios);

        // Set variable to 1-X, where X is its value in the optimal solution */
        if (origX[i] < 0.5)
            v.set(GRB.DoubleAttr.ScenNLB, 1.0);
        else
            v.set(GRB.DoubleAttr.ScenNUB, 0.0);

        scenarios++;
    } else {
        // Add MIP start for all other variables using the optimal
        // solution of the base model
        v.set(GRB.DoubleAttr.Start, origX[i]);
    }
}

// Solve multi-scenario model
model.optimize();

// In case we solved the scenario model to optimality capture the
// sensitivity information
if (model.get(GRB.IntAttr.Status) == GRB.OPTIMAL) {
    // get the model sense (minimization or maximization)
    int modelSense = model.get(GRB.IntAttr.ModelSense);

    scenarios = 0;

    for (int i = 0; i < vars.length; i++) {
        GRBVar v = vars[i];
        char vType = v.get(GRB.CharAttr.VType);
if (v.get(GRB.DoubleAttr.LB) == 0 && v.get(GRB.DoubleAttr.UB) == 1 && (vType == GRB.BINARY || vType == GRB.INTEGER)) {

    // Set scenario parameter to collect the objective value of the corresponding scenario
    model.set(GRB.IntParam.ScenarioNumber, scenarios);

    // Collect objective value and bound for the scenario
    double scenarioObjVal = model.get(GRB.DoubleAttr.ScenNObjVal);
    double scenarioObjBound = model.get(GRB.DoubleAttr.ScenNObjBound);

    System.out.print("Objective sensitivity for variable "+v.get(GRB.StringAttr.VarName)+" is ");

    // Check if we found a feasible solution for this scenario
    if (scenarioObjVal >= modelSense * GRB.INFINITY) {
        // Check if the scenario is infeasible
        if (scenarioObjBound >= modelSense * GRB.INFINITY)
            System.out.println("infeasible");
        else
            System.out.println("unknown (no solution available)");
    } else {
        // Scenario is feasible and a solution is available
        System.out.println("" + modelSense * (scenarioObjVal - origObjVal));
    }

    scenarios++;

    if (scenarios >= MAXSCENARIOS)
        break;
}
}

// Dispose of model and environment
model.dispose();
env.dispose();

} catch (GRBException e) {
    System.out.println("Error code: "+e.getErrorCode());
    System.out.println(e.getMessage());
    e.printStackTrace();
}

Sos.java

/* Copyright 2020, Gurobi Optimization, LLC */

/* This example creates a very simple Special Ordered Set (SOS) model. The model consists of 3 continuous variables, no linear constraints, and a pair of SOS constraints of type 1. */
import gurobi.*;

public class Sos {
    public static void main(String[] args) {
        try {
            GRBEnv env = new GRBEnv();
            GRBModel model = new GRBModel(env);

            // Create variables
            double ub[] = {1, 1, 2};
            double obj[] = {-2, -1, -1};
            String names[] = {"x0", "x1", "x2"};
            GRBVar[] x = model.addVars(null, ub, obj, null, names);

            // Add first SOS1: x0=0 or x1=0
            GRBVar sosv1[] = {x[0], x[1]};
            double soswt1[] = {1, 2};
            model.addSOS(sosv1, soswt1, GRB.SOS_TYPE1);

            // Add second SOS1: x0=0 or x2=0
            GRBVar sosv2[] = {x[0], x[2]};
            double soswt2[] = {1, 2};
            model.addSOS(sosv2, soswt2, GRB.SOS_TYPE1);

            // Optimize model
            model.optimize();

            for (int i = 0; i < 3; i++)
                System.out.println(x[i].get(GRB.StringAttr.VarName) + " " + x[i].get(GRB.DoubleAttr.X));

            // Dispose of model and environment
            model.dispose();
            env.dispose();
        } catch (GRBException e) {
            System.out.println("Error code: " + e.getErrorCode() + ". " + e.getMessage());
        }
    }
}

Sudoku.java

/* Copyright 2020, Gurobi Optimization, LLC */
/*
  Sudoku example.
*/
The Sudoku board is a 9x9 grid, which is further divided into a 3x3 grid of 3x3 grids. Each cell in the grid must take a value from 0 to 9. No two grid cells in the same row, column, or 3x3 subgrid may take the same value.

In the MIP formulation, binary variables $x[i,j,v]$ indicate whether cell $<i,j>$ takes value $'v'$. The constraints are as follows:

1. Each cell must take exactly one value ($\sum_v x[i,j,v] = 1$)
2. Each value is used exactly once per row ($\sum_i x[i,j,v] = 1$)
3. Each value is used exactly once per column ($\sum_j x[i,j,v] = 1$)
4. Each value is used exactly once per 3x3 subgrid ($\sum_{grid} x[i,j,v] = 1$)

Input datasets for this example can be found in examples/data/sudoku*.

```java
import gurobi.*;
import java.io.*;

public class Sudoku {
    public static void main(String[] args) {
        int n = 9;
        int s = 3;

        if (args.length < 1) {
            System.out.println("Usage: java Sudoku filename");
            System.exit(1);
        }

        try {
            GRBEnv env = new GRBEnv();
            GRBModel model = new GRBModel(env);

            // Create 3-D array of model variables
            GRBVar[][][] vars = new GRBVar[n][n][n];

            for (int i = 0; i < n; i++) {
                for (int j = 0; j < n; j++) {
                    for (int v = 0; v < n; v++) {
                        String st = "G_" + String.valueOf(i) + "_" + String.valueOf(j) + "_" + String.valueOf(v);
                        vars[i][j][v] = model.addVar(0.0, 1.0, 0.0, GRB.BINARY, st);
                    }
                }
            }

            // Add constraints
            GRBLinExpr expr;

            // Each cell must take one value
            for (int i = 0; i < n; i++) {
                for (int j = 0; j < n; j++) {
                    expr = new GRBLinExpr();
                    expr.addTerms(null, vars[i][j]);
                }
            }
        }
```
String st = "V_" + String.valueOf(i) + "_" + String.valueOf(j);
model.addConstr(expr, GRB.EQUAL, 1.0, st);
}
}

// Each value appears once per row
for (int i = 0; i < n; i++) {
    for (int v = 0; v < n; v++) {
        expr = new GRBLinExpr();
        for (int j = 0; j < n; j++)
            expr.addTerm(1.0, vars[i][j][v]);
        String st = "R_" + String.valueOf(i) + "_" + String.valueOf(v);
        model.addConstr(expr, GRB.EQUAL, 1.0, st);
    }
}

// Each value appears once per column
for (int j = 0; j < n; j++) {
    for (int v = 0; v < n; v++) {
        expr = new GRBLinExpr();
        for (int i = 0; i < n; i++)
            expr.addTerm(1.0, vars[i][j][v]);
        String st = "C_" + String.valueOf(j) + "_" + String.valueOf(v);
        model.addConstr(expr, GRB.EQUAL, 1.0, st);
    }
}

// Each value appears once per sub-grid
for (int v = 0; v < n; v++) {
    for (int i0 = 0; i0 < s; i0++) {
        for (int j0 = 0; j0 < s; j0++) {
            expr = new GRBLinExpr();
            for (int i1 = 0; i1 < s; i1++)
                expr.addTerm(1.0, vars[i0*s+i1][j0*s+j1][v]);
        }
    }
    String st = "Sub_" + String.valueOf(v) + "_" + String.valueOf(i0) + "_" + String.valueOf(j0);
    model.addConstr(expr, GRB.EQUAL, 1.0, st);
}

// Fix variables associated with pre-specified cells
File file = new File(args[0]);
FileInputStream fis = new FileInputStream(file);
byte[] input = new byte[n];

for (int i = 0; i < n; i++) {
    fis.read(input);
    for (int j = 0; j < n; j++) {
int val = (int) input[j] - 48 - 1; // 0-based

if (val >= 0)
    vars[i][j][val].set(GRB.DoubleAttr.LB, 1.0);

// read the endline byte
fis.read();

// Optimize model
model.optimize();

// Write model to file
model.write("sudoku.lp");

double[][][] x = model.get(GRB.DoubleAttr.X, vars);

System.out.println();
for (int i = 0; i < n; i++) {
    for (int j = 0; j < n; j++) {
        for (int v = 0; v < n; v++) {
            if (x[i][j][v] > 0.5) {
                System.out.print(v+1);
            }
        }
    }
    System.out.println();
}

// Dispose of model and environment
model.dispose();
env.dispose();

} catch (GRBException e) {
    System.out.println("Error code: " + e.getErrorCode() + ". " +
                       e.getMessage());
} catch (IOException e) {
    System.out.println("IO Error");
}

Tsp.java

/* Copyright 2020, Gurobi Optimization, LLC */

// Solve a traveling salesman problem on a randomly generated set of
// points using lazy constraints. The base MIP model only includes
// 'degree-2' constraints, requiring each node to have exactly
// two incident edges. Solutions to this model may contain subtours -
// tours that don't visit every node. The lazy constraint callback
// adds new constraints to cut them off.

import gurobi.*;
```java
public class Tsp extends GRBCallback {
    private GRBVar[][] vars;

    public Tsp(GRBVar[][] xvars) {
        vars = xvars;
    }

    // Subtour elimination callback. Whenever a feasible solution is found,
    // find the subtour that contains node 0, and add a subtour elimination
    // constraint if the tour doesn’t visit every node.

    protected void callback() {
        try {
            if (where == GRB.CB_MIPSOL) {
                // Found an integer feasible solution - does it visit every node?
                int n = vars.length;
                int[] tour = findsubtour(getSolution(vars));

                if (tour.length < n) {
                    // Add subtour elimination constraint
                    GRBLinExpr expr = new GRBLinExpr();
                    for (int i = 0; i < tour.length; i++)
                        for (int j = i + 1; j < tour.length; j++)
                            expr.addTerm(1.0, vars[tour[i]][tour[j]]);
                    addLazy(expr, GRB.LESS_EQUAL, tour.length - 1);
                }
            }
        } catch (GRBException e) {
            System.out.println("Error code: " + e.getErrorCode() + ". " +
                               e.getMessage());
            e.printStackTrace();
        }
    }

    // Given an integer-feasible solution 'sol', return the smallest
    // sub-tour (as a list of node indices).

    protected static int[] findsubtour(double[][] sol) {
        int n = sol.length;
        boolean[] seen = new boolean[n];
        int[] tour = new int[n];
        int bestind, bestlen;
        int i, node, len, start;

        for (i = 0; i < n; i++)
            seen[i] = false;

        start = 0;
        bestlen = n + 1;
        bestind = -1;
        node = 0;
        while (start < n) {
            for (node = 0; node < n; node++)
                if (!seen[node])
                    break;
```
if (node == n)
    break;
for (len = 0; len < n; len++) {
    tour[start+len] = node;
    seen[node] = true;
    for (i = 0; i < n; i++) {
        if (sol[node][i] > 0.5 && !seen[i]) {
            node = i;
            break;
        }
    }
    if (i == n) {
        len ++;
        if (len < bestlen) {
            bestlen = len;
            bestind = start;
        }
        start += len;
        break;
    }
}

int result[] = new int[bestlen];
for (i = 0; i < bestlen; i++)
    result[i] = tour[bestind+i];
return result;

// Euclidean distance between points ‘i’ and ‘j’
protected static double distance(double[] x,
                                    double[] y,
                                    int i,
                                    int j) {
    double dx = x[i]-x[j];
    double dy = y[i]-y[j];
    return Math.sqrt(dx*dx+dy*dy);
}

public static void main(String[] args) {
    if (args.length < 1) {
        System.out.println("Usage: java Tsp ncities");
        System.exit(1);
    }
    int n = Integer.parseInt(args[0]);
    try {
        GRBEnv env = new GRBEnv();
        GRBModel model = new GRBModel(env);

        // Must set LazyConstraints parameter when using lazy constraints
        model.set(GRB.IntParam.LazyConstraints, 1);
double[] x = new double[n];
double[] y = new double[n];

for (int i = 0; i < n; i++) {
    x[i] = Math.random();
    y[i] = Math.random();
}

// Create variables
GRBVar[][] vars = new GRBVar[n][n];

for (int i = 0; i < n; i++) {
    for (int j = 0; j <= i; j++) {
        vars[i][j] = model.addVar(0.0, 1.0, distance(x, y, i, j),
                                  GRB.BINARY,
                                  "x"+String.valueOf(i)+"_"+String.valueOf(j));
        vars[j][i] = vars[i][j];
    }
}

// Degree-2 constraints
for (int i = 0; i < n; i++) {
    GRBLinExpr expr = new GRBLinExpr();
    for (int j = 0; j < n; j++)
        expr.addTerm(1.0, vars[i][j]);
    model.addConstr(expr, GRB.EQUAL, 2.0, "deg2_"+String.valueOf(i));
}

// Forbid edge from node back to itself
for (int i = 0; i < n; i++)
    vars[i][i].set(GRB.DoubleAttr.UB, 0.0);

model.setCallback(new Tsp(vars));
model.optimize();

if (model.get(GRB.IntAttr.SolCount) > 0) {
    int[] tour = findsubtour(model.get(GRB.DoubleAttr.X, vars));
    assert tour.length == n;

    System.out.print("Tour: ");
    for (int i = 0; i < tour.length; i++)
        System.out.print(String.valueOf(tour[i]) + " ");
    System.out.println();
}

// Dispose of model and environment
model.dispose();
env.dispose();

} catch (GRBException e) {
    System.out.println("Error code: " + e.getErrorCode() + ". " +
                       e.getMessage());
e.printStackTrace();
}
import gurobi.*;

public class Tune {
    public static void main(String[] args) {
        if (args.length < 1) {
            System.out.println("Usage: java Tune filename");
            System.exit(1);
        }

        try {
            GRBEnv env = new GRBEnv();

            // Read model from file
            GRBModel model = new GRBModel(env, args[0]);

            // Set the TuneResults parameter to 1
            model.set(GRB.IntParam.TuneResults, 1);

            // Tune the model
            model.tune();

            // Get the number of tuning results
            int resultcount = model.get(GRB.IntAttr.TuneResultCount);

            if (resultcount > 0) {
                // Load the tuned parameters into the model’s environment
                model.getTuneResult(0);

                // Write the tuned parameters to a file
                model.write("tuneprm");

                // Solve the model using the tuned parameters
                model.optimize();
            }

            // Dispose of model and environment
            model.dispose();
            env.dispose();
        } catch (GRBException e) {
            System.out.println("Error code: " + e.getCode() + ". " + e.getMessage());
        }
    }
}
import gurobi.*;

public class Workforce1 {

    public static void main(String[] args) {
        try {

            // Sample data
            // Sets of days and workers
            String Shifts[] =
            String Workers[] =

            int nShifts = Shifts.length;
            int nWorkers = Workers.length;

            // Number of workers required for each shift
            double shiftRequirements[] =
                new double[] { 3, 2, 4, 4, 5, 6, 5, 2, 2, 3, 4, 6, 7, 5 };

            // Amount each worker is paid to work one shift
            double pay[] = new double[] { 10, 12, 10, 8, 8, 9, 11 };

            // Worker availability: 0 if the worker is unavailable for a shift
            double availability[][] =
                new double[][] {
                    { 0, 1, 1, 0, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1 },
                    { 1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1 },
                    { 0, 0, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1 },
                    { 0, 1, 1, 0, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1 },
                    { 1, 1, 1, 1, 1, 1, 0, 0, 1, 0, 1, 0, 1, 1 },
                    { 1, 0, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1 },
                    { 1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1 },
                    { 0, 1, 1, 0, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1 },
                    { 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1 },
                    { 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1 },
                    { 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1 });

            // Model
            GRBEnv env = new GRBEnv();
            GRBModel model = new GRBModel(env);
            model.set(GRB.StringAttr.ModelName, "assignment");

            // Assignment variables: x[w][s] == 1 if worker w is assigned
            // to shift s. Since an assignment model always produces integer
// solutions, we use continuous variables and solve as an LP.
GRBVar[][] x = new GRBVar[nWorkers][nShifts];
for (int w = 0; w < nWorkers; ++w) {
    for (int s = 0; s < nShifts; ++s) {
        x[w][s] =
            model.addVar(0, availability[w][s], pay[w], GRB.CONTINUOUS,
                         Workers[w] + "." + Shifts[s]);
    }
}

// The objective is to minimize the total pay costs
model.set(GRB.IntAttr.ModelSense, GRB.MINIMIZE);

// Constraint: assign exactly shiftRequirements[s] workers to each shift s
// for each shift s
for (int s = 0; s < nShifts; ++s) {
    GRBLinExpr lhs = new GRBLinExpr();
    for (int w = 0; w < nWorkers; ++w) {
        lhs.addTerm(1.0, x[w][s]);
    }
    model.addConstr(lhs, GRB.EQUAL, shiftRequirements[s], Shifts[s]);
}

// Optimize
model.optimize();
int status = model.get(GRB.IntAttr.Status);
if (status == GRB.Status.UNBOUNDED) {
    System.out.println("The model cannot be solved 
                       + "because it is unbounded");
    return;
}
if (status == GRB.Status.OPTIMAL) {
    System.out.println("The optimal objective is " +
                       model.get(GRB.DoubleAttr.ObjVal));
    return;
}
if (status != GRB.Status.INF_OR_UNBD &&
    status != GRB.Status.INFEASIBLE) {
    System.out.println("Optimization was stopped with status " + status);
    return;
}

// Compute IIS
System.out.println("The model is infeasible; computing IIS");
model.computeIIS();
System.out.println("\nThe following constraint(s) "
+ "cannot be satisfied:");
for (GRBConstr c : model.getConstrs()) {
    if (c.get(GRB.IntAttr.IISConstr) == 1) {
        System.out.println(c.get(GRB.StringAttr.ConstrName));
    }
}

// Dispose of model and environment
model.dispose();
env.dispose();
Workforce2.java

/* Copyright 2020, Gurobi Optimization, LLC */

/* Assign workers to shifts; each worker may or may not be available on a particular day. If the problem cannot be solved, use IIS iteratively to find all conflicting constraints. */

import gurobi.*;
import java.util.*;

public class Workforce2 {
    public static void main(String[] args) {
        try {
            // Sample data
            // Sets of days and workers
            String Shifts[] = 
            String Workers[] = 

            int nShifts = Shifts.length;
            int nWorkers = Workers.length;

            // Number of workers required for each shift
            double shiftRequirements[] = 
            new double[] { 3, 2, 4, 4, 5, 6, 5, 2, 2, 3, 4, 6, 7, 5 };

            // Amount each worker is paid to work one shift
            double pay[] = new double[] { 10, 12, 10, 8, 8, 9, 11 };

            // Worker availability: 0 if the worker is unavailable for a shift
            double availability[][] = 
            new double[][] { 
                { 0, 0, 0, 0, 1, 0, 1, 1, 1, 1, 1, 1 },
                { 1, 1, 0, 0, 1, 1, 0, 1, 1, 1, 1, 1 },
                { 0, 0, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1 },
                { 0, 1, 1, 0, 1, 1, 0, 1, 1, 1, 1, 1 },
                { 1, 1, 1, 0, 1, 1, 0, 1, 1, 0, 1, 1 },
                { 1, 1, 1, 0, 0, 1, 0, 1, 1, 1, 1, 1 },
                { 1, 1, 1, 0, 1, 1, 0, 1, 1, 1, 1, 1 } };

            // Model
            GRBEnv env = new GRBEnv();
            GRBModel model = new GRBModel(env);

        }
    }
}

} catch (GRBException e) {
    System.out.println("Error code: " + e.getErrorCode() + ". " +
    e.getMessage());
}
model.set(GRB.StringAttr.ModelName, "assignment");

// Assignment variables: x[w][s] == 1 if worker w is assigned to shift s. Since an assignment model always produces integer solutions, we use continuous variables and solve as an LP.
GRBVar[][] x = new GRBVar[nWorkers][nShifts];
for (int w = 0; w < nWorkers; ++w) {
    for (int s = 0; s < nShifts; ++s) {
        x[w][s] =
            model.addVar(0, availability[w][s], pay[w], GRB.CONTINUOUS,
                        Workers[w] + "." + Shifts[s]);
    }
}

// The objective is to minimize the total pay costs
model.set(GRB.IntAttr.ModelSense, GRB.MINIMIZE);

// Constraint: assign exactly shiftRequirements[s] workers to each shift s
for (int s = 0; s < nShifts; ++s) {
    GRBLinExpr lhs = new GRBLinExpr();
    for (int w = 0; w < nWorkers; ++w) {
        lhs.addTerm(1.0, x[w][s]);
    }
    model.addConstr(lhs, GRB.EQUAL, shiftRequirements[s], Shifts[s]);
}

// Optimize
model.optimize();
int status = model.get(GRB.IntAttr.Status);
if (status == GRB.Status.UNBOUNDED) {
    System.out.println("The model cannot be solved because it is unbounded");
    return;
}
if (status == GRB.Status.OPTIMAL) {
    System.out.println("The optimal objective is "+model.get(GRB.DoubleAttr.ObjVal));
    return;
}
if (status != GRB.Status.INF_OR_UNBD && status != GRB.Status.INFEASIBLE) {
    System.out.println("Optimization was stopped with status "+status);
    return;
}

// Do IIS
System.out.println("The model is infeasible; computing IIS");
LinkedList<String> removed = new LinkedList<String>();

// Loop until we reduce to a model that can be solved
while (true) {
    model.computeIIS();
    System.out.println("The following constraint cannot be satisfied:");
    for (GRBConstr c : model.getConstrs()) {
        if (c.get(GRB.IntAttr.IISConstr) == 1) {
            removed.add(c.getConstrName());
        }
    }
}
System.out.println(c.get(GRB.StringAttr.ConstrName));
// Remove a single constraint from the model
removed.add(c.get(GRB.StringAttr.ConstrName));
model.remove(c);
break;
}
}
System.out.println();
model.optimize();
status = model.get(GRB.IntAttr.Status);

if (status == GRB.Status.UNBOUNDED) {
    System.out.println("The model cannot be solved "
    + "because it is unbounded");
    return;
} 
if (status == GRB.Status.OPTIMAL) {
    break;
}
if (status != GRB.Status.INF_OR_UNBD &&
    status != GRB.Status.INFEASIBLE ) {
    System.out.println("Optimization was stopped with status " +
    status);
    return;
}

System.out.println("\nThe following constraints were removed "
    + "to get a feasible LP:");
for (String s : removed) {
    System.out.print(s + " ");
}
System.out.println();

// Dispose of model and environment
model.dispose();
env.dispose();

} catch (GRBException e) {
    System.out.println("Error code: " + e.getErrorCode() + ". " +
    e.getMessage());
}
}

Workforce3.java

/* Copyright 2020, Gurobi Optimization, LLC */

/* Assign workers to shifts; each worker may or may not be available on a
particular day. If the problem cannot be solved, relax the model
to determine which constraints cannot be satisfied, and how much
they need to be relaxed. */

import gurobi.*;
public class Workforce3 {

  public static void main(String[] args) {
    try {

      // Sample data
      // Sets of days and workers
      String Shifts[] =
      String Workers[] =

      int nShifts = Shifts.length;
      int nWorkers = Workers.length;

      // Number of workers required for each shift
      double shiftRequirements[] =
          new double[] { 3, 2, 4, 4, 5, 6, 5, 2, 2, 3, 4, 6, 7, 5 };

      // Amount each worker is paid to work one shift
      double pay[] =
          new double[] { 10, 12, 10, 8, 8, 9, 11 };

      // Worker availability: 0 if the worker is unavailable for a shift
      double availability[][] =
          new double[][] { { 0, 1, 1, 0, 1, 0, 1, 0, 1, 1, 1, 1, 0, 1 },
                          { 0, 0, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1 },
                          { 0, 1, 1, 0, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1 },
                          { 1, 1, 1, 1, 0, 1, 1, 0, 0, 1, 1, 1, 1, 1 },
                          { 1, 1, 1, 0, 1, 0, 1, 0, 1, 1, 1, 1, 1, 1 },
                          { 1, 1, 0, 1, 1, 0, 1, 0, 1, 1, 1, 1, 1, 1 } };

      // Model
      GRBEnv env = new GRBEnv();
      GRBModel model = new GRBModel(env);
      model.set(GRB.StringAttr.ModelName, "assignment");

      // Assignment variables: x[w][s] == 1 if worker w is assigned
      // to shift s. Since an assignment model always produces integer
      // solutions, we use continuous variables and solve as an LP.
      GRBVar[][] x = model.addVars(nWorkers, nShifts);
      for (int w = 0; w < nWorkers; ++w) {
        for (int s = 0; s < nShifts; ++s) {
          x[w][s] =
              model.addVar(0, availability[w][s], pay[w], GRB.CONTINUOUS, Workers[w] + "." + Shifts[s]);
        }
      }

      // The objective is to minimize the total pay costs
      model.set(GRB.IntAttr.ModelSense, GRB.MINIMIZE);

      // Constraint: assign exactly shiftRequirements[s] workers

    }
  }
}
// to each shift s
for (int s = 0; s < nShifts; ++s) {
    GRBLinExpr lhs = new GRBLinExpr();
    for (int w = 0; w < nWorkers; ++w) {
        lhs.addTerm(1.0, x[w][s]);
    }
    model.addConstr(lhs, GRB.EQUAL, shiftRequirements[s], Shifts[s]);
}

// Optimize
model.optimize();
int status = model.get(GRB.IntAttr.Status);
if (status == GRB.UNBOUNDED) {
    System.out.println("The model cannot be solved "+ "because it is unbounded");
    return;
}
if (status == GRB.OPTIMAL) {
    System.out.println("The optimal objective is "+ model.get(GRB.DoubleAttr.ObjVal));
    return;
}
if (status != GRB.INF_OR_UNBD &&
    status != GRB.INFEASIBLE) {
    System.out.println("Optimization was stopped with status " + status);
    return;
}

// Relax the constraints to make the model feasible
System.out.println("The model is infeasible; relaxing the constraints");
int orignumvars = model.get(GRB.IntAttr.NumVars);
model.feasRelax(0, false, false, true);
model.optimize();
status = model.get(GRB.IntAttr.Status);
if (status == GRB.INF_OR_UNBD ||
    status == GRB.INFEASIBLE ||
    status == GRB.UNBOUNDED) {
    System.out.println("The relaxed model cannot be solved "+ "because it is infeasible or unbounded");
    return;
}
if (status != GRB.OPTIMAL) {
    System.out.println("Optimization was stopped with status " + status);
    return;
}

System.out.println("\nSlack values:");
GRBVar[] vars = model.getVars();
for (int i = orignumvars; i < model.get(GRB.IntAttr.NumVars); ++i) {
    GRBVar sv = vars[i];
    if (sv.get(GRB.DoubleAttr.X) > 1e-6) {
        System.out.println(sv.get(GRB.StringAttr.VarName) + " = " + sv.get(GRB.DoubleAttr.X));
    }
}
// Dispose of model and environment
model.dispose();
env.dispose();

} catch (GRBException e) {
    System.out.println("Error code: "+ e.getErrorCode()+ ". "+ e.getMessage());
}

}

Workforce4.java

/* Copyright 2020, Gurobi Optimization, LLC */

/* Assign workers to shifts; each worker may or may not be available on a particular day. We use Pareto optimization to solve the model: first, we minimize the linear sum of the slacks. Then, we constrain the sum of the slacks, and we minimize a quadratic objective that tries to balance the workload among the workers. */

import gurobi.*;

public class Workforce4 {

    public static void main(String[] args) {
        try {
            // Sample data
            // Sets of days and workers
            String[] Workers = new String[] {"Amy", "Bob", "Cathy", "Dan", "Ed", "Fred", "Gu"};

            int nShifts = Shifts.length;
            int nWorkers = Workers.length;

            // Number of workers required for each shift
double[] shiftRequirements =
                new double[]{3, 2, 4, 4, 5, 6, 5, 2, 2, 3, 4, 6, 7, 5};

            // Worker availability: 0 if the worker is unavailable for a shift
double[] availability =
                new double[]{0, 1, 0, 0, 1, 0, 0, 1, 0, 0, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 0, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1};

            // Model
            GRBEnv env = new GRBEnv();
GRBModel model = new GRBModel(env);
model.set(GRB.StringAttr.ModelName, "assignment");

// Assignment variables: x[w][s] == 1 if worker w is assigned to shift s. This is no longer a pure assignment model, so we must use binary variables.
GRBVar[][] x = new GRBVar[nWorkers][nShifts];
for (int w = 0; w < nWorkers; ++w) {
    for (int s = 0; s < nShifts; ++s) {
        x[w][s] = model.addVar(0, availability[w][s], 0, GRB.BINARY, Workers[w] + "." + Shifts[s]);
    }
}

// Slack variables for each shift constraint so that the shifts can be satisfied
GRBVar[] slacks = new GRBVar[nShifts];
for (int s = 0; s < nShifts; ++s) {
    slacks[s] = model.addVar(0, GRB.INFINITY, 0, GRB.CONTINUOUS, Shifts[s] + "Slack");
}

// Variable to represent the total slack
GRBVar totSlack = model.addVar(0, GRB.INFINITY, 0, GRB.CONTINUOUS, "totSlack");

// Variables to count the total shifts worked by each worker
GRBVar[] totShifts = new GRBVar[nWorkers];
for (int w = 0; w < nWorkers; ++w) {
    totShifts[w] = model.addVar(0, GRB.INFINITY, 0, GRB.CONTINUOUS, Workers[w] + "TotShifts");
}

GRBLinExpr lhs;

// Constraint: assign exactly shiftRequirements[s] workers to each shift s, plus the slack
for (int s = 0; s < nShifts; ++s) {
    lhs = new GRBLinExpr();
    lhs.addTerm(1.0, slacks[s]);
    for (int w = 0; w < nWorkers; ++w) {
        lhs.addTerm(1.0, x[w][s]);
    }
    model.addConstr(lhs, GRB.EQUAL, shiftRequirements[s], Shifts[s]);
}

// Constraint: set totSlack equal to the total slack
lhs = new GRBLinExpr();
lhs.addTerm(-1.0, totSlack);
for (int s = 0; s < nShifts; ++s) {
    lhs.addTerm(1.0, slacks[s]);
}model.addConstr(lhs, GRB.EQUAL, 0, "totSlack");
// Constraint: compute the total number of shifts for each worker
for (int w = 0; w < nWorkers; ++w) {
    lhs = new GRBLinExpr();
    lhs.addTerm(-1.0, totShifts[w]);
    for (int s = 0; s < nShifts; ++s) {
        lhs.addTerm(1.0, x[w][s]);
    }
    model.addConstr(lhs, GRB.EQUAL, 0, "totShifts" + Workers[w]);
}

// Objective: minimize the total slack
GRBLinExpr obj = new GRBLinExpr();
obj.addTerm(1.0, totSlack);
model.setObjective(obj);

// Optimize
int status =
    solveAndPrint(model, totSlack, nWorkers, Workers, totShifts);
if (status != GRB.Status.OPTIMAL) {
    return;
}

// Constrain the slack by setting its upper and lower bounds
totSlack.set(GRB.DoubleAttr.UB, totSlack.get(GRB.DoubleAttr.X));
totSlack.set(GRB.DoubleAttr.LB, totSlack.get(GRB.DoubleAttr.X));

// Variable to count the average number of shifts worked
GRBVar avgShifts =
    model.addVar(0, GRB.INFINITY, 0, GRB.CONTINUOUS, "avgShifts");

// Variables to count the difference from average for each worker;
// note that these variables can take negative values.
GRBVar[] diffShifts = new GRBVar[nWorkers];
for (int w = 0; w < nWorkers; ++w) {
    diffShifts[w] = model.addVar(-GRB.INFINITY, GRB.INFINITY, 0,
                                GRB.CONTINUOUS, Workers[w] + "Diff");
}

// Constraint: compute the average number of shifts worked
lhs = new GRBLinExpr();
lhs.addTerm(-nWorkers, avgShifts);
for (int w = 0; w < nWorkers; ++w) {
    lhs.addTerm(1.0, totShifts[w]);
}
model.addConstr(lhs, GRB.EQUAL, 0, "avgShifts");

// Constraint: compute the difference from the average number of shifts
for (int w = 0; w < nWorkers; ++w) {
    lhs = new GRBLinExpr();
    lhs.addTerm(-1, diffShifts[w]);
    lhs.addTerm(-1, avgShifts);
    lhs.addTerm(1, totShifts[w]);
    model.addConstr(lhs, GRB.EQUAL, 0, Workers[w] + "Diff");
}

// Objective: minimize the sum of the square of the difference from the
// average number of shifts worked
GRBQuadExpr qobj = new GRBQuadExpr();
for (int w = 0; w < nWorkers; ++w) {
    qobj.addTerm(1.0, diffShifts[w], diffShifts[w]);
}
model.setObjective(qobj);

// Optimize
status =
solveAndPrint(model, totSlack, nWorkers, Workers, totShifts);
if (status != GRB.Status.OPTIMAL) {
    return;
}

// Dispose of model and environment
model.dispose();
env.dispose();
}
catch (GRBException e) {
    System.out.println("Error code: " + e.getErrorCode() + ". " + 
                     e.getMessage());
}

private static int solveAndPrint(GRBModel model, GRBVar totSlack,
                                   int nWorkers, String[] Workers,
                                   GRBVar[] totShifts) throws GRBException {

    model.optimize();
    int status = model.get(GRB.IntAttr.Status);
    if ((status == GRB.Status.INF_OR_UNBD ||
         status == GRB.Status.INFEASIBLE ||
         status == GRB.Status.UNBOUNDED) ) {
        System.out.println("The model cannot be solved 
                           + "because it is infeasible or unbounded");
        return status;
    }
    if (status != GRB.Status.OPTIMAL) {
        System.out.println("Optimization was stopped with status " + status);
        return status;
    }

    // Print total slack and the number of shifts worked for each worker
    System.out.println("\nTotal slack required: " +
                       totSlack.get(GRB.DoubleAttr.X));
    for (int w = 0; w < nWorkers; ++w) {
        System.out.println(Workers[w] + " worked " +
                            totShifts[w].get(GRB.DoubleAttr.X) + " shifts");
    }
    System.out.println("\n");
    return status;
}
Workforce5.java

/* Copyright 2020, Gurobi Optimization, LLC */

/* Assign workers to shifts; each worker may or may not be available on a particular day. We use multi-objective optimization to solve the model. The highest-priority objective minimizes the sum of the slacks (i.e., the total number of uncovered shifts). The secondary objective minimizes the difference between the maximum and minimum number of shifts worked among all workers. The second optimization is allowed to degrade the first objective by up to the smaller value of 10% and 2 */

import gurobi.*;

public class Workforce5 {

    public static void main(String[] args) {
        try {
            // Sample data
            // Sets of days and workers

            int nShifts = Shifts.length;
            int nWorkers = Workers.length;

            // Number of workers required for each shift
            double shiftRequirements[] = new double[] { 3, 2, 4, 4, 5, 6, 5, 2, 2, 3, 4, 6, 7, 5 };

            // Worker availability: 0 if the worker is unavailable for a shift
            double availability[][] = new double[][] { 
                { 0, 1, 1, 0, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1 },
                { 1, 1, 0, 1, 0, 1, 0, 1, 1, 0, 1, 0, 1, 0 },
                { 0, 0, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1 },
                { 0, 1, 1, 0, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1 },
                { 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1 },
                { 1, 1, 1, 0, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1 },
                { 0, 1, 1, 1, 0, 1, 1, 1, 0, 1, 1, 0, 1, 1 },
                { 1, 1, 1, 0, 0, 1, 0, 1, 1, 0, 1, 0, 1, 1 },
                { 1, 1, 1, 0, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1 }
            };

            // Create environment
            GRBEnv env = new GRBEnv();

            // Create initial model
            GRBModel model = new GRBModel(env);
            model.set(GRB.StringAttr.ModelName, "Workforce5");

            // Initialize assignment decision variables:
            // x[w][s] == 1 if worker w is assigned to shift s.
// This is no longer a pure assignment model, so we must
// use binary variables.
GRBVar[][] x = new GRBVar[nWorkers][nShifts];
for (int w = 0; w < nWorkers; ++w) {
    for (int s = 0; s < nShifts; ++s) {
        x[w][s] =
            model.addVar(0, availability[w][s], 0, GRB.BINARY,
            Workers[w] + "." + Shifts[s]);
    }
}

// Slack variables for each shift constraint so that the shifts can
// be satisfied
GRBVar[] slacks = new GRBVar[nShifts];
for (int s = 0; s < nShifts; ++s) {
    slacks[s] =
        model.addVar(0, GRB.INFINITY, 0, GRB.CONTINUOUS,
        Shifts[s] + "Slack");
}

// Variable to represent the total slack
GRBVar totSlack = model.addVar(0, GRB.INFINITY, 0, GRB.CONTINUOUS,
    "totSlack");

// Variables to count the total shifts worked by each worker
GRBVar[] totShifts = new GRBVar[nWorkers];
for (int w = 0; w < nWorkers; ++w) {
    totShifts[w] = model.addVar(0, GRB.INFINITY, 0, GRB.CONTINUOUS,
        Workers[w] + "TotShifts");
}

GRBLinExpr lhs;
// Constraint: assign exactly shiftRequirements[s] workers
// to each shift s, plus the slack
for (int s = 0; s < nShifts; ++s) {
    lhs = new GRBLinExpr();
    lhs.addTerm(1.0, slacks[s]);
    for (int w = 0; w < nWorkers; ++w) {
        lhs.addTerm(1.0, x[w][s]);
    }
    model.addConstr(lhs, GRB.EQUAL, shiftRequirements[s], Shifts[s]);
}

// Constraint: set totSlack equal to the total slack
lhs = new GRBLinExpr();
lhs.addTerm(-1.0, totSlack);
for (int s = 0; s < nShifts; ++s) {
    lhs.addTerm(1.0, slacks[s]);
}
model.addConstr(lhs, GRB.EQUAL, 0, "totSlack");

// Constraint: compute the total number of shifts for each worker
for (int w = 0; w < nWorkers; ++w) {
    lhs = new GRBLinExpr();
    lhs.addTerm(-1.0, totShifts[w]);
for (int s = 0; s < nShifts; ++s) {
    lhs.addTerm(1.0, x[w][s]);
}
model.addConstr(lhs, GRB.EQUAL, 0, "totShifts" + Workers[w]);

// Constraint: set minShift/maxShift variable to less <=/= to the
// number of shifts among all workers
GRBVar minShift = model.addVar(0, GRB.INFINITY, 0, GRB.CONTINUOUS, "minShift");
GRBVar maxShift = model.addVar(0, GRB.INFINITY, 0, GRB.CONTINUOUS, "maxShift");
model.addGenConstrMin(minShift, totShifts, GRB.INFINITY, "minShift");
model.addGenConstrMax(maxShift, totShifts, -GRB.INFINITY, "maxShift");

// Set global sense for ALL objectives
model.set(GRB.IntAttr.ModelSense, GRB.MINIMIZE);

// Set primary objective
GRBLinExpr obj0 = new GRBLinExpr();
obj0.addTerm(1.0, totSlack);
model.setObjectiveN(obj0, 0, 2, 1.0, 2.0, 0.1, "TotalSlack");

// Set secondary objective
GRBLinExpr obj1 = new GRBLinExpr();
obj1.addTerm(1.0, maxShift);
obj1.addTerm(-1.0, minShift);
model.setObjectiveN(obj1, 1, 1, 1.0, 0.0, 0.0, "Fairness");

// Save problem
model.write("Workforce5.lp");

// Optimize
int status = solveAndPrint(model, totSlack, nWorkers, Workers, totShifts);
if (status != GRB.OPTIMAL)
    return;

// Dispose of model and environment
model.dispose();
env.dispose();

} catch (GRBException e) {
    System.out.println("Error code: " + e.getErrorCode() + ", " + e.getMessage());
}

private static int solveAndPrint(GRBModel model, GRBVar totSlack,
        int nWorkers, String[] Workers,
        GRBVar[] totShifts) throws GRBException {

    model.optimize();
    int status = model.get(GRB.IntAttr.Status);
    if (status == GRB.Status.INF_OR_UNBD ||
        status == GRB.Status.INFEASIBLE ||
        status == GRB.Status.UNBOUNDED) {
status == GRB.Status.UNBOUNDED ) {
    System.out.println("The model cannot be solved "
    + "because it is infeasible or unbounded");
    return status;
}
if (status != GRB.Status.OPTIMAL ) {
    System.out.println("Optimization was stopped with status " + status);
    return status;
}

// Print total slack and the number of shifts worked for each worker
System.out.println("\nTotal slack required: " +
    totSlack.get(GRB.DoubleAttr.X));
for (int w = 0; w < nWorkers; ++w) {
    System.out.println(Workers[w] + " worked " +
    totShifts[w].get(GRB.DoubleAttr.X) + " shifts");
}
System.out.println("\n");
return status;
}

3.4 C# Examples
This section includes source code for all of the Gurobi C# examples. The same source code can be
found in the examples/c# directory of the Gurobi distribution.

batchmode_cs.cs

/* Copyright 2020, Gurobi Optimization, LLC */

/* This example reads a MIP model from a file, solves it in batch mode,
and prints the JSON solution string.

You will need a Cluster Manager license for this example to work. */

using System;
using Gurobi;

class batchmode_cs
{
    /// <summary>Set-up the environment for batch mode optimization.
    /// </summary>
    /// <remarks>
    /// The function creates an empty environment, sets all necessary
    /// parameters, and returns the ready-to-be-started Env object to
    /// caller.
    /// </remarks>
    static void setupbatchenv(ref GRBEnv env)
    {
        env.CSBatchMode = 1;
        env.CSManager = "http://localhost:61080";
        env.LogFile = "batchmode.log";
        env.ServerPassword = "pass";
    }
env.UserName = "gurobi";

// No network communication happened up to this point. This will happen
// now that we call start().
env.Start();

///<summary>Print batch job error information, if any</summary>
static void printbatcherrorinfo(ref GRBBatch batch)
{
    if (batch.BatchErrorCode == 0)
        return;
    Console.WriteLine("Batch ID: " + batch.BatchID + ", Error code: " +
        batch.BatchErrorCode + "(" +
        batch.BatchErrorMessage + ")");
}

///<summary>Create a batch request for given problem file</summary>
static string newbatchrequest(string filename)
{
    string batchID = "";
    // Create an empty environment
    GRBEnv env = new GRBEnv(true);
    // set environment and build model
    setupbatchenv(ref env);
    GRBModel model = new GRBModel(env, filename);
    try {
        // Set some parameters
        model.Set(GRB.DoubleParam.MIPGap, 0.01);
        model.Set(GRB.IntParam.JSONSolDetail, 1);
        // Define tags for some variables to access their values later
        int count = 0;
        foreach (GRBVar v in model.GetVars()) {
            v.VTag = "Variable" + count;
            count += 1;
            if (count >= 10) break;
        }
        // Submit batch request
        batchID = model.OptimizeBatch();
    }
    finally {
        // Dispose of model and env
        model.Dispose();
        env.Dispose();
    }
    return batchID;
}

///<summary>Wait for the final status of the batch. Initially the
/// status of a batch is <see cref="GRB.BatchStatus.SUBMITTED"/>;
/// the status will change once the batch has been processed
/** (by a compute server).</summary>
static void waitforfinalstatus(string batchID)
{
    // Wait no longer than one hour
    double maxwaittime = 3600;
    DateTime start = DateTime.Now;

    // Setup and start environment, create local Batch handle object
    GRBEnv env = new GRBEnv(true);
    setupbatchenv(ref env);
    GRBBatch batch = new GRBBatch(env, batchID);
    try
    {
        while (batch.BatchStatus == GRB.BatchStatus.SUBMITTED) {
            // Abort this batch if it is taking too long
            TimeSpan interval = DateTime.Now - start;
            if (interval.TotalSeconds > maxwaittime) {
                batch.Abort();
                break;
            }
            // Wait for two seconds
            System.Threading.Thread.Sleep(2000);
        }

        // If the batch failed, we retry it
        if (batch.BatchStatus == GRB.BatchStatus.FAILED) {
            batch.Retry();
            System.Threading.Thread.Sleep(2000);
            batch.Update();
        }
    }
} finally {
    // Print information about error status of the job
    // that processed the batch
    printbatcherrorinfo(ref batch);
    batch.Dispose();
    env.Dispose();
}

/**Final Report for Batch Request*/
static void printfinalreport(string batchID)
{
    // Setup and start environment, create local Batch handle object
    GRBEnv env = new GRBEnv(true);
    setupbatchenv(ref env);
    GRBBatch batch = new GRBBatch(env, batchID);

    switch (batch.BatchStatus) {
        case GRB.BatchStatus.CREATED:
            Console.WriteLine("Batch status is 'CREATED'\n");
            break;
        case GRB.BatchStatus.SUBMITTED:
Console.WriteLine("Batch is 'SUBMITTED\n");
break;
case GRB.BatchStatus.ABORTED:
    Console.WriteLine("Batch is 'ABORTED\n");
    break;
case GRB.BatchStatus.FAILED:
    Console.WriteLine("Batch is 'FAILED\n");
    break;
case GRB.BatchStatus.COMPLETED:
    Console.WriteLine("Batch is 'COMPLETED\n");
    // Get JSON solution as string
    Console.WriteLine("JSON solution: " + batch.GetJSONSolution());

    // Write the full JSON solution string to a file
    batch.WriteJSONSolution("batch-sol.json.gz");
    break;
default:
    // Should not happen
    Console.WriteLine("Unknown BatchStatus " + batch.BatchStatus);
    Environment.Exit(1);
    break;
}
// Cleanup
batch.Dispose();
env.Dispose();
}

///<summary>Instruct the cluster manager to discard all data relating
/// to this BatchID</summary>
static void batchdiscard(string batchID)
{
    // Setup and start environment, create local Batch handle object
    GRBEnv env = new GRBEnv(true);
    setupbatchenv(ref env);
    GRBBatch batch = new GRBBatch(env, batchID);

    // Remove batch request from manager
    batch.Discard();

    // Cleanup
    batch.Dispose();
    env.Dispose();
}

///<summary>Solve a given model using batch optimization</summary>
static void Main(string[] args)
{
    if (args.Length < 1) {
        Console.Out.WriteLine("Usage: batchmode_cs filename");
        return;
    }

    try {
        // Submit new batch request
        string batchSize = newbatchrequest(args[0]);
// Wait for final status
waitForFinalStatus(batchID);

// Report final status info
printfinalreport(batchID);

// Remove batch request from manager
batchdiscard(batchID);

Console.WriteLine("Batch optimization OK");
}
} catch (GRBException e) {
    Console.WriteLine("Error code: "+e.ErrorCode+". "+e.Message);
}
}

bilinear_cs.cs

/* Copyright 2020, Gurobi Optimization, LLC */

/* This example formulates and solves the following simple bilinear model:

maximize x
subject to x + y + z <= 10
    x * y <= 2 (bilinear inequality)
    x * z + y * z == 1 (bilinear equality)
    x, y, z non-negative (x integral in second version)
 */

using System;
using Gurobi;

class bilinear_cs
{
    static void Main()
    {
        try {
            GRBEnv env = new GRBEnv("bilinear.log");
            GRBModel model = new GRBModel(env);

            // Create variables
            GRBVar x = model.AddVar(0.0, GRB.INFINITY, 0.0, GRB.CONTINUOUS, "x");
            GRBVar y = model.AddVar(0.0, GRB.INFINITY, 0.0, GRB.CONTINUOUS, "y");
            GRBVar z = model.AddVar(0.0, GRB.INFINITY, 0.0, GRB.CONTINUOUS, "z");

            // Set objective
            GRBLinExpr obj = x;
            model.SetObjective(obj, GRB.MAXIMIZE);

            // Add linear constraint: x + y + z <= 10
            model.AddConstr(x + y + z <= 10, "c0");
        }
    }
}
// Add bilinear inequality: x * y <= 2
model.AddQConstr(x*y <= 2, "bilinear0");

// Add bilinear equality: x * z + y * z == 1
model.AddQConstr(x*z + y*z == 1, "bilinear1");

// Optimize model
try {
    model.Optimize();
    catch (GRBException e) {
    Console.WriteLine("Failed (as expected) " + e.ErrorCode + ". " + e.Message);
    }
}

model.Set(GRB.IntParam.NonConvex, 2);
model.Optimize();

Console.WriteLine(x.VarName + " " + x.X);
Console.WriteLine(y.VarName + " " + y.X);
Console.WriteLine(z.VarName + " " + z.X);

Console.WriteLine("Obj: " + model.ObjVal + " " + obj.Value);

x.Set(GRB.CharAttr.VType, GRB.INTEGER);
model.Optimize();

Console.WriteLine(x.VarName + " " + x.X);
Console.WriteLine(y.VarName + " " + y.X);
Console.WriteLine(z.VarName + " " + z.X);

Console.WriteLine("Obj: " + model.ObjVal + " " + obj.Value);

// Dispose of model and env
model.Dispose();
env.Dispose();

} catch (GRBException e) {
    Console.WriteLine("Error code: " + e.ErrorCode + ". " + e.Message);
}

}
optimization of a MIP model once at least one of the following two conditions have been satisfied:
1) The optimality gap is less than 10%
2) At least 10000 nodes have been explored, and an integer feasible solution has been found.
Note that termination is normally handled through Gurobi parameters (MIPGap, NodeLimit, etc.). You should only use a callback for termination if the available parameters don’t capture your desired termination criterion.

*/
using System;
using System.IO;
using Gurobi;

class callback_cs : GRBCallback
{
private double lastiter;
private double lastnode;
private GRBVar[] vars;
private StreamWriter logfile;

public callback_cs (GRBVar[] xvars, StreamWriter xlogfile)
{
    lastiter = lastnode = -GRB.INFINITY;
    vars = xvars;
    logfile = xlogfile;
}

protected override void Callback()
{
    try {
        if (where == GRB.Callback.POLLING) {
            // Ignore polling callback
        } else if (where == GRB.Callback.PRESOLVE) {
            // Presolve callback
            int cdels = GetIntInfo(GRB.Callback.PRE_COLDEL);
            int rdels = GetIntInfo(GRB.Callback.PRE_ROWDEL);
            if (cdels != 0 || rdels != 0) {
                Console.WriteLine(cdels + " columns and " + rdels + " rows are removed");
            }
        } else if (where == GRB.Callback.SIMPLEX) {
            // Simplex callback
            double itcnt = GetDoubleInfo(GRB.Callback.SPX_ITRCNT);
            if (itcnt - lastiter >= 100) {
                lastiter = itcnt;
                double obj = GetDoubleInfo(GRB.Callback.SPX_OBJVAL);
                int ispert = GetIntInfo(GRB.Callback.SPX_ISPERT);
                double pinf = GetDoubleInfo(GRB.Callback.SPX_PRIMINF);
                double dinf = GetDoubleInfo(GRB.Callback.SPX_DUALINF);
                char ch;
                if (ispert == 0) ch = ' ';
                else if (ispert == 1) ch = 'S';
                else ch = 'P';
                Console.WriteLine(itcnt + " " + obj + ch + " "

```
    + pinf + " " + dinf);
}
} else if (where == GRB.Callback.MIP) {
    // General MIP callback
    double nodelcnt = GetDoubleInfo(GRB.Callback.MIP_NODCNT);
    double objbst = GetDoubleInfo(GRB.Callback.MIP_OBJBST);
    double objbnd = GetDoubleInfo(GRB.Callback.MIP_OBJBND);
    int solcnt = GetIntInfo(GRB.Callback.MIP_SOLCNT);
    if (nodelcnt - lastnode >= 100) {
        lastnode = nodelcnt;
        int actnodes = (int) GetDoubleInfo(GRB.Callback.MIP_NODLFT);
        int itcnt = (int) GetDoubleInfo(GRB.Callback.MIP_ITRCNT);
        int cutcnt = GetIntInfo(GRB.Callback.MIP_CUTCNT);
        Console.WriteLine(nodelcnt + " " + actnodes + " "
        + itcnt + " " + objbst + " " + objbnd + " "
        + solcnt + " " + cutcnt);
    }
    if (Math.Abs(objbst - objbnd) < 0.1 * (1.0 + Math.Abs(objbst))) {
        Console.WriteLine("Stop early - 10% gap achieved");
        Abort();
    }
    if (nodelcnt >= 10000 && solcnt > 0) {
        Console.WriteLine("Stop early - 10000 nodes explored");
        Abort();
    }
} else if (where == GRB.Callback.MIPSOL) {
    // MIP solution callback
    int nodelcnt = (int) GetDoubleInfo(GRB.Callback.MIPSOL_NODCNT);
    double obj = GetDoubleInfo(GRB.Callback.MIPSOL_OBJ);
    int solcnt = GetIntInfo(GRB.Callback.MIPSOL_SOLCNT);
    double[] x = GetSolution(vars);
    Console.WriteLine("**** New solution at node " + nodelcnt
    + "", obj " + obj + ", sol " + solcnt
    + ", x[0] = " + x[0] + " ****");
} else if (where == GRB.Callback.MIPNODE) {
    // MIP node callback
    Console.WriteLine("**** New node ****");
    if (GetIntInfo(GRB.Callback.MIPNODE_STATUS) == GRB.Status.OPTIMAL) {
        double[] x = GetNodeRel(vars);
        SetSolution(vars, x);
    }
} else if (where == GRB.Callback.BARRIER) {
    // Barrier callback
    int itcnt = GetIntInfo(GRB.Callback.BARRIER_ITRCNT);
    double primobj = GetDoubleInfo(GRB.Callback.BARRIER_PRIMOBJ);
    double dualobj = GetDoubleInfo(GRB.Callback.BARRIER_DUALOBJ);
    double priminf = GetDoubleInfo(GRB.Callback.BARRIER_PRIMINF);
    double dualinf = GetDoubleInfo(GRB.Callback.BARRIER_DUALINF);
    double cmpl = GetDoubleInfo(GRB.Callback.BARRIER_COMPL);
    Console.WriteLine(itcnt + " " + primobj + " " + dualobj + " "
    + priminf + " " + dualinf + " " + cmpl);
} else if (where == GRB.Callback.MESSAGE) {
    // Message callback
    string msg = GetStringInfo(GRB.Callback.MSG_STRING);
    if (msg != null) logfile.Write(msg);
}
} catch (GRBException e) {
    Console.WriteLine("Error code: " + e.ErrorCode);
    Console.WriteLine(e.Message);
    Console.WriteLine(e.StackTrace);
} catch (Exception e) {
    Console.WriteLine("Error during callback");
    Console.WriteLine(e.StackTrace);
}

static void Main(string[] args) {
    if (args.Length < 1) {
        Console.Out.WriteLine("Usage: callback_cs filename");
        return;
    }

    StreamWriter logfile = null;
    try {
        // Create environment
        GRBEnv env = new GRBEnv();

        // Read model from file
        GRBModel model = new GRBModel(env, args[0]);

        // Turn off display and heuristics
        model.Parameters.OutputFlag = 0;
        model.Parameters.Heuristics = 0.0;

        // Open log file
        logfile = new StreamWriter("cb.log");

        // Create a callback object and associate it with the model
        GRBVar[] vars = model.GetVars();
        callback_cs cb = new callback_cs(vars, logfile);
        model.SetCallback(cb);

        // Solve model and capture solution information
        model.Optimize();
        Console.WriteLine("\n");
        Console.WriteLine("Optimization complete");
        if (model.SolCount == 0) {
            Console.WriteLine("No solution found, optimization status = " + model.Status);
        } else {
            Console.WriteLine("Solution found, objective = " + model.ObjVal);
            string[] vnames = model.Get(GRB.StringAttr.VarName, vars);
            double[] x = model.Get(GRB.DoubleAttr.X, vars);

            for (int j = 0; j < vars.Length; j++) {
                if (x[j] != 0.0) Console.WriteLine(vnames[j] + " " + x[j]);
            }
            
            { }
// Dispose of model and environment
model.Dispose();
env.Dispose();

} catch (GRBException e) {
    Console.WriteLine("Error code: " + e.ErrorCode);
    Console.WriteLine(e.Message);
    Console.WriteLine(e.StackTrace);
} catch (Exception e) {
    Console.WriteLine("Error during optimization");
    Console.WriteLine(e.Message);
    Console.WriteLine(e.StackTrace);
} finally {
    // Close log file
    if (logfile != null) logfile.Close();
}

} // dense_cs.cs

/* Copyright 2020, Gurobi Optimization, LLC */

/* This example formulates and solves the following simple QP model:

    minimize    x + y + x^2 + xy + y^2 + yz + z^2
    subject to  x + 2 y + 3 z >= 4
                 x + y >= 1
                 x, y, z non-negative

    The example illustrates the use of dense matrices to store A and Q
    (and dense vectors for the other relevant data). We don’t recommend
    that you use dense matrices, but this example may be helpful if you
    already have your data in this format. */

using System;
using Gurobi;

class dense_cs {

    protected static bool dense_optimize(GRBEnv env,
        int rows, int cols, double[] c, double[,] Q, double[,] A, char[] sense,
        double[] rhs, double[] lb, double[] ub, char[] vtype, double[] solution) {

bool success = false;

try {
  GRBModel model = new GRBModel(env);

  // Add variables to the model
  GRBVar[] vars = model.AddVars(lb, ub, null, vtype, null);

  // Populate A matrix
  for (int i = 0; i < rows; i++) {
    GRBLinExpr expr = new GRBLinExpr();
    for (int j = 0; j < cols; j++)
      if (A[i,j] != 0)
        expr.AddTerm(A[i,j], vars[j]); // Note: ‘+=’ would be much slower
    model.AddConstr(expr, sense[i], rhs[i], "");
  }

  // Populate objective
  GRBQuadExpr obj = new GRBQuadExpr();
  if (Q != null) {
    for (int i = 0; i < cols; i++)
      for (int j = 0; j < cols; j++)
        if (Q[i,j] != 0)
          obj.AddTerm(Q[i,j], vars[i], vars[j]); // Note: ‘+=’ would be much slower
    for (int j = 0; j < cols; j++)
      if (c[j] != 0)
        obj.AddTerm(c[j], vars[j]); // Note: ‘+=’ would be much slower
    model.SetObjective(obj);
  }

  // Solve model
  model.Optimize();

  // Extract solution
  if (model.Status == GRB.Status.OPTIMAL) {
    success = true;
    for (int j = 0; j < cols; j++)
      solution[j] = vars[j].X;
  }

  model.Dispose();
}

catch (GRBException e) {
  Console.WriteLine("Error code: "+ e.ErrorCode + ". " + e.Message);
}

return success;
public static void Main(String[] args) {
    try {
        GRBEnv env = new GRBEnv();

        double[] c = {1d, 1d, 0d};
        double[,] Q = {{1d, 1d, 0d}, {0d, 1d, 1d}, {0d, 0d, 1d}};
        double[,] A = {{1d, 2d, 3d}, {1d, 1d, 0d}};
        char[] sense = new char[] {'>', '>'};
        double[] rhs = {4d, 1d};
        double[] lb = {0d, 0d, 0d};
        bool success;
        double[] sol = {0d, 0d, 0d};

        success = dense_optimize(env, 2, 3, c, Q, A, sense, rhs, lb, null, null, sol);

        if (success) {
        }

        // Dispose of environment
        env.Dispose();
    } catch (GRBException e) {
        Console.WriteLine("Error code: " + e.ErrorCode + ". " + e.Message);
    }
}
}

diet_cs.cs

/* Copyright 2020, Gurobi Optimization, LLC */

/* Solve the classic diet model, showing how to add constraints
to an existing model. */

using System;
using Gurobi;

class diet_cs {
    static void Main() {
        try {

            // Nutrition guidelines, based on
            // USDA Dietary Guidelines for Americans, 2005

            string[] Categories =
                new string[] { "calories", "protein", "fat", "sodium" };
            int nCategories = Categories.Length;
            double[] minNutrition = new double[] { 1800, 91, 0, 0 };
            double[] maxNutrition = new double[] { 2200, GRB.INFINITY, 65, 1779 };

            // Set of foods
            string[] Foods =
        }
new string[] { "hamburger", "chicken", "hot dog", "fries", "macaroni", "pizza", "salad", "milk", "ice cream" };  
int nFoods = Foods.Length;  
double[] cost = 
new double[] { 2.49, 2.89, 1.50, 1.89, 2.09, 1.99, 2.49, 0.89, 1.59 };  

// Nutrition values for the foods 
double[,] nutritionValues = new double[,] {  
{ 410, 24, 26, 730 }, // hamburger  
{ 420, 32, 10, 1190 }, // chicken  
{ 560, 20, 32, 1800 }, // hot dog  
{ 380, 4, 19, 270 }, // fries  
{ 320, 12, 10, 930 }, // macaroni  
{ 320, 15, 12, 820 }, // pizza  
{ 320, 31, 12, 1230 }, // salad  
{ 100, 8, 2.5, 125 }, // milk  
{ 330, 8, 10, 180 } // ice cream  
};  

// Model  
GRBEnv env = new GRBEnv();  
GRBModel model = new GRBModel(env);  
model.ModelName = "diet";  

// Create decision variables for the nutrition information,  
// which we limit via bounds  
GRBVar[] nutrition = new GRBVar[nCategories];  
for (int i = 0; i < nCategories; ++i) {  
nutrition[i] =  
model.AddVar(minNutrition[i], maxNutrition[i], 0, GRB.CONTINUOUS,  
Categories[i]);  
}  

// Create decision variables for the foods to buy  
GRBVar[] buy = new GRBVar[nFoods];  
for (int j = 0; j < nFoods; ++j) {  
buy[j] =  
model.AddVar(0, GRB.INFINITY, cost[j], GRB.CONTINUOUS, Foods[j]);  
}  

// The objective is to minimize the costs  
model.ModelSense = GRB.MINIMIZE;  

// Nutrition constraints  
for (int i = 0; i < nCategories; ++i) {  
GRBLinExpr ntot = 0.0;  
for (int j = 0; j < nFoods; ++j)  
nTot.AddTerm(nutritionValues[j,i], buy[j]);  
model.AddConstr(ntot == nutrition[i], Categories[i]);  
}  

// Solve  
model.Optimize();  
PrintSolution(model, buy, nutrition);
```csharp
Console.WriteLine("Adding constraint: at most 6 servings of dairy");

// Solve
model.Optimize();
PrintSolution(model, buy, nutrition);

// Dispose of model and env
model.Dispose();
env.Dispose();

} catch (GRBException e) {
    Console.WriteLine("Error code: " + e.ErrorCode + ". " + e.Message);
}

private static void PrintSolution(GRBModel model, GRBVar[] buy, GRBVar[] nutrition) {
    if (model.Status == GRB.Status.OPTIMAL) {
        Console.WriteLine("Cost: " + model.ObjVal);
        Console.WriteLine("Buy: ");
        for (int j = 0; j < buy.Length; ++j) {
            if (buy[j].X > 0.0001) {
                Console.WriteLine(buy[j].VarName + " " + buy[j].X);
            }
        }
        Console.WriteLine("Nutrition: ");
        for (int i = 0; i < nutrition.Length; ++i) {
            Console.WriteLine(nutrition[i].VarName + " " + nutrition[i].X);
        }
    } else {
        Console.WriteLine("No solution");
    }
}

facility_cs.cs

/* Copyright 2020, Gurobi Optimization, LLC */

/* Facility location: a company currently ships its product from 5 plants to 4 warehouses. It is considering closing some plants to reduce costs. What plant(s) should the company close, in order to minimize transportation and fixed costs?

Based on an example from Frontline Systems:
http://www.solver.com/disfacility.htm
Used with permission. */

using System;
using Gurobi;

class facility_cs
```
{  
    static void Main()
    {
        try {

            // Warehouse demand in thousands of units
            double[] Demand = new double[] { 15, 18, 14, 20 };  

            // Plant capacity in thousands of units
            double[] Capacity = new double[] { 20, 22, 17, 19, 18 };  

            // Fixed costs for each plant
            double[] FixedCosts = new double[] { 12000, 15000, 17000, 13000, 16000 };  

            // Transportation costs per thousand units
            double[,] TransCosts =
                new double[,] {
                    { 4000, 2000, 3000, 2500, 4500 },
                    { 2500, 2600, 3400, 3000, 4000 },
                    { 1200, 1800, 2600, 4100, 3000 },
                    { 2200, 2600, 3100, 3700, 3200 }  
                };

            // Number of plants and warehouses
            int nPlants = Capacity.Length;
            int nWarehouses = Demand.Length;

            // Model
            GRBEnv env = new GRBEnv();
            GRBModel model = new GRBModel(env);

            model.ModelName = "facility";

            // Plant open decision variables: open[p] == 1 if plant p is open.
            GRBVar[] open = new GRBVar[nPlants];
            for (int p = 0; p < nPlants; ++p) {
                open[p] = model.AddVar(0, 1, FixedCosts[p], GRB.BINARY, "Open" + p);
            }

            // Transportation decision variables: how much to transport from
            // a plant p to a warehouse w
            GRBVar[,] transport = new GRBVar[nWarehouses, nPlants];
            for (int w = 0; w < nWarehouses; ++w) {
                for (int p = 0; p < nPlants; ++p) {
                    transport[w, p] =
                        model.AddVar(0, GRB.INFINITY, TransCosts[w, p], GRB.CONTINUOUS,
                                    "Trans" + p + "," + w);
                }
            }

            // The objective is to minimize the total fixed and variable costs
            model.ModelSense = GRB.MINIMIZE;

            // Production constraints
            // Note that the right-hand limit sets the production to zero if
            // the plant is closed
            for (int p = 0; p < nPlants; ++p) {
            
        
    }
    
}
GRBLinExpr ptot = 0.0;
for (int w = 0; w < nWarehouses; ++w)
    ptot.AddTerm(1.0, transport[w,p]);
model.AddConstr(ptot <= Capacity[p] * open[p], "Capacity" + p);
}

// Demand constraints
for (int w = 0; w < nWarehouses; ++w) {
    GRBLinExpr dtot = 0.0;
    for (int p = 0; p < nPlants; ++p)
        dtot.AddTerm(1.0, transport[w,p]);
    model.AddConstr(dtot == Demand[w], "Demand" + w);
}

// Guess at the starting point: close the plant with the highest fixed costs; open all others

// First, open all plants
for (int p = 0; p < nPlants; ++p) {
    open[p].Start = 1.0;
}

// Now close the plant with the highest fixed cost
Console.WriteLine("Initial guess:");
double maxFixed = -GRB.INFINITY;
for (int p = 0; p < nPlants; ++p) {
    if (FixedCosts[p] > maxFixed) {
        maxFixed = FixedCosts[p];
    }
}
for (int p = 0; p < nPlants; ++p) {
    if (FixedCosts[p] == maxFixed) {
        open[p].Start = 0.0;
        Console.WriteLine("Closing plant "+ p + "\n");
        break;
    }
}

// Use barrier to solve root relaxation
model.Parameters.Method = GRB.METHOD_BARRIER;

// Solve
model.Optimize();

// Print solution
Console.WriteLine("\nTOTAL COSTS: " + model.ObjVal);
Console.WriteLine("SOLUTION:");
for (int p = 0; p < nPlants; ++p) {
    if (open[p].X > 0.99) {
        Console.WriteLine("Plant "+ p + " open:");
        for (int w = 0; w < nWarehouses; ++w) {
            if (transport[w,p].X > 0.0001) {
                Console.WriteLine(" Transport " +
                    transport[w,p].X + " units to warehouse " + w);
            }
        }
    }
}
else {
    Console.WriteLine("Plant " + p + " closed!");
}

// Dispose of model and env
model.Dispose();
env.Dispose();

} catch (GRBException e) {
    Console.WriteLine("Error code: " + e.ErrorCode + ". " + e.Message);
}

}  

 feasopt_cs.cs

/* Copyright 2020, Gurobi Optimization, LLC */

/* This example reads a MIP model from a file, adds artificial variables to each constraint, and then minimizes the sum of the artificial variables. A solution with objective zero corresponds to a feasible solution to the input model. We can also use FeasRelax feature to do it. In this example, we use minrelax=1, i.e. optimizing the returned model finds a solution that minimizes the original objective, but only from among those solutions that minimize the sum of the artificial variables. */

using Gurobi;
using System;

class feasopt_cs{
    static void Main(string[] args)
    {
    if (args.Length < 1) {
        Console.Out.WriteLine("Usage: feasopt_cs filename");
        return;
    }

    try {
        GRBEnv env = new GRBEnv();
        GRBModel feasmodel = new GRBModel(env, args[0]);

        // Create a copy to use FeasRelax feature later */
        GRBModel feasmodel1 = new GRBModel(feasmodel);

        // Clear objective
        feasmodel.SetObjective(new GRBLinExpr());

        // Add slack variables
        GRBConstr[] c = feasmodel.GetConstrs();
        for (int i = 0; i < c.Length; ++i) {
            char sense = c[i].Sense;
            if (sense != '>') {
                GRBConstr[] constrs = new GRBConstr[] { c[i] };
            }
        }
    }

    static void Main(string[] args)
    {{
double[] coeffs = new double[] { -1 };  
feasmodel.AddVar(0.0, GRB.INFINITY, 1.0, GRB.CONTINUOUS, conrs,  
coeffs, "ArtN_" + c[i].ConstrName);

if (sense != '<') {
    GRBConstr[] conrs = new GRBConstr[] { c[i] };  
double[] coeffs = new double[] { 1 };  
feasmodel.AddVar(0.0, GRB.INFINITY, 1.0, GRB.CONTINUOUS, conrs,  
coeffs, "ArtP_" +  
c[i].ConstrName);
}

// Optimize modified model  
feasmodel.Optimize();  
feasmodel.Write("feasopt.lp");

// Use FeasRelax feature */  
feasmodel1.FeasRelax(GRB.FEASRELAX_LINEAR, true, false, true);  
feasmodel1.Write("feasopt1.lp");  
feasmodel1.Optimize();

// Dispose of model and env  
feasmodel1.Dispose();  
feasmodel.Dispose();  
env.Dispose();

} catch (GRBException e) {
    Console.WriteLine("Error code: " + e.ErrorCode + ". " + e.Message);
}

fixanddive_cs.cs

/* Copyright 2020, Gurobi Optimization, LLC */

/* Implement a simple MIP heuristic. Relax the model,  
sort variables based on fractionality, and fix the 25% of  
the fractional variables that are closest to integer variables.  
Repeat until either the relaxation is integer feasible or  
linearly infeasible. */

using System;
using System.Collections.Generic;
using Gurobi;

class fixanddive_cs
{
    // Comparison class used to sort variable list based on relaxation  
    // fractionality

class FractionalCompare : IComparer<GRBVar>
{
    public int Compare(GRBVar v1, GRBVar v2)
    {
        // Implementation of Compare method
try {
    double sol1 = Math.Abs(v1.X);
    double sol2 = Math.Abs(v2.X);
    double frac1 = Math.Abs(sol1 - Math.Floor(sol1 + 0.5));
    double frac2 = Math.Abs(sol2 - Math.Floor(sol2 + 0.5));
    if (frac1 < frac2) {
        return -1;
    } else if (frac1 > frac2) {
        return 1;
    } else {
        return 0;
    }
} catch (GRBException e) {
    Console.WriteLine("Error code: "+ e.ErrorCode + ". " + e.Message);
}
return 0;

static void Main(string[] args) {
    if (args.Length < 1) {
        Console.Out.WriteLine("Usage: fixanddive_cs filename");
        return;
    }

    try {
        // Read model
        GRBEnv env = new GRBEnv();
        GRBModel model = new GRBModel(env, args[0]);

        // Collect integer variables and relax them
        List<GRBVar> intvars = new List<GRBVar>();
        foreach (GRBVar v in model.GetVars()) {
            if (v.VType != GRB.CONTINUOUS) {
                intvars.Add(v);
                v.VType = GRB.CONTINUOUS;
            }
        }

        model.Parameters.OutputFlag = 0;
        model.Optimize();

        // Perform multiple iterations. In each iteration, identify the first
        // quartile of integer variables that are closest to an integer value
        // in the relaxation, fix them to the nearest integer, and repeat.
        for (int iter = 0; iter < 1000; ++iter) {

            // create a list of fractional variables, sorted in order of
            // increasing distance from the relaxation solution to the nearest
            // integer value
            List<GRBVar> fractional = new List<GRBVar>();
            foreach (GRBVar v in intvars) {

```csharp
double sol = Math.Abs(v.X);
if (Math.Abs(sol - Math.Floor(sol + 0.5)) > 1e-5) {
    fractional.Add(v);
}
}

Console.WriteLine("Iteration "+ iter+", obj "+
    model.ObjVal+", fractional "+ fractional.Count);

if (fractional.Count == 0) {
    Console.WriteLine("Found feasible solution - objective "+
        model.ObjVal);
    break;
}

// Fix the first quartile to the nearest integer value
fractional.Sort(new FractionalCompare());
int nfix = Math.Max(fractional.Count / 4, 1);
for (int i = 0; i < nfix; ++i) {
    GRBVar v = fractional[i];
    double fixval = Math.Floor(v.X + 0.5);
    v.LB = fixval;
    v.UB = fixval;
    Console.WriteLine(" Fix "+ v.VarName+
        " to "+ fixval + " ( rel " + v.X + " )");
}
model.Optimize();

// Check optimization result
if (model.Status != GRB.Status.OPTIMAL) {
    Console.WriteLine("Relaxation is infeasible");
    break;
}

// Dispose of model and env
model.Dispose();
env.Dispose();

} catch (GRBException e) {
    Console.WriteLine("Error code: "+ e.ErrorCode+". "+
        e.Message);
}
```

gc_pwl_cs.cs

/* Copyright 2020, Gurobi Optimization, LLC

This example formulates and solves the following simple model
with PWL constraints:

```
maximize
    \sum c[j] * x[j]
subject to
    \sum A[i,j] * x[j] <= 0, for i = 0, ..., m-1
    \sum y[j] <= 3
    y[j] = \text{pwl}(x[j]), for j = 0, ..., n-1
    x[j] free, y[j] >= 0, for j = 0, ..., n-1
where \text{pwl}(x) = 0, if x = 0
    = 1+|x|, if x != 0

Note
1. \sum \text{pwl}(x[j]) <= b is to bound x vector and also to favor sparse x vector. Here b = 3 means that at most two x[j] can be nonzero and if two, then \sum x[j] <= 1
2. \text{pwl}(x) jumps from 1 to 0 and from 0 to 1, if x moves from negative 0 to 0, then to positive 0, so we need three points at x = 0. x has infinite bounds on both sides, the piece defined with two points (-1, 2) and (0, 1) can extend x to -infinite. Overall we can use five points (-1, 2), (0, 1), (0, 0), (0, 1) and (1, 2) to define y = \text{pwl}(x)

*/

using System;
using Gurobi;

public class gc_pwl_cs {

    public static void Main() {
        try {
            int n = 5;
            int m = 5;
            double[] c = new double[] { 0.5, 0.8, 0.5, 0.1, -1 };
            double[,] A = new double[,] {
                {0, 0, 0, 1, -1},
                {0, 0, 1, 1, -1},
                {1, 1, 0, 0, -1},
                {1, 0, 1, 0, -1},
                {1, 0, 0, 1, -1} };
            double[] xpts = new double[] {-1, 0, 0, 0, 1};
            double[] ypts = new double[] {2, 1, 0, 1, 2};

            // Env and model
            GRBEnv env = new GRBEnv();
            GRBModel model = new GRBModel(env);
            model.ModelName = "gc_pwl_cs";

            // Add variables, set bounds and obj coefficients
            GRBVar[] x = model.AddVars(n, GRB.CONTINUOUS);
            for (int i = 0; i < n; i++) {
                x[i].LB = -GRB.INFINITY;
                x[i].Obj = c[i];
            }
            GRBVar[] y = model.AddVars(n, GRB.CONTINUOUS);

            // Set objective to maximize
            model.ModelSense = GRB.MAXIMIZE;
        }
    }
}
// Add linear constraints
for (int i = 0; i < m; i++) {
    GRBLinExpr le = 0.0;
    for (int j = 0; j < n; j++) {
        le.AddTerm(A[i,j], x[j]);
    }
    model.AddConstr(le, GRB.LESS_EQUAL, 0, "cx" + i);
}

GRBLinExpr le1 = 0.0;
for (int j = 0; j < n; j++) {
    le1.AddTerm(1.0, y[j]);
}model.AddConstr(le1, GRB.LESS_EQUAL, 3, "cy");

// Add piecewise constraints
for (int j = 0; j < n; j++) {
    model.AddGenConstrPWL(x[j], y[j], xpts, ypts, "pwl" + j);
}

// Optimize model
model.Optimize();

for (int j = 0; j < n; j++) {
    Console.WriteLine("x[" + j + "] = " + x[j].X);
}Console.WriteLine("Obj: " + model.ObjVal);

// Dispose of model and environment
model.Dispose();
env.Dispose();
} catch (GRBException e) {
    Console.WriteLine("Error code: " + e.ErrorCode + ". " + e.Message);
}
}

gc_pwl_func_cs.cs

/* Copyright 2020, Gurobi Optimization, LLC

This example considers the following nonconvex nonlinear problem

maximize 2 x + y
subject to exp(x) + 4 sqrt(y) <= 9
       x, y >= 0

We show you two approaches to solve this:

1) Use a piecewise-linear approach to handle general function constraints (such as exp and sqrt).
   a) Add two variables
      u = exp(x)
      v = sqrt(y)
   b) Compute points (x, u) of u = exp(x) for some step length (e.g., x = 0, 1e-3, 2e-3, ..., xmax) and points (y, v) of v = sqrt(y) for

322
some step length (e.g., \( y = 0, 1e-3, 2e-3, \ldots, ymax \)). We need to compute \( xmax \) and \( ymax \) (which is easy for this example, but this does not hold in general).

c) Use the points to add two general constraints of type piecewise-linear.

2) Use the Gurobi's built-in general function constraints directly (\( \text{EXP} \) and \( \text{POW} \)). Here, we do not need to compute the points and the maximal possible values, which will be done internally by Gurobi. In this approach, we show how to "zoom in" on the optimal solution and tighten tolerances to improve the solution quality.

```csharp
using System;
using Gurobi;

class gc_pwl_func_cs {
    private static double f(double u) { return Math.Exp(u); }
    private static double g(double u) { return Math.Sqrt(u); }
    private static void printsol(GRBModel m, GRBVar x, GRBVar y, GRBVar u, GRBVar v) {
        Console.WriteLine("x = " + x.X + ", u = " + u.X);
        Console.WriteLine("y = " + y.X + ", v = " + v.X);
        Console.WriteLine(" Obj = " + m.ObjVal);

        // Calculate violation of \( \text{EXP}(x) + 4 \text{sqrt}(y) \leq 9 \)
        double vio = f(x.X) + 4 * g(y.X) - 9;
        if (vio < 0.0) vio = 0.0;
        Console.WriteLine(" Vio = " + vio);
    }

    static void Main() {
        try {

            // Create environment
            GRBEnv env = new GRBEnv();
            GRBModel m = new GRBModel(env);

            // Create a new m
            double lb = 0.0, ub = GRB.INFINITY;
            GRBVar x = m.AddVar(lb, ub, 0.0, GRB.CONTINUOUS, "x");
            GRBVar y = m.AddVar(lb, ub, 0.0, GRB.CONTINUOUS, "y");
            GRBVar u = m.AddVar(lb, ub, 0.0, GRB.CONTINUOUS, "u");
            GRBVar v = m.AddVar(lb, ub, 0.0, GRB.CONTINUOUS, "v");

            // Set objective
            m.SetObjective(2*x + y, GRB.MAXIMIZE);

            // Add linear constraint
```
m.AddConstr(u + 4*v <= 9, "l1");

// Approach 1) PWL constraint approach

double intv = 1e-3;
double xmax = Math.Log(9.0);
int len = (int) Math.Ceiling(xmax/intv) + 1;
double[] xpts = new double[len];
double[] upts = new double[len];
for (int i = 0; i < len; i++) {
    xpts[i] = i*intv;
    upts[i] = f(i*intv);
}
GRBGenConstr gc1 = m.AddGenConstrPWL(x, u, xpts, upts, "gc1");

double ymax = (9.0/4.0)*(9.0/4.0);
len = (int) Math.Ceiling(ymax/intv) + 1;
double[] ypts = new double[len];
double[] vpts = new double[len];
for (int i = 0; i < len; i++) {
    ypts[i] = i*intv;
    vpts[i] = g(i*intv);
}
GRBGenConstr gc2 = m.AddGenConstrPWL(y, v, ypts, vpts, "gc2");

// Optimize the model and print solution
m.Optimize();
printsol(m, x, y, u, v);

// Approach 2) General function constraint approach with auto PWL translation by Gurobi

// restore unsolved state and get rid of PWL constraints
m.Reset();
m.Remove(gc1);
m.Remove(gc2);
m.Update();

GRBGenConstr gcf1 = m.AddGenConstrExp(x, u, "gcf1", "");
GRBGenConstr gcf2 = m.AddGenConstrPow(y, v, 0.5, "gcf2", "");

m.Parameters.FuncPieceLength = 1e-3;

// Optimize the model and print solution
m.Optimize();
printsol(m, x, y, u, v);

// Zoom in, use optimal solution to reduce the ranges and use a smaller pclen=1e-5 to solve it
x.LB = Math.Max(x.LB, x.X-0.01);
x.UB = Math.Min(x.UB, x.X+0.01);
y.LB = Math.Max(y.LB, y.X-0.01);
y.UB = Math.Min(y.UB, y.X+0.01);
m. Update();
m. Reset();

m. Parameters.FuncPieceLength = 1e-5;

// Optimize the model and print solution
m. Optimize();
printsol(m, x, y, u, v);

// Dispose of model and environment
m. Dispose();
env. Dispose();
} catch (GRBException e) {
    Console.WriteLine("Error code: " + e.ErrorCode + ". " + e.Message);
}
}

genconstr_cs.cs

/* Copyright 2020, Gurobi Optimization, LLC */

/* In this example we show the use of general constraints for modeling some common expressions. We use as an example a SAT-problem where we want to see if it is possible to satisfy at least four (or all) clauses of the logical for

L = (x0 or ~x1 or x2) and (x1 or ~x2 or x3) and (x2 or ~x3 or x0) and (x3 or ~x0 or x1) and
(-x0 or ~x1 or x2) and (-x1 or ~x2 or x3) and (-x2 or ~x3 or x0) and (-x3 or ~x0 or x1)

We do this by introducing two variables for each literal (itself and its negated value), a variable for each clause, and then two variables for indicating if we can satisfy four, and another to identify the minimum of the clauses (so if it one, we can satisfy all clauses) and put these two variables in the objective. i.e. the Objective function will be

maximize Obj0 + Obj1

Obj0 = MIN(Clause1, ..., Clause8) 
Obj1 = 1 -> Clause1 + ... + Clause8 >= 4

thus, the objective value will be two if and only if we can satisfy all clauses; one if and only if at least four clauses can be satisfied, and zero otherwise. */

using System;
using Gurobi;

class genconstr_cs {

public const int n = 4;
public const int NLITERALS = 4; // same as n
public const int NCLAUSES = 8;
public const int NOBJ = 2;

static void Main() {
    try {
        // Example data:
        // e.g. {0, n+1, 2} means clause (x0 or -x1 or x2)
        int[,] Clauses = new int[,] {
            {0, n+1, 2}, {1, n+2, 3},
            {2, n+3, 0}, {3, n+0, 1},
            {n+0, n+1, 2}, {n+1, n+2, 3},
            {n+2, n+3, 0}, {n+3, n+0, 1}};

        int i, status;

        // Create environment
        GRBEnv env = new GRBEnv("genconstr_cs.log");

        // Create initial model
        GRBModel model = new GRBModel(env);
        model.ModelName = "genconstr_cs";

        // Initialize decision variables and objective
        GRBVar[] Lit = new GRBVar[NLITERALS];
        GRBVar[] NotLit = new GRBVar[NLITERALS];
        for (i = 0; i < NLITERALS; i++) {
            Lit[i] = model.AddVar(0.0, 1.0, 0.0, GRB.BINARY, string.Format("X{i}", i));
            NotLit[i] = model.AddVar(0.0, 1.0, 0.0, GRB.BINARY, string.Format("notX{i}", i));
        }

        GRBVar[] Cla = new GRBVar[NCLAUSES];
        for (i = 0; i < NCLAUSES; i++) {
            Cla[i] = model.AddVar(0.0, 1.0, 0.0, GRB.BINARY, string.Format("Clause{i}", i));
        }

        GRBVar[] Obj = new GRBVar[NOBJ];
        for (i = 0; i < NOBJ; i++) {
            Obj[i] = model.AddVar(0.0, 1.0, 1.0, GRB.BINARY, string.Format("Obj{i}", i));
        }

        // Link Xi and notXi
        GRBLinExpr lhs;
        for (i = 0; i < NLITERALS; i++) {
            lhs = new GRBLinExpr();
            lhs.AddTerm(1.0, Lit[i]);
            lhs.AddTerm(1.0, NotLit[i]);
            model.AddConstr(lhs, GRB.EQUAL, 1.0, string.Format("CNSTR_X{i}", i));
        }

        // Link clauses and literals
        for (i = 0; i < NCLAUSES; i++) {
            GRBVar[] clause = new GRBVar[3];
        }
    }
}
for (int j = 0; j < 3; j++) {
    if (Clauses[i,j] >= n) clause[j] = NotLit[Clauses[i,j]-n];
    else clause[j] = Lit[Clauses[i,j]];
}
model.AddGenConstrOr(Cla[i], clause, string.Format("CNSTR_Cla{0}", i));

// Link objs with clauses
model.AddGenConstrMin(Obj[0], Cla, GRB.INFINITY, "CNSTR_Obj0");
    lhs = new GRBLinExpr();
    for (i = 0; i < NCLAUSES; i++) {
        lhs.AddTerm(1.0, Cla[i]);
    }
model.AddGenConstrIndicator(Obj[1], 1, lhs, GRB.GREATER_EQUAL, 4.0, "CNSTR_Obj1");

// Set global objective sense
model.ModelSense = GRB.MAXIMIZE;

// Save problem
model.Write("genconstr_cs.mps");
model.Write("genconstr_cs.lp");

// Optimize
model.Optimize();

// Status checking
status = model.Status;

if (status == GRB.Status.INF_OR_UNBD ||
     status == GRB.Status.INFEASIBLE ||
     status == GRB.Status.UNBOUNDED ) {
    Console.WriteLine("The model cannot be solved " +
                      "because it is infeasible or unbounded");
    return;
}
if (status != GRB.Status.OPTIMAL) {
    Console.WriteLine("Optimization was stopped with status {0}", status);
    return;
}

// Print result
double objval = model.ObjVal;

if (objval > 1.9)
    Console.WriteLine("Logical expression is satisfiable");
else if (objval > 0.9)
    Console.WriteLine("At least four clauses can be satisfied");
else
    Console.WriteLine("Not even three clauses can be satisfied");

// Dispose of model and environment
model.Dispose();
env.Dispose();

} catch (GRBException e) {
    Console.WriteLine("Error code: {0}. {1}", e.ErrorCode, e.Message);
/* Copyright 2020, Gurobi Optimization, LLC */

/* This example reads an LP model from a file and solves it.
   If the model is infeasible or unbounded, the example turns off
   presolve and solves the model again. If the model is infeasible,
   the example computes an Irreducible Inconsistent Subsystem (IIS),
   and writes it to a file. */

using System;
using Gurobi;

class lp_cs {
    static void Main(string[] args)
    {
        if (args.Length < 1) {
            Console.Out.WriteLine("Usage: lp_cs filename");
            return;
        }

        try {
            GRBEnv env = new GRBEnv();
            GRBModel model = new GRBModel(env, args[0]);

            model.Optimize();

            int optimstatus = model.Status;

            if (optimstatus == GRB.Status.INF_OR_UNBD) {
                model.Parameters.Presolve = 0;
                model.Optimize();
                optimstatus = model.Status;
            }

            if (optimstatus == GRB.Status.OPTIMAL) {
                double objval = model.ObjVal;
                Console.WriteLine("Optimal objective: " + objval);
            } else if (optimstatus == GRB.Status.INFEASIBLE) {
                Console.WriteLine("Model is infeasible");

                // compute and write out IIS

                model.ComputeIIS();
                model.Write("model.ilp");
            } else if (optimstatus == GRB.Status.UNBOUNDED) {
                Console.WriteLine("Model is unbounded");
            } else {
                Console.WriteLine("Optimization was stopped with status = " + optimstatus);
            }
        }
    }
}

lp_cs.cs
// Dispose of model and env
model.Dispose();
env.Dispose();

} catch (GRBException e) {
    Console.WriteLine("Error code: " + e.ErrorCode + ". " + e.Message);
}
}

lpmethod_cs.cs

/* Copyright 2020, Gurobi Optimization, LLC */

/* Solve a model with different values of the Method parameter;
   show which value gives the shortest solve time. */

using System;
using Gurobi;

class lpmethod_cs {
    static void Main(string[] args) {
        if (args.Length < 1) {
            Console.Out.WriteLine("Usage: lpmethod_cs filename");
            return;
        }

        try {
            // Read model
            GRBEnv env = new GRBEnv();
            GRBModel model = new GRBModel(env, args[0]);

            // Solve the model with different values of Method
            int bestMethod = -1;
            double bestTime = model.Parameters.TimeLimit;
            for (int i = 0; i <= 2; ++i) {
                model.Reset();
                model.Parameters.Method = i;
                model.Optimize();
                if (model.Status == GRB.Status.OPTIMAL) {
                    bestTime = model.Runtime;
                    bestMethod = i;
                    // Reduce the TimeLimit parameter to save time
                    // with other methods
                    model.Parameters.TimeLimit = bestTime;
                }
            }

            // Report which method was fastest
            if (bestMethod == -1) {
                Console.WriteLine("Unable to solve this model");
            }
        }
    }
}
else {
    Console.WriteLine("Solved in " + bestTime
        + " seconds with Method: " + bestMethod);
}

// Dispose of model and env
model.Dispose();
env.Dispose();

} catch (GRBException e) {
    Console.WriteLine("Error code: " + e.ErrorCode + ". " + e.Message);
}
}

lpmod_cs.cs

/* Copyright 2020, Gurobi Optimization, LLC */

/* This example reads an LP model from a file and solves it. If the model can be solved, then it finds the smallest positive variable, sets its upper bound to zero, and resolves the model two ways: first with an advanced start, then without an advanced start (i.e. 'from scratch'). */

using System;
using Gurobi;

class lpmod_cs
{
    static void Main(string[] args)
    {
        if (args.Length < 1) {
            Console.Out.WriteLine("Usage: lpmod_cs filename");
            return;
        }

        try {
            // Read model and determine whether it is an LP
            GRBEnv env = new GRBEnv();
            GRBModel model = new GRBModel(env, args[0]);
            if (model.IsMIP != 0) {
                Console.WriteLine("The model is not a linear program");
                Environment.Exit(1);
            }

            model.Optimize();

            int status = model.Status;

            if ((status == GRB.Status.INF_OR_UNBD) ||
                (status == GRB.Status.INFEASIBLE) ||
                (status == GRB.Status.UNBOUNDED)) {
                Console.WriteLine("The model cannot be solved because it is "
                    + "infeasible or unbounded");
                Environment.Exit(1);
            }

            // Dispose of model and env
            model.Dispose();
            env.Dispose();
        }
    }
}
if (status != GRB.Status.OPTIMAL) {
    Console.WriteLine("Optimization was stopped with status "+ status);
    Environment.Exit(0);
}

// Find the smallest variable value
double minVal = GRB.INFINITY;
GRBVar minVar = null;
foreach (GRBVar v in model.GetVars()) {
    double sol = v.X;
    if ((sol > 0.0001) && (sol < minVal) && (v.LB == 0.0)) {
        minVal = sol;
        minVar = v;
    }
}

Console.WriteLine("\n *** Setting " +
    minVar.VarName + " from " + minVal +
    " to zero ***\n");
minVar.UB = 0.0;

// Solve from this starting point
model.Optimize();

// Save iteration & time info
double warmCount = model.IterCount;
double warmTime = model.Runtime;

// Reset the model and resolve
Console.WriteLine("\n *** Resetting and solving "
    + "without an advanced start ***\n");
model.Reset();
model.Optimize();

double coldCount = model.IterCount;
double coldTime = model.Runtime;

Console.WriteLine("\n *** Warm start: " + warmCount + " iterations, " +
    warmTime + " seconds");
Console.WriteLine("*** Cold start: " + coldCount + " iterations, " +
    coldTime + " seconds");

// Dispose of model and env
model.Dispose();
env.Dispose();

} catch (GRBException e) {
    Console.WriteLine("Error code: " + e.ErrorCode + ". " + e.Message);
}
}
mip1_cs.cs

/* Copyright 2020, Gurobi Optimization, LLC */

/* This example formulates and solves the following simple MIP model :
maximize x + y + 2 z
subject to x + 2 y + 3 z <= 4
        x + y    >= 1
               x, y, z binary
*/

using System;
using Gurobi;

class mip1_cs
{
    static void Main()
    {
        try {
            // Create an empty environment, set options and start
            GRBEnv env = new GRBEnv(true);
            env.Set("LogFile", "mip1.log");
            env.Start();

            // Create empty model
            GRBModel model = new GRBModel(env);

            // Create variables
            GRBVar x = model.AddVar(0.0, 1.0, 0.0, GRB.BINARY, "x");
            GRBVar y = model.AddVar(0.0, 1.0, 0.0, GRB.BINARY, "y");
            GRBVar z = model.AddVar(0.0, 1.0, 0.0, GRB.BINARY, "z");

            // Set objective: maximize x + y + 2 z
            model.SetObjective(x + y + 2 * z, GRB.MAXIMIZE);

            // Add constraint: x + 2 y + 3 z <= 4
            model.AddConstr(x + 2 * y + 3 * z <= 4.0, "c0");

            // Add constraint: x + y >= 1
            model.AddConstr(x + y >= 1.0, "c1");

            // Optimize model
            model.Optimize();

            Console.WriteLine(x.VarName + " " + x.X);
            Console.WriteLine(y.VarName + " " + y.X);
            Console.WriteLine(z.VarName + " " + z.X);

            Console.WriteLine("Obj: " + model.ObjVal);

            // Dispose of model and env
            model.Dispose();
            env.Dispose();
        }
    }
}
} catch (GRBException e) {
    Console.WriteLine("Error code: " + e.ErrorCode + ". " + e.Message);
}
}

mip2_cs.cs

/* Copyright 2020, Gurobi Optimization, LLC */

/* This example reads a MIP model from a file, solves it and
prints the objective values from all feasible solutions
generated while solving the MIP. Then it creates the fixed
model and solves that model. */

using System;
using Gurobi;

class mip2_cs
{
    static void Main(string[] args)
    {
        if (args.Length < 1) {
            Console.Out.WriteLine("Usage: mip2_cs filename");
            return;
        }

        try {
            GRBEnv env = new GRBEnv();
            GRBModel model = new GRBModel(env, args[0]);
            if (model.IsMIP == 0) {
                Console.WriteLine("Model is not a MIP");
                return;
            }

            model.Optimize();

            int optimstatus = model.Status;
            double objval = 0;
            if (optimstatus == GRB.Status.OPTIMAL) {
                objval = model.ObjVal;
                Console.WriteLine("Optimal objective: " + objval);
            } else if (optimstatus == GRB.Status.INF_OR_UNBD) {
                Console.WriteLine("Model is infeasible or unbounded");
                return;
            } else if (optimstatus == GRB.Status.INFEASIBLE) {
                Console.WriteLine("Model is infeasible");
                return;
            } else if (optimstatus == GRB.Status.UNBOUNDED) {
                Console.WriteLine("Model is unbounded");
                return;
            } else {
                Console.WriteLine("Optimization was stopped with status = "
                                + optimstatus);
                return;
            }
        }
    }
}
/* Iterate over the solutions and compute the objectives */

GRBVar[] vars = model.GetVars();
model.Parameters.OutputFlag = 0;

Console.WriteLine();
for (int k = 0; k < model.SolCount; ++k) {
    model.Parameters.SolutionNumber = k;
    double objn = 0.0;
    for (int j = 0; j < vars.Length; j++) {
        objn += vars[j].Obj * vars[j].Xn;
    }
    Console.WriteLine("Solution " + k + " has objective: " + objn);
}

model.Parameters.OutputFlag = 1;

/* Create a fixed model, turn off presolve and solve */

GRBModel fixedmodel = model.FixedModel();
fixedmodel.Parameters.Presolve = 0;
fixedmodel.Optimize();

int foptimstatus = fixedmodel.Status;
if (foptimstatus != GRB.Status.OPTIMAL) {
    Console.WriteLine("Error: fixed model isn't optimal");
    return;
}

double fobjval = fixedmodel.ObjVal;
if (Math.Abs(fobjval - objval) > 1.0e-6 * (1.0 + Math.Abs(objval))) {
    Console.WriteLine("Error: objective values are different");
    return;
}

GRBVar[] fvars = fixedmodel.GetVars();
double[] x = fixedmodel.Get(GRB.DoubleAttr.X, fvars);
string[] vnames = fixedmodel.Get(GRB.StringAttr.VarName, fvars);
for (int j = 0; j < fvars.Length; j++) {
    if (x[j] != 0.0) Console.WriteLine(vnames[j] + " = " + x[j]);
}

// Dispose of models and env
fixedmodel.Dispose();
model.Dispose();
env.Dispose();
}
catch (GRBException e) {

try {
    // Sample data
    int groundSetSize = 20;
    int nSubsets = 4;
    int Budget = 12;
    double[,] Set = new double[,] {
        { 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0 },
        { 0, 0, 0, 0, 0, 1, 1, 1, 1, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1 },
        { 0, 0, 0, 1, 1, 0, 1, 1, 0, 0, 0, 0, 1, 1, 0, 1, 1, 0, 0 },
        { 0, 0, 0, 1, 1, 1, 0, 0, 1, 1, 1, 0, 0, 0, 1, 1, 1, 0, 0 } },
    int[] SetObjPriority = new int[] {3, 2, 2, 1};
    double[] SetObjWeight = new double[] {1.0, 0.25, 1.25, 1.0};
    int e, i, status, nSolutions;

    // Create environment
    GRBEnv env = new GRBEnv("multiobj_cs.log");

    // Create initial model
    GRBModel model = new GRBModel(env);
    model.ModelName = "multiobj_cs";

    // Initialize decision variables for ground set:
    // x[e] == 1 if element e is chosen for the covering.
    GRBVar[] Elem = model.AddVars(groundSetSize, GRB.BINARY);
    for (e = 0; e < groundSetSize; e++) {
        string vname = string.Format("El{0}", e);
        Elem[e].VarName = vname;
    }

    // Constraint: limit total number of elements to be picked to be at most
    // Budget
    GRBLinExpr lhs = new GRBLinExpr();
    for (e = 0; e < groundSetSize; e++) {
        lhs.AddTerm(1.0, Elem[e]);
    }
    model.AddConstr(lhs, GRB.LESS_EQUAL, Budget, "Budget");
}
// Set global sense for ALL objectives
model.ModelSense = GRB.MAXIMIZE;

// Limit how many solutions to collect
model.Parameters.PoolSolutions = 100;

// Set and configure i-th objective
for (i = 0; i < nSubsets; i++) {
    string vname = string.Format("Set{0}", i);
    GRBLinExpr objn = new GRBLinExpr();
    for (e = 0; e < groundSetSize; e++) {
        objn.AddTerm(Set[i,e], Elem[e]);
    }
    model.SetObjectiveN(objn, i, SetObjPriority[i], SetObjWeight[i],
        1.0 + i, 0.01, vname);
}

// Save problem
model.Write("multiobj_cs.lp");

// Optimize
model.Optimize();

// Status checking
status = model.Status;

if (status == GRB.Status.INF_OR_UNBD ||
    status == GRB.Status.INFEASIBLE ||
    status == GRB.Status.UNBOUNDED) {
    Console.WriteLine("The model cannot be solved " +
        "because it is infeasible or unbounded");
    return;
}
if (status != GRB.Status.OPTIMAL) {
    Console.WriteLine("Optimization was stopped with status {0}" , status);
    return;
}

// Print best selected set
Console.WriteLine("Selected elements in best solution:");
Console.Write("\t");
for (e = 0; e < groundSetSize; e++) {
    if (Elem[e].X < .9) continue;
    Console.Write("El{0} ", e);
}
Console.WriteLine();

// Print number of solutions stored
nSolutions = model.SolCount;
Console.WriteLine("Number of solutions found: {0}" , nSolutions);

// Print objective values of solutions
if (nSolutions > 10) nSolutions = 10;
Console.WriteLine("Objective values for first {0} solutions:" , nSolutions);
for (i = 0; i < nSubsets; i++) {
model.Parameters.ObjNumber = i;

Console.Write("\tSet " + i);
for (e = 0; e < nSolutions; e++) {
    model.Parameters.SolutionNumber = e;
    Console.Write("{0,8} " , model.ObjNVals);
}
Console.WriteLine();
model.Dispose();
env.Dispose();
} catch (GRBException e) {
    Console.WriteLine("Error code = {0}" , e);
    Console.WriteLine(e.Message);
}
}

multiscenario_cs.cs

// Copyright 2020, Gurobi Optimization, LLC

// Facility location: a company currently ships its product from 5 plants
// to 4 warehouses. It is considering closing some plants to reduce
// costs. What plant(s) should the company close, in order to minimize
// transportation and fixed costs?
//
// Since the plant fixed costs and the warehouse demands are uncertain, a
// scenario approach is chosen.
//
// Note that this example is similar to the facility_cs.cs example. Here we
// added scenarios in order to illustrate the multi-scenario feature.
//
// Based on an example from Frontline Systems:
// http://www.solver.com/disfacility.htm
// Used with permission.

using System;
using Gurobi;

class multiscenario_cs
{
    static void Main()
    {
        try {

            // Warehouse demand in thousands of units
            double[] Demand = new double[] { 15, 18, 14, 20 };  

            // Plant capacity in thousands of units
            double[] Capacity = new double[] { 20, 22, 17, 19, 18 };  

            // Fixed costs for each plant
            double[] FixedCosts =
                new double[] { 12000, 15000, 17000, 13000, 16000 };  

        } catch (GRBException e) {
            Console.WriteLine("Error code = {0}" , e);
            Console.WriteLine(e.Message);
        }
    }
}
// Transportation costs per thousand units
double[,] TransCosts =
    new double[,] { { 4000, 2000, 3000, 2500, 4500 },
                    { 2500, 2600, 3400, 3000, 4000 },
                    { 1200, 1800, 2600, 4100, 3000 },
                    { 2200, 2600, 3100, 3700, 3200 } };

// Number of plants and warehouses
int nPlants = Capacity.Length;
int nWarehouses = Demand.Length;

double maxFixed = -GRB.INFINITY;
double minFixed = GRB.INFINITY;
for (int p = 0; p < nPlants; ++p) {
    if (FixedCosts[p] > maxFixed)
        maxFixed = FixedCosts[p];
    if (FixedCosts[p] < minFixed)
        minFixed = FixedCosts[p];
}

// Model
GRBEnv env = new GRBEnv();
GRBModel model = new GRBModel(env);
model.ModelName = "multiscenario";

// Plant open decision variables: open[p] == 1 if plant p is open.
GRBVar[] open = new GRBVar[nPlants];
for (int p = 0; p < nPlants; ++p) {
    open[p] = model.AddVar(0, 1, FixedCosts[p], GRB.BINARY, "Open" + p);
}

// Transportation decision variables: how much to transport from
// a plant p to a warehouse w
GRBVar[,] transport = new GRBVar[nWarehouses,nPlants];
for (int w = 0; w < nWarehouses; ++w) {
    for (int p = 0; p < nPlants; ++p) {
        transport[w,p] = model.AddVar(0, GRB.INFINITY, TransCosts[w,p],
                                       GRB.CONTINUOUS, "Trans" + p + "." + w);
    }
}

// The objective is to minimize the total fixed and variable costs
model.ModelSense = GRB.MINIMIZE;

// Production constraints
// Note that the right-hand limit sets the production to zero if
// the plant is closed
for (int p = 0; p < nPlants; ++p) {
    GRBLinExpr ptot = 0.0;
    for (int w = 0; w < nWarehouses; ++w)
        ptot.AddTerm(1.0, transport[w,p]);
    model.AddConstr(ptot <= Capacity[p] * open[p], "Capacity" + p);
}
// Demand constraints
GRBConstr[] demandConstr = new GRBConstr[nWarehouses];
for (int w = 0; w < nWarehouses; ++w) {
    GRBLinExpr dtot = 0.0;
    for (int p = 0; p < nPlants; ++p)
        dtot.AddTerm(1.0, transport[w,p]);
    demandConstr[w] = model.AddConstr(dtot == Demand[w], "Demand" + w);
}

// We constructed the base model, now we add 7 scenarios
//
// Scenario 0: Represents the base model, hence, no manipulations.
// Scenario 1: Manipulate the warehouses demands slightly (constraint right
// hand sides).
// Scenario 2: Double the warehouses demands (constraint right hand sides).
// Scenario 3: Manipulate the plant fixed costs (objective coefficients).
// Scenario 4: Manipulate the warehouses demands and fixed costs.
// Scenario 5: Force the plant with the largest fixed cost to stay open
// (variable bounds).
// Scenario 6: Force the plant with the smallest fixed cost to be closed
// (variable bounds).

model.NumScenarios = 7;

// Scenario 0: Base model, hence, nothing to do except giving the
// scenario a name
model.Parameters.SenarioNumber = 0;
model.ScenNName = "Base model";

// Scenario 1: Increase the warehouse demands by 10%
model.Parameters.SenarioNumber = 1;
model.ScenNName = "Increased warehouse demands";
for (int w = 0; w < nWarehouses; w++) {
    demandConstr[w].ScenNRHS = Demand[w] * 1.1;
}

// Scenario 2: Double the warehouse demands
model.Parameters.SenarioNumber = 2;
model.ScenNName = "Double the warehouse demands";
for (int w = 0; w < nWarehouses; w++) {
    demandConstr[w].ScenNRHS = Demand[w] * 2.0;
}

// Scenario 3: Decrease the plant fixed costs by 5%
model.Parameters.SenarioNumber = 3;
model.ScenNName = "Decreased plant fixed costs";
for (int p = 0; p < nPlants; p++) {
    open[p].ScenNObj = FixedCosts[p] * 0.95;
}

// Scenario 4: Combine scenario 1 and scenario 3 */
model.Parameters.SenarioNumber = 4;
model.ScenNName = "Increased warehouse demands and decreased plant fixed costs";
for (int w = 0; w < nWarehouses; w++) {
    demandConstr[w].ScenNRHS = Demand[w] * 1.1;
}
for (int p = 0; p < nPlants; p++) {
    open[p].ScenNObj = FixedCosts[p] * 0.95;
}

// Scenario 5: Force the plant with the largest fixed cost to stay
// open
model.Parameters.ScenarioNumber = 5;
model.ScenNName = "Force plant with largest fixed cost to stay open";

    for (int p = 0; p < nPlants; p++) {
        if (FixedCosts[p] == maxFixed) {
            open[p].ScenNLB = 1.0;
            break;
        }
    }

// Scenario 6: Force the plant with the smallest fixed cost to be
// closed
model.Parameters.ScenarioNumber = 6;
model.ScenNName = "Force plant with smallest fixed cost to be closed";

    for (int p = 0; p < nPlants; p++) {
        if (FixedCosts[p] == minFixed) {
            open[p].ScenNUB = 0.0;
            break;
        }
    }

// Guess at the starting point: close the plant with the highest
// fixed costs; open all others

// First, open all plants
for (int p = 0; p < nPlants; ++p) {
    open[p].Start = 1.0;
}

// Now close the plant with the highest fixed cost
Console.WriteLine("Initial guess:");
for (int p = 0; p < nPlants; ++p) {
    if (FixedCosts[p] == maxFixed) {
        open[p].Start = 0.0;
        Console.WriteLine("Closing plant " + p + "\n");
        break;
    }
}

// Use barrier to solve root relaxation
model.Parameters.Method = GRB.METHOD_BARRIER;

// Solve multi-scenario model
model.Optimize();
int nScenarios = model.NumScenarios;

for (int s = 0; s < nScenarios; s++) {
    int modelSense = GRB.MINIMIZE;

    // Set the scenario number to query the information for this scenario
    model.Parameters.ScenarioNumber = s;

    // collect result for the scenario
    double scenNObjBound = model.ScenNObjBound;
    double scenNObjVal = model.ScenNObjVal;

    Console.WriteLine("\n\n------ Scenario " + s
    + " (" + model.ScenNName + ")\n");

    // Check if we found a feasible solution for this scenario
    if (scenNObjVal >= modelSense * GRB.INFINITY)
        if (scenNObjBound >= modelSense * GRB.INFINITY)
            // Scenario was proven to be infeasible
            Console.WriteLine("\nINFEASIBLE\n");
        else
            // We did not find any feasible solution - should not happen in
            // this case, because we did not set any limit (like a time
            // limit) on the optimization process
            Console.WriteLine("\nNO SOLUTION\n");
    else {
        Console.WriteLine("\nTOTAL COSTS: " + scenNObjVal);
        Console.WriteLine("SOLUTION:\n");
        for (int p = 0; p < nPlants; p++) {
            double scenNX = open[p].ScenNX;
            if (scenNX > 0.5) {
                Console.WriteLine("Plant " + p + " open");
                for (int w = 0; w < nWarehouses; w++) {
                    scenNX = transport[w,p].ScenNX;
                    if (scenNX > 0.0001)
                        Console.WriteLine(" Transport " + scenNX
                        + " units to warehouse " + w);
                }
            } else
                Console.WriteLine("Plant " + p + " closed!\n");
        }
    }

    // Print a summary table: for each scenario we add a single summary
    // line
    Console.WriteLine("\n\nSummary: Closed plants depending on scenario\n");
    Console.WriteLine("{0,8} | {1,17} {2,13} | {3,6} Name | {4,6} Costs\n");
    Console.Write("{0,8} | {1,17} {2,13} | {3,6} Name | {4,6} Costs\n");
}
for (int s = 0; s < nScenarios; s++) {
    int modelSense = GRB.MINIMIZE;
    // Set the scenario number to query the information for this scenario
    model.Parameters.ScenarioNumber = s;
    // Collect result for the scenario
    double scenNObjBound = model.ScenNObjBound;
    double scenNObjVal = model.ScenNObjVal;
    Console.Write("{0,-8} |", s);
    // Check if we found a feasible solution for this scenario
    if (scenNObjVal >= modelSense * GRB.INFINITY) {
        if (scenNObjBound >= modelSense * GRB.INFINITY)
            // Scenario was proven to be infeasible
            Console.WriteLine("{0,-30}| {1,-6} " + model.ScenNName,
                             "infeasible", ";
        else
            // We did not find any feasible solution - should not happen in
            // this case, because we did not set any limit (like a time
            // limit) on the optimization process
            Console.WriteLine("{0,-30}| {1,-6} " + model.ScenNName,
                             "no solution found", "");
    } else {
        for (int p = 0; p < nPlants; p++) {
            double scenNX = open[p].ScenNX;
            if (scenNX > 0.5)
                Console.Write("{0,6}" , "");
            else
                Console.Write("{0,6}" , "x");
        }
        Console.WriteLine("| {0,6}" + model.ScenNName, scenNObjVal);
    }
    // Dispose of model and env
    model.Dispose();
    env.Dispose();
}
catch (GRBException e) {
    Console.WriteLine("Error code: "+ e.ErrorCode + ". " + e.Message);
}

params_cs.cs

/* Copyright 2020, Gurobi Optimization, LLC */

/* Use parameters that are associated with a model.

A MIP is solved for a few seconds with different sets of parameters.
The one with the smallest MIP gap is selected, and the optimization
is resumed until the optimal solution is found. */
using System;
using Gurobi;

class params_cs {
    static void Main(string[] args) {
        if (args.Length < 1) {
            Console.Out.WriteLine("Usage: params_cs filename");
            return;
        }

        try {
            // Read model and verify that it is a MIP
            GRBEnv env = new GRBEnv();
            GRBModel m = new GRBModel(env, args[0]);
            if (m.IsMIP == 0) {
                Console.WriteLine("The model is not an integer program");
                Environment.Exit(1);
            }

            // Set a 2 second time limit
            m.Parameters.TimeLimit = 2.0;

            // Now solve the model with different values of MIPFocus
            GRBModel bestModel = new GRBModel(m);
            bestModel.Optimize();
            for (int i = 1; i <= 3; ++i) {
                m.Reset();
                m.Parameters.MIPFocus = i;
                m.Optimize();
                if (bestModel.MIPGap > m.MIPGap) {
                    GRBModel swap = bestModel;
                    bestModel = m;
                    m = swap;
                }
            }

            // Finally, delete the extra model, reset the time limit and
            // continue to solve the best model to optimality
            m.Dispose();
            bestModel.Parameters.TimeLimit = GRB.INFINITY;
            bestModel.Optimize();
            Console.WriteLine("Solved with MIPFocus: " +
                bestModel.Parameters.MIPFocus);

            // Clean up bestModel and environment
            bestModel.Dispose();
            env.Dispose();
        } catch (GRBException e) {
            Console.WriteLine("Error code: " + e.ErrorCode + ". " +
                e.Message);
        }
    }
}
/* Copyright 2020, Gurobi Optimization, LLC */

/* This example considers the following separable, convex problem:

minimize \( f(x) - y + g(z) \)
subject to \( x + 2y + 3z \leq 4 \)
\( x + y \geq 1 \)
\( x, y, z \leq 1 \)

where \( f(u) = \exp(-u) \) and \( g(u) = 2u^2 - 4u \), for all real \( u \). It formulates and solves a simpler LP model by approximating \( f \) and \( g \) with piecewise-linear functions. Then it transforms the model into a MIP by negating the approximation for \( f \), which corresponds to a non-convex piecewise-linear function, and solves it again. */

using System;
using Gurobi;

class piecewise_cs{

    private static double f(double u) { return Math.Exp(-u); }
    private static double g(double u) { return 2 * u * u - 4 * u; }

    static void Main()
    {
        try {
            // Create environment
            GRBEnv env = new GRBEnv();
            // Create a new model
            GRBModel model = new GRBModel(env);
            // Create variables
            double lb = 0.0, ub = 1.0;
            GRBVar x = model.AddVar(lb, ub, 0.0, GRB.CONTINUOUS, "x");
            GRBVar y = model.AddVar(lb, ub, 0.0, GRB.CONTINUOUS, "y");
            GRBVar z = model.AddVar(lb, ub, 0.0, GRB.CONTINUOUS, "z");

            // Set objective for y
            model.SetObjective(-y);

            // Add piecewise-linear objective functions for x and z
            int npts = 101;
        }
    }
}
double[] ptu = new double[npts];
double[] ptf = new double[npts];
double[] ptg = new double[npts];

for (int i = 0; i < npts; i++) {
    ptu[i] = lb + (ub - lb) * i / (npts - 1);
    ptf[i] = f(ptu[i]);
    ptg[i] = g(ptu[i]);
}

model.SetPWLObj(x, ptu, ptf);
model.SetPWLObj(z, ptu, ptg);

// Add constraint: x + 2 y + 3 z <= 4
model.AddConstr(x + 2 * y + 3 * z <= 4.0, "c0");

// Add constraint: x + y >= 1
model.AddConstr(x + y >= 1.0, "c1");

// Optimize model as an LP
model.Optimize();

Console.WriteLine("IsMIP: " + model.IsMIP);

Console.WriteLine(x.VarName + " " + x.X);
Console.WriteLine(y.VarName + " " + y.X);
Console.WriteLine(z.VarName + " " + z.X);

Console.WriteLine("Obj: " + model.ObjVal);

Console.WriteLine();

// Negate piecewise-linear objective function for x
for (int i = 0; i < npts; i++) {
    ptf[i] = -ptf[i];
}

model.SetPWLObj(x, ptu, ptf);

// Optimize model as a MIP
model.Optimize();

Console.WriteLine("IsMIP: " + model.IsMIP);

Console.WriteLine(x.VarName + " " + x.X);
Console.WriteLine(y.VarName + " " + y.X);
Console.WriteLine(z.VarName + " " + z.X);

Console.WriteLine("Obj: " + model.ObjVal);

// Dispose of model and environment
model.Dispose();
env.Dispose();

} catch (GRBException e) {
    Console.WriteLine("Error code: " + e.ErrorCode + ". " + e.Message);
}

}

poolsearch_cs.cs

/* Copyright 2020, Gurobi Optimization, LLC */

/* We find alternative epsilon-optimal solutions to a given knapsack problem by using PoolSearchMode */

using System;
using Gurobi;

class poolsearch_cs {

    static void Main() {
        try{
            // Sample data
            int groundSetSize = 10;
            double[] objCoef = new double[] {32, 32, 15, 15, 6, 6, 1, 1, 1, 1};
            double[] knapsackCoef = new double[] {16, 16, 8, 8, 4, 4, 2, 2, 1, 1};
            double Budget = 33;
            int e, status, nSolutions;

            // Create environment
            GRBEnv env = new GRBEnv("poolsearch_cs.log");

            // Create initial model
            GRBModel model = new GRBModel(env);
            model.ModelName = "poolsearch_cs";

            // Initialize decision variables for ground set:
            // x[e] == 1 if element e is chosen
            GRBVar[] Elem = model.AddVars(groundSetSize, GRB.BINARY);
            model.Set(GRB.DoubleAttr.Obj, Elem, objCoef, 0, groundSetSize);

            for (e = 0; e < groundSetSize; e++) {
                Elem[e].VarName = string.Format("El\{0\}", e);
            }

            // Constraint: limit total number of elements to be picked to be at most Budget
            GRBLinExpr lhs = new GRBLinExpr();
            for (e = 0; e < groundSetSize; e++) {
                lhs.AddTerm(knapsackCoef[e], Elem[e]);
            }
            model.AddConstr(lhs, GRB.LESS_EQUAL, Budget, "Budget");

        } catch (GRBException e) {
            Console.WriteLine("Error code: " + e.ErrorCode + ". " + e.Message);
        }
    }

}
// set global sense for ALL objectives
model.ModelSense = GRB.MAXIMIZE;

// Limit how many solutions to collect
model.Parameters.PoolSolutions = 1024;

// Limit the search space by setting a gap for the worst possible solution that will be accepted
model.Parameters.PoolGap = 0.10;

// do a systematic search for the k-best solutions
model.Parameters.PoolSearchMode = 2;

// save problem
model.Write("poolsearch_cs.lp");

// Optimize
model.Optimize();

// Status checking
status = model.Status;
if (status == GRB.Status.INF_OR_UNBD ||
    status == GRB.Status.INFEASIBLE ||
    status == GRB.Status.UNBOUNDED) {
    Console.WriteLine("The model cannot be solved " +
                      "because it is infeasible or unbounded");
    return;
}
if (status != GRB.Status.OPTIMAL) {
    Console.WriteLine("Optimization was stopped with status {0}", status);
    return;
}

// Print best selected set
Console.WriteLine("Selected elements in best solution:");
for (e = 0; e < groundSetSize; e++) {
    if (Elem[e].X < .9) continue;
    Console.Write("El{0} ", e);
}
Console.WriteLine();

// Print number of solutions stored
nSolutions = model.SolCount;
Console.WriteLine("Number of solutions found: {0}", nSolutions);

// Print objective values of solutions
for (e = 0; e < nSolutions; e++) {
    model.Parameters.SolutionNumber = e;
    Console.WriteLine("{0} ", model.PoolObjVal);
    if (e%15 == 14) Console.WriteLine();
}
Console.WriteLine();

// Print fourth best set if available
if (nSolutions >= 4) {

model.Parameters.SolutionNumber = 3;

Console.WriteLine("Selected elements in fourth best solution:");
Console.Write("\t");
for (e = 0; e < groundSetSize; e++) {
    if (Elem[e].Xn < .9) continue;
    Console.Write("El{0} ", e);
}
Console.WriteLine();
}

model.Dispose();
env.Dispose();
} catch (GRBException e) {
    Console.WriteLine("Error code: {0}. {1}", e.ErrorCode, e.Message);
}
}

qcp_cs.cs

/* Copyright 2020, Gurobi Optimization, LLC */

/* This example formulates and solves the following simple QCP model:

maximize       x
subject to     x + y + z = 1
               x^2 + y^2 <= z^2 (second-order cone)
               x^2 <= yz         (rotated second-order cone)
               x, y, z non-negative
*/

using System;
using Gurobi;

class qcp_cs
{
    static void Main()
    {
        try {
            GRBEnv env = new GRBEnv("qcp.log");
            GRBModel model = new GRBModel(env);

            // Create variables
            GRBVar x = model.AddVar(0.0, GRB.INFINITY, 0.0, GRB.CONTINUOUS, "x");
            GRBVar y = model.AddVar(0.0, GRB.INFINITY, 0.0, GRB.CONTINUOUS, "y");
            GRBVar z = model.AddVar(0.0, GRB.INFINITY, 0.0, GRB.CONTINUOUS, "z");

            // Set objective
            GRBLinExpr obj = x;
            model.SetObjective(obj, GRB.MAXIMIZE);

            // Add linear constraint: x + y + z = 1

        }
    }
}
model.AddConstr(x + y + z == 1.0, "c0");

// Add second-order cone: x^2 + y^2 <= z^2
model.AddQConstr(x*x + y*y <= z*z, "qc0");

// Add rotated cone: x^2 <= yz
model.AddQConstr(x*x <= y*z, "qc1");

// Optimize model
model.Optimize();

Console.WriteLine(x.VarName + " " + x.X);
Console.WriteLine(y.VarName + " " + y.X);
Console.WriteLine(z.VarName + " " + z.X);

Console.WriteLine("Obj: " + model.ObjVal + " " + obj.Value);

// Dispose of model and env
model.Dispose();
env.Dispose();

} catch (GRBException e) {
    Console.WriteLine("Error code: " + e.ErrorCode + ". " + e.Message);
}

} // end Main

qp_cs.cs

/* Copyright 2020, Gurobi Optimization, LLC */

/* This example formulates and solves the following simple QP model:

    minimize x^2 + x*y + y^2 + y*z + z^2 + 2 x
    subject to x + 2 y + 3 z >= 4
            x + y >= 1
            x, y, z non-negative

    It solves it once as a continuous model, and once as an integer model. */

using System;
using Gurobi;

class qp_cs
{
    static void Main()
    {
        try {
            GRBEnv env = new GRBEnv("qp.log");
            GRBModel model = new GRBModel(env);

            model.AddConstr(x + y + z == 1.0, "c0");

            // Add second-order cone: x^2 + y^2 <= z^2
            model.AddQConstr(x*x + y*y <= z*z, "qc0");

            // Add rotated cone: x^2 <= yz
            model.AddQConstr(x*x <= y*z, "qc1");

            // Optimize model
            model.Optimize();

            Console.WriteLine(x.VarName + " " + x.X);
            Console.WriteLine(y.VarName + " " + y.X);
            Console.WriteLine(z.VarName + " " + z.X);

            Console.WriteLine("Obj: " + model.ObjVal + " " + obj.Value);

            // Dispose of model and env
            model.Dispose();
            env.Dispose();
        }
    }
}
// Create variables
GRBVar x = model.AddVar(0.0, 1.0, 0.0, GRB.CONTINUOUS, "x");
GRBVar y = model.AddVar(0.0, 1.0, 0.0, GRB.CONTINUOUS, "y");
GRBVar z = model.AddVar(0.0, 1.0, 0.0, GRB.CONTINUOUS, "z");

// Set objective
GRBQuadExpr obj = x*x + x*y + y*y + y*z + z*z + 2*x;
model.SetObjective(obj);

// Add constraint: x + 2 y + 3 z >= 4
model.AddConstr(x + 2 * y + 3 * z >= 4.0, "c0");

// Add constraint: x + y >= 1
model.AddConstr(x + y >= 1.0, "c1");

// Optimize model
model.Optimize();

Console.WriteLine(x.VarName + " " + x.X);
Console.WriteLine(y.VarName + " " + y.X);
Console.WriteLine(z.VarName + " " + z.X);

Console.WriteLine("Obj: " + model.ObjVal + " " + obj.Value);

// Change variable types to integer
x.VType = GRB.INTEGER;
y.VType = GRB.INTEGER;
z.VType = GRB.INTEGER;

// Optimize model
model.Optimize();

Console.WriteLine(x.VarName + " " + x.X);
Console.WriteLine(y.VarName + " " + y.X);
Console.WriteLine(z.VarName + " " + z.X);

Console.WriteLine("Obj: " + model.ObjVal + " " + obj.Value);

// Dispose of model and env
model.Dispose();
e Dispose();

} catch (GRBException e) {
    Console.WriteLine("Error code: " + e.ErrorCode + ". " + e.Message);
}
}
sensitivity_cs.cs

// Copyright 2020, Gurobi Optimization, LLC
// A simple sensitivity analysis example which reads a MIP model from a
// file and solves it. Then uses the scenario feature to analyze the impact
// w.r.t. the objective function of each binary variable if it is set to
// 1-X, where X is its value in the optimal solution.
// Usage:
// sensitivity_cs <model filename>

using System;
using Gurobi;

class sensitivity_cs
{
    // Maximum number of scenarios to be considered
    public const int MAXSCENARIOS = 100;

    static void Main(string[] args)
    {
        const int maxscenarios = sensitivity_cs.MAXSCENARIOS;

        if (args.Length < 1) {
            Console.Out.WriteLine("Usage: sensitivity_cs filename");
            return;
        }

        try {

            // Create environment
            GRBEnv env = new GRBEnv();

            // Read model
            GRBModel model = new GRBModel(env, args[0]);

            int scenarios;

            if (model.IsMIP == 0) {
                Console.WriteLine("Model is not a MIP");
                return;
            }

            // Solve model
            model.optimize();

            if (model.Status != GRB.Status.OPTIMAL) {
                Console.WriteLine("Optimization ended with status " + model.Status);
                return;
            }

            // Store the optimal solution
            double origObjVal = model.ObjVal;
        }
    }
}
GRBVar[] vars = model.GetVars();
double[] origX = model.Get(GRB.DoubleAttr.X, vars);

scenarios = 0;

// Count number of unfixed, binary variables in model. For each we
// create a scenario.
for (int i = 0; i < vars.Length; i++) {
    GRBVar v = vars[i];
    char vType = v.VType;

    if (v.LB == 0.0 && v.UB == 1.0 &&
        (vType == GRB.BINARY || vType == GRB.INTEGER)) {
        scenarios++;

        if (scenarios >= maxscenarios)
            break;
    }
}

Console.WriteLine("### construct multi-scenario model with "+ scenarios + " scenarios");

// Set the number of scenarios in the model */
model.NumScenarios = scenarios;

scenarios = 0;

// Create a (single) scenario model by iterating through unfixed
// binary variables in the model and create for each of these
// variables a scenario by fixing the variable to 1-X, where X is its
// value in the computed optimal solution
for (int i = 0; i < vars.Length; i++) {
    GRBVar v = vars[i];
    char vType = v.VType;

    if (v.LB == 0.0 && v.UB == 1.0 &&
        (vType == GRB.BINARY || vType == GRB.INTEGER) &&
        scenarios < maxscenarios) {
        // Set ScenarioNumber parameter to select the corresponding
        // scenario for adjustments
        model.Parameters.ScenarioNumber = scenarios;

        // Set variable to 1-X, where X is its value in the optimal solution */
        if (origX[i] < 0.5)
            v.ScenNLB = 1.0;
        else
            v.ScenNUB = 0.0;

        scenarios++;
    } else {
        // Add MIP start for all other variables using the optimal solution
        // of the base model
        v.Start = origX[i];
    }
}
Solve multi-scenario model

model.Optimize();

In case we solved the scenario model to optimality capture the sensitivity information

if (model.Status == GRB.Status.OPTIMAL) {

    // get the model sense (minimization or maximization)
    int modelSense = model.ModelSense;

    scenarios = 0;

    for (int i = 0; i < vars.Length; i++) {
        GRBVar v = vars[i];
        char vType = v.VType;

        if (v.LB == 0.0 && v.UB == 1.0 &&
            (vType == GRB.BINARY || vType == GRB.INTEGER)) {

            // Set scenario parameter to collect the objective value of the corresponding scenario
            model.Parameters.ScenarioNumber = scenarios;

            double scenarioObjVal = model.ScenNObjVal;
            double scenarioObjBound = model.ScenNObjBound;

            Console.WriteLine("Objective sensitivity for variable " + v.VarName + " is ");

            // Check if we found a feasible solution for this scenario
            if (scenarioObjVal >= modelSense * GRB.INFINITY) {
                // Check if the scenario is infeasible
                if (scenarioObjBound >= modelSense * GRB.INFINITY)
                    Console.WriteLine("infeasible");
                else
                    Console.WriteLine("unknown (no solution available)");
            } else {
                // Scenario is feasible and a solution is available
                Console.WriteLine(modelSense * (scenarioObjVal - origObjVal));
            }

            scenarios++;

            if (scenarios >= maxscenarios)
                break;
        }
    }

    // Dispose of model and environment
    model.Dispose();
    env.Dispose();
}

} catch (GRBException e) {
sos_cs.cs

/* Copyright 2020, Gurobi Optimization, LLC */

/* This example creates a very simple Special Ordered Set (SOS) model.
    The model consists of 3 continuous variables, no linear constraints,
    and a pair of SOS constraints of type 1. */

using System;
using Gurobi;

class sos_cs
{
    static void Main()
    {
        try
        {
            GRBEnv env = new GRBEnv();
            GRBModel model = new GRBModel(env);

            // Add first SOS1: x0=0 or x1=0
            GRBVar[] x = model.AddVars(null, ub, obj, null, names);
            double[] sosv1 = {x[0], x[1]};
            double[] soswt1 = {1, 2};
            model.AddSOS(sosv1, soswt1, GRB.SOS_TYPE1);

            // Add second SOS1: x0=0 or x2=0
            GRBVar[] sosv2 = {x[0], x[2]};
            double[] soswt2 = {1, 2};
            model.AddSOS(sosv2, soswt2, GRB.SOS_TYPE1);

            // Optimize model
            model.Optimize();

            for (int i = 0; i < 3; i++)
            {
                Console.WriteLine(x[i].VarName + " " + x[i].X);
            }
        }
    }
}
```csharp
// Dispose of model and env
model.Dispose();
env.Dispose();

} catch (GRBException e) {
    Console.WriteLine("Error code: " + e.ErrorCode + ". " + e.Message);
}
}

sudoku_cs.cs

/* Copyright 2020, Gurobi Optimization, LLC */

/*
Sudoku example.

The Sudoku board is a 9x9 grid, which is further divided into a 3x3 grid of 3x3 grids. Each cell in the grid must take a value from 0 to 9. No two grid cells in the same row, column, or 3x3 subgrid may take the same value.

In the MIP formulation, binary variables x[i,j,v] indicate whether cell <i,j> takes value 'v'. The constraints are as follows:
1. Each cell must take exactly one value (sum_v x[i,j,v] = 1)
2. Each value is used exactly once per row (sum_i x[i,j,v] = 1)
3. Each value is used exactly once per column (sum_j x[i,j,v] = 1)
4. Each value is used exactly once per 3x3 subgrid (sum_grid x[i,j,v] = 1)

Input datasets for this example can be found in examples/data/sudoku*.
*/

using System;
using System.IO;
using Gurobi;

class sudoku_cs {
    static void Main(string[] args) {
        int n = 9;
        int s = 3;

        if (args.Length < 1) {
            Console.Out.WriteLine("Usage: sudoku_cs filename");
            return;
        }

        try {
            GRBEnv env = new GRBEnv();
            GRBModel model = new GRBModel(env);

            // Create 3-D array of model variables
            GRBVar[,] vars = new GRBVar[n,n,n];
        }
    }
}
```
for (int i = 0; i < n; i++) {
    for (int j = 0; j < n; j++) {
        for (int v = 0; v < n; v++) {
            string st = "G_" + i.ToString() + "_" + j.ToString()
                        + "_" + v.ToString();
            vars[i,j,v] = model.AddVar(0.0, 1.0, 0.0, GRB.BINARY, st);
        }
    }
}

// Add constraints
GRBLinExpr expr;

// Each cell must take one value
for (int i = 0; i < n; i++) {
    for (int j = 0; j < n; j++) {
        expr = 0.0;
        for (int v = 0; v < n; v++)
            expr.AddTerm(1.0, vars[i,j,v]);
        string st = "V_" + i.ToString() + "_" + j.ToString();
        model.AddConstr(expr == 1.0, st);
    }
}

// Each value appears once per row
for (int i = 0; i < n; i++) {
    for (int v = 0; v < n; v++) {
        expr = 0.0;
        for (int j = 0; j < n; j++)
            expr.AddTerm(1.0, vars[i,j,v]);
        string st = "R_" + i.ToString() + "_" + v.ToString();
        model.AddConstr(expr == 1.0, st);
    }
}

// Each value appears once per column
for (int j = 0; j < n; j++) {
    for (int v = 0; v < n; v++) {
        expr = 0.0;
        for (int i = 0; i < n; i++)
            expr.AddTerm(1.0, vars[i,j,v]);
        string st = "C_" + j.ToString() + "_" + v.ToString();
        model.AddConstr(expr == 1.0, st);
    }
}

// Each value appears once per sub-grid
for (int v = 0; v < n; v++) {
    for (int i0 = 0; i0 < s; i0++) {
        for (int j0 = 0; j0 < s; j0++) {
            expr = 0.0;
        }
    }
}
for (int i1 = 0; i1 < s; i1++) {
    for (int j1 = 0; j1 < s; j1++) {
        expr.AddTerm(1.0, vars[i0*s+i1,j0*s+j1,v]);
    }
}

string st = "Sub_" + v.ToString() + "," + i0.ToString() + "," + j0.ToString();
model.AddConstr(expr == 1.0, st);
}
}

// Fix variables associated with pre-specified cells
StreamReader sr = File.OpenText(args[0]);
for (int i = 0; i < n; i++) {
    string input = sr.ReadLine();
    for (int j = 0; j < n; j++) {
        int val = (int) input[j] - 48 - 1; // 0-based
        if (val >= 0)
            vars[i,j,val].LB = 1.0;
    }
}

// Optimize model
model.Optimize();

// Write model to file
model.Write("sudoku.lp");

double[,] x = model.Get(GRB.DoubleAttr.X, vars);
Console.WriteLine();
for (int i = 0; i < n; i++) {
    for (int j = 0; j < n; j++) {
        for (int v = 0; v < n; v++) {
            if (x[i,j,v] > 0.5) {
                Console.Write(v+1);
            }
        }
    }
    Console.WriteLine();
}

// Dispose of model and env
model.Dispose();
env.Dispose();
}
catch (GRBException e) {
    Console.WriteLine("Error code: " + e.ErrorCode + ", " + e.Message);
}
}
tsp_cs.cs

/* Copyright 2020, Gurobi Optimization, LLC */

// Solve a traveling salesman problem on a randomly generated set of
// points using lazy constraints. The base MIP model only includes
// 'degree-2' constraints, requiring each node to have exactly
// two incident edges. Solutions to this model may contain subtours -
// tours that don't visit every node. The lazy constraint callback
// adds new constraints to cut them off.

using System;
using Gurobi;

class tsp_cs : GRBCallback {
    private GRBVar[,] vars;

    public tsp_cs(GRBVar[,] xvars) {
        vars = xvars;
    }

    // Subtour elimination callback. Whenever a feasible solution is found,
    // find the smallest subtour, and add a subtour elimination
    // constraint if the tour doesn’t visit every node.

    protected override void Callback() {
        try {
            if (where == GRB.Callback.MIPSOL) {
                // Found an integer feasible solution - does it visit every node?

                int n = vars.GetLength(0);
                int[] tour = findsubtour(GetSolution(vars));

                if (tour.Length < n) {
                    // Add subtour elimination constraint
                    GRBLinExpr expr = 0;
                    for (int i = 0; i < tour.Length; i++)
                        for (int j = i + 1; j < tour.Length; j++)
                            expr.AddTerm(1.0, vars[tour[i], tour[j]]);
                    AddLazy(expr <= tour.Length - 1);
                }
            } catch (GRBException e) {
                Console.WriteLine("Error code: " + e.ErrorCode + ". " + e.Message);
                Console.WriteLine(e.StackTrace);
            }
        }
    }

    // Given an integer-feasible solution 'sol', return the smallest
    // sub-tour (as a list of node indices).

    protected static int[] findsubtour(double[,] sol) {
        int n = sol.GetLength(0);
        bool[] seen = new bool[n];
        int[] tour = new int[n];
        }
```java
int bestind, bestlen;
int i, node, len, start;

for (i = 0; i < n; i++)
    seen[i] = false;

start = 0;
bestlen = n+1;
bestind = -1;
node = 0;
while (start < n) {
    for (node = 0; node < n; node++)
        if (!seen[node])
            break;
    if (node == n)
        break;
    for (len = 0; len < n; len++) {
        tour[start+len] = node;
        seen[node] = true;
        for (i = 0; i < n; i++)
            if (sol[node, i] > 0.5 && !seen[i]) {
                node = i;
                break;
            }
    }
    if (i == n) {
        len++;
        if (len < bestlen) {
            bestlen = len;
            bestind = start;
        }
        start += len;
        break;
    }
}

for (i = 0; i < bestlen; i++)
    tour[i] = tour[bestind+i];
System.Array.Resize(ref tour, bestlen);
return tour;
}

// Euclidean distance between points 'i' and 'j'
protected static double distance(double[] x, double[] y, int i, int j) {
    double dx = x[i]-x[j];
    double dy = y[i]-y[j];
    return Math.Sqrt(dx*dx+dy*dy);
}

public static void Main(String[] args) {
```
if (args.Length < 1) {
    Console.WriteLine("Usage: tsp_cs nnodes");
    return;
}

int n = Convert.ToInt32(args[0]);

try {
    GRBEnv env = new GRBEnv();
    GRBModel model = new GRBModel(env);

    // Must set LazyConstraints parameter when using lazy constraints
    model.Parameters.LazyConstraints = 1;

double[] x = new double[n];
double[] y = new double[n];

    Random r = new Random();
    for (int i = 0; i < n; i++) {
        x[i] = r.NextDouble();
        y[i] = r.NextDouble();
    }

    // Create variables
    GRBVar[,] vars = new GRBVar[n, n];
    for (int i = 0; i < n; i++) {
        for (int j = 0; j <= i; j++) {
            vars[i, j] = model.AddVar(0.0, 1.0, distance(x, y, i, j),
                                      GRB.BINARY, "x" + i + "_" + j);
            vars[j, i] = vars[i, j];
        }
    }

    // Degree-2 constraints
    for (int i = 0; i < n; i++) {
        GRBLinExpr expr = 0;
        for (int j = 0; j < n; j++)
            expr.AddTerm(1.0, vars[i, j]);
        model.AddConstr(expr == 2.0, "deg2_" + i);
    }

    // Forbid edge from node back to itself
    for (int i = 0; i < n; i++)
        vars[i, i].UB = 0.0;

    model.SetCallback(new tsp_cs(vars));
    model.Optimize();

    if (model.SolCount > 0) {
        int[] tour = findsubtour(model.Get(GRB.DoubleAttr.X, vars));
    }
}
Console.Write("Tour: ");
for (int i = 0; i < tour.Length; i++)
    Console.Write(tour[i] + " ");
Console.WriteLine();

// Dispose of model and environment
model.Dispose();
env.Dispose();

} catch (GRBException e) {
    Console.WriteLine("Error code: "+ e.ErrorCode + ", " + e.Message);
    Console.WriteLine(e.StackTrace);
}

} // End Main()

/*
 * Copyright 2020, Gurobi Optimization, LLC
 */

tune_cs.cs

/* This example reads a model from a file and tunes it. 
   It then writes the best parameter settings to a file 
   and solves the model using these parameters. */

using System;
using Gurobi;

class tune_cs
{
    static void Main(string[] args)
    {
        if (args.Length < 1) {
            Console.Out.WriteLine("Usage: tune_cs filename");
            return;
        }

        try {
            GRBEnv env = new GRBEnv();

            // Read model from file
            GRBModel model = new GRBModel(env, args[0]);

            // Set the TuneResults parameter to 1
            model.Parameters.TuneResults = 1;

            // Tune the model
            model.Tune();

            // Get the number of tuning results
            int resultcount = model.TuneResultCount;

            if (resultcount > 0) {
                // Load the tuned parameters into the model’s environment

361
model.GetTuneResult(0);

// Write the tuned parameters to a file
model.Write("tune.prm");

// Solve the model using the tuned parameters
model.Optimize();
}

// Dispose of model and environment
model.Dispose();
env.Dispose();

} catch (GRBException e) {
  Console.WriteLine("Error code : " + e.ErrorCode + ". " + e.Message);
}
}

workforce1_cs.cs

/* Copyright 2020, Gurobi Optimization, LLC */

/* Assign workers to shifts; each worker may or may not be available on a particular day. If the problem cannot be solved, use IIS to find a set of conflicting constraints. Note that there may be additional conflicts besides what is reported via IIS. */

using System;
using Gurobi;

class workforce1_cs {
  static void Main() {
    try {
      // Sample data
      // Sets of days and workers
      string[] Shifts =
      string[] Workers =

      int nShifts = Shifts.Length;
      int nWorkers = Workers.Length;

      // Number of workers required for each shift
double[] shiftRequirements =
      new double[] { 3, 2, 4, 4, 5, 6, 5, 2, 2, 3, 4, 6, 7, 5 };

      // Amount each worker is paid to work one shift
double[] pay = new double[] { 10, 12, 10, 8, 8, 9, 11 };
  }
}
// Worker availability: 0 if the worker is unavailable for a shift
double[,] availability =
    new double[,] {{ 0, 1, 1, 0, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1 },
                   { 1, 1, 0, 1, 1, 0, 1, 0, 1, 0, 1, 0, 1, 0 },
                   { 0, 0, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1 },
                   { 0, 1, 1, 0, 1, 1, 0, 1, 1, 0, 1, 1, 1, 1 },
                   { 1, 1, 1, 1, 0, 1, 1, 0, 1, 0, 1, 0, 1, 1 },
                   { 1, 1, 1, 0, 0, 1, 0, 1, 1, 0, 0, 1, 1, 1 },
                   { 1, 1, 1, 0, 1, 0, 1, 0, 1, 1, 0, 1, 1, 1 }};

// Model
GRBEnv env = new GRBEnv();
GRBModel model = new GRBModel(env);
model.ModelName = "assignment";

// Assignment variables: x[w][s] == 1 if worker w is assigned
// to shift s. Since an assignment model always produces integer
// solutions, we use continuous variables and solve as an LP.
GRBVar[,] x = new GRBVar[nWorkers,nShifts];
for (int w = 0; w < nWorkers; ++w) {
    for (int s = 0; s < nShifts; ++s) {
        x[w,s] =
            model.AddVar(0, availability[w,s], pay[w], GRB.CONTINUOUS,
                         Workers[w] + "," + Shifts[s]);
    }
}

// The objective is to minimize the total pay costs
model.ModelSense = GRB.MINIMIZE;

// Constraint: assign exactly shiftRequirements[s] workers
// to each shift s
for (int s = 0; s < nShifts; ++s) {
    GRBLinExpr lhs = 0.0;
    for (int w = 0; w < nWorkers; ++w)
        lhs.AddTerm(1.0, x[w,s]);
    model.AddConstr(lhs == shiftRequirements[s], Shifts[s]);
}

// Optimize
model.Optimize();
int status = model.Status;
if (status == GRB.Status.UNBOUNDED) {
    Console.WriteLine("The model cannot be solved "
                      + " because it is unbounded");
    return;
}
if (status == GRB.Status.OPTIMAL) {
    Console.WriteLine("The optimal objective is " + model.ObjVal);
    return;
}
if ((status != GRB.Status.INF_OR_UNBD) &&
    (status != GRB.Status.INFEASIBLE)) {
    Console.WriteLine("Optimization was stopped with status " + status);
    return;
}
// Do IIS
Console.WriteLine("The model is infeasible; computing IIS");
model.ComputeIIS();
Console.WriteLine("\nThe following constraint(s) "
    + "cannot be satisfied:");
foreach (GRBConstr c in model.GetConstrs()) {
    if (c.IISConstr == 1) {
        Console.WriteLine(c.ConstrName);
    }
}

// Dispose of model and env
model.Dispose();
env.Dispose();
}

} catch (GRBException e) {
    Console.WriteLine("Error code: " + e.ErrorCode + ". " +
        e.Message);
}

workforce2_cs.cs

/* Copyright 2020, Gurobi Optimization, LLC */

/* Assign workers to shifts; each worker may or may not be available on a
particular day. If the problem cannot be solved, use IIS iteratively to
find all conflicting constraints. */

using System;
using System.Collections.Generic;
using Gurobi;

class workforce2_cs
{
    static void Main()
    {
        try {

            // Sample data
            // Sets of days and workers
            string[] Shifts =
                new string[] { "Mon1", "Tue2", "Wed3", "Thu4", "Fri5", "Sat6",
                              "Sun7", "Mon8", "Tue9", "Wed10", "Thu11", "Fri12", "Sat13",
                              "Sun14" };
            string[] Workers =

            int nShifts = Shifts.Length;
            int nWorkers = Workers.Length;

            // Number of workers required for each shift
            double[] shiftRequirements =
new double[] { 3, 2, 4, 4, 5, 6, 5, 2, 2, 3, 4, 6, 7, 5 };

// Amount each worker is paid to work one shift
double[] pay = new double[] { 10, 12, 10, 8, 8, 9, 11 };

// Worker availability: 0 if the worker is unavailable for a shift
double[,] availability =
    new double[,] { 0, 1, 1, 0, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1 },
    { 1, 1, 0, 1, 1, 0, 1, 0, 1, 0, 1, 0, 0, 1 },
    { 0, 1, 1, 0, 1, 0, 1, 1, 0, 1, 1, 1, 1, 1 },
    { 1, 1, 1, 0, 1, 1, 0, 1, 1, 1, 1, 1, 0, 0 },
    { 0, 1, 1, 0, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1 },
    { 1, 1, 1, 0, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1 },
    { 1, 1, 1, 0, 1, 1, 1, 1, 1, 0, 1, 1, 1, 0 };

// Model
GRBEnv env = new GRBEnv();
GRBModel model = new GRBModel(env);

model.ModelName = "assignment";

// Assignment variables: x[w][s] == 1 if worker w is assigned
// to shift s. Since an assignment model always produces integer
// solutions, we use continuous variables and solve as an LP.
GRBVar[,] x = new GRBVar[nWorkers, nShifts];
for (int w = 0; w < nWorkers; ++w)
    for (int s = 0; s < nShifts; ++s)
        x[w,s] =
            model.AddVar(0, availability[w,s], pay[w], GRB.CONTINUOUS,
            Workers[w] + "." + Shifts[s]);

// The objective is to minimize the total pay costs
model.ModelSense = GRB.MINIMIZE;

// Constraint: assign exactly shiftRequirements[s] workers
// to each shift s
for (int s = 0; s < nShifts; ++s)
    for (int w = 0; w < nWorkers; ++w)
        lhs.AddTerm(1.0, x[w, s]);
model.AddConstr(lhs == shiftRequirements[s], Shifts[s]);

// Optimize
model.Optimize();
int status = model.Status;
if (status == GRB.Status.UNBOUNDED) {
    Console.WriteLine("The model cannot be solved "
                      + "because it is unbounded");
    return;
}
if (status == GRB.Status.OPTIMAL) {
    Console.WriteLine("The optimal objective is " + model.ObjVal);
    return;
if ((status != GRB.Status.INF_OR_UNBD) &&
    (status != GRB.Status.INFEASIBLE)) {
    Console.WriteLine("Optimization was stopped with status " + status);
    return;
}

// Do IIS
Console.WriteLine("The model is infeasible; computing IIS");
LinkedList<string> removed = new LinkedList<string>();

// Loop until we reduce to a model that can be solved
while (true) {
    model.ComputeIIS();
    Console.WriteLine("The following constraint cannot be satisfied:");
    foreach (GRBConstr c in model.GetConstrs()) {
        if (c.IISConstr == 1) {
            Console.WriteLine(c.ConstrName);
            // Remove a single constraint from the model
            removed.AddFirst(c.ConstrName);
            model.Remove(c);
            break;
        }
    }
    Console.WriteLine();
    model.Optimize();
    status = model.Status;
    if (status == GRB.Status.UNBOUNDED) {
        Console.WriteLine("The model cannot be solved "
            + "because it is unbounded");
        return;
    }
    if (status == GRB.Status.OPTIMAL) {
        break;
    }
    if ((status != GRB.Status.INF_OR_UNBD) &&
        (status != GRB.Status.INFEASIBLE)) {
        Console.WriteLine("Optimization was stopped with status " +
            status);
        return;
    }
}

Console.WriteLine("The following constraints were removed "
    + "to get a feasible LP:");
foreach (string s in removed) {
    Console.Write(s + " ");
}
Console.WriteLine();

// Dispose of model and env
model.Dispose();
env.Dispose();
catch (GRBException e) {
  Console.WriteLine("Error code: "+ e.ErrorCode + ". " + e.Message);
}
}

workforce3_cs.cs

/* Copyright 2020, Gurobi Optimization, LLC */

/* Assign workers to shifts; each worker may or may not be available on a
particular day. If the problem cannot be solved, relax the model
to determine which constraints cannot be satisfied, and how much
they need to be relaxed. */

using System;
using Gurobi;

class workforce3_cs
{
  static void Main()
  {
    try {

      // Sample data
      // Sets of days and workers
      string[] Shifts =
        new string[] { "Mon1", "Tue2", "Wed3", "Thu4", "Fri5", "Sat6",
        "Sun7", "Mon8", "Tue9", "Wed10", "Thu11", "Fri12", "Sat13",
        "Sun14" };
      string[] Workers =

      int nShifts = Shifts.Length;
      int nWorkers = Workers.Length;

      // Number of workers required for each shift
      double[] shiftRequirements =
        new double[] { 3, 2, 4, 4, 5, 6, 5, 2, 2, 3, 4, 6, 7, 5 };

      // Amount each worker is paid to work one shift
      double[] pay = new double[] { 10, 12, 10, 8, 8, 9, 11 };

      // Worker availability: 0 if the worker is unavailable for a shift
      double[,] availability =
        new double[,] {
          { 0, 1, 1, 0, 1, 0, 1, 0, 1, 1, 1, 1, 1, 1 },
          { 1, 1, 0, 0, 1, 1, 0, 1, 0, 1, 0, 1, 0, 1 },
          { 0, 0, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1 },
          { 0, 1, 1, 0, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1 },
          { 1, 1, 1, 1, 0, 1, 1, 1, 0, 1, 0, 1, 0, 1 },
          { 1, 1, 1, 0, 1, 1, 0, 1, 0, 1, 0, 1, 1, 1 },
          { 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1 } };

      // Model
      GRBEnv env = new GRBEnv();
GRBModel model = new GRBModel(env);

model.ModelName = "assignment";

// Assignment variables: x[w][s] == 1 if worker w is assigned // to shift s. Since an assignment model always produces integer // solutions, we use continuous variables and solve as an LP.
GRBVar[,] x = new GRBVar[nWorkers,nShifts];
for (int w = 0; w < nWorkers; ++w) {
  for (int s = 0; s < nShifts; ++s) {
    x[w,s] =
      model.AddVar(0, availability[w,s], pay[w], GRB.CONTINUOUS,
                   Workers[w] + "." + Shifts[s]);
  }
}

// The objective is to minimize the total pay costs
model.ModelSense = GRB.MINIMIZE;

// Constraint: assign exactly shiftRequirements[s] workers // to each shift s
for (int s = 0; s < nShifts; ++s) {
  GRBLinExpr lhs = 0.0;
  for (int w = 0; w < nWorkers; ++w) {
    lhs.AddTerm(1.0, x[w,s]);
  }
  model.AddConstr(lhs == shiftRequirements[s], Shifts[s]);
}

// Optimize
model.Optimize();
int status = model.Status;
if (status == GRB.Status.UNBOUNDED) {
  Console.WriteLine("The model cannot be solved "+ "because it is unbounded");
  return;
}
if (status == GRB.Status.OPTIMAL) {
  Console.WriteLine("The optimal objective is " + model.ObjVal);
  return;
}
if ((status != GRB.Status.INF_OR_UNBD) &&
    (status != GRB.Status.INFEASIBLE)) {
  Console.WriteLine("Optimization was stopped with status " + status);
  return;
}

// Relax the constraints to make the model feasible
Console.WriteLine("The model is infeasible; relaxing the constraints");
int originumvars = model.NumVars;
model.FeasRelax(0, false, false, true);
model.Optimize();
status = model.Status;
if ((status == GRB.Status.INF_OR_UNBD) ||
    (status == GRB.Status.INFEASIBLE) ||
    (status == GRB.Status.UNBOUNDED)) {
Console.WriteLine("The relaxed model cannot be solved "+"because it is infeasible or unbounded");
return;
}
if (status != GRB.Status.OPTIMAL) {
    Console.WriteLine("Optimization was stopped with status " + status);
    return;
}

Console.WriteLine("\nSlack values:");
GRBVar[] vars = model.GetVars();
for (int i = orignumvars; i < model.NumVars; ++i) {
    GRBVar sv = vars[i];
    if (sv.X > 1e-6) {
        Console.WriteLine(sv.VarName + " = " + sv.X);
    }
}

// Dispose of model and environment
model.Dispose();
env.Dispose();

} catch (GRBException e) {
    Console.WriteLine("Error code: " + e.ErrorCode + ". " + e.Message);
}
}

workforce4_cs.cs

/* Copyright 2020, Gurobi Optimization, LLC */

/* Assign workers to shifts; each worker may or may not be available on a particular day. We use Pareto optimization to solve the model: first, we minimize the linear sum of the slacks. Then, we constrain the sum of the slacks, and we minimize a quadratic objective that tries to balance the workload among the workers. */

using System;
using Gurobi;

class workforce4_cs
{
    static void Main()
    {
        try {

            // Sample data
            // Sets of days and workers
            string[] Shifts =
            string[] Workers =
int nShifts = Shifts.Length;
int nWorkers = Workers.Length;

// Number of workers required for each shift
double[] shiftRequirements =
    new double[] { 3, 2, 4, 4, 6, 5, 2, 2, 3, 4, 6, 7, 5 };

// Worker availability: 0 if the worker is unavailable for a shift
double[,] availability =
    new double[,] { { 0, 1, 1, 0, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1 },
                    { 1, 1, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1 },
                    { 0, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1 },
                    { 1, 1, 1, 0, 0, 1, 1, 1, 1, 0, 1, 0, 1, 1 },
                    { 1, 1, 1, 0, 1, 0, 1, 1, 1, 0, 1, 0, 1, 1 },
                    { 1, 1, 1, 0, 0, 1, 1, 1, 1, 0, 1, 1, 1, 1 },
                    { 1, 1, 1, 0, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1 } };

// Model
GRBEnv env = new GRBEnv();
GRBModel model = new GRBModel(env);
model.ModelName = "assignment";

// Assignment variables: x[w][s] == 1 if worker w is assigned
// to shift s. This is no longer a pure assignment model, so we must
// use binary variables.
GRBVar[,] x = new GRBVar[nWorkers, nShifts];
for (int w = 0; w < nWorkers; ++w) {
    for (int s = 0; s < nShifts; ++s) {
        x[w,s] =
            model.AddVar(0, availability[w,s], 0, GRB.BINARY, 
                         Workers[w] + "." + Shifts[s]);
    }
}

// Slack variables for each shift constraint so that the shifts can
// be satisfied
GRBVar[] slacks = new GRBVar[nShifts];
for (int s = 0; s < nShifts; ++s) {
    slacks[s] =
        model.AddVar(0, GRB.INFINITY, 0, GRB.CONTINUOUS, 
                    Shifts[s] + "Slack");
}

// Variable to represent the total slack
GRBVar totSlack = model.AddVar(0, GRB.INFINITY, 0, GRB.CONTINUOUS, 
                                "totSlack");

// Variables to count the total shifts worked by each worker
GRBVar[] totShifts = new GRBVar[nWorkers];
for (int w = 0; w < nWorkers; ++w) {
    totShifts[w] = model.AddVar(0, GRB.INFINITY, 0, GRB.CONTINUOUS, 
                                Workers[w] + "TotShifts");
}
GRBLinExpr lhs;

// Constraint: assign exactly shiftRequirements[s] workers to each shift s, plus the slack
for (int s = 0; s < nShifts; ++s) {
    lhs = new GRBLinExpr();
    lhs.AddTerm(1.0, slacks[s]);
    for (int w = 0; w < nWorkers; ++w) {
        lhs.AddTerm(1.0, x[w, s]);
    }
    model.AddConstr(lhs == shiftRequirements[s], Shifts[s]);
}

// Constraint: set totSlack equal to the total slack
lhs = new GRBLinExpr();
for (int s = 0; s < nShifts; ++s) {
    lhs.AddTerm(1.0, slacks[s]);
}
model.AddConstr(lhs == totSlack, "totSlack");

// Constraint: compute the total number of shifts for each worker
for (int w = 0; w < nWorkers; ++w) {
    lhs = new GRBLinExpr();
    for (int s = 0; s < nShifts; ++s) {
        lhs.AddTerm(1.0, x[w, s]);
    }
    model.AddConstr(lhs == totShifts[w], "totShifts" + Workers[w]);
}

// Objective: minimize the total slack
model.SetObjective(1.0*totSlack);

// Optimize
int status = solveAndPrint(model, totSlack, nWorkers, Workers, totShifts);
if (status != GRB.Status.OPTIMAL) {
    return;
}

// Constrain the slack by setting its upper and lower bounds
totSlack.UB = totSlack.X;
totSlack.LB = totSlack.X;

// Variable to count the average number of shifts worked
GRBVar avgShifts =
    model.AddVar(0, GRB.INFINITY, 0, GRB.CONTINUOUS, "avgShifts");

// Variables to count the difference from average for each worker;
// note that these variables can take negative values.
GRBVar[] diffShifts = new GRBVar[nWorkers];
for (int w = 0; w < nWorkers; ++w) {
    diffShifts[w] = model.AddVar(-GRB.INFINITY, GRB.INFINITY, 0,
                              GRB.CONTINUOUS, Workers[w] + "Diff");
}

// Constraint: compute the average number of shifts worked
lhs = new GRBLinExpr();
for (int w = 0; w < nWorkers; ++w) {
    lhs.AddTerm(1.0, totShifts[w]);
}
model.AddConstr(lhs == nWorkers * avgShifts, "avgShifts");

// Constraint: compute the difference from the average number of shifts
for (int w = 0; w < nWorkers; ++w) {
    model.AddConstr(totShifts[w] - avgShifts == diffShifts[w], Workers[w] + " Diff");
}

// Objective: minimize the sum of the square of the difference from the
// average number of shifts worked
GRBQuadExpr qobj = new GRBQuadExpr();
for (int w = 0; w < nWorkers; ++w) {
    qobj.AddTerm(1.0, diffShifts[w], diffShifts[w]);
}
model.SetObjective(qobj);

// Optimize
status = solveAndPrint(model, totSlack, nWorkers, Workers, totShifts);
if (status != GRB.Status.OPTIMAL) {
    return;
}

// Dispose of model and env
model.Dispose();
env.Dispose();
}

private static int solveAndPrint(GRBModel model, GRBVar totSlack, int nWorkers, String[] Workers, GRBVar[] totShifts)
{
    model.Optimize();
    int status = model.Status;
    if ((status == GRB.Status.INF_OR_UNBD) ||
        (status == GRB.Status.INFEASIBLE) ||
        (status == GRB.Status.UNBOUNDED)) {
        Console.WriteLine("The model cannot be solved "+ "because it is infeasible or unbounded");
        return status;
    }
    if (status != GRB.Status.OPTIMAL) {
        Console.WriteLine("Optimization was stopped with status " + status);
        return status;
    }

    // Print total slack and the number of shifts worked for each worker
    Console.WriteLine("Total slack required: " + totSlack.X);
    for (int w = 0; w < nWorkers; ++w) {
Console.WriteLine(Workers[w] + " worked " +
    totShifts[w].X + " shifts");
}
Console.WriteLine("\n");
return status;
}
}

workforce5_cs.cs

/* Copyright 2020, Gurobi Optimization, LLC */

/* Assign workers to shifts; each worker may or may not be available on a
   particular day. We use multi-objective optimization to solve the model.
The highest-priority objective minimizes the sum of the slacks
(i.e., the total number of uncovered shifts). The secondary objective
minimizes the difference between the maximum and minimum number of
shifts worked among all workers. The second optimization is allowed
to degrade the first objective by up to the smaller value of 10% and 2 */

using System;
using Gurobi;

class workforce5_cs
{
    static void Main()
    {
        try {

            // Sample data
            // Sets of days and workers
            string[] Shifts =
                new string[] { "Mon1", "Tue2", "Wed3", "Thu4", "Fri5", "Sat6",
                    "Sun7", "Mon8", "Tue9", "Wed10", "Thu11", "Fri12", "Sat13",
                    "Sun14" };
            string[] Workers =

            int nShifts = Shifts.Length;
            int nWorkers = Workers.Length;

            // Number of workers required for each shift
            double[] shiftRequirements =
                new double[] { 3, 2, 4, 5, 6, 5, 2, 2, 3, 4, 6, 7, 5 };

            // Worker availability: 0 if the worker is unavailable for a shift
            double[,] availability =
                new double[,] { { 0, 1, 1, 0, 1, 0, 1, 1, 1, 1, 1, 1 },
                                { 1, 1, 0, 0, 1, 1, 0, 1, 0, 1, 0, 0 },
                                { 0, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1 },
                                { 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1 },
                                { 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1 },
                                { 1, 1, 0, 0, 1, 1, 0, 1, 0, 1, 0, 1 },
                                { 1, 1, 1, 0, 1, 1, 0, 1, 0, 1, 0, 1 },
                                { 0, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1 },
                                { 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1 },
                                { 0, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1 } );
// Create environment
GRBEnv env = new GRBEnv();

// Create initial model
GRBModel model = new GRBModel(env);
model.ModelName = "workforce5_cs";

// Initialize assignment decision variables:
// x[w][s] == 1 if worker w is assigned to shift s.
// This is no longer a pure assignment model, so we must
// use binary variables.
GRBVar[,] x = new GRBVar[nWorkers, nShifts];
for (int w = 0; w < nWorkers; ++w) {
    for (int s = 0; s < nShifts; ++s) {
        x[w,s] = model.AddVar(0, availability[w,s], 0, GRB.BINARY, string.Format("{0}.{1}" , Workers[w], Shifts[s]));
    }
}

// Slack variables for each shift constraint so that the shifts can
// be satisfied
GRBVar[] slacks = new GRBVar[nShifts];
for (int s = 0; s < nShifts; ++s) {
    slacks[s] = model.AddVar(0, GRB.INFINITY, 0, GRB.CONTINUOUS, string.Format("{0} Slack", Shifts[s]));
}

// Variable to represent the total slack
GRBVar totSlack = model.AddVar(0, GRB.INFINITY, 0, GRB.CONTINUOUS, "totSlack");

// Variables to count the total shifts worked by each worker
GRBVar[] totShifts = new GRBVar[nWorkers];
for (int w = 0; w < nWorkers; ++w) {
    totShifts[w] = model.AddVar(0, GRB.INFINITY, 0, GRB.CONTINUOUS, string.Format("{0} TotShifts", Workers[w]));
}

GRBLinExpr lhs;

// Constraint: assign exactly shiftRequirements[s] workers
// to each shift s, plus the slack
for (int s = 0; s < nShifts; ++s) {
    lhs = new GRBLinExpr();
    lhs.AddTerm(1.0, slacks[s]);
    for (int w = 0; w < nWorkers; ++w) {
        lhs.AddTerm(1.0, x[w,s]);
    }
    model.AddConstr(lhs, GRB.EQUAL, shiftRequirements[s], Shifts[s]);
}

// Constraint: set totSlack equal to the total slack
lhs = new GRBLinExpr();
lhs.AddTerm(-1.0, totSlack);
for (int s = 0; s < nShifts; ++s) {
    lhs.AddTerm(1.0, slacks[s]);
}
model.AddConstr(lhs, GRB.EQUAL, 0, "totSlack");

// Constraint: compute the total number of shifts for each worker
for (int w = 0; w < nWorkers; ++w) {
    lhs = new GRBLinExpr();
    lhs.AddTerm(-1.0, totShifts[w]);
    for (int s = 0; s < nShifts; ++s) {
        lhs.AddTerm(1.0, x[w][s]);
    }
    model.AddConstr(lhs, GRB.EQUAL, 0, string.Format("totShifts{0}", Workers[w]));
}

// Constraint: set minShift/maxShift variable to less <=/=> to the
// number of shifts among all workers
GRBVar minShift = model.AddVar(0, GRB.INFINITY, 0, GRB.CONTINUOUS, "minShift");
GRBVar maxShift = model.AddVar(0, GRB.INFINITY, 0, GRB.CONTINUOUS, "maxShift");
model.AddGenConstrMin(minShift, totShifts, GRB.INFINITY, "minShift");
model.AddGenConstrMax(maxShift, totShifts, -GRB.INFINITY, "maxShift");

// Set global sense for ALL objectives
model.ModelSense = GRB.MINIMIZE;

// Set primary objective
model.SetObjectiveN(totSlack, 0, 2, 1.0, 2.0, 0.1, "TotalSlack");

// Set secondary objective
model.SetObjectiveN(maxShift - minShift, 1, 1, 1.0, 0, 0, "Fairness");

// Save problem
model.Write("workforce5_cs.lp");

// Optimize
int status = solveAndPrint(model, totSlack, nWorkers, Workers, totShifts);
if (status != GRB.Status.OPTIMAL)
    return;

// Dispose of model and environment
model.Dispose();
env.Dispose();
}

private static int solveAndPrint(GRBModel model, GRBVar totSlack, int nWorkers, String[] Workers, GRBVar[] totShifts)
{
    ...
}
model.Optimize();
int status = model.Status;
if (status == GRB.Status.INF_OR_UNBD ||
    status == GRB.Status.INFEASIBLE ||
    status == GRB.Status.UNBOUNDED) {
    Console.WriteLine("The model cannot be solved "+ "because it is infeasible or unbounded");
    return status;
}
if (status != GRB.Status.OPTIMAL) {
    Console.WriteLine("Optimization was stopped with status {0} ", status);
    return status;
}

// Print total slack and the number of shifts worked for each worker
Console.WriteLine("Total slack required: {0} ", totSlack.X);
for (int w = 0; w < nWorkers; ++w) {
    Console.WriteLine("{0} worked {1} shifts ", Workers[w], totShifts[w].X);
}
Console.WriteLine("\n");
return status;
}

3.5 Visual Basic Examples

This section includes source code for all of the Gurobi Visual Basic examples. The same source code can be found in the examples/vb directory of the Gurobi distribution.

batchmode_vb.vb

' Copyright 2020, Gurobi Optimization, LLC
',
' This example reads a MIP model from a file, solves it in batch mode,
' and prints the JSON solution string.
',
' You will need a Cluster Manager license for this example to work. */

Imports System
Imports Gurobi

Class batchmode_vb

' Set-up the environment for batch mode optimization.
',
' The function creates an empty environment, sets all neccessary
' parameters, and returns the ready-to-be-started Env object to caller.
' It is the caller's responsibility to dispose of this environment when
' it's no longer needed.
Private Shared Function setupbatchenv() As GRBEnv
    Dim env As GRBEnv = New GRBEnv(True)
    env.CSBatchMode = 1
    env.CSManager = "http://localhost:61080"
    envLogFile = "batchmode.log"

End Class
env.ServerPassword = "pass"
env.UserName = "gurobi"

' No network communication happened up to this point. This will happen once the caller invokes the start() method of the returned Env Object.

Return env
End Function

' Print batch job error information, if any
Private Shared Sub printbatcherrorinfo(ByRef batch As GRBBatch)
    If batch.BatchErrorCode = 0 Then Return
End Sub

' Create a batch request for given problem file
Private Shared Function newbatchrequest(ByVal filename As String) As String
    Dim batchID As String = 

    ' Start environment, create Model object from file
    Dim env As GRBEnv = setupbatchenv()
    env.Start()
    Dim model As GRBModel = New GRBModel(env, filename)

    Try
        ' Set some parameters
        model[[Set](GRB.DoubleParam.MIPGap, 0.01)
        model[[Set](GRB.IntParam.JSONSolDetail, 1)

        ' Define tags for some variables in order to access their values later
        Dim count As Integer = 0
        For Each v As GRBVar In model.GetVars()
            v.VTag = "Variable" & count
            count += 1
            If count >= 10 Then Exit For
        Next

        ' submit batch request
        batchID = model.OptimizeBatch()
    Finally
        model.Dispose()
        env.Dispose()
    End Try

    Return batchID
End Function

' Wait for the final status of the batch.
' Initially the status of a batch is "submitted"; the status will change once the batch has been processed (by a compute server).
Private Shared Sub waitforfinalstatus(ByVal batchID As String)
    ' Wait no longer than one hour
    Dim maxwaittime As Double = 3600
    Dim start As DateTime = DateTime.Now

' Setup and start environment, create local Batch handle object
Dim env As GRBEnv = setupbatchenv()
env.Start()
Dim batch As GRBBatch = New GRBBatch(env, batchID)

Try

While batch.BatchStatus = GRB.BatchStatus.SUBMITTED

' Abort this batch if it is taking too long
Dim interval As TimeSpan = DateTime.Now - start
If interval.TotalSeconds > maxwaittime Then
    batch.Abort()
    Exit While
End If

' Wait for two seconds

' Update the resident attribute cache of the Batch object with the
' latest values from the cluster manager.
batch.Update()

' If the batch failed, we retry it
If batch.BatchStatus = GRB.BatchStatus.FAILED Then
    batch.Retry()
batch.Update()
End If
End While

Finally

' Print information about error status of the job that
' processed the batch
printbatcherrorinfo(batch)
batch.Dispose()
env.Dispose()
End Try
End Sub

Private Shared Sub printfinalreport(ByVal batchID As String)
' Setup and start environment, create local Batch handle object
Dim env As GRBEnv = setupbatchenv()
env.Start()
Dim batch As GRBBatch = New GRBBatch(env, batchID)

Select Case batch.BatchStatus
    Case GRB.BatchStatus.CREATED
        Console.WriteLine("Batch status is 'CREATED'")
    Case GRB.BatchStatus.SUBMITTED
        Console.WriteLine("Batch is 'SUBMITTED'")
    Case GRB.BatchStatus.ABORTED
        Console.WriteLine("Batch is 'ABORTED'")
    Case GRB.BatchStatus.FAILED
        Console.WriteLine("Batch is 'FAILED'")
    Case GRB.BatchStatus.COMPLETED
        Console.WriteLine("Batch is 'COMPLETED'")
End Select
' Pretty printing the general solution information
Console.WriteLine("JSON solution:" & batch.GetJSONSolution())

' Write the full JSON solution string to a file
batch.WriteJSONSolution("batch-sol.json.gz")

Case Else
' Should not happen
Console.WriteLine("Unknown BatchStatus" & batch.BatchStatus)
Environment.Exit(1)
End Select

batch.Dispose()
env.Dispose()

End Sub

' Instruct cluster manager to discard all data relating to this BatchID
Private Shared Sub batchdiscard(ByVal batchID As String)
' Setup and start environment, create local Batch handle object
Dim env As GRBEnv = setupbatchenv()
env.Start()
Dim batch As GRBBatch = New GRBBatch(env, batchID)

' Remove batch request from manager
batch.Discard()
batch.Dispose()
env.Dispose()
End Sub

' Solve a given model using batch optimization
Shared Sub Main(ByVal args As String())
' Ensure we have an input file
If args.Length < 1 Then
    Console.Out.WriteLine("Usage: batchmode_vb filename")
    Return
End If

Try
' Submit new batch request
Dim batchID As String = newbatchrequest(args(0))

' Wait for final status
waitforfinalstatus(batchID)

' Report final status info
printfinalreport(batchID)

' Remove batch request from manager
batchdiscard(batchID)

Console.WriteLine("Batch optimization OK")
Catch e As GRBException
End Try
End Sub
End Class
This example formulates and solves the following simple bilinear model:

maximize \( x \)
subject to \( x + y + z \leq 10 \)
\( x \cdot y \leq 2 \)  (bilinear inequality)
\( x \cdot z + y \cdot z = 1 \)  (bilinear equality)
\( x, y, z \) non-negative (\( x \) integral in second version)

Imports Gurobi

Class bilinear_vb
    Shared Sub Main()
        Try
            Dim env As New GRBEnv("bilinear.log")
            Dim model As New GRBModel(env)

            ' Create variables
            Dim x As GRBVar = model.AddVar(0, GRB_INFINITY, 0, GRB.CONTINUOUS, "x")
            Dim y As GRBVar = model.AddVar(0, GRB_INFINITY, 0, GRB.CONTINUOUS, "y")
            Dim z As GRBVar = model.AddVar(0, GRB_INFINITY, 0, GRB.CONTINUOUS, "z")

            ' Set objective
            Dim obj As GRBLinExpr = x
            model.SetObjective(obj, GRB.MAXIMIZE)

            ' Add linear constraint: \( x + y + z \leq 10 \)
            model.AddConstr(x + y + z <= 10, "c0")

            ' Add bilinear inequality: \( x \cdot y \leq 2 \)
            model.AddQConstr(x * y <= 2, "bilinear0")

            ' Add bilinear equality: \( x \cdot z + y \cdot z = 1 \)
            model.AddQConstr(x * z + y * z = 1, "bilinear1")

        ' Optimize model
        Try
            model.Optimize()
            Catch e As GRBException
                Console.WriteLine("Failed (as expected)")
            End Try

            model.Set(GRB.IntParam.NonConvex, 2)
            model.Optimize()

            Console.WriteLine(x.VarName & " " & x.X)
            Console.WriteLine(y.VarName & " " & y.X)
            Console.WriteLine(z.VarName & " " & z.X)
        End Try
    End Sub
End Class

x.Set(GLB.CharAttr.VType, GLB.INTEGER)
model.Optimize()

Console.WriteLine(x.VarName & " " & x.X)
Console.WriteLine(y.VarName & " " & y.X)
Console.WriteLine(z.VarName & " " & z.X)


' Dispose of model and env
model.Dispose()
env.Dispose()

Catch e As GRBException
End Try

End Sub
End Class

callback_vb.vb

' Copyright 2020, Gurobi Optimization, LLC

' This example reads a model from a file, sets up a callback that
' monitors optimization progress and implements a custom
' termination strategy, and outputs progress information to the
' screen and to a log file.
' The termination strategy implemented in this callback stops the
' optimization of a MIP model once at least one of the following two
' conditions have been satisfied:
' 1) The optimality gap is less than 10%
' 2) At least 10000 nodes have been explored, and an integer feasible
'    solution has been found.
' Note that termination is normally handled through Gurobi parameters
' (MIPGap, NodeLimit, etc.). You should only use a callback for
' termination if the available parameters don’t capture your desired
' termination criterion.

Imports System
Imports Gurobi

Class callback_vb
    Inherits GRBCallback
    Private vars As GRBVar()
    Private lastnode As Double
    Private lastiter As Double

    Public Sub New(ByVal xvars As GRBVar())
        vars = xvars
        lastnode = lastiter = -1
    End Sub
End Class
Protected Overloads Overrides Sub Callback()
Try
  If where = GRB.Callback.PRESOLVE Then
    ' Presolve callback
    Dim cdels As Integer = GetIntInfo(GRB.Callback.PRE_COLDEL)
    Dim rdels As Integer = GetIntInfo(GRB.Callback.PRE_ROWDEL)
    Console.WriteLine(cdels & " columns and " & rdels & " rows are removed")
  ElseIf where = GRB.Callback.SIMPLEX Then
    ' Simplex callback
    Dim itcnt As Double = GetDoubleInfo(GRB.Callback.SPX_ITRCNT)
    If itcnt Mod lastiter >= 100 Then
      lastiter = itcnt
      Dim obj As Double = GetDoubleInfo(GRB.Callback.SPX_OBJVAL)
      Dim pinf As Double = GetDoubleInfo(GRB.Callback.SPX_PRIMINF)
      Dim dinf As Double = GetDoubleInfo(GRB.Callback.SPX_DUALINF)
      Dim ispert As Integer = GetIntInfo(GRB.Callback.SPX_ISPERT)
      Dim ch As Char
      If ispert = 0 Then
        ch = "c"
      ElseIf ispert = 1 Then
        ch = "S"c
      Else
        ch = "P"c
      End If
      Console.WriteLine(itcnt & " & obj & ch & " & pinf & " & dinf)
    End If
  ElseIf where = GRB.Callback.MIP Then
    ' General MIP callback
    Dim nodecnt As Double = GetDoubleInfo(GRB.Callback.MIP_NODCNT)
    If nodecnt - lastnode >= 100 Then
      lastnode = nodecnt
      Dim objbst As Double = GetDoubleInfo(GRB.Callback.MIP_OBJBST)
      Dim objbnd As Double = GetDoubleInfo(GRB.Callback.MIP_OBJBND)
      If Math.Abs(objbst - objbnd) < 0.1 * (1.0R + Math.Abs(objbst)) Then
        Abort()
      End If
      Dim actnodes As Integer = CInt(GetDoubleInfo(GRB.Callback.MIP_NODLFT))
      Dim itcnt As Integer = CInt(GetDoubleInfo(GRB.Callback.MIP_ITRCNT))
      Dim solcnt As Integer = GetIntInfo(GRB.Callback.MIP_SOLCNT)
      Dim cutcnt As Integer = GetIntInfo(GRB.Callback.MIP_CUTCNT)
      Console.WriteLine(nodecnt & " & actnodes & " & itcnt & " & _
                       objbst & " & objbnd & " & solcnt & " & cutcnt)
    End If
  ElseIf where = GRB.Callback.MIPSOL Then
    ' MIP solution callback
    Dim obj As Double = GetDoubleInfo(GRB.Callback.MIPSOL_OBJ)
    Dim nodecnt As Integer = CInt(GetDoubleInfo(GRB.Callback.MIPSOL_NODCNT))
    Dim x As Double() = GetSolution(vars)
    Console.WriteLine("**** New solution at node " & nodecnt & ", obj " & _
                       obj & ", x(0) = " & x(0) & "****")
  End If
Catch e As GRBException
  Console.WriteLine(e.StackTrace)
End Sub
End Try
End Sub

Shared Sub Main(ByVal args As String())

If args.Length < 1 Then
    Console.WriteLine("Usage: callback_vb filename")
    Return
End If

Try
    Dim env As New GRBEnv()
    Dim model As New GRBModel(env, args(0))
    Dim vars As GRBVar() = model.GetVars()

    ' Create a callback object and associate it with the model
    model.SetCallback(New callback_vb(vars))
    model.Optimize()

    Dim x As Double() = model.Get(GRB.DoubleAttr.X, vars)
    Dim vnames As String() = model.Get(GRB.StringAttr.VarName, vars)

    For j As Integer = 0 To vars.Length - 1
        If x(j) <> 0.0R Then
            Console.WriteLine(vnames(j) & " " & x(j))
        End If
    Next

    ' Dispose of model and env
    model.Dispose()
    env.Dispose()

    Catch e As GRBException
        Console.WriteLine(e.StackTrace)
    End Try
End Sub
End Class

dense_vb.vb

' Copyright 2020, Gurobi Optimization, LLC
'
' This example formulates and solves the following simple QP model:
'
' minimize    x + y + x^2 + x*y + y^2 + y*z + z^2
' subject to  x + 2 y + 3 z >= 4
'            x + y >= 1
'            x, y, z non-negative
'
' The example illustrates the use of dense matrices to store A and Q
' (and dense vectors for the other relevant data). We don't recommend
' that you use dense matrices, but this example may be helpful if you
' already have your data in this format.
Imports Gurobi

Class dense_vb

Protected Shared Function dense_optimize (env As GRBEnv, rows As Integer, cols As Integer, c As Double(), Q As Double(,), A As Double(,), sense As Char(), rhs As Double(), lb As Double(), ub As Double(), vtype As Char(), solution As Double()) As Boolean

Dim success As Boolean = False

Try
  Dim model As New GRBModel(env)

    ' Add variables to the model
    Dim vars As GRBVar() = model.AddVars(lb, ub, Nothing, vtype, Nothing)

    ' Populate A matrix
    For i As Integer = 0 To rows - 1
        Dim expr As New GRBLinExpr()
        For j As Integer = 0 To cols - 1
            If A(i, j) <> 0 Then
                expr.AddTerm(A(i, j), vars(j))
            End If
        Next
        model.AddConstr(expr, sense(i), rhs(i), "")
    Next

    ' Populate objective
    Dim obj As New GRBQuadExpr()
    If Q IsNot Nothing Then
        For i As Integer = 0 To cols - 1
            For j As Integer = 0 To cols - 1
                If Q(i, j) <> 0 Then
                    obj.AddTerm(Q(i, j), vars(i), vars(j))
                End If
            Next
        Next
        model.SetObjective(obj)
    End If

  End Try

success

success = dense_optimize(env, rows, cols, c, Q, A, sense, rhs, lb, ub, vtype, solution)

success
End If

' Solve model
model.Optimize()

' Extract solution
If model.Status = GRB.Status.OPTIMAL Then
    success = True
    For j As Integer = 0 To cols - 1
        solution(j) = vars(j).X
    Next
End If

model.Dispose()
Imports System
Imports Gurobi

Class diet_vb
    Shared Sub Main()
        Try

            ' Nutrition guidelines, based on
            ' USDA Dietary Guidelines for Americans, 2005
            Dim Categories As String() = New String() {"calories", "protein", "fat", "sodium"}

            Dim nCategories As Integer = Categories.Length
            Dim minNutrition As Double() = New Double() {1800, 91, 0, 0}
            Dim maxNutrition As Double() = New Double() {2200, GRB.INFINITY, 65, 1779}

            ' Set of foods
            Dim Foods As String() = New String() {"hamburger", "chicken", "hot dog", "fries", "macaroni", "pizza", "salad", "milk", "ice cream"}

            Dim nFoods As Integer = Foods.Length
            Dim cost As Double() = New Double() {2.49, 2.89, 1.5R, 1.89, 2.09, 1.99, 2.49, 0.89, 1.59}

            ' Nutrition values for the foods
            ' hamburger
            ' chicken
            ' hot dog
            ' fries
            ' macaroni
            ' pizza
            ' salad
            ' milk
            ' ice cream
            Dim nutritionValues As Double(,) = New Double(,) {{410, 24, 26, 730}, _
            {420, 32, 10, 1190}, _
            {560, 20, 32, 1800}, _
            {380, 4, 19, 270}, _
            {320, 12, 10, 930}, _
            {320, 15, 12, 820}, _
            {320, 31, 12, 1230}, _
            {100, 8, 2.5, 125}, _
            {330, 8, 10, 180}}

            ' Model
            Dim env As New GRBEnv()
            Dim model As New GRBModel(env)

            model.ModelName = "diet"
        Catch ex
            Console.WriteLine(ex.Message)
        End Try
    End Sub
End Class
' Create decision variables for the nutrition information, 
' which we limit via bounds
Dim nutrition As GRBVar() = New GRBVar(nCategories - 1) {}
For i As Integer = 0 To nCategories - 1
    nutrition(i) = model.AddVar(minNutrition(i), maxNutrition(i), 0, _
                               GRB.CONTINUOUS, Categories(i))
Next

' Create decision variables for the foods to buy
Dim buy As GRBVar() = New GRBVar(nFoods - 1) {}
For j As Integer = 0 To nFoods - 1
    buy(j) = model.AddVar(0, GRB.INFINITY, cost(j), GRB.CONTINUOUS, _
                          Foods(j))
Next

' The objective is to minimize the costs
model.ModelSense = GRB.MINIMIZE

' Nutrition constraints
For i As Integer = 0 To nCategories - 1
    Dim ntot As GRBLinExpr = 0
    For j As Integer = 0 To nFoods - 1
        ntot.AddTerm(nutritionValues(j, i), buy(j))
    Next
    model.AddConstr(ntot = nutrition(i), Categories(i))
Next

' Solve
model.Optimize()
PrintSolution(model, buy, nutrition)

Console.WriteLine(vbLf & "Adding constraint: at most 6 servings of dairy")
model.AddConstr(buy(7) + buy(8) <= 6, "limit_dairy")

' Solve
model.Optimize()
PrintSolution(model, buy, nutrition)

' Dispose of model and env
model.Dispose()
env.Dispose()

Catch e As GRBException
End Try
End Sub

Private Shared Sub PrintSolution(ByVal model As GRBModel, ByVal buy As GRBVar(), _
                                 ByVal nutrition As GRBVar())
    If model.Status = GRB.Status.OPTIMAL Then
        Console.WriteLine(vbLf & "Cost: " & model.ObjVal)
        Console.WriteLine(vbLf & "Buy:")
        For j As Integer = 0 To buy.Length - 1
            If buy(j).X > 0.0001 Then
                Console.WriteLine(" Buy ")
            End If
        Next
    End If
End Sub
Imports System
Imports Gurobi

Class facility_vb
    Shared Sub Main()
        Try

            ' Warehouse demand in thousands of units
            Dim Demand As Double() = New Double() {15, 18, 14, 20}

            ' Plant capacity in thousands of units
            Dim Capacity As Double() = New Double() {20, 22, 17, 19, 18}

            ' Fixed costs for each plant
            Dim FixedCosts As Double() = New Double() {12000, 15000, 17000, 13000, 16000}

            ' Transportation costs per thousand units
            Dim TransCosts As Double(,) = New Double(,) {{4000, 2000, 3000, 2500, 4500},
                                                      {2500, 2600, 3400, 3000, 4000},
                                                      {1200, 1800, 2600, 4100, 3000},
                                                      {2200, 2600, 3100, 3700, 3200}}

            ' Number of plants and warehouses
            Dim nPlants As Integer = Capacity.Length
            Dim nWarehouses As Integer = Demand.Length

            ' Model
            Dim env As New GRBEnv()
            Dim model As New GRBModel(env)
        End Try
    End Sub
End Class
model.ModelName = "facility"

' Plant open decision variables: open(p) == 1 if plant p is open.
Dim open As GRBVar() = New GRBVar(nPlants - 1) {}
For p As Integer = 0 To nPlants - 1
    open(p) = model.AddVar(0, 1, FixedCosts(p), GRB.BINARY, "Open" & p)
Next

' Transportation decision variables: how much to transport from
' a plant p to a warehouse w
Dim transport As GRBVar(,) = New GRBVar(nWarehouses - 1, nPlants - 1) {}
For w As Integer = 0 To nWarehouses - 1
    For p As Integer = 0 To nPlants - 1
        transport(w, p) = model.AddVar(0, GRB.INFINITY, _
            TransCosts(w, p), GRB.CONTINUOUS, _
            "Trans" & p & "." & w)
    Next
Next

' The objective is to minimize the total fixed and variable costs
model.ModelSense = GRB.MINIMIZE

' Production constraints
' Note that the right-hand limit sets the production to zero if
' the plant is closed
For p As Integer = 0 To nPlants - 1
    Dim ptot As GRBLinExpr = 0
    For w As Integer = 0 To nWarehouses - 1
        ptot.AddTerm(1.0, transport(w, p))
    Next
    model.AddConstr(ptot <= Capacity(p) * open(p), "Capacity" & p)
Next

' Demand constraints
For w As Integer = 0 To nWarehouses - 1
    Dim dtot As GRBLinExpr = 0
    For p As Integer = 0 To nPlants - 1
        dtot.AddTerm(1.0, transport(w, p))
    Next
    model.AddConstr(dtot = Demand(w), "Demand" & w)
Next

' Guess at the starting point: close the plant with the highest
' fixed costs; open all others

' First, open all plants
For p As Integer = 0 To nPlants - 1
    open(p).Start = 1.0
Next

' Now close the plant with the highest fixed cost
Console.WriteLine("Initial guess:")
Dim maxFixed As Double = -GRB.INFINITY
For p As Integer = 0 To nPlants - 1
    If FixedCosts(p) > maxFixed Then
maxFixed = FixedCosts(p)
End If
Next
For p As Integer = 0 To nPlants - 1
If FixedCosts(p) = maxFixed Then
    open(p).Start = 0.0
    Console.WriteLine("Closing plant " & p & vbCrLf)
    Exit For
End If
Next

' Use barrier to solve root relaxation
model.Parameters.Method = GRB.METHOD_BARRIER

' Solve
model.Optimize()

' Print solution
Console.WriteLine(vbCrLf & "TOTAL COSTS: " & model.ObjVal)
Console.WriteLine("SOLUTION:")
For p As Integer = 0 To nPlants - 1
    If open(p).X > 0.99 Then
        Console.WriteLine("Plant " & p & " open:")
        For w As Integer = 0 To nWarehouses - 1
            If transport(w, p).X > 0.0001 Then
                Console.WriteLine(" Transport " & transport(w, p).X & " units to warehouse " & w)
            End If
        Next
    Else
        Console.WriteLine("Plant " & p & " closed!")
    End If
Next

' Dispose of model and env
model.Dispose()
env.Dispose()

Catch e As GRBException
End Try
End Sub
End Class

 feasopt_vb.vb

' Copyright 2020, Gurobi Optimization, LLC
'
' This example reads a MIP model from a file, adds artificial
' variables to each constraint, and then minimizes the sum of the
' artificial variables. A solution with objective zero corresponds
' to a feasible solution to the input model.
' We can also use FeasRelax feature to do it. In this example, we
' use minrelax=1, i.e. optimizing the returned model finds a solution
Imports Gurobi
Imports System

Class feasopt_vb
    Shared Sub Main(ByVal args As String())
        If args.Length < 1 Then
            Console.WriteLine("Usage: feasopt_vb filename")
            Return
        End If
        Try
            Dim env As New GRBEnv()
            Dim feasmodel As New GRBModel(env, args(0))

            ' Create a copy to use FeasRelax feature later
            Dim feasmodel1 As New GRBModel(feasmodel)

            ' Clear objective
            feasmodel.SetObjective(New GRBLinExpr())

            ' Add slack variables
            Dim c As GRBConstr() = feasmodel.GetConstrs()
            For i As Integer = 0 To c.Length - 1
                Dim sense As Char = c(i).Sense
                If sense <> “>”c Then
                    Dim constrs As GRBConstr() = New GRBConstr() {c(i)}
                    Dim coeffs As Double() = New Double() {-1}
                    feasmodel.AddVar(0.0, GRB.INFINITY, 1.0, GRB.CONTINUOUS, _
                                      constrs, coeffs, _
                                      “ArtN_” & c(i).ConstrName)
                End If
                If sense <> “<”c Then
                    Dim constrs As GRBConstr() = New GRBConstr() {c(i)}
                    Dim coeffs As Double() = New Double() {1}
                    feasmodel.AddVar(0.0, GRB.INFINITY, 1.0, GRB.CONTINUOUS, _
                                      constrs, coeffs, _
                                      “ArtP_” & c(i).ConstrName)
                End If
            Next

            ' Optimize modified model
            feasmodel.Optimize()
            feasmodel.Write("feasopt.lp")

            ' Use FeasRelax feature */
            feasmodel1.FeasRelax(GRB.FEASRELAX_LINEAR, true, false, true)
            feasmodel1.Write("feasopt1.lp")
            feasmodel1.Optimize()

            ' Dispose of model and env
            feasmodel1.Dispose()
            feasmodel.Dispose()
        End Try
    End Sub
End Class
env.Dispose()

Catch e As GRBException
End Try
End Sub
End Class

fixanddive_vb.vb

' Copyright 2020, Gurobi Optimization, LLC
'
' Implement a simple MIP heuristic. Relax the model,
' sort variables based on fractionality, and fix the 25% of
' the fractional variables that are closest to integer variables.
' Repeat until either the relaxation is integer feasible or
' linearly infeasible.

Imports System
Imports System.Collections.Generic
Imports Gurobi
Class fixanddive_vb
    ' Comparison class used to sort variable list based on relaxation
    ' fractionality
    Private Class FractionalCompare : Implements IComparer(Of GRBVar)
        Public Function Compare(ByVal v1 As GRBVar, ByVal v2 As GRBVar) As Integer
            Implements IComparer(Of Gurobi.GRBVar).Compare
            Try
                Dim sol1 As Double = Math.Abs(v1.X)
                Dim sol2 As Double = Math.Abs(v2.X)
                Dim frac1 As Double = Math.Abs(sol1 - Math.Floor(sol1 + 0.5))
                Dim frac2 As Double = Math.Abs(sol2 - Math.Floor(sol2 + 0.5))
                If frac1 < frac2 Then
                    Return -1
                ElseIf frac1 > frac2 Then
                    Return 1
                Else
                    Return 0
                End If
            Catch e As GRBException
            End Try
            Return 0
        End Function
    End Class
    Shared Sub Main(ByVal args As String())
        If args.Length < 1 Then
            Console.WriteLine("Usage: fixanddive_vb filename")
            Return
        End If
        Try
            ' Implementation...
        End Try
    End Sub
End Class
' Read model
Dim env As New GRBEnv()
Dim model As New GRBModel(env, args(0))

' Collect integer variables and relax them
Dim intvars As New List(Of GRBVar)()
For Each v As GRBVar In model.GetVars()
    If v.VType <> GRB.CONTINUOUS Then
        intvars.Add(v)
        v.VType = GRB.CONTINUOUS
    End If
Next

model.Parameters.OutputFlag = 0
model.Optimize()

' Perform multiple iterations. In each iteration, identify the first
' quartile of integer variables that are closest to an integer value
' in the relaxation, fix them to the nearest integer, and repeat.
For iter As Integer = 0 To 999
    ' create a list of fractional variables, sorted in order of
    ' increasing distance from the relaxation solution to the nearest
    ' integer value
    Dim fractional As New List(Of GRBVar)()
    For Each v As GRBVar In intvars
        Dim sol As Double = Math.Abs(v.X)
        If Math.Abs(sol - Math.Floor(sol + 0.5)) > 0.00001 Then
            fractional.Add(v)
        End If
    Next
    fractional.Sort(New FractionalCompare())
    Dim nfix As Integer = Math.Max(fractional.Count / 4, 1)
    For i As Integer = 0 To nfix - 1
        Dim v As GRBVar = fractional(i)
        Dim fixval As Double = Math.Floor(v.X + 0.5)
        v.LB = fixval
        v.UB = fixval
        Console.WriteLine(" Fix " & v.VarName & " to " & fixval & " ( rel " & v.X & ")")
    Next

    Console.WriteLine(" Iteration " & iter & ", obj " & _
    model.ObjVal & ", fractional " & fractional.Count)
    If fractional.Count = 0 Then
        Console.WriteLine(" Found feasible solution - objective " & _
        model.ObjVal)
        Exit For
    End If

    fractional.Sort(New FractionalCompare())
    Dim nfix As Integer = Math.Max(fractional.Count / 4, 1)
    For i As Integer = 0 To nfix - 1
        Dim v As GRBVar = fractional(i)
        Dim fixval As Double = Math.Floor(v.X + 0.5)
        v.LB = fixval
        v.UB = fixval
        Console.WriteLine(" Fix " & v.VarName & " to " & fixval & " ( rel " & v.X & ")")
    Next

393
model.Optimize()

' Check optimization result
If model.Status <> GRB.Status.OPTIMAL Then
    Console.WriteLine("Relaxation is infeasible")
    Exit For
End If
Next

' Dispose of model and env
model.Dispose()
env.Dispose()

Catch e As GRBException
End Try
End Sub
End Class
gc_pwl_vb.vb

' Copyright 2020, Gurobi Optimization, LLC
'
' This example formulates and solves the following simple model
' with PWL constraints:
'
' maximize
'     sum c[j] * x[j]
' subject to
'     sum A[i,j] * x[j] <= 0, for i = 0, ..., m-1
'     sum y[j] <= 3
'     y[j] = pwl(x[j]), for j = 0, ..., n-1
'     x[j] free, y[j] >= 0, for j = 0, ..., n-1
' where pwl(x) = 0, if x = 0
'     = 1+|x|, if x != 0
'
' Note
' 1. sum pwl(x[j]) <= b is to bound x vector and also to favor sparse x vector.
' Here b = 3 means that at most two x[j] can be nonzero and if two, then
' sum x[j] <= 1
' 2. pwl(x) jumps from 1 to 0 and from 0 to 1, if x moves from negative 0 to 0,
' then to positive 0, so we need three points at x = 0. x has infinite bounds
' on both sides, the piece defined with two points (-1, 2) and (0, 1) can
' extend x to -infinite. Overall we can use five points (-1, 2), (0, 1),
' (0, 0), (0, 1) and (1, 2) to define y = pwl(x)

Imports System
Imports Gurobi

Class gc_pwl_vb
    Shared Sub Main()
        Try
            Dim n As Integer = 5
            Dim m As Integer = 5
            Dim c As Double() = New Double() {0.5, 0.8, 0.5, 0.1, -1}

            model.Optimize()
            ' Check optimization result
            If model.Status <> GRB.Status.OPTIMAL Then
                Console.WriteLine("Relaxation is infeasible")
                Exit For
            End If
            Next
            ' Dispose of model and env
            model.Dispose()
            env.Dispose()

            Catch e As GRBException
            End Try
        End Sub
    End Class
Dim A As Double(,) = New Double(,) {{0, 0, 0, 1, -1}, _
    {0, 0, 1, 1, -1}, _
    {1, 1, 0, 0, -1}, _
    {1, 0, 1, 0, -1}, _
    {1, 0, 0, 1, -1}}

Dim xpts As Double() = New Double() {-1, 0, 0, 0, 1}
Dim ypts As Double() = New Double() {2, 1, 0, 1, 2}

' Env and model
Dim env As GRBEnv = New GRBEnv()
Dim model As GRBModel = New GRBModel(env)
model.ModelName = "gc_pwl_cs"

' Add variables, set bounds and obj coefficients
Dim x As GRBVar() = model.AddVars(n, GRB.CONTINUOUS)
For i As Integer = 0 To n - 1
    x(i).LB = -GRB.INFINITY
    x(i).Obj = c(i)
Next

Dim y As GRBVar() = model.AddVars(n, GRB.CONTINUOUS)

' Set objective to maximize
model.ModelSense = GRB.MAXIMIZE

' Add linear constraints
For i As Integer = 0 To m - 1
    Dim le As GRBLinExpr = 0.0
    For j As Integer = 0 To n - 1
        le.AddTerm(A(i, j), x(j))
    Next
    model.AddConstr(le, GRB.LESS_EQUAL, 0, "cx" & i)
Next

Dim le1 As GRBLinExpr = 0.0
For j As Integer = 0 To n - 1
    le1.AddTerm(1.0, y(j))
Next
model.AddConstr(le1, GRB.LESS_EQUAL, 3, "cy")

' Add piecewise constraints
For j As Integer = 0 To n - 1
    model.AddGenConstrPWL(x(j), y(j), xpts, ypts, "pwl" & j)
Next

' Optimize model
model.Optimize()

For j As Integer = 0 To n - 1
    Console.WriteLine("x[" & j & "] = " & x(j).X)
Next
Console.WriteLine("Obj: " & model.ObjVal)

' Dispose of model and environment
model.Dispose()
env.Dispose()
Try
Catch e As GRBException
End Try
End Sub

Class gc_pwl_func_vb

    ' Copyright 2020, Gurobi Optimization, LLC
    
    ' This example considers the following nonconvex nonlinear problem
    
    ' maximize 2 x + y
    ' subject to exp(x) + 4 sqrt(y) <= 9
    ' x, y >= 0
    
    ' We show you two approaches to solve this:
    
    ' 1) Use a piecewise-linear approach to handle general function
    '    constraints (such as exp and sqrt).
    '    a) Add two variables
    '    u = exp(x)
    '    v = sqrt(y)
    '    b) Compute points (x, u) of u = exp(x) for some step length (e.g., x
    '       = 0, 1e-3, 2e-3, ..., xmax) and points (y, v) of v = sqrt(y) for
    '       some step length (e.g., y = 0, 1e-3, 2e-3, ..., ymax). We need to
    '       compute xmax and ymax (which is easy for this example, but this
    '       does not hold in general).
    '    c) Use the points to add two general constraints of type
    '       piecewise-linear.
    
    ' 2) Use the Gurobi built-in general function constraints directly (EXP
    '    and POW). Here, we do not need to compute the points and the maximal
    '    possible values, which will be done internally by Gurobi. In this
    '    approach, we show how to "zoom in" on the optimal solution and
    '    tighten tolerances to improve the solution quality.

Imports System
Imports Gurobi

Class gc_pwl_func_vb

    Shared Function f(u As Double) As Double
        Return Math.Exp(u)
    End Function

    Shared Function g(u As Double) As Double
        Return Math.Sqrt(u)
    End Function

    Shared Sub printsol(m As GRBModel, x As GRBVar, _
        y As GRBVar, u As GRBVar, v As GRBVar)
        Console.WriteLine("x = " & x.X & ", y = " & y.X & ", u = " & u.X)
        Console.WriteLine("y = " & y.X & ", v = " & v.X)
        Console.WriteLine("Obj = " & m.ObjVal)
    End Sub
' Calculate violation of exp(x) + 4 sqrt(y) <= 9
Dim vio As Double = f(x.X) + 4 * g(y.X) - 9
If vio < 0.0 Then
    vio = 0.0
End If
Console.WriteLine("Vio = " & vio)
End Sub

Shared Sub Main()
    Try

    ' Create environment
    Dim env As New GRBEnv()

    ' Create a new m
    Dim m As New GRBModel(env)

    Dim lb As Double = 0.0
    Dim ub As Double = GRB.INFINITY

    Dim x As GRBVar = m.AddVar(lb, ub, 0.0, GRB.CONTINUOUS, "x")
    Dim y As GRBVar = m.AddVar(lb, ub, 0.0, GRB.CONTINUOUS, "y")
    Dim u As GRBVar = m.AddVar(lb, ub, 0.0, GRB.CONTINUOUS, "u")
    Dim v As GRBVar = m.AddVar(lb, ub, 0.0, GRB.CONTINUOUS, "v")

    ' Set objective
    m.SetObjective(2*x + y, GRB.MAXIMIZE)

    ' Add linear constraint
    m.AddConstr(u + 4*v <= 9, "l1")

    ' PWL constraint approach
    Dim intv As Double = 1e-3
    Dim xmax As Double = Math.Log(9.0)
    Dim npts As Integer = Math.Ceiling(xmax/intv) + 1
    Dim xpts As Double() = new Double(npts -1) {}
    Dim upts As Double() = new Double(npts -1) {}
    For i As Integer = 0 To npts - 1
        xpts(i) = i*intv
        upts(i) = f(i*intv)
    Next

    Dim gc1 As GRBGenConstr = m.AddGenConstrPWL(x, u, xpts, upts, "gc1")

    Dim ymax As Double = (9.0/4.0)*(9.0/4.0)
    npts = Math.Ceiling(ymax/intv) + 1
    Dim ypts As Double() = new Double(npts -1) {}
    Dim vpts As Double() = new Double(npts -1) {}
    For i As Integer = 0 To npts - 1

ypts(i) = i*intv
vpts(i) = g(i*intv)

Next

Dim gc2 As GRBGenConstr = m.AddGenConstrPWL(y, v, ypts, vpts, "gc2")

' Optimize the model and print solution
m.Optimize()
printsol(m, x, y, u, v)

' General function approach with auto PWL translation by Gurobi
m.Reset()
m.Remove(gc1)
m.Remove(gc2)
m.Update()

Dim gcf1 As GRBGenConstr = m.AddGenConstrExp(x, u, "gcf1", "")
Dim gcf2 As GRBGenConstr = m.AddGenConstrPow(y, v, 0.5, "gcf2", "")

m.Parameters.FuncPieceLength = 1e-3

' Optimize the model and print solution
m.Optimize()
printsol(m, x, y, u, v)

' Use optimal solution to reduce the ranges and use smaller pclen to solve
x.LB = Math.Max(x.LB, x.X-0.01)
x.UB = Math.Min(x.UB, x.X+0.01)
y.LB = Math.Max(y.LB, y.X-0.01)
y.UB = Math.Min(y.UB, y.X+0.01)
m.Update()
m.Reset()

m.Parameters.FuncPieceLength = 1e-5

' Optimize the model and print solution
m.Optimize()
printsol(m, x, y, u, v)

' Dispose of model and environment
m.Dispose()
env.Dispose()
Catch e As GRBException
End Try
End Sub
End Class

genconstr_vb.vb
In this example we show the use of general constraints for modeling some common expressions. We use as an example a SAT-problem where we want to see if it is possible to satisfy at least four (or all) clauses of the logical for:

\[ L = (x_0 \text{ or } \neg x_1 \text{ or } x_2) \text{ and } (x_1 \text{ or } \neg x_2 \text{ or } x_3) \text{ and } (-x_0 \text{ or } \neg x_1 \text{ or } x_2) \text{ and } (-x_1 \text{ or } \neg x_2 \text{ or } x_3) \text{ and } (-x_2 \text{ or } \neg x_3 \text{ or } x_0) \text{ and } (-x_3 \text{ or } \neg x_0 \text{ or } x_1) \]

We do this by introducing two variables for each literal (itself and its negated value), a variable for each clause, and then two variables for indicating if we can satisfy four, and another to identify the minimum of the clauses (so if it one, we can satisfy all clauses) and put these two variables in the objective. i.e. the Objective function will be:

maximize Obj0 + Obj1

\[ \text{Obj0} = \text{MIN(Clause}1, \ldots, \text{Clause}8) \]
\[ \text{Obj1} = 1 \Rightarrow \text{Clause}1 + \ldots + \text{Clause}8 \geq 4 \]

thus, the objective value will be two if and only if we can satisfy all clauses; one if and only of at least four clauses can be satisfied, and zero otherwise.

Imports Gurobi

Class genconstr_vb

Public Const n As Integer = 4
Public Const NLITERALS As Integer = 4 'same as n
Public Const NCLAUSES As Integer = 8
Public Const NOBJ As Integer = 2

Shared Sub Main()

Try

' Example data:
' e.g. {0, n+1, 2} means clause (x0 or \neg x1 or x2)
Dim Clauses As Integer(,) = New Integer(,) { _
  { 0, n + 1, 2}, { 1, n + 2, 3}, _,
  { 2, n + 3, 0}, { 3, n + 0, 1}, _,
  {n + 0, n + 1, 2}, {n + 1, n + 2, 3}, _,
  {n + 2, n + 3, 0}, {n + 3, n + 0, 1}}

Dim i As Integer, status As Integer

' Create environment
Dim env As New GRBEnv("genconstr_vb.log")


' Create initial model
Dim model As New GRBModel(env)
model.ModelName = "genconstr_vb"

' Initialize decision variables and objective
Dim Lit As GRBVar() = New GRBVar(NLITERALS - 1) {}
Dim NotLit As GRBVar() = New GRBVar(NLITERALS - 1) {}
For i = 0 To NLITERALS - 1
    Lit(i) = model.AddVar(0.0, 1.0, 0.0, GRB.BINARY, String.Format("X{0}", i))
    NotLit(i) = model.AddVar(0.0, 1.0, 0.0, GRB.BINARY, String.Format("notX{0}", i))
Next

Dim Cla As GRBVar() = New GRBVar(NCLAUSES - 1) {}
For i = 0 To NCLAUSES - 1
    Cla(i) = model.AddVar(0.0, 1.0, 0.0, GRB.BINARY, String.Format("Clause{0}", i))
Next

Dim Obj As GRBVar() = New GRBVar(NOBJ - 1) {}
For i = 0 To NOBJ - 1
    Obj(i) = model.AddVar(0.0, 1.0, 1.0, GRB.BINARY, String.Format("Obj{0}", i))
Next

' Link Xi and notXi
Dim lhs As GRBLinExpr
For i = 0 To NLITERALS - 1
    lhs = New GRBLinExpr()
    lhs.AddTerm(1.0, Lit(i))
    lhs.AddTerm(1.0, NotLit(i))
    model.AddConstr(lhs, GRB.EQUAL, 1.0, String.Format("CNSTR_X{0}", i))
Next

' Link clauses and literals
For i = 0 To NCLAUSES - 1
    Dim clause As GRBVar() = New GRBVar(2) {}
    For j As Integer = 0 To 2
        If Clauses(i, j) >= n Then
            clause(j) = NotLit(Clauses(i, j) - n)
        Else
            clause(j) = Lit(Clauses(i, j))
        End If
    Next
    model.AddGenConstrOr(Cla(i), clause, String.Format("CNSTR_Clause{0}", i))
Next

' Link objs with clauses
model.AddGenConstrMin(Obj(0), Cla, GRB.INFINITY, "CNSTR_Obj0")
lhs = New GRBLinExpr()
For i = 0 To NCLAUSES - 1
    lhs.AddTerm(1.0, Cla(i))
Next
model.AddGenConstrIndicator(Obj(1), 1, lhs, GRB.GREATER_EQUAL, 4.0, "CNSTR_Obj1")

' Set global objective sense
model.ModelSense = GRB.MAXIMIZE

' Save problem
model.Write("genconstr_vb.mps")
model.Write("genconstr_vb.lp")

' Optimize
model.Optimize()

' Status checking
status = model.Status

If status = GRB.Status.INF_OR_UNBD OrElse _
    status = GRB.Status.INFEASIBLE OrElse _
    status = GRB.Status.UNBOUNDED Then
    Console.WriteLine("The model cannot be solved " & _
        "because it is infeasible or unbounded")
    Return
End If

If status <> GRB.Status.OPTIMAL Then
    Console.WriteLine("Optimization was stopped with status {0}", status)
    Return
End If

' Print result
Dim objval As Double = model.ObjVal

If objval > 1.9 Then
    Console.WriteLine("Logical expression is satisfiable")
ElseIf objval > 0.9 Then
    Console.WriteLine("At least four clauses can be satisfied")
Else
    Console.WriteLine("Not even three clauses can be satisfied")
End If

' Dispose of model and environment
model.Dispose()
env.Dispose()

Catch e As GRBException
    Console.WriteLine("Error code: {0}. {1}", e.ErrorCode, e.Message)
End Try

End Sub

End Class

lp_vb.vb

' Copyright 2020, Gurobi Optimization, LLC
'
' This example reads an LP model from a file and solves it.
' If the model is infeasible or unbounded, the example turns off
' presolve and solves the model again. If the model is infeasible,
' the example computes an Irreducible Inconsistent Subsystem (IIS),
' and writes it to a file.

Imports System
Imports Gurobi
Class lp_vb
    Shared Sub Main(ByVal args As String())

        If args.Length < 1 Then
            Console.WriteLine("Usage: lp_vb filename")
            Return
        End If

        Try
            Dim env As GRBEnv = New GRBEnv("lp1.log")
            Dim model As GRBModel = New GRBModel(env, args(0))

            model.Optimize()

            Dim optimstatus As Integer = model.Status

            If optimstatus = GRB.Status.INF_OR_UNBD Then
                model.Parameters.Presolve = 0
                model.Optimize()
                optimstatus = model.Status
            End If

            If optimstatus = GRB.Status.OPTIMAL Then
                Dim objval As Double = model.ObjVal
                Console.WriteLine("Optimal objective: " & objval)
            ElseIf optimstatus = GRB.Status.INFEASIBLE Then
                Console.WriteLine("Model is infeasible")
                model.ComputeIIS()
                model.Write("model.ilp")
            ElseIf optimstatus = GRB.Status.UNBOUNDED Then
                Console.WriteLine("Model is unbounded")
            Else
                Console.WriteLine("Optimization was stopped with status = " & _
                    optimstatus)
            End If

            ' Dispose of model and env
            model.Dispose()
            env.Dispose()
        Catch e As GRBException
        End Try
    End Sub
End Class

lpmethod_vb.vb

' Copyright 2020, Gurobi Optimization, LLC
'
' Solve a model with different values of the Method parameter;
' show which value gives the shortest solve time.

Imports System
Imports Gurobi
Class lpmethod_vb

Shared Sub Main(ByVal args As String())

    If args.Length < 1 Then
        Console.WriteLine("Usage: lpmethod_vb filename")
        Return
    End If

    Try

        ' Read model and verify that it is a MIP
        Dim env As New GRBEnv()
        Dim model As New GRBModel(env, args(0))

        ' Solve the model with different values of Method
        Dim bestMethod As Integer = -1
        Dim bestTime As Double = model.get(GRB.DoubleParam.TimeLimit)
        For i As Integer = 0 To 2
            model.Reset()
            model.Parameters.Method = i
            model.Optimize()
            If model.Status = GRB.Status.OPTIMAL Then
                bestTime = model.Runtime
                bestMethod = i
                ' Reduce the TimeLimit parameter to save time
                ' with other methods
                model.Parameters.TimeLimit = bestTime
            End If
        Next

        ' Report which method was fastest
        If bestMethod = -1 Then
            Console.WriteLine("Unable to solve this model")
        Else
        End If

        ' Dispose of model and env
        model.Dispose()
        env.Dispose()

    Catch e As GRBException
    End Try

    End Sub
End Class

lpmod_vb.vb

' Copyright 2020, Gurobi Optimization, LLC
'
' This example reads an LP model from a file and solves it.
' If the model can be solved, then it finds the smallest positive variable,
' sets its upper bound to zero, and resolves the model two ways:
' first with an advanced start, then without an advanced start
' (i.e. from scratch).

Imports System
Imports Gurobi

Class lpmod_vb
    Shared Sub Main(ByVal args As String())
        If args.Length < 1 Then
            Console.WriteLine("Usage: lpmod_vb filename")
            Return
        End If
        Try
            ' Read model and determine whether it is an LP
            Dim env As New GRBEnv()
            Dim model As New GRBModel(env, args(0))
            If model.IsMIP <> 0 Then
                Console.WriteLine("The model is not a linear program")
                Environment.Exit(1)
            End If
            model.Optimize()

            Dim status As Integer = model.Status
            If (status = GRB.Status.INF_OR_UNBD) OrElse _
                (status = GRB.Status.INFEASIBLE) OrElse _
                (status = GRB.Status.UNBOUNDED) Then
                Console.WriteLine("The model cannot be solved because it is " & _
                "infeasible or unbounded")
                Environment.Exit(1)
            End If

            If status <> GRB.Status.OPTIMAL Then
                Console.WriteLine("Optimization was stopped with status " & status)
                Environment.Exit(0)
            End If

            ' Find the smallest variable value
            Dim minVal As Double = GRB.INFINITY
            Dim minVar As GRBVar = Nothing
            For Each v As GRBVar In model.GetVars()
                Dim sol As Double = v.X
                If (sol > 0.0001) AndAlso _
                    (sol < minVal) AndAlso _
                    (v.LB = 0.0) Then
                    minVal = sol
                    minVar = v
            Next
            Console.WriteLine(vbCrLf & "*** Setting " & _
                minVar.VarName & " from " & minVal & " to zero ***" & vbCrLf)
            minVar.UB = 0
        End Try
    End Sub
End Class
' Solve from this starting point
model.Optimize()

' Save iteration & time info
Dim warmCount As Double = model.IterCount
Dim warmTime As Double = model.Runtime

' Reset the model and resolve
Console.WriteLine(vbLf & " *** Resetting and solving " & _
    "without an advanced start *** " & vbCrLf)
model.Reset()
model.Optimize()

Dim coldCount As Double = model.IterCount
Dim coldTime As Double = model.Runtime

Console.WriteLine(vbLf & " *** Warm start : " & warmCount & _
    " iterations , " & warmTime & " seconds")
Console.WriteLine(" *** Cold start : " & coldCount & " iterations , " & _
    coldTime & " seconds")

' Dispose of model and env
model.Dispose()
env.Dispose()

Catch e As GRBException
End Try
End Sub

End Class

mip1_vb.vb

' Copyright 2020, Gurobi Optimization, LLC
',
' This exampleformulates and solves the following simple MIP model:
',
' maximize x + y + 2 z
',
' subject to x + 2 y + 3 z <= 4
',
' x + y >= 1
',
' x, y, z binary

Imports System
Imports Gurobi

Class mip1_vb
    Shared Sub Main()
        Try
            Dim env As GRBEnv = New GRBEnv("mip1.log")
            Dim model As GRBModel = New GRBModel(env)

            ' Create variables
            Dim x As GRBVar = model.AddVar(0.0, 1.0, 0.0, GRB.BINARY, "x")
        End Try
    End Sub
End Class
Dim y As GRBVar = model.AddVar(0.0, 1.0, 0.0, GRB.BINARY, "y")
Dim z As GRBVar = model.AddVar(0.0, 1.0, 0.0, GRB.BINARY, "z")

' Set objective: maximize x + y + 2 z
model.SetObjective(x + y + 2 * z, GRB.MAXIMIZE)

' Add constraint: x + 2 y + 3 z <= 4
model.AddConstr(x + 2 * y + 3 * z <= 4.0, "c0")

' Add constraint: x + y >= 1
model.AddConstr(x + 2 * y + 3 * z <= 4.0, "c1")

' Optimize model
model.Optimize()

Console.WriteLine(x.VarName & " " & x.X)
Console.WriteLine(y.VarName & " " & y.X)
Console.WriteLine(z.VarName & " " & z.X)

Console.WriteLine("Obj: " & model.ObjVal)

' Dispose of model and env
model.Dispose()
env.Dispose()

Catch e As GRBException
End Try
End Sub

Class mip2_vb

' Copyright 2020, Gurobi Optimization, LLC
'
' This example reads a MIP model from a file, solves it and
' prints the objective values from all feasible solutions
' generated while solving the MIP. Then it creates the fixed
' model and solves that model.

Imports System
Imports Gurobi

Class mip2_vb
    Shared Sub Main(ByVal args As String())
        If args.Length < 1 Then
            Console.WriteLine("Usage: mip2_vb filename")
            Return
        End If
    End Sub
End Class
Try

```vba
Dim env As GRBEnv = New GRBEnv("lp1.log")
Dim model As GRBModel = New GRBModel(env, args(0))

If model.IsMIP = 0 Then
    Console.WriteLine("Model is not a MIP")
    Return
End If

model.Optimize()

Dim optimstatus As Integer = model.Status

If optimstatus = GRB.Status.INF_OR_UNBD Then
    model.Parameters.Presolve = 0
    model.Optimize()
    optimstatus = model.Status
End If

Dim objval As Double

If optimstatus = GRB.Status.OPTIMAL Then
    objval = model.ObjVal
    Console.WriteLine("Optimal objective: " & objval)
ElseIf optimstatus = GRB.Status.INFEASIBLE Then
    Console.WriteLine("Model is infeasible")
    model.ComputeIIS()
    model.Write("model.ilp")
    Return
ElseIf optimstatus = GRB.Status.UNBOUNDED Then
    Console.WriteLine("Model is unbounded")
    Return
Else
    Console.WriteLine("Optimization was stopped with status = " & _
                   optimstatus)
    Return
End If

' Iterate over the solutions and compute the objectives
Dim vars() As GRBVar = model.GetVars()
model.Parameters.OutputFlag = 0

Console.WriteLine()
For k As Integer = 0 To model.SolCount - 1
    model.Parameters.SolutionNumber = k
    Dim objn As Double = 0.0
    For j As Integer = 0 To vars.Length - 1
        objn += vars(j).Obj * vars(j).Xn
    Next
    Console.WriteLine("Solution " & k & " has objective: " & objn)
Next
Console.WriteLine()
model.Parameters.OutputFlag = 1
```

407
' Solve fixed model
Dim fixedmodel As GRBModel = model.FixedModel()
fixedmodel.Parameters.Presolve = 0
fixedmodel.Optimize()

Dim foptimstatus As Integer = fixedmodel.Status
If foptimstatus <> GRB.Status.OPTIMAL Then
   _console.WriteLine("Error: fixed model isn’t optimal")
    Return
End If

Dim fobjval As Double = fixedmodel.ObjVal
If Math.Abs(fobjval - objval) > 0.000001 * (1.0 + Math.Abs(objval)) Then
End If

Dim fvars() As GRBVar = fixedmodel.GetVars()
Dim x() As Double = fixedmodel.Get(GRB.DoubleAttr.X, fvars)
Dim vnames() As String = fixedmodel.Get(GRB.StringAttr.VarName, fvars)

For j As Integer = 0 To fvars.Length - 1
    If x(j) <> 0 Then
        Console.WriteLine(vnames(j) & " " & x(j))
    End If
Next

' Dispose of models and env
fixedmodel.Dispose()
model.Dispose()
env.Dispose()

Catch e As GRBException
End Try
End Sub

End Class

multiobj_vb.vb

' Copyright 2020, Gurobi Optimization, LLC

' Want to cover three different sets but subject to a common budget of
' elements allowed to be used. However, the sets have different priorities to
' be covered; and we tackle this by using multi-objective optimization.

Imports Gurobi

Class multiobj_vb

    Shared Sub Main()

        Try

            ' Sample data
            Dim groundSetSize As Integer = 20

        End Try

    End Sub

End Class
Dim nSubsets As Integer = 4
Dim Budget As Integer = 12
Dim [Set] As Double(,) = New Double(,) {
_    {1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0}, _
_    {0, 0, 0, 0, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 1}, _
_    {0, 0, 0, 1, 1, 0, 1, 1, 0, 0, 0, 0, 1, 0, 1, 1, 0, 1, 0}, _
_    {0, 0, 0, 1, 1, 1, 0, 0, 0, 1, 1, 1, 0, 0, 0, 1, 1, 1, 0, 0}}
Dim SetObjPriority As Integer() = New Integer() {3, 2, 2, 1}
Dim SetObjWeight As Double() = New Double() {1.0, 0.25, 1.25, 1.0}
Dim e As Integer, i As Integer, status As Integer, nSolutions As Integer
' Create environment
Dim env As New GRBEnv("multiobj_vb.log")
' Create initial model
Dim model As New GRBModel(env)
model.ModelName = "multiobj_vb"
' Initialize decision variables for ground set:
' x[e] == 1 if element e is chosen for the covering.
Dim Elem As GRBVar() = model.AddVars(groundSetSize, GRB.BINARY)
For e = 0 To groundSetSize - 1
    Dim vname As String = "El" & e.ToString()
    Elem(e).VarName = vname
    model.AddConstr(model.AddVar(1.0, Elem(e)) , GRB.LESS_EQUAL , Budget , "Budget")
Next
' Constraint: limit total number of elements to be picked to be at most
' Budget
Dim lhs As New GRBLinExpr()
For e = 0 To groundSetSize - 1
    lhs.AddTerm(1.0, Elem(e))
Next
model.SetObjectiveN(lhs , 1, 0.01, vname )
model.ModelSense = GRB.MAXIMIZE
model.Parameters.PoolSolutions = 100
' Set and configure i-th objective
For i = 0 To nSubsets - 1
    Dim objn As New GRBLinExpr()
    For e = 0 To groundSetSize - 1
        objn.AddTerm([Set](i, e), Elem(e))
    Next
    model.SetObjectiveN(objn , i, SetObjPriority(i), SetObjWeight(i), _
        1.0 + i, 0.01, vname)
Next
' Save problem
model.Write("multiobj_vb.lp")
' Optimize
model.Optimize()

' Status checking
status = model.Status

If status = GRB.Status.INF_OR_UNBD OrElse _
    status = GRB.Status.INFEASIBLE OrElse _
    status = GRB.Status.UNBOUNDED Then
    Console.WriteLine("The model cannot be solved " & _
        "because it is infeasible or unbounded")
    Return
End If
If status <> GRB.Status.OPTIMAL Then
    Console.WriteLine("Optimization was stopped with status {0}", status)
    Return
End If

' Print best selected set
Console.WriteLine("Selected elements in best solution:")
Console.Write(vbTab)
For e = 0 To groundSetSize - 1
    If Elem(e).X < 0.9 Then
        Continue For
    End If
    Console.Write("E{l}{0} ", e)
Next
Console.WriteLine()

' Print number of solutions stored
nSolutions = model.SolCount
Console.WriteLine("Number of solutions found: {0}", nSolutions)

' Print objective values of solutions
If nSolutions > 10 Then
    nSolutions = 10
End If
Console.WriteLine("Objective values for first {0} solutions:", nSolutions)
For i = 0 To nSubsets - 1
    model.Parameters.ObjNumber = i
    Console.Write(vbTab & "Set" & i)
    For e = 0 To nSolutions - 1
        model.Parameters.SolutionNumber = e
        Console.Write("{0,8}", model.ObjNVal)
    Next
    Console.WriteLine()
Next
Console.WriteLine()
model.Dispose()
env.Dispose()

Catch e As GRBException
    Console.WriteLine("Error code = {0}", e)
    Console.WriteLine(e.Message)
End Try
End Sub
End Class

multiscenario_vb.vb

' Copyright 2020, Gurobi Optimization, LLC

' Facility location: a company currently ships its product from 5 plants
to 4 warehouses. It is considering closing some plants to reduce
costs. What plant(s) should the company close, in order to minimize
transportation and fixed costs?
'
' Since the plant fixed costs and the warehouse demands are uncertain, a
scenario approach is chosen.
'
' Note that this example is similar to the facility_vb.vb example. Here we
added scenarios in order to illustrate the multi-scenario feature.
'
' Based on an example from Frontline Systems:
http://www.solver.com/disfacility.htm
' Used with permission.

Imports System
Imports Gurobi

Class multiscenario_vb
    Shared Sub Main()
        Try

' Warehouse demand in thousands of units
        Dim Demand As Double() = New Double() {15, 18, 14, 20}

' Plant capacity in thousands of units
        Dim Capacity As Double() = New Double() {20, 22, 17, 19, 18}

' Fixed costs for each plant
        Dim FixedCosts As Double() = New Double() {12000, 15000, 17000, 13000, 16000}

' Transportation costs per thousand units
        Dim TransCosts As Double(,) = New Double(,) {{4000, 2000, 3000, 2500, 4500}, _
          {2500, 2600, 3400, 3000, 4000}, _
          {1200, 1800, 2600, 4100, 3000}, _
          {2200, 2600, 3100, 3700, 3200}}

' Number of plants and warehouses
        Dim nPlants As Integer = Capacity.Length
        Dim nWarehouses As Integer = Demand.Length

        Dim maxFixed As Double = -GRB.INFINITY
        Dim minFixed As Double = GRB.INFINITY
        For p As Integer = 0 To nPlants - 1
            If FixedCosts(p) > maxFixed Then maxFixed = FixedCosts(p)
            If FixedCosts(p) < minFixed Then minFixed = FixedCosts(p)
        Next

    End Sub
End Class
' Model
Dim env As GRBEnv = New GRBEnv()
Dim model As GRBModel = New GRBModel(env)

model.ModelName = "multiscenario"

' Plant open decision variables: open(p) == 1 if plant p is open.
Dim open As GRBVar() = New GRBVar(nPlants - 1) {}
For p As Integer = 0 To nPlants - 1
   open(p) = model.AddVar(0, 1, FixedCosts(p), GRB.BINARY, "Open" & p)
Next

' Transportation decision variables: how much to transport from a plant ' p to a warehouse w
Dim transport As GRBVar(,) = New GRBVar(nWarehouses - 1, nPlants - 1) {}
For w As Integer = 0 To nWarehouses - 1
   For p As Integer = 0 To nPlants - 1
      transport(w, p) = model.AddVar(0, GRB.INFINITY, TransCosts(w, p), _
         GRB.CONTINUOUS, "Trans" & p & "." & w)
   Next
Next

' The objective is to minimize the total fixed and variable costs
model.ModelSense = GRB.MINIMIZE

' Production constraints
' Note that the right-hand limit sets the production to zero if ' the plant is closed
For p As Integer = 0 To nPlants - 1
   Dim ptot As GRBLinExpr = 0.0
   For w As Integer = 0 To n Warehouses - 1
      ptot.AddTerm(1.0, transport(w, p))
   Next
   model.AddConstr(ptot <= Capacity(p) * open(p), "Capacity" & p)
Next

' Demand constraints
Dim demandConstr As GRBConstr() = New GRBConstr(nWarehouses - 1) {}
For w As Integer = 0 To nWarehouses - 1
   Dim dtot As GRBLinExpr = 0.0
   For p As Integer = 0 To nPlants - 1
      dtot.AddTerm(1.0, transport(w, p))
   Next
   demandConstr(w) = model.AddConstr(dtot = Demand(w), "Demand" & w)
Next

' We constructed the base model, now we add 7 scenarios
'
' Scenario 0: Represents the base model, hence, no manipulations.
' Scenario 1: Manipulate the warehouses demands slightly (constraint right ' hand sides).
Scenario 2: Double the warehouses demands (constraint right hand sides).
Scenario 3: Manipulate the plant fixed costs (objective coefficients).
Scenario 4: Manipulate the warehouses demands and fixed costs.
Scenario 5: Force the plant with the largest fixed cost to stay open
( variable bounds ).
Scenario 6: Force the plant with the smallest fixed cost to be closed
( variable bounds ).

model.NumScenarios = 7

Scenario 0: Base model, hence, nothing to do except giving the
scenario a name
model.Parameters.ScenarioNumber = 0
model.ScenNName = "Base model"

Scenario 1: Increase the warehouse demands by 10%
model.Parameters.ScenarioNumber = 1
model.ScenNName = "Increased warehouse demands"

For w As Integer = 0 To nWarehouses - 1
    demandConstr(w).ScenNRHS = Demand(w) * 1.1
Next

Scenario 2: Double the warehouse demands
model.Parameters.ScenarioNumber = 2
model.ScenNName = "Double the warehouse demands"

For w As Integer = 0 To nWarehouses - 1
    demandConstr(w).ScenNRHS = Demand(w) * 2.0
Next

Scenario 3: Decrease the plant fixed costs by 5%
model.Parameters.ScenarioNumber = 3
model.ScenNName = "Decreased plant fixed costs"

For p As Integer = 0 To nPlants - 1
    open(p).ScenNObj = FixedCosts(p) * 0.95
Next

Scenario 4: Combine scenario 1 and scenario 3
model.Parameters.ScenarioNumber = 4
model.ScenNName = "Increased warehouse demands and decreased plant fixed costs"

For w As Integer = 0 To nWarehouses - 1
    demandConstr(w).ScenNRHS = Demand(w) * 1.1
Next

For p As Integer = 0 To nPlants - 1
    open(p).ScenNObj = FixedCosts(p) * 0.95
Next

Scenario 5: Force the plant with the largest fixed cost to stay open
model.Parameters.ScenarioNumber = 5
model.ScenNName = "Force plant with largest fixed cost to stay open"
For p As Integer = 0 To nPlants - 1
    If FixedCosts(p) = maxFixed Then
        open(p).ScenNLB = 1.0
        Exit For
    End If
Next

' Scenario 6: Force the plant with the smallest fixed cost to be closed
model.Parameters.ScenarioNumber = 6
model.ScenNName = "Force plant with smallest fixed cost to be closed"

For p As Integer = 0 To nPlants - 1
    If FixedCosts(p) = minFixed Then
        open(p).ScenNUB = 0.0
        Exit For
    End If
Next

' Guess at the starting point: close the plant with the highest fixed costs; open all others

' First, open all plants
For p As Integer = 0 To nPlants - 1
    open(p).Start = 1.0
Next

' Now close the plant with the highest fixed cost
Console.WriteLine("Initial guess:")
For p As Integer = 0 To nPlants - 1
    If FixedCosts(p) = maxFixed Then
        open(p).Start = 0.0
        Console.WriteLine("Closing plant " & p & vbCrLf)
        Exit For
    End If
Next

' Use barrier to solve root relaxation
model.Parameters.Method = GRB.METHOD_BARRIER

' Solve multi-scenario model
model.Optimize()
Dim nScenarios As Integer = model.NumScenarios

For s As Integer = 0 To nScenarios - 1
    Dim modelSense As Integer = GRB.MINIMIZE

    ' Set the scenario number to query the information for this scenario
    model.Parameters.ScenarioNumber = s

    ' collect result for the scenario
    Dim scenNObjBound As Double = model.ScenNObjBound
    Dim scenNObjVal As Double = model.ScenNObjVal
' Check if we found a feasible solution for this scenario
If scenNObjVal >= modelSense * GRB.INFINITY Then
  If scenNObjBound >= modelSense * GRB.INFINITY Then
    ' Scenario was proven to be infeasible
    Console.WriteLine(vbLf & "INFEASIBLE")
  Else
    ' We did not find any feasible solution - should not happen in
    ' this case, because we did not set any limit (like a time
    ' limit) on the optimization process
    Console.WriteLine(vbLf & "NO SOLUTION")
  End If
Else
  Console.WriteLine(vbLf & "TOTAL COSTS: " & scenNObjVal)
End If

For p As Integer = 0 To nPlants - 1
  Dim scenNX As Double = open(p).ScenNX
  If scenNX > 0.5 Then
    Console.WriteLine("Plant " & p & " open")
    For w As Integer = 0 To nWarehouses - 1
      scenNX = transport(w, p).ScenNX
      If scenNX > 0.0001 Then Console.WriteLine(" Transport " & scenNX & " units to ")
    Next
  Else
    Console.WriteLine(" Plant " & p & " closed!")
  End If
Next

' Print a summary table: for each scenario we add a single summary line
Console.WriteLine(vbLf & vbLf & "Summary: Closed plants depending on scenario" & vbCrLf)
Console.WriteLine("{0,8} | {1,17} {2,13} ", "|")
Console.Write("{0,8} |", "Scenario")
For p As Integer = 0 To nPlants - 1
  Console.Write("{0,6}"", p)
Next
Console.WriteLine(" | {0,6} Name", "Costs")

For s As Integer = 0 To nScenarios - 1
  Dim modelSense As Integer = GRB.MINIMIZE
  Set the scenario number to query the information for this scenario
  model.Parameters.ScenarioNumber = s
  Collect result for the scenario
  Dim scenNObjBound As Double = model.ScenNObjBound
  Dim scenNObjVal As Double = model.ScenNObjVal
Check if a feasible solution was found for this scenario
If scenNObjVal >= modelSense * GRB.INFINITY Then
    If scenNObjBound >= modelSense * GRB.INFINITY Then
        Scenario was proven to be infeasible
        Console.WriteLine("{0,-30}| {1,6} " & model.ScenNName, "infeasible", ",-"
    Else
        We did not find any feasible solution - should not happen in
        this case, because we did not set any limit (like a Time
        limit) on the optimization process
        Console.WriteLine("{0,-30}| {1,6} " & model.ScenNName, "no solution found", ",-")
    End If
Else
    For p As Integer = 0 To nPlants - 1
        Dim scenNX As Double = open(p).ScenNX
        If scenNX > 0.5 Then
            Console.Write("{0,6} ", "x")
        Else
            Console.Write("{0,6} ", " ")
        End If
    Next
    Console.WriteLine(" | {0,6} ", model.ScenNName, scenNObjVal)
End If
Next
model.Dispose()
env.Dispose()
Catch e As GRBException
End Try
End Sub
End Class

params_vb.vb

' Copyright 2020, Gurobi Optimization, LLC */

' Use parameters that are associated with a model.
' A MIP is solved for a few seconds with different sets of parameters.
' The one with the smallest MIP gap is selected, and the optimization
' is resumed until the optimal solution is found.

Imports System
Imports Gurobi

Class params_vb
    Shared Sub Main(args As String())
        If args.Length < 1 Then
            Console.Out.WriteLine("Usage: params_vb filename")
            Return
        End If
End Sub

416
Try
' Read model and verify that it is a MIP
Dim env As New GRBEnv()
Dim m As New GRBModel(env, args(0))
If m.IsMIP = 0 Then
    Console.WriteLine("The model is not an integer program")
    Environment.Exit(1)
End If

' Set a 2 second time limit
m.Parameters.TimeLimit = 2.0

' Now solve the model with different values of MIPFocus
Dim bestModel As New GRBModel(m)
bestModel.Optimize()
For i As Integer = 1 To 3
    m.Reset()
    m.Parameters.MIPFocus = i
    m.Optimize()
    If bestModel.MIPGap > m.MIPGap Then
        Dim swap As GRBModel = bestModel
        bestModel = m
        m = swap
    End If
Next

' Finally, delete the extra model, reset the time limit and
' continue to solve the best model to optimality
m.Dispose()
bestModel.Parameters.TimeLimit = GRB.INFINITY
bestModel.Optimize()
Console.WriteLine("Solved with MIPFocus: " & bestModel.Parameters.MIPFocus)
Catch e As GRBException
End Try
End Sub
End Class

piecewise_vb.vb

' Copyright 2020, Gurobi Optimization, LLC

' This example considers the following separable, convex problem:
'
' minimize  f(x) - y + g(z)
' subject to  x + 2 y + 3 z <= 4
'       x, y, z <= 1
'
' where f(u) = exp(-u) and g(u) = 2 u^2 - 4 u, for all real u. It
' formulates and solves a simpler LP model by approximating f and
' g with piecewise-linear functions. Then it transforms the model
' into a MIP by negating the approximation for f, which corresponds
' to a non-convex piecewise-linear function, and solves it again.
Imports System
Imports Gurobi

Class piecewise_vb
    Shared Function f(u As Double) As Double
        Return Math.Exp(-u)
    End Function

    Shared Function g(u As Double) As Double
        Return 2 * u * u - 4 * u
    End Function

    Shared Sub Main()
        Try
            ' Create environment
            Dim env As New GRBEnv()

            ' Create a new model
            Dim model As New GRBModel(env)

            ' Create variables
            Dim lb As Double = 0.0, ub As Double = 1.0

            Dim x As GRBVar = model.AddVar(lb, ub, 0.0, GRB.CONTINUOUS, "x")
            Dim y As GRBVar = model.AddVar(lb, ub, 0.0, GRB.CONTINUOUS, "y")
            Dim z As GRBVar = model.AddVar(lb, ub, 0.0, GRB.CONTINUOUS, "z")

            ' Set objective for y
            model.SetObjective(-y)

            ' Add piecewise-linear objective functions for x and z
            Dim npts As Integer = 101
            Dim ptu As Double() = New Double(npts - 1) {}
            Dim ptf As Double() = New Double(npts - 1) {}
            Dim ptg As Double() = New Double(npts - 1) {}

            For i As Integer = 0 To npts - 1
                ptu(i) = lb + (ub - lb) * i / (npts - 1)
                ptf(i) = f(ptu(i))
                ptg(i) = g(ptu(i))
            Next

            model.SetPWLObj(x, ptu, ptf)
            model.SetPWLObj(z, ptu, ptg)

            ' Add constraint: x + 2 y + 3 z <= 4
            model.AddConstr(x + 2 * y + 3 * z <= 4.0, "c0")

            ' Add constraint: x + y >= 1
        Catch ex As Exception
            Console.WriteLine(ex.Message)
        End Try
    End Sub
End Class
model.AddConstr(x + y >= 1.0, "c1")

' Optimize model as an LP
model.Optimize()

Console.WriteLine("IsMIP: " & model.IsMIP)

Console.WriteLine(x.VarName & " " & x.X)
Console.WriteLine(y.VarName & " " & y.X)
Console.WriteLine(z.VarName & " " & z.X)

Console.WriteLine("Obj: " & model.ObjVal)

Console.WriteLine()

' Negate piecewise-linear objective function for x
For i As Integer = 0 To npts - 1
    ptf(i) = -ptf(i)
Next

model.SetPWLObj(x, ptu, ptf)

' Optimize model as a MIP
model.Optimize()

Console.WriteLine("IsMIP: " & model.IsMIP)

Console.WriteLine(x.VarName & " " & x.X)
Console.WriteLine(y.VarName & " " & y.X)
Console.WriteLine(z.VarName & " " & z.X)

Console.WriteLine("Obj: " & model.ObjVal)

' Dispose of model and environment
model.Dispose()
env.Dispose()

Catch e As GRBException
End Try

End Sub

End Class

poolsearch_vb.vb

' Copyright 2020, Gurobi Optimization, LLC

' We find alternative epsilon-optimal solutions to a given knapsack problem by using PoolSearchMode

Imports Gurobi
Class poolsearch_vb

Shared Sub Main()

Try

' Sample data
Dim groundSetSize As Integer = 10

Dim objCoef As Double() = New Double() { 32, 32, 15, 15, 6, 6, 1, 1, 1, 1 }

Dim knapsackCoef As Double() = New Double() { 16, 16, 8, 8, 4, 4, 2, 2, 1, 1 }

Dim Budget As Double = 33
Dim e As Integer, status As Integer, nSolutions As Integer

' Create environment
Dim env As New GRBEnv("poolsearch_vb.log")

' Create initial model
Dim model As New GRBModel(env)
model.ModelName = "poolsearch_vb"

' Initialize decision variables for ground set:
' x[e] == k if element e is chosen k-times.
Dim Elem As GRBVar() = model.AddVars(groundSetSize, GRB.BINARY)
model.Set(GRB.DoubleAttr.Obj, Elem, objCoef, 0, groundSetSize)

For e = 0 To groundSetSize - 1
    Elem(e).VarName = String.Format("El{0}", e)
Next

' Constraint: limit total number of elements to be picked to be at most Budget
Dim lhs As New GRBLinExpr()
For e = 0 To groundSetSize - 1
    lhs.AddTerm(knapsackCoef(e), Elem(e))
Next
model.AddConstr(lhs, GRB.LESS_EQUAL, Budget, "Budget")

' set global sense for ALL objectives
model.ModelSense = GRB.MAXIMIZE

' Limit how many solutions to collect
model.Parameters.PoolSolutions = 1024

' Limit how many solutions to collect
model.Parameters.PoolGap = 0.1

' Limit how many solutions to collect
model.Parameters.PoolSearchMode = 2

' save problem
model.Write("poolsearch_vb.lp")
' Optimize
model.Optimize()

' Status checking
status = model.Status

If status = GRB.Status.INF_OR_UNBD OrElse 
    status = GRB.Status.INFEASIBLE OrElse 
    status = GRB.Status.UNBOUNDED Then
    Console.WriteLine("The model cannot be solved because it is infeasible or unbounded")
    Return
End If
If status <> GRB.Status.OPTIMAL Then
    Console.WriteLine("Optimization was stopped with status {0}", status)
    Return
End If

' Print best selected set
Console.WriteLine("Selected elements in best solution:")
Console.Write(vbTab)
For e = 0 To groundSetSize - 1
    If Elem(e).X < 0.9 Then
        Continue For
    End If
    Console.Write("El{0} ", e)
Next
Console.WriteLine()

' Print number of solutions stored
nSolutions = model.SolCount
Console.WriteLine("Number of solutions found: ", nSolutions)

' Print objective values of solutions
For e = 0 To nSolutions - 1
    model.Parameters.SolutionNumber = e
    Console.WriteLine("{0} ", model.PoolObjVal)
    If e Mod 15 = 14 Then
        Console.WriteLine()
    End If
Next
Console.WriteLine()
model.Dispose()
env.Dispose()

Catch e As GRBException
    Console.WriteLine("Error code: {0}. {1}", e.ErrorCode, e.Message)
End Try
End Sub

End Class

cp_vb.vb
' This example formulates and solves the following simple QCP model:

maximize        x
subject to  x + y + z = 1
            x^2 + y^2 <= z^2 (second-order cone)
            x^2 <= yz (rotated second-order cone)
            x, y, z non-negative

Imports Gurobi

Class qcp_vb
    Shared Sub Main()
        Try
            Dim env As New GRBEnv("qcp.log")
            Dim model As New GRBModel(env)

            ' Create variables
            Dim x As GRBVar = model.AddVar(0.0, GRB.INFINITY, 0.0, GRB.CONTINUOUS, "x")
            Dim y As GRBVar = model.AddVar(0.0, GRB.INFINITY, 0.0, GRB.CONTINUOUS, "y")
            Dim z As GRBVar = model.AddVar(0.0, GRB.INFINITY, 0.0, GRB.CONTINUOUS, "z")

            ' Set objective
            Dim obj As GRBLinExpr = x
            model.SetObjective(obj, GRB.MAXIMIZE)

            ' Add linear constraint: x + y + z = 1
            model.AddConstr(x + y + z = 1.0, "c0")

            ' Add second-order cone: x^2 + y^2 <= z^2
            model.AddQConstr(x * x + y * y <= z * z, "qc0")

            ' Add rotated cone: x^2 <= yz
            model.AddQConstr(x * x <= y * z, "qc1")

            ' Optimize model
            model.Optimize()

            Console.WriteLine(x.VarName & " = " & x.X)
            Console.WriteLine(y.VarName & " = " & y.X)
            Console.WriteLine(z.VarName & " = " & z.X)


        ' Dispose of model and env
        model.Dispose()
        env.Dispose()
Class qp_vb

' Copyright 2020, Gurobi Optimization, LLC
'
' This example formulates and solves the following simple QP model:
' minimize x^2 + x*y + y^2 + y*z + z^2 + 2 x
' subject to x + 2 y + 3 z >= 4
' x + y >= 1
' x, y, z non-negative
'
' It solves it once as a continuous model, and once as an integer model.
'
Imports Gurobi

Class qp_vb

Shared Sub Main()
    Try
        Dim env As New GRBEnv("qp.log")
        Dim model As New GRBModel(env)

        ' Create variables
        Dim x As GRBVar = model.AddVar(0.0, 1.0, 0.0, GRB.CONTINUOUS, "x")
        Dim y As GRBVar = model.AddVar(0.0, 1.0, 0.0, GRB.CONTINUOUS, "y")
        Dim z As GRBVar = model.AddVar(0.0, 1.0, 0.0, GRB.CONTINUOUS, "z")

        ' Set objective
        Dim obj As New GRBQuadExpr()
        obj = x*x + x*y + y*y + y*z + z*z + 2*x
        model.SetObjective(obj)

        ' Add constraint: x + 2 y + 3 z >= 4
        model.AddConstr(x + 2 * y + 3 * z >= 4.0, "c0")

        ' Add constraint: x + y >= 1
        model.AddConstr(x + y >= 1.0, "c1")

        ' Optimize model
        model.Optimize()

        Console.WriteLine(x.VarName & " " & x.X)
        Console.WriteLine(y.VarName & " " & y.X)
        Console.WriteLine(z.VarName & " " & z.X)
    End Try
End Sub

End Class

' Change variable types to integer
x.VType = GRB.INTEGER
y.VType = GRB.INTEGER
z.VType = GRB.INTEGER

' Optimize model
model.Optimize()

Console.WriteLine(x.VarName & " " & x.X)
Console.WriteLine(y.VarName & " " & y.X)
Console.WriteLine(z.VarName & " " & z.X)


' Dispose of model and env
model.Dispose()
env.Dispose()

Catch e As GRBException
End Try
End Sub

sensitivity_vb.vb

' Copyright 2020, Gurobi Optimization, LLC

' A simple sensitivity analysis example which reads a MIP model from a
' file and solves it. Then uses the scenario feature to analyze the impact
' w.r.t. the objective function of each binary variable if it is set to
' 1-X, where X is its value in the optimal solution.
',
', Usage:
', ' sensitivity_cs <model filename>

Imports System
Imports Gurobi

Class sensitivity_vb
    Shared Sub Main(args As String())
        If args.Length < 1 Then
            Console.Out.WriteLine("Usage: sensitivity_vb filename")
            Return
        End If
        Try
            ' Maximum number of scenarios to be considered
            Dim MAXSCENARIOS as Integer = 100
            ' Maximum number of scenarios to be considered
            Dim MAXSCENARIOS as Integer = 100
        Catch e As GRBException
        End Try
    End Sub
End Class
' Create environment
Dim env As New GRBEnv()

' Read model
Dim model As New GRBModel(env, args(0))

Dim scenarios As Integer
If model.IsMIP = 0 Then
    Console.WriteLine("Model is not a MIP")
    Return
End If

' Solve model
model.Optimize()
If model.Status <> GRB.Status.OPTIMAL Then
    Console.WriteLine("Optimization ended with status ", model.Status)
    Return
End If

' Store the optimal solution
Dim origObjVal As Double = model.ObjVal
Dim vars As GRBVar() = model.GetVars()
Dim origX As Double() = model.Get(GRB.DoubleAttr.X, vars)

scenarios = 0

' Count number of unfixed, binary variables in model. For each we
' create a scenario.
For i As Integer = 0 To vars.Length - 1
    Dim v As GRBVar = vars(i)
    Dim vType As Char = v.VType
    If v.LB = 0.0 AndAlso v.UB = 1.0 AndAlso (vType = GRB.BINARY OrElse vType = GRB.INTEGER) Then
        scenarios += 1
        If scenarios >= MAXSCENARIOS Then Exit For
    End If
End If

Console.WriteLine("### construct multi-scenario model with ", scenarios
& " scenarios")

' Set the number of scenarios in the model */
model.NumScenarios = scenarios

scenarios = 0

' Create a (single) scenario model by iterating through unfixed
binary variables in the model and create for each of these
variables a scenario by fixing the variable to 1-X, where X is its
value in the computed optimal solution

For i As Integer = 0 To vars.Length - 1
    Dim v As GRBVar = vars(i)
    Dim vType As Char = v.VType

    If v.LB = 0.0 AndAlso v.UB = 1.0 AndAlso (vType = GRB.BINARY OrElse vType = GRB.INTEGER) AndAlso scenarios < MAXSCENARIOS Then

        ' Set ScenarioNumber parameter to select the corresponding
        ' scenario for adjustments
        model.Parameters.ScenarioNumber = scenarios

        ' Set variable to 1-X, where X is its value in the optimal solution */
        If origX(i) < 0.5 Then
            v.ScnNLB = 1.0
        Else
            v.ScnNUB = 0.0
        End If

        scenarios += 1
    Else

        ' Add MIP start for all other variables using the optimal solution
        ' of the base model
        v.Start = origX(i)
    End If

Next

' Solve multi-scenario model
model.Optimize()

' In case we solved the scenario model to optimality capture the
' sensitivity information
If model.Status = GRB.Status.OPTIMAL Then
    Dim modelSense As Integer = model.ModelSense

    scenarios = 0

    For i As Integer = 0 To vars.Length - 1
        Dim v As GRBVar = vars(i)
        Dim vType As Char = v.VType

        If v.LB = 0.0 AndAlso v.UB = 1.0 AndAlso (vType = GRB.BINARY OrElse vType = GRB.INTEGER) Then

            ' Set scenario parameter to collect the objective value of the
            ' corresponding scenario
            model.Parameters.ScenarioNumber = scenarios

            ' Collect objective value and bound for the scenario
            Dim scenarioObjVal As Double = model.ScenObjVal
            Dim scenarioObjBound As Double = model.ScenObjBound

        End If

    Next

Console.Write("Objective sensitivity for variable " + 
v.VarName & ")

' Check if we found a feasible solution for this scenario
If scenarioObjVal >= modelSense * GRB.INFINITY Then
    ' Check if the scenario is infeasible
    If scenarioObjBound >= modelSense * GRB.INFINITY Then
        Console.WriteLine("infeasible")
    Else
        Console.WriteLine("unknown (no solution available")
    End If
Else
    ' Scenario is feasible and a solution is available
    Console.WriteLine(modelSense * (scenarioObjVal - origObjVal))
End If

scenarios += 1
If scenarios >= MAXSCENARIOS Then
    Exit For
End If

End If
Next
End If

' Dispose of model and environment
model.Dispose()
env.Dispose()

Catch e As GRBException
    Console.WriteLine("Error code: " + e.ErrorCode)
    Console.WriteLine(e.Message)
    Console.WriteLine(e.StackTrace)
End Try
End Sub
End Class

sos_vb.vb

' Copyright 2020, Gurobi Optimization, LLC
'
' This example creates a very simple Special Ordered Set (SOS) model.
' The model consists of 3 continuous variables, no linear constraints,
' and a pair of SOS constraints of type 1.

Imports System
Imports Gurobi

Class sos_vb
    Shared Sub Main()
        Try
            Dim env As New GRBEnv()
            Dim model As New GRBModel(env)
        End Try
    End Sub
End Class
' Create variables
Dim ub As Double() = {1, 1, 2}
Dim obj As Double() = {-2, -1, -1}
Dim names As String() = {"x0", "x1", "x2"}
Dim x As GRBVar() = model.AddVars(Nothing, ub, obj, Nothing, names)
'
' Add first SOS1: x0=0 or x1=0
Dim sosv1 As GRBVar() = {x(0), x(1)}
Dim soswt1 As Double() = {1, 2}
model.AddSOS(sosv1, soswt1, GRB.SOS_TYPE1)
'
' Add second SOS1: x0=0 or x2=0
Dim sosv2 As GRBVar() = {x(0), x(2)}
Dim soswt2 As Double() = {1, 2}
model.AddSOS(sosv2, soswt2, GRB.SOS_TYPE1)
'
' Optimize model
model.Optimize()
For i As Integer = 0 To 2
    Console.WriteLine(x(i).VarName & " " & x(i).X)
Next
'
' Dispose of model and env
model.Dispose()
env.Dispose()

Catch e As GRBException
End Try
End Sub
End Class

sudoku_vb.vb

' Copyright 2020, Gurobi Optimization, LLC
',
' Sudoku example.
',
' The Sudoku board is a 9x9 grid, which is further divided into a 3x3 grid
' of 3x3 grids. Each cell in the grid must take a value from 0 to 9.
' No two grid cells in the same row, column, or 3x3 subgrid may take the
' same value.
'
' In the MIP formulation, binary variables x(i,j,v) indicate whether
' cell <i,j> takes value 'v'. The constraints are as follows:
' 1. Each cell must take exactly one value (sum_v x(i,j,v) = 1)
' 2. Each value is used exactly once per row (sum_i x(i,j,v) = 1)
' 3. Each value is used exactly once per column (sum_j x(i,j,v) = 1)
Imports System
Imports System.IO
Imports Gurobi

Class sudoku_vb
    Shared Sub Main(ByVal args as String())
        Dim n As Integer = 9
        Dim s As Integer = 3

        If args.Length < 1 Then
            Console.WriteLine("Usage: sudoku_vb filename")
            Return
        End If

        Try
            Dim env As New GRBEnv()
            Dim model As New GRBModel(env)

            ' Create 3-D array of model variables
            Dim vars As GRBVar(,,) = New GRBVar(n - 1, n - 1, n - 1) {}

            For i As Integer = 0 To n - 1
                For j As Integer = 0 To n - 1
                    For v As Integer = 0 To n - 1
                        Dim st As String = "G_" & i & "_" & j & "_" & v
                        vars(i, j, v) = model.AddVar(0.0, 1.0, 0.0, GRB.BINARY, st)
                    Next
                Next
            Next

            ' Add constraints
            Dim expr As GRBLinExpr

            ' Each cell must take one value
            For i As Integer = 0 To n - 1
                For j As Integer = 0 To n - 1
                    expr = 0
                    For v As Integer = 0 To n - 1
                        expr.AddTerm(1.0, vars(i, j, v))
                    Next
                    Dim st As String = "V_" & i & "_" & j
                    model.AddConstr(expr = 1, st)
                Next
            Next

            ' Each value appears once per row
            For i As Integer = 0 To n - 1
                For v As Integer = 0 To n - 1
                    For j As Integer = 0 To n - 1
                        vars(i, j, v) = model.AddVar(0.0, 1.0, 0.0, GRB.BINARY, st)
                    Next
                Next
            Next
        End Try
    End Sub
End Class
expr = 0
For j As Integer = 0 To n - 1
    expr.AddTerm(1.0, vars(i, j, v))
Next
Dim st As String = "R_" & i & "_" & v
model.AddConstr(expr = 1, st)
Next
Next

' Each value appears once per column
For j As Integer = 0 To n - 1
    For v As Integer = 0 To n - 1
        expr = 0
        For i As Integer = 0 To n - 1
            expr.AddTerm(1.0, vars(i, j, v))
        Next
        Dim st As String = "C_" & j & "_" & v
        model.AddConstr(expr = 1, st)
    Next
Next

' Each value appears once per sub-grid
For v As Integer = 0 To n - 1
    For i0 As Integer = 0 To s - 1
        For j0 As Integer = 0 To s - 1
            expr = 0
            For i1 As Integer = 0 To s - 1
                For j1 As Integer = 0 To s - 1
                    expr.AddTerm(1.0, vars(i0 * s + i1, j0 * s + j1, v))
                Next
            Next
            Dim st As String = "Sub_" & v & "_" & i0 & "_" & j0
            model.AddConstr(expr = 1, st)
        Next
    Next
Next

' Fix variables associated with pre-specified cells
Dim sr As StreamReader = File.OpenText(args(0))
For i As Integer = 0 To n - 1
    Dim input As String = sr.ReadLine()
    For j As Integer = 0 To n - 1
        Dim val As Integer = Microsoft.VisualBasic.Asc(input(j)) - 48 - 1
        ' 0-based
        If val >= 0 Then
            vars(i, j, val).LB = 1.0
        End If
    Next
Next

' Optimize model
model.Optimize()

' Write model to file
model.Write("sudoku.lp")

Dim x As Double(,,) = model.Get(GRB.DoubleAttr.X, vars)

Console.WriteLine()
For i As Integer = 0 To n - 1
    For j As Integer = 0 To n - 1
        For v As Integer = 0 To n - 1
            If x(i, j, v) > 0.5 Then
                Console.Write(v + 1)
            End If
        Next
    Next
Next

Console.WriteLine()

' Dispose of model and env
model.Dispose()
env.Dispose()

Catch e As GRBException
End Try
End Sub
End Class
tsp_vb.vb

' Copyright 2020, Gurobi Optimization, LLC
'
' Solve a traveling salesman problem on a randomly generated set of
' points using lazy constraints. The base MIP model only includes
' 'degree-2' constraints, requiring each node to have exactly
' two incident edges. Solutions to this model may contain subtours -
' tours that don't visit every node. The lazy constraint callback
' adds new constraints to cut them off.

Imports Gurobi

Class tsp_vb
    Inherits GRBCallback

    Private vars As GRBVar(,,)

    Public Sub New(xvars As GRBVar(,,))
        vars = xvars
    End Sub

    ' Subtour elimination callback. Whenever a feasible solution is found,
    ' find the smallest subtour, and add a subtour elimination constraint
    ' if the tour doesn't visit every node.

    Protected Overrides Sub Callback()
        Try
If where = GRB.Callback.MIPSOL Then
  ' Found an integer feasible solution - does it visit every node?
  Dim n As Integer = vars.GetLength(0)
  Dim tour As Integer() = findsubtour(GetSolution(vars))

  If tour.Length < n Then
    ' Add subtour elimination constraint
    Dim expr As GRBLinExpr = 0
    For i As Integer = 0 To tour.Length - 1
      For j As Integer = i + 1 To tour.Length - 1
        expr.AddTerm(1.0, vars(tour(i), tour(j)))
      Next
    Next
    AddLazy(expr <= tour.Length - 1)
  End If
End If
Catch e As GRBException
  Console.WriteLine(e.StackTrace)
End Try
End Sub

' Given an integer-feasible solution 'sol', returns the smallest
' sub-tour (as a list of node indices).
Protected Shared Function findsubtour(sol As Double(),) As Integer()
  Dim n As Integer = sol.GetLength(0)
  Dim seen As Boolean() = New Boolean(n - 1) {}
  Dim tour As Integer() = New Integer(n - 1) {}
  Dim bestind As Integer, bestlen As Integer
  Dim i As Integer, node As Integer, len As Integer, start As Integer

  For i = 0 To n - 1
    seen(i) = False
  Next

  start = 0
  bestlen = n + 1
  bestind = -1
  node = 0
  While start < n
    For node = 0 To n - 1
      if Not seen(node)
        Exit For
      End if
    Next
    if node = n
      Exit While
    End if
    For len = 0 To n - 1
      tour(start + len) = node
      seen(node) = true
      For i = 0 To n - 1
        if sol(node, i) > 0.5 AndAlso Not seen(i)
          node = i
          seen(node) = true
          start = start + len
          bestind = node
          bestlen = len
          Exit For
        End if
      Next
    Next
  End While
  Return bestind
End Function
Exit For
End If
Next
If i = n
    len = len + 1
    If len < bestlen
        bestlen = len
        bestind = start
    End If
    start = start + len
    Exit For
End If
Next
End While

For i = 0 To bestlen - 1
    tour(i) = tour(bestind+i)
Next
System.Array.Resize(tour, bestlen)

Return tour
End Function

' Euclidean distance between points 'i' and 'j'

Protected Shared Function distance(x As Double(), y As Double(), _
    i As Integer, j As Integer) As Double
    Dim dx As Double = x(i) - x(j)
    Dim dy As Double = y(i) - y(j)
    Return Math.Sqrt(dx * dx + dy * dy)
End Function

Public Shared Sub Main(args As String())

    If args.Length < 1 Then
        Console.WriteLine("Usage: tsp_vb nnodes")
        Return
    End If

    Dim n As Integer = Convert.ToInt32(args(0))

    Try
        Dim env As New GRBEnv()
        Dim model As New GRBModel(env)

        ' Must set LazyConstraints parameter when using lazy constraints
        model.Parameters.LazyConstraints = 1

        Dim x As Double() = New Double(n - 1) {}
        Dim y As Double() = New Double(n - 1) {}

        Dim r As New Random()
        For i As Integer = 0 To n - 1
            x(i) = r.NextDouble()
            y(i) = r.NextDouble()
    End Try

End Sub
' Create variables
Dim vars As GRBVar(,) = New GRBVar(n - 1, n - 1) {}  
For i As Integer = 0 To n - 1
    For j As Integer = 0 To i
        vars(i, j) = model.AddVar(0.0, 1.0, distance(x, y, i, j), _
            GRB.BINARY, "x" & i & "_" & j)
        vars(j, i) = vars(i, j)
    Next
Next

' Degree-2 constraints
For i As Integer = 0 To n - 1
    Dim expr As GRBLinExpr = 0
    For j As Integer = 0 To n - 1
        expr.AddTerm(1.0, vars(i, j))
    Next
    model.AddConstr(expr = 2.0, "deg2_" & i)
Next

' Forbid edge from node back to itself
For i As Integer = 0 To n - 1
    vars(i, i).UB = 0.0
Next

model.SetCallback(New tsp_vb(vars))
model Optimize()

If model.SolCount > 0 Then
    Dim tour As Integer() = findsubtour(model.Get(GRB.DoubleAttr.X, vars))
    Console.Write("Tour: ")
    For i As Integer = 0 To tour.Length - 1
        Console.Write(tour(i) & " ")
    Next
    Console.WriteLine()
End If

' Dispose of model and environment
model.Dispose()
env.Dispose()

Catch e As GRBException
    Console.WriteLine(e.StackTrace)
End Try
End Sub
End Class
tune_vb.vb
Imports System
Imports Gurobi

Class tune_vb
    Shared Sub Main(ByVal args As String())
        If args.Length < 1 Then
            Console.Out.WriteLine("Usage: tune_vb filename")
            Return
        End If
        Try
            Dim env As New GRBEnv()
            Dim model As New GRBModel(env, args(0))
            model.Parameters.TuneResults = 1
            model.Tune()
            Dim resultcount As Integer = model.TuneResultCount
            If resultcount > 0 Then
                model.GetTuneResult(0)
                model.Write("tune.prm")
                model.Optimize()
            End If
            model.Dispose()
            env.Dispose()
        Catch e As GRBException
        End Try
    End Sub
End Class

workforce1_vb.vb
Assign workers to shifts; each worker may or may not be available on a particular day. If the problem cannot be solved, use IIS to find a set of conflicting constraints. Note that there may be additional conflicts besides what is reported via IIS.

Imports System
Imports Gurobi

Class workforce1_vb
    Shared Sub Main()
        Try

            ' Sample data
            ' Sets of days and workers
            Dim Shifts As String() = New String() {"Mon1", "Tue2", "Wed3", "Thu4", _
                "Fri5", "Sat6", "Sun7", "Mon8", _
                "Tue9", "Wed10", "Thu11", _
                "Fri12", "Sat13", "Sun14"}  
            Dim Workers As String() = New String() {"Amy", "Bob", "Cathy", "Dan", _
                "Ed", "Fred", "Gu"}

            Dim nShifts As Integer = Shifts.Length
            Dim nWorkers As Integer = Workers.Length

            ' Number of workers required for each shift
            Dim shiftRequirements As Double() = New Double() {3, 2, 4, 4, 5, 6, _
                5, 2, 2, 3, 4, 6, _
                7, 5}

            ' Amount each worker is paid to work one shift
            Dim pay As Double() = New Double() {10, 12, 10, 8, 8, 9, 11}

            ' Worker availability: 0 if the worker is unavailable for a shift
            Dim availability As Double(,) = New Double(,) { _
                {0, 1, 1, 0, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1}, _
                {1, 1, 0, 0, 1, 1, 0, 1, 0, 1, 0, 1, 0, 1}, _
                {0, 0, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1}, _
                {0, 1, 1, 0, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1}, _
                {1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 0, 1, 1}, _
                {1, 1, 1, 0, 0, 1, 0, 1, 1, 1, 0, 1, 1, 1}, _
                {1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1}}

            ' Model
            Dim env As New GRBEnv()
            Dim model As New GRBModel(env)

            model.ModelName = "assignment"

            ' Assignment variables: x(w)(s) == 1 if worker w is assigned to shift s. Since an assignment model always produces integer solutions, we use continuous variables and solve as an LP.
            Dim x As GRBVar(,) = New GRBVar(nWorkers - 1, nShifts - 1) {} 
            For w As Integer = 0 To nWorkers - 1
                For s As Integer = 0 To nShifts - 1
                    
                Next
            Next
        
```
\[ x(w, s) = \text{model.AddVar}(0, \text{availability}(w, s), \text{pay}(w), \_ \text{GRB.CONTINUOUS, } \_ \text{Workers}(w) \& \text{"."} \& \text{Shifts}(s)) \]

Next
Next

' The objective is to minimize the total pay costs
model.ModelSense = \text{GRB.MINIMIZE}

' Constraint: assign exactly shiftRequirements(s) workers
' to each shift s
For s As Integer = 0 To nShifts - 1
  Dim lhs As \text{GRBLinExpr} = 0
  For w As Integer = 0 To nWorkers - 1
    lhs.AddTerm(1.0, x(w, s))
  Next
  model.AddConstr(lhs = shiftRequirements(s), Shifts(s))
Next

' Optimize
model.Optimize()
Dim status As Integer = model.Status
If status = GRB.Status.UNBOUNDED Then
  Console.WriteLine("The model cannot be solved " & 
  "because it is unbounded")
  Exit Sub
End If
If status = GRB.Status.OPTIMAL Then
  Console.WriteLine("The optimal objective is " & model.ObjVal)
  Exit Sub
End If
If (status <> GRB.Status.INF_OR_UNBD) AndAlso 
  (status <> GRB.Status.INFEASIBLE) Then
  Console.WriteLine("Optimization was stopped with status " & status)
  Exit Sub
End If

' Do IIS
Console.WriteLine("The model is infeasible; computing IIS")
model.ComputeIIS()
Console.WriteLine(vbLf & "The following constraint(s) " & 
  "cannot be satisfied:"
For Each c As GRBConstr In model.GetConstrs()
  If c.IISConstr = 1 Then
    Console.WriteLine(c.ConstrName)
  End If
Next

' Dispose of model and env
model.Dispose()
env.Dispose()

Catch e As GRBException
End Try
' Copyright 2020, Gurobi Optimization, LLC
'
' Assign workers to shifts; each worker may or may not be available on a
' particular day. If the problem cannot be solved, use IIS iteratively to
' find all conflicting constraints.

Imports System
Imports System.Collections.Generic
Imports Gurobi

Class workforce2_vb
    Shared Sub Main()
        Try

            ' Sample data
            ' Sets of days and workers
            Dim Shifts As String() = New String() {"Mon1", "Tue2", "Wed3", "Thu4", _
                                               "Fri5", "Sat6", "Sun7", "Mon8", _
                                               "Tue9", "Wed10", "Thu11", _
                                               "Fri12", "Sat13", "Sun14"}
            Dim Workers As String() = New String() {"Amy", "Bob", "Cathy", "Dan", _
                                               "Ed", "Fred", "Gu"}
            Dim nShifts As Integer = Shifts.Length
            Dim nWorkers As Integer = Workers.Length

            ' Number of workers required for each shift
            Dim shiftRequirements As Double() = New Double() {3, 2, 4, 4, 5, 6, _
                                                        5, 2, 2, 3, 4, 6, _
                                                        7, 5}

            ' Amount each worker is paid to work one shift
            Dim pay As Double() = New Double() {10, 12, 10, 8, 8, 9, 11}

            ' Worker availability: 0 if the worker is unavailable for a shift
            Dim availability As Double(,,) = New Double(,,) {
                {0, 1, 1, 0, 1, 0, 1, 0, 1, 1, 1, 1, 1, 1}, _
                {1, 1, 0, 0, 1, 1, 0, 1, 0, 1, 0, 1, 1, 1}, _
                {0, 0, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1}, _
                {0, 1, 1, 0, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1}, _
                {1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1}, _
                {1, 1, 1, 0, 1, 0, 1, 0, 1, 1, 0, 1, 1, 1}, _
                {1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1}}

            ' Model
            Dim env As New GRBEnv()
            Dim model As New GRBModel(env)
            model.ModelName = "assignment"

            ' Assignment variables: x(w)(s) == 1 if worker w is assigned
        End Try
    End Sub
End Class

end_class
' to shift s. Since an assignment model always produces integer 
solutions, we use continuous variables and solve as an LP.
Dim x As GRBVar() = New GRBVar(nWorkers - 1, nShifts - 1) {}
For w As Integer = 0 To nWorkers - 1
    For s As Integer = 0 To nShifts - 1
        x(w, s) = model.AddVar(0, availability(w, s), pay(w), _
            GRB.CONTINUOUS, _
            Workers(w) & "." & Shifts(s))
    Next
Next

' The objective is to minimize the total pay costs
model.ModelSense = GRB.MINIMIZE

' Constraint: assign exactly shiftRequirements(s) workers 
' to each shift s
For s As Integer = 0 To nShifts - 1
    Dim lhs As GRBLinExpr = 0
    For w As Integer = 0 To nWorkers - 1
        lhs.AddTerm(1.0, x(w, s))
    Next
    model.AddConstr(lhs = shiftRequirements(s), Shifts(s))
Next

' Optimize
model.Optimize()
Dim status As Integer = model.Status
If status = GRB.Status.UNBOUNDED Then
    Console.WriteLine("The model cannot be solved " & _
        "because it is unbounded")
    Exit Sub
End If
If status = GRB.Status.OPTIMAL Then
    Console.WriteLine("The optimal objective is " & model.ObjVal)
    Exit Sub
End If
If (status <> GRB.Status.INF_OR_UNBD) AndAlso _
    (status <> GRB.Status.INFEASIBLE) Then
    Console.WriteLine("Optimization was stopped with status " & status)
    Exit Sub
End If

' Do IIS
Console.WriteLine("The model is infeasible; computing IIS")
Dim removed As LinkedList(Of String) = New LinkedList(Of String)()

' Loop until we reduce to a model that can be solved
While True
    model.ComputeIIS()
    Console.WriteLine(vbLf & "The following constraint cannot be satisfied:")
    For Each c As GRBConstr In model.GetConstrs()
        If c.IISConstr = 1 Then
            Console.WriteLine(c.ConstrName)
            ' Remove a single constraint from the model
            removed.AddFirst(c.ConstrName)
            model.Remove(c)
        End If
    Next
    ' Do IIS again
End While
439
Exit For
End If
Next

Console.WriteLine()
model.Optimize()
status = model.Status

If status = GRB.Status.UNBOUNDED Then
    Console.WriteLine("The model cannot be solved " & _
        "because it is unbounded")
    Exit Sub
End If
If status = GRB.Status.OPTIMAL Then
    Exit While
End If
If (status <> GRB.Status.INF_OR_UNBD) AndAlso _
    (status <> GRB.Status.INFEASIBLE) Then
    Console.WriteLine("Optimization was stopped with status " & _
        status)
    Exit Sub
End If
End While

Console.WriteLine(vbLf & "The following constraints were removed " & _
    "to get a feasible LP:"
For Each s As String In removed
    Console.Write(s & " ")
Next

Console.WriteLine()
' Dispose of model and env
model.Dispose()
env.Dispose()

Catch e As GRBException
End Try
End Sub
End Class

workforce3_vb.vb

' Copyright 2020, Gurobi Optimization, LLC

' Assign workers to shifts; each worker may or may not be available on a
' particular day. If the problem cannot be solved, relax the model
' to determine which constraints cannot be satisfied, and how much
' they need to be relaxed.

Imports System
Imports Gurobi

Class workforce3_vb
    Shared Sub Main()

73

440
Try

' Sample data
' Sets of days and workers
Dim Shifts As String() = New String() {"Mon1", "Tue2", "Wed3", "Thu4", _
"Fri5", "Sat6", "Sun7", "Mon8", _
"Tue9", "Wed10", "Thu11", _
"Fri12", "Sat13", "Sun14"}
Dim Workers As String() = New String() {"Amy", "Bob", "Cathy", "Dan", _
"Ed", "Fred", "Gu"}

Dim nShifts As Integer = Shifts.Length
Dim nWorkers As Integer = Workers.Length

' Number of workers required for each shift
Dim shiftRequirements As Double() = New Double() {3, 2, 4, 4, 5, 6, _
5, 2, 2, 3, 4, 6, _
7, 5}

' Amount each worker is paid to work one shift
Dim pay As Double() = New Double() {10, 12, 10, 8, 8, 9, 11}

' Worker availability: 0 if the worker is unavailable for a shift
Dim availability As Double(,) = New Double(,) { _
{0, 1, 1, 0, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1}, _
{1, 1, 0, 0, 1, 1, 0, 1, 0, 0, 1, 0, 1, 0}, _
{0, 0, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1}, _
{0, 1, 1, 0, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1}, _
{1, 1, 1, 1, 1, 0, 1, 1, 1, 0, 0, 1, 0, 1}, _
{1, 1, 1, 0, 0, 1, 0, 1, 1, 0, 1, 1, 1, 1}}

' Model
Dim env As New GRBEnv()
Dim model As New GRBModel(env)
model.ModelName = "assignment"

' Assignment variables: x[w][s] == 1 if worker w is assigned ' to shift s. Since an assignment model always produces integer ' solutions, we use continuous variables and solve as an LP.
Dim x As GRBVar(,) = New GRBVar(nWorkers - 1, nShifts - 1) {}
For w As Integer = 0 To nWorkers - 1
   For s As Integer = 0 To nShifts - 1
      x(w, s) = model.AddVar(0, availability(w, s), pay(w), _
         GRB.CONTINUOUS, _
         Workers(w) & "." & Shifts(s))
   Next
Next

' The objective is to minimize the total pay costs
model.ModelSense = GRB.MINIMIZE

' Constraint: assign exactly shiftRequirements[s] workers ' to each shift s
For s As Integer = 0 To nShifts - 1
   model.AddConstr(x(:, s) = shiftRequirements(s), _
      "shift" & s + 1)
Next
Dim lhs As GRBLinExpr = 0.0
For w As Integer = 0 To nWorkers - 1
    lhs.AddTerm(1.0, x(w, s))
Next
model.AddConstr(lhs = shiftRequirements(s), Shifts(s))
Next

' Optimize
model.Optimize()
Dim status As Integer = model.Status
If status = GRB.Status.UNBOUNDED Then
    Console.WriteLine("The model cannot be solved " & _
    "because it is unbounded")
    Return
End If
If status = GRB.Status.OPTIMAL Then
    Console.WriteLine("The optimal objective is " & model.ObjVal)
    Return
End If
If (status <> GRB.Status.INF_OR_UNBD) AndAlso _
    (status <> GRB.Status.INFEASIBLE) Then
    Console.WriteLine("Optimization was stopped with status " & _
    status)
    Return
End If

' Relax the constraints to make the model feasible
Console.WriteLine("The model is infeasible; relaxing the constraints")
Dim orignumvars As Integer = model.NumVars
model.FeasRelax(0, False, False, True)
model.Optimize()
status = model.Status
If (status = GRB.Status.INF_OR_UNBD) OrElse _
    (status = GRB.Status.INFEASIBLE) OrElse _
    (status = GRB.Status.UNBOUNDED) Then
    Console.WriteLine("The relaxed model cannot be solved " & _
    "because it is infeasible or unbounded")
    Return
End If
If status <> GRB.Status.OPTIMAL Then
    Console.WriteLine("Optimization was stopped with status " & status)
    Return
End If

Console.WriteLine(vbLf & "Slack values:")
Dim vars As GRBVar() = model.GetVars()
For i As Integer = orignumvars To model.NumVars - 1
    Dim sv As GRBVar = vars(i)
    If sv.X > 1E-06 Then
        Console.WriteLine(sv.VarName & " = " & sv.X)
    End If
Next

' Dispose of model and environment
model.Dispose()
env.Dispose()
CATCH e AS GRBException
End Try
End Sub
End Class

workforce4_vb.vb

' Copyright 2020, Gurobi Optimization, LLC
'
' Assign workers to shifts; each worker may or may not be available on a
' particular day. We use Pareto optimization to solve the model:
' first, we minimize the linear sum of the slacks. Then, we constrain
' the sum of the slacks, and we minimize a quadratic objective that
' tries to balance the workload among the workers.

Imports System
Imports Gurobi

Class workforce4_vb
    Shared Sub Main()
        Try

            ' Sample data
            ' Sets of days and workers
            Dim Shifts As String() = New String() {"Mon1", "Tue2", "Wed3", "Thu4", _
                "Fri5", "Sat6", "Sun7", "Mon8", _
                "Tue9", "Wed10", "Thu11", _
                "Fri12", "Sat13", "Sun14"}
            Dim Workers As String() = New String() {"Amy", "Bob", "Cathy", "Dan", _
                "Ed", "Fred", "Gu"}
            Dim nShifts As Integer = Shifts.Length
            Dim nWorkers As Integer = Workers.Length

            ' Number of workers required for each shift
            Dim shiftRequirements As Double() = New Double() {3, 2, 4, 4, 5, 6, _
                5, 2, 2, 3, 4, 6, _
                7, 5}

            ' Worker availability: 0 if the worker is unavailable for a shift
            Dim availability As Double(,) = New Double(,) { _
                {0, 1, 1, 0, 1, 0, 1, 1, 1, 1, 1, 1}, _
                {1, 1, 0, 0, 1, 1, 0, 1, 0, 1, 0, 1}, _
                {0, 0, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1}, _
                {0, 1, 1, 0, 1, 1, 0, 1, 1, 1, 1, 1}, _
                {1, 1, 1, 1, 1, 0, 1, 1, 0, 1, 0, 1}, _
                {1, 1, 1, 0, 0, 1, 0, 1, 0, 0, 1, 1}, _
                {1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1}}

            ' Model
            Dim env As New GRBEnv()
            Dim model As New GRBModel(env)

            model.ModelName = "assignment"
Assignment variables: \( x(w)(s) = 1 \) if worker \( w \) is assigned to shift \( s \). This is no longer a pure assignment model, so we must use binary variables.

```vba
Dim x As GRBVar(,,) = New GRBVar(nWorkers - 1, nShifts - 1) {}
For w As Integer = 0 To nWorkers - 1
    For s As Integer = 0 To nShifts - 1
        x(w, s) = model.AddVar(0, availability(w, s), 0, _
            GRB.BINARY, _
            Workers(w) & "." & Shifts(s))
    Next
Next
```

Add a new slack variable to each shift constraint so that the shifts can be satisfied

```vba
Dim slacks As GRBVar() = New GRBVar(nShifts - 1) {}
For s As Integer = 0 To nShifts - 1
    slacks(s) = model.AddVar(0, GRB.INFINITY, 0, GRB.CONTINUOUS, _
        Shifts(s) & "Slack")
Next
```

Variable to represent the total slack

```vba
Dim totSlack As GRBVar = model.AddVar(0, GRB.INFINITY, 0, _
        GRB.CONTINUOUS, "totSlack")
```

Variables to count the total shifts worked by each worker

```vba
Dim totShifts As GREVar() = New GREVar(nWorkers - 1) {}
For w As Integer = 0 To nWorkers - 1
    totShifts(w) = model.AddVar(0, GRB.INFINITY, 0, GRB.CONTINUOUS, _
        Workers(w) & "TotShifts")
Next
```

Constraint: assign exactly \( shiftRequirements(s) \) workers to each shift \( s \), plus the slack

```vba
Dim lhs As GRBLinExpr
For s As Integer = 0 To nShifts - 1
    lhs = 0
    For w As Integer = 0 To nWorkers - 1
        lhs.AddTerm(1.0, x(w, s))
    Next
    model.AddConstr(lhs = shiftRequirements(s), Shifts(s))
Next
```

Constraint: set \( totSlack \) equal to the total slack

```vba
For s As Integer = 0 To nShifts - 1
    lhs.AddTerm(1.0, slacks(s))
Next
model.AddConstr(lhs = totSlack, "totSlack")
```

Constraint: compute the total number of shifts for each worker

```vba
For w As Integer = 0 To nWorkers - 1
```


lhs = 0
For s As Integer = 0 To nShifts - 1
    lhs.AddTerm(1.0, x(w, s))
Next
model.AddConstr(lhs = totShifts(w), "totShifts" & Workers(w))

' Objective: minimize the total slack
model.SetObjective(1.0*totSlack)

' Optimize
Dim status As Integer = _
    solveAndPrint(model, totSlack, nWorkers, Workers, totShifts)
If status <> GRB.Status.OPTIMAL Then
    Exit Sub
End If

' Constrain the slack by setting its upper and lower bounds
totSlack.UB = totSlack.X
totSlack.LB = totSlack.X

' Variable to count the average number of shifts worked
Dim avgShifts As GRBVar = model.AddVar(0, GRB.INFINITY, 0, _
    GRB.CONTINUOUS, "avgShifts")

' Variables to count the difference from average for each worker;
' note that these variables can take negative values.
Dim diffShifts As GRBVar() = New GRBVar(nWorkers - 1) {}
For w As Integer = 0 To nWorkers - 1
    diffShifts(w) = _
        model.AddVar(-GRB.INFINITY, GRB.INFINITY, 0, _
            GRB.CONTINUOUS, Workers(w) & "Diff")
Next

' Constraint: compute the average number of shifts worked
lhs = 0
For w As Integer = 0 To nWorkers - 1
    lhs.AddTerm(1.0, totShifts(w))
Next
model.AddConstr(lhs = nWorkers * avgShifts, "avgShifts")

' Constraint: compute the difference from the average number of shifts
For w As Integer = 0 To nWorkers - 1
    model.AddConstr(totShifts(w) - avgShifts = diffShifts(w), _
        Workers(w) & "Diff")
Next

' Objective: minimize the sum of the square of the difference
' from the average number of shifts worked
Dim qobj As GRBQuadExpr = New GRBQuadExpr
For w As Integer = 0 To nWorkers - 1
    qobj.AddTerm(1.0, diffShifts(w), diffShifts(w))
Next
model.SetObjective(qobj)

' Optimize
status = solveAndPrint(model, totSlack, nWorkers, Workers, totShifts)
If status <> GRB.Status.OPTIMAL Then
    Exit Sub
End If

' Dispose of model and env
model.Dispose()
env.Dispose()

Catch e As GRBException
' shifts worked among all workers. The second optimization is allowed ' to degrade the first objective by up to the smaller value of 10% and 2 */

Imports System
Imports Gurobi

Class workforce5_vb
    Shared Sub Main()
        Try
            ' Sample data ' Sets of days and workers
            Dim Shifts As String() = New String() { _
                "Mon1", "Tue2", "Wed3", "Thu4", "Fri5", "Sat6", "Sun7", _
                "Mon8", "Tue9", "Wed10", "Thu11", "Fri12", "Sat13", "Sun14"}

            Dim Workers As String() = New String() { _

            Dim nShifts As Integer = Shifts.Length
            Dim nWorkers As Integer = Workers.Length

            ' Number of workers required for each shift
            Dim shiftRequirements As Double() = New Double() { _
                3, 2, 4, 4, 5, 6, 5, 2, 2, 3, 4, 6, 7, 5}

            ' Worker availability: 0 if the worker is unavailable for a shift
            Dim availability As Double(,) = New Double(,) { _
                {0, 1, 1, 0, 1, 0, 1, 1, 1, 1, 1, 1}, _
                {1, 1, 0, 0, 1, 1, 0, 0, 1, 1, 1, 1}, _
                {0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1}, _
                {0, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1}, _
                {1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1}, _
                {1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1}, _
                {0, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1}, _
                {1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1}, _
                {1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0}, _
                {0, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0}, _
                {0, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0}, _
                {1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0}}

            ' Create environment
            Dim env As New GRBEnv()

            ' Create initial model
            Dim model As New GRBModel(env)
            model.ModelName = "workforce5_vb"

            ' Initialize assignment decision variables:
            ' x[w][s] == 1 if worker w is assigned to shift s ' This is no longer a pure assignment model, so we must
            ' use binary variables.
            Dim x As GRBVar(,) = New GRBVar(nWorkers - 1, nShifts - 1) {}
            For w As Integer = 0 To nWorkers - 1
                For s As Integer = 0 To nShifts - 1
                    x(w, s) = model.AddVar(0, availability(w, s), 0, GRB.BINARY, _
                        String.Format("{0}.{1}", Workers(w), Shifts(s)))
                Next
            Next
' Slack variables for each shift constraint so that the shifts can be satisfied
Dim slacks As GRBVar() = New GRBVar(nShifts - 1) {}
For s As Integer = 0 To nShifts - 1
    slacks(s) = model.AddVar(0, GRB.INFINITY, 0, GRB.CONTINUOUS, 
        String.Format("{0} Slack", Shifts(s)))
Next

' Variable to represent the total slack
Dim totSlack As GRBVar = model.AddVar(0, GRB.INFINITY, 0, GRB.CONTINUOUS, "totSlack")

' Variables to count the total shifts worked by each worker
Dim totShifts As GRBVar() = New GRBVar(nWorkers - 1) {}
For w As Integer = 0 To nWorkers - 1
    totShifts(w) = model.AddVar(0, GRB.INFINITY, 0, GRB.CONTINUOUS, 
        String.Format("{0} TotShifts", Workers(w)))
Next

Dim lhs As GRBLinExpr

' Constraint: assign exactly shiftRequirements[s] workers to each shift s, plus the slack
For s As Integer = 0 To nShifts - 1
    lhs = New GRBLinExpr()
    lhs.AddTerm(1.0, slacks(s))
    For w As Integer = 0 To nWorkers - 1
        lhs.AddTerm(1.0, x(w, s))
    Next
    model.AddConstr(lhs, GRB.EQUAL, shiftRequirements(s), Shifts(s))
Next

' Constraint: set totSlack equal to the total slack
lhs = New GRBLinExpr()
lhs.AddTerm(-1.0, totSlack)
For s As Integer = 0 To nShifts - 1
    lhs.AddTerm(1.0, slacks(s))
Next
model.AddConstr(lhs, GRB.EQUAL, 0, "totSlack")

' Constraint: compute the total number of shifts for each worker
For w As Integer = 0 To nWorkers - 1
    lhs = New GRBLinExpr()
    lhs.AddTerm(-1.0, totShifts(w))
    For s As Integer = 0 To nShifts - 1
        lhs.AddTerm(1.0, x(w, s))
    Next
    model.AddConstr(lhs, GRB.EQUAL, 0, String.Format("totShifts{0}" , Workers(w)))
Next

' Constraint: set minShift/maxShift variable to less <=/>= to the number of shifts among all workers
Dim minShift As GRBVar = model.AddVar(0, GRB.INFINITY, 0, GRB.CONTINUOUS, "minShift")
Dim maxShift As GRBVar = model.AddVar(0, GRB.INFINITY, 0, GRB.CONTINUOUS, "maxShift")
model.AddGenConstrMin(minShift, totShifts, GRB.INFINITY, "minShift")
model.AddGenConstrMax(maxShift, totShifts, -GRB.INFINITY, "maxShift")
' Set global sense for ALL objectives
model.ModelSense = GRB.MINIMIZE

' Set primary objective
model.SetObjectiveN(totSlack, 0, 2, 1.0, 2.0, 0.1, "TotalSlack")

' Set secondary objective
model.SetObjectiveN(maxShift - minShift, 1, 1, 1.0, 0, 0, "Fairness")

' Save problem
model.Write("workforce5_vb.lp")

' Optimize
Dim status As Integer = _
solveAndPrint(model, totSlack, nWorkers, Workers, totShifts)
If status <> GRB.Status.OPTIMAL Then
    Return
End If

' Dispose of model and environment
model.Dispose()
env.Dispose()
Catch e As GRBException
    Console.WriteLine("Error code: {0}. {1}", e.ErrorCode, e.Message)
End Try
End Sub

Private Shared Function solveAndPrint(ByVal model As GRBModel, _
    ByVal totSlack As GRBVar, _
    ByVal nWorkers As Integer, _
    ByVal Workers As String(), _
    ByVal totShifts As GRBVar()) As Integer

    model.Optimize()
    Dim status As Integer = model.Status
    If status = GRB.Status.INF_OR_UNBD OrElse _
        status = GRB.Status.INFEASIBLE OrElse _
        status = GRB.Status.UNBOUNDED Then
        Console.WriteLine("The model cannot be solved " & _
            "because it is infeasible or unbounded")
        Return status
    End If
    If status <> GRB.Status.OPTIMAL Then
        Console.WriteLine("Optimization was stopped with status {0}", status)
        Return status
    End If

    ' Print total slack and the number of shifts worked for each worker
    Console.WriteLine(vbLf & "Total slack required: {0}", totSlack.X)
    For w As Integer = 0 To nWorkers - 1
        Console.WriteLine("{0} worked {1} shifts", Workers(w), totShifts(w).X)
    Next
    Console.WriteLine(vbLf)
3.6 Python Examples

This section includes source code for all of the Gurobi Python examples. The same source code can be found in the examples/python directory of the Gurobi distribution.

batchmode.py

#!/usr/bin/env python3.7
# Copyright 2020, Gurobi Optimization, LLC
#
# This example reads a MIP model from a file, solves it in batch mode,
# and prints the JSON solution string.
# You will need a Cluster Manager license for this example to work.

import sys
import time
import json
import gurobipy as gp
from gurobipy import GRB

# Set up the environment for batch mode optimization.
#
# The function creates an empty environment, sets all necessary parameters,
# and returns the ready-to-be-started Env object to caller. It is the
# caller's responsibility to dispose of this environment when it's no
# longer needed.
def setupbatchenv():
    env = gp.Env(empty=True)
    env.setParam('LogFile', 'batchmode.log')
    env.setParam('CSManager', 'http://localhost:61080')
    env.setParam('UserName', 'gurobi')
    env.setParam('ServerPassword', 'pass')
    env.setParam('CSBatchMode', 1)

    return env

# Print batch job error information, if any
def printbatcherrorinfo(batch):
    if batch is None or batch.BatchErrorCode == 0:
        return


```python

# Create a batch request for given problem file
def newbatchrequest(filename):
    # Start environment, create Model object from file
    # By using the context handlers for env and model, it is ensured that
    # model.dispose() and env.dispose() are called automatically
    with setupbatchenv().start() as env, gp.read(filename, env=env) as model:
        # Set some parameters
        model.Params.MIPGap = 0.01
        model.Params.JSONSolDetail = 1

        # Define tags for some variables in order to access their values later
        for count, v in enumerate(model.getVars()):
            v.VTag = "Variable{}".format(count)
            if count >= 10:
                break

        # Submit batch request
        batchID = model.optimizeBatch()
        return batchID

    # Wait for the final status of the batch.
    # Initially the status of a batch is "submitted"; the status will change
    # once the batch has been processed (by a compute server).
    def waitforfinalstatus(batchID):
        # Wait no longer than one hour
        maxwaittime = 3600

        # Setup and start environment, create local Batch handle object
        with setupbatchenv().start() as env, gp.Batch(batchID, env) as batch:
            starttime = time.time()
            while batch.BatchStatus == GRB.BATCH_SUBMITTED:
                # Abort this batch if it is taking too long
                curtime = time.time()
                if curtime - starttime > maxwaittime:
                    batch.abort()
                    break

            # Wait for two seconds
            time.sleep(2)

            # Update the resident attribute cache of the Batch object with the
            # latest values from the cluster manager.
            batch.update()

            # If the batch failed, we retry it
            if batch.BatchStatus == GRB.BATCH_FAILED:
                batch.retry()
```

451
# Print information about error status of the job that processed the batch
printbatcherrorinfo(batch)

def printfinalreport(batchID):
    # Setup and start environment, create local Batch handle object
    with setupbatchenv().start() as env, gp.Batch(batchID, env) as batch:
        if batch.BatchStatus == GRB.BATCH_CREATED:
            print("Batch status is 'CREATED'\")
        elif batch.BatchStatus == GRB.BATCH_SUBMITTED:
            print("Batch is 'SUBMITTED'\")
        elif batch.BatchStatus == GRB.BATCH_ABORTED:
            print("Batch is 'ABORTED'\")
        elif batch.BatchStatus == GRB.BATCH_FAILED:
            print("Batch is 'FAILED'\")
        elif batch.BatchStatus == GRB.BATCH_COMPLETED:
            print("Batch is 'COMPLETED'\")
            print("JSON solution:"
    # Get JSON solution as string, create dict from it
    sol = json.loads(batch.getJSONSolution())

    # Pretty printing the general solution information
    print(json.dumps(sol["SolutionInfo"], indent=4))

    # Write the full JSON solution string to a file
    batch.writeJSONSolution('batch-sol.json.gz')
    else:
        # Should not happen
        print("Batch has unknown BatchStatus")

    printbatcherrorinfo(batch)

# Instruct the cluster manager to discard all data relating to this BatchID
def batchdiscard(batchID):
    # Setup and start environment, create local Batch handle object
    with setupbatchenv().start() as env, gp.Batch(batchID, env) as batch:
        # Remove batch request from manager
        batch.discard()

# Solve a given model using batch optimization
if __name__ == '__main__':

    # Ensure we have an input file
    if len(sys.argv) < 2:
        print("Usage: {} filename".format(sys.argv[0]))
        quit()

    # Submit new batch request
    batchID = newbatchrequest(sys.argv[1])

    # Wait for final status
    waitforfinalstatus(batchID)
# Report final status info
printfinalreport(batchID)

# Remove batch request from manager
batchdiscard(batchID)

print('Batch optimization OK')

bilinear.py

#!/usr/bin/env python3.7

# Copyright 2020, Gurobi Optimization, LLC

# This example formulates and solves the following simple bilinear model:
# maximize x
# subject to x + y + z <= 10
# x * y <= 2 (bilinear inequality)
# x * z + y * z = 1 (bilinear equality)
# x, y, z non-negative (x integral in second version)

import gurobipy as gp
from gurobipy import GRB

# Create a new model
m = gp.Model("bilinear")

# Create variables
x = m.addVar(name="x")
y = m.addVar(name="y")
z = m.addVar(name="z")

# Set objective: maximize x
m.setObjective(1.0*x, GRB.MAXIMIZE)

# Add linear constraint: x + y + z <= 10
m.addConstr(x + y + z <= 10, "c0")

# Add bilinear inequality constraint: x * y <= 2
m.addConstr(x*y <= 2, "bilinear0")

# Add bilinear equality constraint: x * z + y * z == 1
m.addConstr(x*z + y*z == 1, "bilinear1")

# First optimize() call will fail - need to set NonConvex to 2
try:
    m.optimize()
except gp.GurobiError:
    print("Optimize failed due to non-convexity")

# Solve bilinear model
m.params.NonConvex = 2
m.optimize()

m.printAttr('x')
# Constrain 'x' to be integral and solve again
x.vType = GRB.INTEGER
m.optimize()
m.printAttr('x')

callback.py

#!/usr/bin/env python3.7

# Copyright 2020, Gurobi Optimization, LLC

# This example reads a model from a file, sets up a callback that
# monitors optimization progress and implements a custom
# termination strategy, and outputs progress information to the
# screen and to a log file.
# The termination strategy implemented in this callback stops the
# optimization of a MIP model once at least one of the following two
# conditions have been satisfied:
# 1) The optimality gap is less than 10%
# 2) At least 10000 nodes have been explored, and an integer feasible
#    solution has been found.
# Note that termination is normally handled through Gurobi parameters
# (MIPGap, NodeLimit, etc.). You should only use a callback for
# termination if the available parameters don't capture your desired
# termination criterion.

import sys
import gurobipy as gp
from gurobipy import GRB

# Define my callback function
def mycallback(model, where):
    if where == GRB.Callback.POLLING:
        # Ignore polling callback
        pass
    elif where == GRB.Callback.PRESOLVE:
        # Presolve callback
cdels = model.cbGet(GRB.Callback.PRE_COLDEL)
rdels = model.cbGet(GRB.Callback.PRE_ROWDEL)
if cdels or rdels:
    print('%d columns and %d rows are removed' % (cdels, rdels))
    elif where == GRB.Callback.SIMPLEX:
        # Simplex callback
itcnt = model.cbGet(GRB.Callback.SPX_ITRCNT)
if itcnt - model._lastiter >= 100:
    model._lastiter = itcnt
    obj = model.cbGet(GRB.Callback.SPX_OBJVAL)
    ispert = model.cbGet(GRB.Callback.SPX_ISPERT)
pinf = model.cbGet(GRB.Callback.SPX_PRIMINF)
dinf = model.cbGet(GRB.Callback.SPX_DUALINF)
if ispert == 0:
    ch = ' ',
elif ispert == 1:
    ch = 'S'
else:
    ch = 'P'
print('%d %g%s %g %g' % (int(itcnt), obj, ch, pinf, dinf))

elif where == GRB.Callback.MIP:
    # General MIP callback
    nodedcnt = model.cbGet(GRB.Callback.MIP_NODCNT)
    objbst = model.cbGet(GRB.Callback.MIP_OBJBST)
    objbnd = model.cbGet(GRB.Callback.MIP_OBJBND)
    solcnt = model.cbGet(GRB.Callback.MIP_SOLCNT)
    if nodedcnt - model._lastnode >= 100:
        model._lastnode = nodedcnt
        actnodes = model.cbGet(GRB.Callback.MIP_NODLFT)
        itcnt = model.cbGet(GRB.Callback.MIP_ITRCNT)
        cutcnt = model.cbGet(GRB.Callback.MIP_CUTCNT)
        print('%d %d %d %g %g %d %d' % (nodedcnt, actnodes, itcnt, objbst, objbnd, solcnt, cutcnt))
        if abs(objbst - objbnd) < 0.1 * (1.0 + abs(objbst)):
            print('Stop early - 10% gap achieved')
            model.terminate()
        if nodedcnt >= 10000 and solcnt:
            print('Stop early - 10000 nodes explored')
            model.terminate()

elif where == GRB.Callback.MIPSOL:
    # MIP solution callback
    nodedcnt = model.cbGet(GRB.Callback.MIPSOL_NODCNT)
    obj = model.cbGet(GRB.Callback.MIPSOL_OBJ)
    solcnt = model.cbGet(GRB.Callback.MIPSOL_SOLCNT)
    x = model.cbGetSolution(model._vars)
    print('**** New solution at node %d, obj %g, sol %d, %x[0] = %g ****' % (nodedcnt, obj, solcnt, x[0]))

elif where == GRB.Callback.MIPNODE:
    # MIP node callback
    print('**** New node ****
    if model.cbGet(GRB.Callback.MIPNODE_STATUS) == GRB.OPTIMAL:
        x = model.cbGetNodeRel(model._vars)
        model.cbSetSolution(model.getVars(), x)

elif where == GRB.Callback.BARRIER:
    # Barrier callback
    itcnt = model.cbGet(GRB.Callback.BARRIER_ITRCNT)
    primobj = model.cbGet(GRB.Callback.BARRIER_PRIMOBJ)
    dualobj = model.cbGet(GRB.Callback.BARRIER_DUALOBJ)
    priminf = model.cbGet(GRB.Callback.BARRIER_PRIMINF)
    dualinf = model.cbGet(GRB.Callback.BARRIER_DUALINF)
    cmpl = model.cbGet(GRB.Callback.BARRIER_COMPL)
    print('%d %g %g %g %g %g' % (itcnt, primobj, dualobj, priminf, dualinf, cmpl))

elif where == GRB.Callback.MESSAGE:
    # Message callback
    msg = model.cbGet(GRB.Callback.MSG_STRING)
    model._logfile.write(msg)

if len(sys.argv) < 2:
    print('Usage: callback.py filename')
quit()

# Turn off display and heuristics

gp.setParam('OutputFlag', 0)
gp.setParam('Heuristics', 0)

# Read model from file

model = gp.read(sys.argv[1])

# Open log file

logfile = open('cb.log', 'w')

# Pass data into my callback function

model._lastiter = -GRB.INFINITY
model._lastnode = -GRB.INFINITY
model._logfile = logfile
model._vars = model.getVars()

# Solve model and capture solution information

model.optimize(mycallback)

print('')
print('Optimization complete')
if model.SolCount == 0:
    print('No solution found, optimization status = %d' % model.Status)
else:
    print('Solution found, objective = %g' % model.ObjVal)
    for v in model.getVars():
        if v.X != 0.0:
            print('%s %g' % (v.VarName, v.X))

# Close log file

logfile.close()

custom.py

# Copyright 2020, Gurobi Optimization, LLC
#
# Interactive shell customization example
#
# Define a set of customizations for the Gurobi shell. Type 'from custom import *' to import them into your shell.
#
from gurobipy import *

# custom read command -- change directory as appropriate
def myread(name):
    return read('/home/jones/models/' + name)

# Custom termination criterion: Quit optimization
# - after 5s if a high quality (1% gap) solution has been found, or
# - after 10s if a feasible solution has been found.

def mycallback(model, where):
    if where == GRB.Callback.MIP:
        time = model.cbGet(GRB.Callback.RUNTIME)
        best = model.cbGet(GRB.Callback.MIP_OBJBST)
        bound = model.cbGet(GRB.Callback.MIP_OBJBND)

        if best < GRB.INFINITY:
            # We have a feasible solution
            if time > 5 and abs(bound - best) < 0.01 * abs(bound):
                model.terminate()

        if time > 10:
            model.terminate()

# custom optimize() function that uses callback

def myopt(model):
    model.optimize(mycallback)

if __name__ == "__main__":
    # Use as customized command line tool
    import sys
    if len(sys.argv) != 2:
        print("Usage: python custom.py <model>")

    m = read(sys.argv[1])
    myopt(m)

dense.py

#!/usr/bin/env python3.7

# Copyright 2020, Gurobi Optimization, LLC

# This example formulates and solves the following simple QP model:
# #
# # minimize     x + y + x^2 + x*y + y^2 + y*z + z^2
# # subject to  x + 2 y + 3 z >= 4
# #              x + y >= 1
# #              x, y, z non-negative
# #
# # The example illustrates the use of dense matrices to store A and Q
# # (and dense vectors for the other relevant data). We don't recommend
# # that you use dense matrices, but this example may be helpful if you
# # already have your data in this format.
import sys
import gurobipy as gp
from gurobipy import GRB

def dense_optimize(rows, cols, c, Q, A, sense, rhs, lb, ub, vtype, solution):
    model = gp.Model()
    # Add variables to model
    vars = []
    for j in range(cols):
        vars.append(model.addVar(lb=lb[j], ub=ub[j], vtype=vtype[j]))
    # Populate A matrix
    for i in range(rows):
        expr = gp.LinExpr()
        for j in range(cols):
            if A[i][j] != 0:
                expr += A[i][j]*vars[j]
        model.addConstr(expr, sense[i], rhs[i])
    # Populate objective
    obj = gp.QuadExpr()
    for i in range(cols):
        for j in range(cols):
            if Q[i][j] != 0:
                obj += Q[i][j]*vars[i]*vars[j]
        for j in range(cols):
            if c[j] != 0:
                obj += c[j]*vars[j]
    model.setObjective(obj)
    # Solve
    model.optimize()
    # Write model to a file
    model.write('dense.lp')
    if model.status == GRB.OPTIMAL:
        x = model.getAttr('x', vars)
        for i in range(cols):
            solution[i] = x[i]
        return True
    else:
        return False

# Put model data into dense matrices

c = [1, 1, 0]
Q = [[1, 1, 0], [0, 1, 1], [0, 0, 1]]
A = [[1, 2, 3], [1, 1, 0]]
sense = [GRB.GREATER_EQUAL, GRB.GREATER_EQUAL]
rhs = [4, 1]
lb = [0, 0, 0]
ub = [GRB.INFINITY, GRB.INFINITY, GRB.INFINITY]
vtype = [GRB.CONTINUOUS, GRB.CONTINUOUS, GRB.CONTINUOUS]
sol = [0]*3

# Optimize

success = dense_optimize(2, 3, c, Q, A, sense, rhs, lb, ub, vtype, sol)

if success:
    print('x: %g, y: %g, z: %g' % (sol[0], sol[1], sol[2]))

# diet.py

#! /usr/bin/env python3.7

# Copyright 2020, Gurobi Optimization, LLC

# Solve the classic diet model, showing how to add constraints
# to an existing model.

import gurobipy as gp
from gurobipy import GRB

categories, minNutrition, maxNutrition = gp.multidict({
    'calories': [1800, 2200],
    'protein': [91, GRB.INFINITY],
    'fat': [0, 65],
    'sodium': [0, 1779])

foods, cost = gp.multidict({
    'hamburger': 2.49,
    'chicken': 2.89,
    'hot dog': 1.50,
    'fries': 1.89,
    'macaroni': 2.09,
    'pizza': 1.99,
    'salad': 2.49,
    'milk': 0.89,
    'ice cream': 1.59})

# Nutrition values for the foods
nutritionValues = {
    ('hamburger', 'calories'): 410,
    ('hamburger', 'protein'): 24,
    ('hamburger', 'fat'): 26,
    ('hamburger', 'sodium'): 730,
    ('chicken', 'calories'): 420,
    ('chicken', 'protein'): 32,
    ('chicken', 'fat'): 10,
    ('chicken', 'sodium'): 1190,
# Model
m = gp.Model("diet")

# Create decision variables for the foods to buy
buy = m.addVars(foods, name="buy")

# You could use Python looping constructs and m.addVar() to create
# these decision variables instead. The following would be equivalent
#
# buy = {}
# for f in foods:
#     buy[f] = m.addVar(name=f)

# The objective is to minimize the costs
m.setObjective(buy.prod(cost), GRB.MINIMIZE)

# Using looping constructs, the preceding statement would be:
#
# m.setObjective(sum(buy[f]*cost[f] for f in foods), GRB.MINIMIZE)

# Nutrition constraints
m.addConstrs((gp.quicksum(nutritionValues[f, c] * buy[f] for f in foods) == [minNutrition[c], maxNutrition[c]]
    for c in categories), "_")

# Using looping constructs, the preceding statement would be:
#
# for c in categories:
#   m.addRange(sum(nutritionValues[f, c] * buy[f] for f in foods),
#             minNutrition[c], maxNutrition[c], c)

def printSolution():
    if m.status == GRB.OPTIMAL:
        print('\nCost: %g' % m.objVal)
        print('\nBuy: ')
        buyx = m.getAttr('x', buy)
        for f in foods:
            if buyx[f].x > 0.0001:
                print('%s %g' % (f, buyx[f]))
            else:
                print('No solution')

# Solve
m.optimize()
printSolution()

print('\nAdding constraint: at most 6 servings of dairy')
m.addConstr(buy.sum(['milk', 'ice cream']) <= 6, "limit_dairy")

# Solve
m.optimize()
printSolution()

diet2.py

#!/usr/bin/env python3.7

# Copyright 2020, Gurobi Optimization, LLC

# Separate the model (dietmodel.py) from the data file (diet2.py), so
# that the model can be solved with different data files.
#
# Nutrition guidelines, based on
# USDA Dietary Guidelines for Americans, 2005

import dietmodel
import gurobipy as gp
from gurobipy import GRB

categories, minNutrition, maxNutrition = gp.multidict({
    'calories': [1800, 2200],
    'protein': [91, GRB.INFINITY],
    'fat': [0, 65],
    'sodium': [0, 1779]})

foods, cost = gp.multidict({
    'hamburger': 2.49,
    'chicken': 2.89,
    'hot dog': 1.50,
{'fries': 1.89,
'macaroni': 2.09,
'pizza': 1.99,
'salad': 2.49,
'milk': 0.89,
'ice cream': 1.59}

# Nutrition values for the foods
nutritionValues = {
    ('hamburger', 'calories'): 410,
    ('hamburger', 'protein'): 24,
    ('hamburger', 'fat'): 26,
    ('hamburger', 'sodium'): 730,
    ('chicken', 'calories'): 420,
    ('chicken', 'protein'): 32,
    ('chicken', 'fat'): 10,
    ('chicken', 'sodium'): 1190,
    ('hot dog', 'calories'): 560,
    ('hot dog', 'protein'): 20,
    ('hot dog', 'fat'): 32,
    ('hot dog', 'sodium'): 1800,
    ('fries', 'calories'): 380,
    ('fries', 'protein'): 4,
    ('fries', 'fat'): 19,
    ('fries', 'sodium'): 270,
    ('macaroni', 'calories'): 320,
    ('macaroni', 'protein'): 12,
    ('macaroni', 'fat'): 10,
    ('macaroni', 'sodium'): 930,
    ('pizza', 'calories'): 320,
    ('pizza', 'protein'): 15,
    ('pizza', 'fat'): 12,
    ('pizza', 'sodium'): 820,
    ('salad', 'calories'): 320,
    ('salad', 'protein'): 31,
    ('salad', 'fat'): 12,
    ('salad', 'sodium'): 1230,
    ('milk', 'calories'): 100,
    ('milk', 'protein'): 8,
    ('milk', 'fat'): 2.5,
    ('milk', 'sodium'): 125,
    ('ice cream', 'calories'): 330,
    ('ice cream', 'protein'): 8,
    ('ice cream', 'fat'): 10,
    ('ice cream', 'sodium'): 180
}

dietmodel.solve(categories, minNutrition, maxNutrition,
    foods, cost, nutritionValues)

diet3.py

#!/usr/bin/env python3.7

# Copyright 2020, Gurobi Optimization, LLC

# Use a SQLite database with the diet model (dietmodel.py). The database
(diet.db) can be recreated using the included SQL script (diet.sql).

Note that this example reads an external data file (..\data\diet.db).
As a result, it must be run from the Gurobi examples/python directory.

```python
import os
import sqlite3
import dietmodel
import gurobipy as gp

con = sqlite3.connect(os.path.join '..', 'data', 'diet.db'))
cur = con.cursor()

cur.execute('select category, minnutrition, maxnutrition from categories')
result = cur.fetchall()
categories, minNutrition, maxNutrition = gp.multidict(
    (cat, [minv, maxv]) for cat, minv, maxv in result)

cur.execute('select food, cost from foods')
result = cur.fetchall()
foods, cost = gp.multidict(result)

cur.execute('select food, category, value from nutrition')
result = cur.fetchall()
nutritionValues = dict(((f, c), v) for f, c, v in result)

con.close()

dietmodel.solve(categories, minNutrition, maxNutrition,
    foods, cost, nutritionValues)
```

diet4.py

#!/usr/bin/env python3.7

# Copyright 2020, Gurobi Optimization, LLC

# Read diet model data from an Excel spreadsheet (diet.xls).
# Pass the imported data into the diet model (dietmodel.py).
# Note that this example reads an external data file (..\data\diet.xls).
# As a result, it must be run from the Gurobi examples/python directory.
# This example requires Python package 'xlrd', which isn't included
# in most Python distributions. You can obtain it from

import os
import xlrd
import dietmodel

book = xlrd.open_workbook(os.path.join '..', 'data', 'diet.xls'))
sh = book.sheet_by_name("Categories")
categories = []
minNutrition = {}
maxNutrition = {}
i = 1
while True:
    try:
        c = sh.cell_value(i, 0)
        categories.append(c)
        minNutrition[c] = sh.cell_value(i, 1)
        maxNutrition[c] = sh.cell_value(i, 2)
        i = i + 1
    except IndexError:
        break

sh = book.sheet_by_name("Foods")
foods = []
cost = {}
i = 1
while True:
    try:
        f = sh.cell_value(i, 0)
        foods.append(f)
        cost[f] = sh.cell_value(i, 1)
        i = i + 1
    except IndexError:
        break

sh = book.sheet_by_name("Nutrition")
nutritionValues = {}
i = 1
for food in foods:
    j = 1
    for cat in categories:
        nutritionValues[food, cat] = sh.cell_value(i, j)
        j += 1
    i += 1

dietmodel.solve(categories, minNutrition, maxNutrition, foods, cost, nutritionValues)

dietmodel.py

#!/usr/bin/env python3.7

# Copyright 2020, Gurobi Optimization, LLC

# Solve the classic diet model. This file implements
# a function that formulates and solves the model,
# but it contains no model data. The data is
# passed in by the calling program. Run example 'diet2.py',
# 'diet3.py', or 'diet4.py' to invoke this function.

import gurobipy as gp
from gurobipy import GRB
def solve(categories, minNutrition, maxNutrition, foods, cost, nutritionValues):
    # Model
    m = gp.Model("diet")

    # Create decision variables for the foods to buy
    buy = m.addVars(foods, name="buy")

    # The objective is to minimize the costs
    m.setObjective(buy.prod(cost), GRB.MINIMIZE)

    # Nutrition constraints
    m.addConstrs((gp.quicksum(nutritionValues[f, c] * buy[f] for f in foods) ==
                   [minNutrition[c], maxNutrition[c]] for c in categories), "_")

    def printSolution():
        if m.status == GRB.OPTIMAL:
            print(\nCost: %g' % m.objVal)
            print(\nBuy:)
            buyx = m.getAttr('x', buy)
            for f in foods:
                if buy[f].x > 0.0001:
                    print(\%s %g' % (f, buyx[f]))
        else:
            print(\No solution'）

    # Solve
    m.optimize()
    printSolution()

    print(\nAdding constraint: at most 6 servings of dairy')
    m.addConstr(buy.sum(['milk', 'ice cream']) <= 6, "limit_dairy")

    # Solve
    m.optimize()
    printSolution()

facility.py

#!/usr/bin/env python3.7

# Copyright 2020, Gurobi Optimization, LLC

# Facility location: a company currently ships its product from 5 plants
# to 4 warehouses. It is considering closing some plants to reduce
# costs. What plant(s) should the company close, in order to minimize
# transportation and fixed costs?
# #
# # Note that this example uses lists instead of dictionaries. Since
# # it does not work with sparse data, lists are a reasonable option.
# #
# # Based on an example from Frontline Systems:
# # http://www.solver.com/disfacility.htm
# # Used with permission.

import gurobipy as gp
from gurobipy import GRB

# Warehouse demand in thousands of units
demand = [15, 18, 14, 20]

# Plant capacity in thousands of units
capacity = [20, 22, 17, 19, 18]

# Fixed costs for each plant
fixedCosts = [12000, 15000, 17000, 13000, 16000]

# Transportation costs per thousand units
transCosts = [[4000, 2000, 3000, 2500, 4500],
              [2500, 2600, 3400, 3000, 4000],
              [1200, 1800, 2600, 4100, 3000],
              [2200, 2600, 3100, 3700, 3200]]

# Range of plants and warehouses
plants = range(len(capacity))
warehouses = range(len(demand))

# Model
m = gp.Model("facility")

# Plant open decision variables: open[p] == 1 if plant p is open.
open = m.addVars(plants,
                 vtype=GRB.BINARY,
                 obj=fixedCosts,
                 name="open")

# Transportation decision variables: transport[w,p] captures the
# optimal quantity to transport to warehouse w from plant p
transport = m.addVars(warehouses, plants,
                      obj=transCosts,
                      name="trans")

# You could use Python looping constructs and m.addVar() to create
# these decision variables instead. The following would be equivalent
# to the preceding two statements...

# open = []
# for p in plants:
#     open.append(m.addVar(vtype=GRB.BINARY,
#                          obj=fixedCosts[p],
#                          name="open[%d]" % p))
#
# transport = []
# for w in warehouses:
#     transport.append([])
# for p in plants:
#     transport[w].append(m.addVar(obj=transCosts[w][p],
#                                   name="trans[%d,%d]" % (w, p)))

# The objective is to minimize the total fixed and variable costs
m.modelSense = GRB.MINIMIZE

# Production constraints
# Note that the right-hand limit sets the production to zero if the plant
# is closed
m.addConstrs(
    (transport.sum('i', p) <= capacity[p]*open[p] for p in plants), "Capacity")

# Using Python looping constructs, the preceding would be...
#
# for p in plants:
#    m.addConstr(sum(transport[w][p] for w in warehouses) 
#    <= capacity[p] * open[p], "Capacity[%d]" % p)

# Demand constraints
m.addConstrs(
    (transport.sum(w) == demand[w] for w in warehouses),
    "Demand")

# ... and the preceding would be ...
# for w in warehouses:
#    m.addConstr(sum(transport[w][p] for p in plants) == demand[w],
#    "Demand[%d]" % w)

# Save model
m.write('facilityPY.lp')

# Guess at the starting point: close the plant with the highest fixed costs;
# open all others

# First open all plants
for p in plants:
    open[p].start = 1.0

# Now close the plant with the highest fixed cost
print('Initial guess: ')
maxFixed = max(fixedCosts)
for p in plants:
    if fixedCosts[p] == maxFixed:
        open[p].start = 0.0
        print('Closing plant %s' % p)
        break
print('')

# Use barrier to solve root relaxation
m.Params.method = 2
m.optimize()

# Print solution
print('\nTOTAL COSTS: %g' % m.objVal)
print('SOLUTION: ')
for p in plants:
    if open[p].x > 0.99:
        print('Plant %s open' % p)
        for w in warehouses:
            if transport[w, p].x > 0:
                print(' Transport %g units to warehouse %s' % 
...
(transport[w, p].x, w))

else:
    print('Plant %s closed!' % p)

feasopt.py

#!/usr/bin/env python3.7

# Copyright 2020, Gurobi Optimization, LLC

# This example reads a MIP model from a file, adds artificial
# variables to each constraint, and then minimizes the sum of the
# artificial variables. A solution with objective zero corresponds
# to a feasible solution to the input model.
#
# We can also use FeasRelax feature to do it. In this example, we
# use minrelax=1, i.e. optimizing the returned model finds a solution
# that minimizes the original objective, but only from among those
# solutions that minimize the sum of the artificial variables.

import sys
import gurobipy as gp

if len(sys.argv) < 2:
    print('Usage: feasopt.py filename')
    quit()

feasmodel = gp.read(sys.argv[1])

# create a copy to use FeasRelax feature later

feasmodel1 = feasmodel.copy()

# clear objective

feasmodel.setObjective(0.0)

# add slack variables

for c in feasmodel.getConstrs():
    sense = c.sense
    if sense != '>':
        feasmodel.addVar(obj=1.0, name="ArtN_" + c.constrName,
                         column=gp.Column([-1], [c]))
    if sense != '<':
        feasmodel.addVar(obj=1.0, name="ArtP_" + c.constrName,
                         column=gp.Column([1], [c]))

# optimize modified model

feasmodel.optimize()

feasmodel.write('feasopt.lp')

# use FeasRelax feature

468
feasmodel1.feasRelaxS(0, True, False, True)
feasmodel1.write("feaso1.lp")
feasmodel1.optimize()

fixanddive.py

#!/usr/bin/env python3.7
# Copyright 2020, Gurobi Optimization, LLC
#
# Implement a simple MIP heuristic. Relax the model, sort variables
# based on fractionality, and fix the 25% of the fractional variables
# that are closest to integer variables. Repeat until either the
# relaxation is integer feasible or linearly infeasible.

import sys
import gurobipy as gp
from gurobipy import GRB

# Key function used to sort variables based on relaxation fractionality
def sortkey(v1):
    sol = v1.x
    return abs(sol - int(sol + 0.5))

if len(sys.argv) < 2:
    print('Usage: fixanddive.py filename')
    quit()

# Read model
model = gp.read(sys.argv[1])

# Collect integer variables and relax them
intvars = []
for v in model.getVars():
    if v.vType != GRB.CONTINUOUS:
        intvars += [v]
        v.vType = GRB.CONTINUOUS

model.Params.outputFlag = 0
model.optimize()

# Perform multiple iterations. In each iteration, identify the first
# quartile of integer variables that are closest to an integer value in the
# relaxation, fix them to the nearest integer, and repeat.
for iter in range(1000):
    # create a list of fractional variables, sorted in order of increasing
# distance from the relaxation solution to the nearest integer value

    fractional = []
    for v in intvars:
        sol = v.x
        if abs(sol - int(sol+0.5)) > 1e-5:
            fractional += [v]
    fractional.sort(key=sortkey)

    print('Iteration %d, obj %g, fractional %d' %
          (iter, model.objVal, len(fractional)))

    if len(fractional) == 0:
        print('Found feasible solution - objective %g' % model.objVal)
        break

# Fix the first quartile to the nearest integer value
    nfix = max(int(len(fractional)/4), 1)
    for i in range(nfix):
        v = fractional[i]
        fixval = int(v.x+0.5)
        v.lb = fixval
        v.ub = fixval
        print(' Fix %s to %g ( rel %g) ' % (v.varName, fixval, v.x))
    model.optimize()

# Check optimization result
    if model.status != GRB.OPTIMAL:
        print('Relaxation is infeasible')
        break

---

gc_pwl.py

#!/usr/bin/env python3.7

# Copyright 2020, Gurobi Optimization, LLC

# This example formulates and solves the following simple model
# with PWL constraints:
#  
# maximize
#   sum c[j] * x[j]
# subject to
#   sum A[i,j] * x[j] <= 0, for i = 0, ..., m-1
#   sum y[j] <= 3
#   y[j] = pwl(x[j]), for j = 0, ..., n-1
#   x[j] free, y[j] >= 0, for j = 0, ..., n-1
# where pwl(x) = 0, if x = 0
#     = 1+|x|, if x != 0
#  
# Note
#  1. sum pwl(x[j]) <= b is to bound x vector and also to favor sparse x vector.
Here $b = 3$ means that at most two $x[j]$ can be nonzero and if two, then
\[ \sum x[j] \leq 1 \]

2. \(pwl(x)\) jumps from 1 to 0 and from 0 to 1, if $x$ moves from negative 0 to 0,
then to positive 0, so we need three points at $x = 0$. $x$ has infinite bounds
on both sides, the piece defined with two points $(-1, 2)$ and $(0, 1)$ can
extend $x$ to $-\infty$. Overall we can use five points $(-1, 2), (0, 1),
(0, 0), (0, 1)$ and $(1, 2)$ to define $y = pwl(x)$

```python
import gurobipy as gp
from gurobipy import GRB

try:
    n = 5
    m = 5
    c = [0.5, 0.8, 0.5, 0.1, -1]
    A = [[0, 0, 0, 1, -1],
         [0, 0, 1, 1, -1],
         [1, 1, 0, 0, -1],
         [1, 0, 1, 0, -1],
         [1, 0, 0, 1, -1]]

    # Create a new model
    model = gp.Model("gc_pwl")

    # Create variables
    x = model.addVars(n, lb=-GRB.INFINITY, name="x")
    y = model.addVars(n, name="y")

    # Set objective
    model.setObjective(gp.quicksum(c[j]*x[j] for j in range(n)), GRB.MAXIMIZE)

    # Add Constraints
    for i in range(m):
        model.addConstr(gp.quicksum(A[i][j]*x[j] for j in range(n)) <= 0)
    model.addConstr(y.sum() <= 3)

    for j in range(n):
        model.addGenConstrPWL(x[j], y[j], [-1, 0, 0, 0, 1], [2, 1, 0, 1, 2])

    # Optimize model
    model.optimize()

    for j in range(n):
        print('%s = %g' % (x[j].varName, x[j].x))

    print('Obj: %g' % model.objVal)

except gp.GurobiError as e:
    print('Error code ' + str(e.errno) + ': ' + str(e))

except AttributeError:
    print('Encountered an attribute error')
```

#!/usr/bin/env python3.7

# Copyright 2020, Gurobi Optimization, LLC

# This example considers the following nonconvex nonlinear problem
# maximize 2 x + y
# subject to exp(x) + 4 sqrt(y) <= 9
# x, y >= 0
#
# We show you two approaches to solve this:
#
# 1) Use a piecewise-linear approach to handle general function
# constraints (such as exp and sqrt).
# a) Add two variables
#    u = exp(x)
#    v = sqrt(y)
# b) Compute points (x, u) of u = exp(x) for some step length (e.g., x
#    = 0, 1e-3, 2e-3, ..., xmax) and points (y, v) of v = sqrt(y) for
#    some step length (e.g., y = 0, 1e-3, 2e-3, ..., ymax). We need to
#    compute xmax and ymax (which is easy for this example, but this
#    does not hold in general).
# c) Use the points to add two general constraints of type
#    piecewise-linear.
#
# 2) Use the Gurobi's built-in general function constraints directly (EXP
#    and POW). Here, we do not need to compute the points and the maximal
#    possible values, which will be done internally by Gurobi. In this
#    approach, we show how to "zoom in" on the optimal solution and
#    tighten tolerances to improve the solution quality.
#
import math
import gurobipy as gp
from gurobipy import GRB

def printsol(m, x, y, u, v):
    print('x = ' + str(x.x) + ', u = ' + str(u.x))
    print('y = ' + str(y.x) + ', v = ' + str(v.x))
    print('Obj = ' + str(m.objVal))

    # Calculate violation of exp(x) + 4 sqrt(y) <= 9
    vio = math.exp(x.x) + 4 * math.sqrt(y.x) - 9
    if vio < 0:
        vio = 0
    print('Vio = ' + str(vio))

try:

    # Create a new model
    m = gp.Model()
# Create variables
x = m.addVar(name='x')
y = m.addVar(name='y')
u = m.addVar(name='u')
v = m.addVar(name='v')

# Set objective
m.setObjective(2*x + y, GRB.MAXIMIZE)

# Add constraints
lc = m.addConstr(u + 4*v <= 9)

# Approach 1) PWL constraint approach
xpts = []
ypts = []
upts = []
vpts = []
intv = 1e-3
xmax = math.log(9)
t = 0.0
while t < xmax + intv:
xpts.append(t)
upts.append(math.exp(t))
t += intv

ymax = (9.0/4)*(9.0/4)
t = 0.0
while t < ymax + intv:
ypts.append(t)
vpts.append(math.sqrt(t))
t += intv

gc1 = m.addGenConstrPWL(x, u, xpts, upts, "gc1")
gc2 = m.addGenConstrPWL(y, v, ypts, vpts, "gc2")

# Optimize the model
m.optimize()

printsol(m, x, y, u, v)

# Approach 2) General function constraint approach with auto PWL translation by Gurobi

# restore unsolved state and get rid of PWL constraints
m.reset()
m.remove(gc1)
m.remove(gc2)
m.update()

# u = exp(x)
afc1 = m.addGenConstrExp(x, u, name="afc1")
# v = x^0.5
afc2 = m.addGenConstrPow(y, v, 0.5, name="afc2")
# Use the equal piece length approach with the length = 1e-3
m.params.FuncPieces = 1
m.params.FuncPieceLength = 1e-3

# Optimize the model
m.optimize()

printsol(m, x, y, u, v)

# Zoom in, use optimal solution to reduce the ranges and use a smaller
# pclen=1-5 to solve it
x.lb = max(x.lb, x.x-0.01)
x.ub = min(x.ub, x.x+0.01)
y.lb = max(y.lb, y.x-0.01)
y.ub = min(y.ub, y.x+0.01)
m.update()
m.reset()

m.params.FuncPieceLength = 1e-5

# Optimize the model
m.optimize()

printsol(m, x, y, u, v)

except gp.GurobiError as e:
    print("Error code ' + str(e.errno) + ': " + str(e))
except AttributeError:
    print('Encountered an attribute error')

genconstr.py

#!/usr/bin/env python3.7

# Copyright 2020, Gurobi Optimization, LLC

# In this example we show the use of general constraints for modeling
# some common expressions.  We use as an example a SAT problem where we
# want to see if it is possible to satisfy at least four (or all) clauses
# of the logical for
#
# L = (x0 or ~x1 or x2) and (x1 or ~x2 or x3) and
#    (x2 or ~x3 or x0) and (x3 or ~x0 or x1) and
#    (~x0 or ~x1 or x2) and (~x1 or ~x2 or x3) and
#    (~x2 or ~x3 or x0) and (~x3 or ~x0 or x1)
#
# We do this by introducing two variables for each literal (itself and its
# negated value), a variable for each clause, and then two
# variables for indicating if we can satisfy four, and another to identify
# the minimum of the clauses (so if it is one, we can satisfy all clauses)
# and put these two variables in the objective.
# i.e. the Objective function will be
#
# maximize Obj0 + Obj1
#
# Obj0 = MIN(Clause1, ..., Clause8)
# Obj1 = 1 -> Clause1 + ... + Clause8 >= 4
#
# thus, the objective value will be two if and only if we can satisfy all
# clauses; one if and only if at least four clauses can be satisfied, and
# zero otherwise.

import gurobipy as gp
from gurobipy import GRB
import sys

try:
    NLITERALS = 4
    n = NLITERALS

    # Example data:
    # e.g. {0, n+1, 2} means clause (x0 or ~x1 or x2)
    Clauses = [[0, n+1, 2],
                [1, n+2, 3],
                [2, n+3, 0],
                [3, n+0, 1],
                [n+0, n+1, 2],
                [n+1, n+2, 3],
                [n+2, n+3, 0],
                [n+3, n+0, 1]]

    # Create a new model
    model = gp.Model("Genconstr")

    # initialize decision variables and objective
    Lit = model.addVars(NLITERALS, vtype=GRB.BINARY, name="X")
    NotLit = model.addVars(NLITERALS, vtype=GRB.BINARY, name="NotX")

    Cla = model.addVars(len(Clauses), vtype=GRB.BINARY, name="Clause")

    Obj0 = model.addVar(vtype=GRB.BINARY, name="Obj0")
    Obj1 = model.addVar(vtype=GRB.BINARY, name="Obj1")

    # Link Xi and notXi
    model.addConstrs((Lit[i] + NotLit[i] == 1.0 for i in range(NLITERALS)),
                     name="CNSTR_X")

    # Link clauses and literals
    for i, c in enumerate(Clauses):
        clause = []
        for l in c:
            if l >= n:
                clause.append(NotLit[l-n])
            else:
                clause.append(Lit[l])
        model.addConstr(Cla[i] == gp.or_(clause), "CNSTR_Clause" + str(i))

    # Link objs with clauses
model.addConstr(Obj0 == gp.min_(Cla), name="CNSTR_Obj0")
model.addConstr((Obj1 == 1) >> (Cla.sum() >= 4.0), name="CNSTR_Obj1")

# Set optimization objective
model.setObjective(Obj0 + Obj1, GRB.MAXIMIZE)

# Save problem
model.write("genconstr.mps")
model.write("genconstr.lp")

# Optimize
model.optimize()

# Status checking
status = model.getAttr(GRB.Attr.Status)

if status in (GRB.INF_OR_UNBD, GRB.INFEASIBLE, GRB.UNBOUNDED):
    print("The model cannot be solved because it is infeasible or "
          "unbounded")
sys.exit(1)

if status != GRB.OPTIMAL:
    print("Optimization was stopped with status ", status)
sys.exit(1)

# Print result
objval = model.getAttr(GRB.Attr.ObjVal)

if objval > 1.9:
    print("Logical expression is satisfiable")
elif objval > 0.9:
    print("At least four clauses can be satisfied")
else:
    print("Not even three clauses can be satisfied")

except gp.GurobiError as e:
    print('Error code ' + str(e.errno) + ': ' + str(e))

except AttributeError:
    print('Encountered an attribute error')

lp.py

#!/usr/bin/env python3.7

# Copyright 2020, Gurobi Optimization, LLC

# This example reads an LP model from a file and solves it.
# If the model is infeasible or unbounded, the example turns off
# presolve and solves the model again. If the model is infeasible,
# the example computes an Irreducible Inconsistent Subsystem (IIS),
# and writes it to a file

import sys
import gurobipy as gp
from gurobipy import GRB
if len(sys.argv) < 2:
    print('Usage: lp.py filename')
    quit()

# Read and solve model
model = gp.read(sys.argv[1])
model.optimize()

if model.status == GRB.INF_OR_UNBD:
    # Turn presolve off to determine whether model is infeasible
    # or unbounded
    model.setParam(GRB.Param.Presolve, 0)
    model.optimize()

if model.status == GRB.OPTIMAL:
    print('Optimal objective: %g' % model.objVal)
    model.write('model.sol')
    sys.exit(0)
else:
    print('Optimization was stopped with status %d' % model.status)
    sys.exit(0)

# Model is infeasible - compute an Irreducible Inconsistent Subsystem (IIS)

print('Model is infeasible')
model.computeIIS()
model.write("model.ilp")
print("IIS written to file 'model.ilp'")

lpmethod.py

#!/usr/bin/env python3.7

# Copyright 2020, Gurobi Optimization, LLC

# Solve a model with different values of the Method parameter;
# show which value gives the shortest solve time.

import sys
import gurobipy as gp
from gurobipy import GRB

if len(sys.argv) < 2:
    print('Usage: lpmethod.py filename')
    quit()

# Read model
m = gp.read(sys.argv[1])

# Solve the model with different values of Method
bestTime = m.Params.timeLimit
bestMethod = -1
for i in range(3):
    m.reset()
    m.Params.method = i
    m.optimize()
    if m.status == GRB.OPTIMAL:
        bestTime = m.Runtime
        bestMethod = i
        # Reduce the TimeLimit parameter to save time with other methods
        m.Params.timeLimit = bestTime

# Report which method was fastest
if bestMethod == -1:
    print('Unable to solve this model')
else:
    print('Solved in %g seconds with Method %d' % (bestTime, bestMethod))

lpmod.py

#!/usr/bin/env python3.7

# Copyright 2020, Gurobi Optimization, LLC

# This example reads an LP model from a file and solves it.
# If the model can be solved, then it finds the smallest positive variable,
# sets its upper bound to zero, and resolves the model two ways:
# first with an advanced start, then without an advanced start
# (i.e. 'from scratch').

import sys
import gurobipy as gp
from gurobipy import GRB

if len(sys.argv) < 2:
    print('Usage: lpmod.py filename')
    quit()

# Read model and determine whether it is an LP

model = gp.read(sys.argv[1])
if model.isMIP == 1:
    print('The model is not a linear program')
    sys.exit(1)

model.optimize()

status = model.status

if status == GRB.INF_OR_UNBD or status == GRB.INFEASIBLE \
    or status == GRB.UNBOUNDED:
    print('The model cannot be solved because it is infeasible or unbounded')
    sys.exit(1)

if status != GRB.OPTIMAL:
    print('Optimization was stopped with status %d' % status)
    sys.exit(0)
# Find the smallest variable value
minVal = GRB.INFINITY
for v in model.getVars():
    if v.x > 0.0001 and v.x < minVal and v.lb == 0.0:
        minVal = v.x
        minVar = v

print('
*** Setting %s from %g to zero ***

' % (minVar.varName, minVal))
minVar.ub = 0.0

# Solve from this starting point
model.optimize()

# Save iteration & time info
warmCount = model.IterCount
warmTime = model.Runtime

# Reset the model and resolve
print('
*** Resetting and solving without an advanced start ***

')
model.reset()
model.optimize()

coldCount = model.IterCount
coldTime = model.Runtime

print('')
print('*** Warm start: %g iterations, %g seconds

' % (warmCount, warmTime))
print('*** Cold start: %g iterations, %g seconds

' % (coldCount, coldTime))
# Set objective
obj = np.array([1.0, 1.0, 2.0])
m.setObjective(obj @ x, GRB.MAXIMIZE)

# Build (sparse) constraint matrix
data = np.array([1.0, 2.0, 3.0, -1.0, -1.0])
row = np.array([0, 0, 0, 1, 1])
col = np.array([0, 1, 2, 0, 1])
A = sp.csr_matrix((data, (row, col)), shape=(2, 3))

# Build rhs vector
rhs = np.array([4.0, -1.0])

# Add constraints
m.addConstr(A @ x <= rhs, name="c")

# Optimize model
m.optimize()

print(x.X)
print('Obj : %g' % m.objVal)

except gp.GurobiError as e:
    print('Error code ' + str(e.errno) + ': ' + str(e))

except AttributeError:
    print('Encountered an attribute error')

matrix2.py

#!/usr/bin/env python3.7

# Copyright 2020, Gurobi Optimization, LLC

# This example uses the Python matrix API to formulate the n-queens
# problem; it maximizes the number queens placed on an n x n
# chessboard without threatening each other.
#
# This example demonstrates NumPy slicing.

import numpy as np
import scipy.sparse as sp
import gurobipy as gp
from gurobipy import GRB

# Size of the n x n chess board
n = 8

try:
    # Create a new model
    m = gp.Model("matrix2")

    # Create a 2-D array of binary variables
# x[i,j]=1 means that a queen is placed at square (i,j)
x = m.addMVar((n, n), vtype=GRB.BINARY, name="x")

# Set objective - maximize number of queens
m.setObjective(x.sum(), GRB.MAXIMIZE)

# Add row and column constraints
for i in range(n):
    # At most one queen per row
    m.addConstr(x[i, :].sum() <= 1, name="row"+str(i))

    # At most one queen per column
    m.addConstr(x[:, i].sum() <= 1, name="col"+str(i))

# Add diagonal constraints
for i in range(1, 2*n):
    # At most one queen per diagonal
    diagn = (range(max(0, i-n), min(n, i)), range(min(n, i)-1, max(0, i-n)-1, -1))
    m.addConstr(x[diagn].sum() <= 1, name="diag"+str(i))

    # At most one queen per anti-diagonal
    adiagn = (range(max(0, i-n), min(n, i)), range(max(0, n-i), min(n, 2*n-i)))
    m.addConstr(x[adiagn].sum() <= 1, name="adiag"+str(i))

# Optimize model
m.optimize()

print(x.X)
print(' Queens placed: %g' % m.objVal)

except gp.GurobiError as e:
    print('Error code ' + str(e.errno) + ': ' + str(e))

except AttributeError:
    print('Encountered an attribute error')

mip1.py

#!/usr/bin/env python3.7

# Copyright 2020, Gurobi Optimization, LLC

# This example formulates and solves the following simple MIP model:
# maximize
#   x + y + 2 z
# subject to
#   x + 2 y + 3 z <= 4
#   x + y >= 1
# x, y, z binary

import gurobipy as gp
from gurobipy import GRB

try:
# Create a new model
m = gp.Model("mip1")

# Create variables
x = m.addVar(vtype=GRB.BINARY, name="x")
y = m.addVar(vtype=GRB.BINARY, name="y")
z = m.addVar(vtype=GRB.BINARY, name="z")

# Set objective
m.setObjective(x + y + 2 * z, GRB.MAXIMIZE)

# Add constraint: x + 2 y + 3 z <= 4
m.addConstr(x + 2 * y + 3 * z <= 4, "c0")

# Add constraint: x + y >= 1
m.addConstr(x + y >= 1, "c1")

# Optimize model
m.optimize()

for v in m.getVars():
    print('%s %g' % (v.varName, v.x))

print('Obj : %g' % m.objVal)

except gp.GurobiError as e:
    print("Error code \%d: \%s" % (e.errno, e))

except AttributeError:
    print("Encountered an attribute error")

mip2.py

#!/usr/bin/env python3.7

# Copyright 2020, Gurobi Optimization, LLC

# This example reads a MIP model from a file, solves it and prints
# the objective values from all feasible solutions generated while
# solving the MIP. Then it creates the associated fixed model and
# solves that model.

import sys
import gurobipy as gp
from gurobipy import GRB

if len(sys.argv) < 2:
    print('Usage: mip2.py filename')
    quit()

# Read and solve model
model = gp.read(sys.argv[1])

if model.isMIP == 0:
print('Model is not a MIP')
sys.exit(0)

model.optimize()

if model.status == GRB.OPTIMAL:
    print('Optimal objective: %g' % model.objVal)
elif model.status == GRB.INF_OR_UNBD:
    print('Model is infeasible or unbounded')
    sys.exit(0)
elif model.status == GRB.INFEASIBLE:
    print('Model is infeasible')
    sys.exit(0)
elif model.status == GRB.UNBOUNDED:
    print('Model is unbounded')
    sys.exit(0)
else:
    print('Optimization ended with status %d' % model.status)
    sys.exit(0)

# Iterate over the solutions and compute the objectives
model.Params.outputFlag = 0
print('')
for k in range(model.solCount):
    model.Params.solutionNumber = k
    objn = 0
    for v in model.getVars():
        objn += v.obj * v.xn
    print('Solution %d has objective %g' % (k, objn))
print('')
model.Params.outputFlag = 1

fixed = model.fixed()
fixed.Params.presolve = 0
fixed.optimize()

if fixed.status != GRB.OPTIMAL:
    print("Error: fixed model isn't optimal")
    sys.exit(1)

diff = model.objVal - fixed.objVal

if abs(diff) > 1e-6 * (1.0 + abs(model.objVal)):
    print('Error: objective values are different')
    sys.exit(1)

# Print values of nonzero variables
for v in fixed.getVars():
    if v.x != 0:
        print('%s %g' % (v.varName, v.x))

multiobj.py

#!/usr/bin/env python3.7

# Copyright 2020, Gurobi Optimization, LLC
import gurobipy as gp
from gurobipy import GRB
import sys

try:
    # Sample data
    Groundset = range(20)
    Subsets = range(4)
    Budget = 12
    Set = [[1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0],
           [0, 0, 0, 0, 1, 1, 1, 1, 0, 0, 0, 0, 1, 1, 1, 1, 1],
           [0, 0, 0, 1, 1, 0, 1, 1, 0, 0, 0, 0, 1, 1, 0, 1, 0, 0],
           [0, 0, 0, 1, 1, 1, 0, 0, 0, 1, 1, 1, 0, 0, 1, 1, 0, 0]]
    SetObjPriority = [3, 2, 2, 1]
    SetObjWeight = [1.0, 0.25, 1.25, 1.0]
    # Create initial model
    model = gp.Model('multiobj')
    # Initialize decision variables for ground set:
    # x[e] == 1 if element e is chosen for the covering.
    Elem = model.addVars(Groundset, vtype=GRB.BINARY, name='El')
    # Constraint: limit total number of elements to be picked to be at most
    # Budget
    model.addConstr(Elem.sum() <= Budget, name='Budget')
    # Set global sense for ALL objectives
    model.ModelSense = GRB.MAXIMIZE
    # Limit how many solutions to collect
    model.setParam(GRB.Param.PoolSolutions, 100)
    # Set and configure i-th objective
    for i in Subsets:
        objn = sum(Elem[k]*Set[i][k] for k in range(len(Elem)))
        model.setObjectiveN(objn, i, SetObjPriority[i], SetObjWeight[i],
                1.0 + i, 0.01, 'Set' + str(i))
    # Save problem
    model.write('multiobj.lp')
    # Optimize
    model.optimize()
    model.setParam(GRB.Param.OutputFlag, 0)
    # Status checking
    status = model.Status
    if status in (GRB.INF_OR_UNBD, GRB.INFEASIBLE, GRB.UNBOUNDED):
        print("The model cannot be solved because it is infeasible or ")
if status != GRB.OPTIMAL:
    print('Optimization was stopped with status ' + str(status))
sys.exit(1)

# Print best selected set
print('Selected elements in best solution: ')
selected = [e for e in Groundset if Elem[e].X > 0.9]
print(" ".join("El{}".format(e) for e in selected))

# Print number of solutions stored
nSolutions = model.SolCount
print('Number of solutions found: ' + str(nSolutions))

# Print objective values of solutions
if nSolutions > 10:
    nSolutions = 10
print('Objective values for first ' + str(nSolutions) + ' solutions: ')
for i in Subsets:
    model.setParam(GRB.Param.ObjNumber, i)
    objvals = []
    for e in range(nSolutions):
        model.setParam(GRB.Param.SolutionNumber, e)
        objvals.append(model.ObjNVal)
    print('	Set {} {:6g} {:6g} {:6g}'.format(i, *objvals))

except gp.GurobiError as e:
    print('Error code ' + str(e.errno) + ": " + str(e))
except AttributeError as e:
    print('Encountered an attribute error: ' + str(e))
import gurobipy as gp
from gurobipy import GRB

# Warehouse demand in thousands of units
demand = [15, 18, 14, 20]

# Plant capacity in thousands of units
capacity = [20, 22, 17, 19, 18]

# Fixed costs for each plant
fixedCosts = [12000, 15000, 17000, 13000, 16000]
maxFixed = max(fixedCosts)
minFixed = min(fixedCosts)

# Transportation costs per thousand units
transCosts = [[4000, 2000, 3000, 2500, 4500],
              [2500, 2600, 3400, 3000, 4000],
              [1200, 1800, 2600, 4100, 3000],
              [2200, 2600, 3100, 3700, 3200]]

# Range of plants and warehouses
plants = range(len(capacity))
warehouses = range(len(demand))

# Model
m = gp.Model('multiscenario')

# Plant open decision variables: open[p] == 1 if plant p is open.
open = m.addVars(plants,
                 vtype=GRB.BINARY,
                 obj=fixedCosts,
                 name="open")

# Transportation decision variables: transport[w,p] captures the
# optimal quantity to transport to warehouse w from plant p
transport = m.addVars(warehouses, plants, obj=transCosts, name="trans")

# You could use Python looping constructs and m.addVar() to create
# these decision variables instead. The following would be equivalent
# to the preceding two statements...
#
# open = []
# for p in plants:
#     open.append(m.addVar(vtype=GRB.BINARY,
#                          obj=fixedCosts[p],
#                          name="open[%d]" % p))
#
# transport = []
# for w in warehouses:
#     transport.append([])
#     for p in plants:
#         transport[w].append(m.addVar(obj=transCosts[w][p],
#                       vtype=GRB.INTEGER,
#                       lb=0, ub=1e10))
# The objective is to minimize the total fixed and variable costs
m.modelSense = GRB.MINIMIZE

# Production constraints
# Note that the right-hand limit sets the production to zero if the plant
# is closed
m.addConstrs(
    (transport.sum('*', p) <= capacity[p]*open[p] for p in plants),
    "Capacity")

# Using Python looping constructs, the preceding would be...
# for p in plants:
#    m.addConstr ( sum ( transport[w][p] for w in warehouses )
#                  <= capacity[p] * open[p], "Capacity[%d]" % p)

# Demand constraints
demandConstr = m.addConstrs(
    (transport.sum(w) == demand[w] for w in warehouses), "Demand")

# ... and the preceding would be...  
# for w in warehouses:
#    m.addConstr ( sum ( transport[w][p] for p in plants ) == demand[w],
#                   "Demand[%d]" % w)

# We constructed the base model, now we add 7 scenarios
# Scenario 0: Represents the base model, hence, no manipulations.
# Scenario 1: Manipulate the warehouses demands slightly (constraint right
#        hand sides).
# Scenario 2: Double the warehouses demands (constraint right hand sides).
# Scenario 3: Manipulate the plant fixed costs (objective coefficients).
# Scenario 4: Manipulate the warehouses demands and fixed costs.
# Scenario 5: Force the plant with the largest fixed cost to stay open
#            (variable bounds).
# Scenario 6: Force the plant with the smallest fixed cost to be closed
#            (variable bounds).

m.NumScenarios = 7

# Scenario 0: Base model, hence, nothing to do except giving the scenario a
# name
m.Params.ScenarioNumber = 0
m.ScenNNName = 'Base model'

# Scenario 1: Increase the warehouse demands by 10%
m.Params.ScenarioNumber = 1
m.ScenNNName = 'Increased warehouse demands'
for w in warehouses:
    demandConstr[w].ScenNRhs = demand[w] * 1.1

# Scenario 2: Double the warehouse demands
m.Params.ScenarioNumber = 2
m.ScenNNName = 'Double the warehouse demands'
for w in warehouses:
    demandConstr[w].ScenNRhs = demand[w] * 2.0

# Scenario 3: Decrease the plant fixed costs by 5%
m.Params.ScenarioNumber = 3
m.ScenNName = 'Decreased plant fixed costs'
for p in plants:
    open[p].ScenNObj = fixedCosts[p] * 0.95

# Scenario 4: Combine scenario 1 and scenario 3
m.Params.ScenarioNumber = 4
m.ScenNName = 'Increased warehouse demands and decreased plant fixed costs'
for w in warehouses:
    demandConstr[w].ScenNRhs = demand[w] * 1.1
for p in plants:
    open[p].ScenNObj = fixedCosts[p] * 0.95

# Scenario 5: Force the plant with the largest fixed cost to stay open
m.Params.ScenarioNumber = 5
m.ScenNName = 'Force plant with largest fixed cost to stay open'
open[fixedCosts.index(maxFixed)].ScenNLB = 1.0

# Scenario 6: Force the plant with the smallest fixed cost to be closed
m.Params.ScenarioNumber = 6
m.ScenNName = 'Force plant with smallest fixed cost to be closed'
open[fixedCosts.index(minFixed)].ScenNUB = 0.0

# Save model
m.write('multiscenario.lp')

# Guess at the starting point: close the plant with the highest fixed costs;
# open all others

# First open all plants
for p in plants:
    open[p].start = 1.0

# Now close the plant with the highest fixed cost
p = fixedCosts.index(maxFixed)
open[p].start = 0.0
print('Initial guess: Closing plant %d
' % p)

# Use barrier to solve root relaxation
m.Params.method = 2

# Solve multi-scenario model
m.optimize()

# Print solution for each scenario
for s in range(m.NumScenarios):
    # Set the scenario number to query the information for this scenario
    m.Params.ScenarioNumber = s
    print('

------ Scenario %d (%s)' % (s, m.ScenNName))
    # Check if we found a feasible solution for this scenario
if m.ScenNObjVal >= m.ModelSense * GRB.INFINITY:
    if m.ScenNObjBound >= m.ModelSense * GRB.INFINITY:
        # Scenario was proven to be infeasible
        print('
INFEASIBLE')
    else:
        # We did not find any feasible solution - should not happen in
        # this case, because we did not set any limit (like a time
        # limit) on the optimization process
        print('
NO SOLUTION')
else:
    print('
TOTAL COSTS: %g' % m.ScenNObjVal)
    print('SOLUTION:
    for p in plants:
        if open[p].ScenNX > 0.5:
            print('Plant %s open' % p)
            for w in warehouses:
                if transport[w, p].ScenNX > 0:
                    print(' Transport %g units to warehouse %s' %
                        (transport[w, p].ScenNX, w))
        else:
            print('Plant %s closed !' % p)

    # Print a summary table: for each scenario we add a single summary line
    print('
Summary: Closed plants depending on scenario')
    tableStr = '%8s | %17s %13s' % ('', 'Plant ', '|')
    print(tableStr)
    tableStr = '%8s |' % ' Scenario '
    for p in plants:
        tableStr = tableStr + ' %5d' % p
    tableStr = tableStr + ' | %6s %-s' % ('Costs ', 'Name ')  
    print(tableStr)
    for s in range(m.NumScenarios):
        # Set the scenario number to query the information for this scenario
        m.Params.ScenarioNumber = s

        tableStr = '%-8d |' % s

        # Check if we found a feasible solution for this scenario
        if m.ScenNObjVal >= m.ModelSense * GRB.INFINITY:
            if m.ScenNObjBound >= m.ModelSense * GRB.INFINITY:
                # Scenario was proven to be infeasible
                print(tableStr + ' %30s | %6s %-s' %
                        ('infeasible', '-', m.ScenNName))
            else:
                # We did not find any feasible solution - should not happen in
                # this case, because we did not set any limit (like a time
                # limit) on the optimization process
                print(tableStr + ' %30s | %6s %-s' %
                        ('no solution found', '-', m.ScenNName))
        else:
            for p in plants:
                if open[p].ScenNX > 0.5:
                    tableStr = tableStr + ' %5s' % '
                else:
                    ....

489
netflow.py

#!/usr/bin/env python3.7

# Copyright 2020, Gurobi Optimization, LLC

# Solve a multi-commodity flow problem. Two products ('Pencils' and 'Pens')
# are produced in 2 cities ('Detroit' and 'Denver') and must be sent to
# warehouses in 3 cities ('Boston', 'New York', and 'Seattle') to
# satisfy demand ('inflow[h,i]').
# Flows on the transportation network must respect arc capacity constraints
# ('capacity[i,j]'). The objective is to minimize the sum of the arc
# transportation costs ('cost[i,j]').

import gurobipy as gp
from gurobipy import GRB

# Base data
commodities = ['Pencils', 'Pens']
nodes = ['Detroit', 'Denver', 'Boston', 'New York', 'Seattle']
arcs, capacity = gp.multidict({
    ('Detroit', 'Boston'): 100,
    ('Detroit', 'New York'): 80,
    ('Detroit', 'Seattle'): 120,
    ('Denver', 'Boston'): 120,
    ('Denver', 'New York'): 120,
    ('Denver', 'Seattle'): 120})

# Cost for triplets commodity-source-destination
cost = {
    ('Pencils', 'Detroit', 'Boston'): 10,
    ('Pencils', 'Detroit', 'New York'): 20,
    ('Pencils', 'Detroit', 'Seattle'): 60,
    ('Pencils', 'Denver', 'Boston'): 40,
    ('Pencils', 'Denver', 'New York'): 40,
    ('Pencils', 'Denver', 'Seattle'): 30,
    ('Pens', 'Detroit', 'Boston'): 20,
    ('Pens', 'Detroit', 'New York'): 20,
    ('Pens', 'Detroit', 'Seattle'): 80,
    ('Pens', 'Denver', 'Boston'): 60,
    ('Pens', 'Denver', 'New York'): 70,
    ('Pens', 'Denver', 'Seattle'): 30}

# Demand for pairs of commodity-city
inflow = {
    ('Pencils', 'Detroit'): 50,
    ('Pencils', 'Denver'): 60,
    ('Pencils', 'Boston'): -50,
    ('Pencils', 'New York'): -50,
    ('Pencils', 'Seattle'): -10,
(('Pens', 'Detroit'): 60,
('Pens', 'Denver'): 40,
('Pens', 'Boston'): -40,
('Pens', 'New York'): -30,
('Pens', 'Seattle'): -30)

# Create optimization model
m = gp.Model('netflow')

# Create variables
flow = m.addVars(commodities, arcs, obj=cost, name="flow")

# Arc-capacity constraints
m.addConstrs(
    (flow.sum('*', i, j) <= capacity[i, j] for i, j in arcs), "cap")

# Equivalent version using Python looping
# for i, j in arcs :
#     m.addConstr ( sum ( flow[h, i, j] for h in commodities ) <= capacity[i, j],
#                    "cap[%s, %s]") % (i, j))

# Flow-conservation constraints
m.addConstrs(
    (flow.sum(h, '*', j) + inflow[h, j] == flow.sum(h, j, '*')
     for h in commodities for j in nodes), "node")

# Alternate version :
# m.addConstrs(
#     (gp.quicksum(flow[h, i, j] for i, j in arcs.select('*', j)) + inflow[h, j] ==
#      gp.quicksum(flow[h, j, k] for j, k in arcs.select(j, '*'))
#     for h in commodities for j in nodes), "node")

# Compute optimal solution
m.optimize()

# Print solution
if m.status == GRB.OPTIMAL:
    solution = m.getAttr('x', flow)
    for h in commodities:
        print('
Optimal flows for %s:' % h)
        for i, j in arcs:
            if solution[h, i, j] > 0:
                print('%s -> %s: %g' % (i, j, solution[h, i, j]))

params.py

#!/usr/bin/env python3.7

# Copyright 2020, Gurobi Optimization, LLC

# Use parameters that are associated with a model.
#
# A MIP is solved for a few seconds with different sets of parameters.
# The one with the smallest MIP gap is selected, and the optimization
# is resumed until the optimal solution is found.
import sys
import gurobipy as gp

if len(sys.argv) < 2:
    print('Usage: params.py filename')
    quit()

# Read model and verify that it is a MIP
m = gp.read(sys.argv[1])
if m.isMIP == 0:
    print('The model is not an integer program')
sys.exit(1)

# Set a 2 second time limit
m.Params.timeLimit = 2

# Now solve the model with different values of MIPFocus
bestModel = m.copy()
bestModel.optimize()
for i in range(1, 4):
    m.reset()
    m.Params.MIPFocus = i
    m.optimize()
    if bestModel.MIPGap > m.MIPGap:
        bestModel, m = m, bestModel  # swap models

# Finally, delete the extra model, reset the time limit and
# continue to solve the best model to optimality
del m
bestModel.Params.timeLimit = "default"
bestModel.optimize()
print('Solved with MIPFocus: %d' % bestModel.Params.MIPFocus)

piecewise.py

#!/usr/bin/env python3.7

# Copyright 2020, Gurobi Optimization, LLC

# This example considers the following separable, convex problem:
#
# minimize  \quad f(x) - y + g(z)
# subject to  \quad x + 2 y + 3 z <= 4
#          \quad x + y >= 1
#          \quad x, y, z <= 1
#
# where f(u) = \exp(-u) and g(u) = 2 u^2 - 4 u, for all real u. It
# formulates and solves a simpler LP model by approximating f and
# g with piecewise-linear functions. Then it transforms the model
# into a MIP by negating the approximation for f, which corresponds
# to a non-convex piecewise-linear function, and solves it again.

import gurobipy as gp
from math import exp

def f(u):
    return exp(-u)

def g(u):
    return 2 * u * u - 4 * u

try:
    # Create a new model
    m = gp.Model()
    # Create variables
    lb = 0.0
    ub = 1.0
    x = m.addVar(lb, ub, name='x')
    y = m.addVar(lb, ub, name='y')
    z = m.addVar(lb, ub, name='z')
    # Set objective for y
    m.setObjective(-y)
    # Add piecewise-linear objective functions for x and z
    npts = 101
    ptu = []
    ptf = []
    ptg = []
    for i in range(npts):
        ptu.append(lb + (ub - lb) * i / (npts - 1))
        ptf.append(f(ptu[i]))
        ptg.append(g(ptu[i]))
    m.setPWLObj(x, ptu, ptf)
    m.setPWLObj(z, ptu, ptg)
    # Add constraint: x + 2 y + 3 z <= 4
    m.addConstr(x + 2 * y + 3 * z <= 4, 'c0')
    # Add constraint: x + y >= 1
    m.addConstr(x + y >= 1, 'c1')
    # Optimize model as an LP
    m.optimize()
print('IsMIP: %d' % m.IsMIP)
for v in m.getVars():
    print('%s %g' % (v.VarName, v.X))
print('Obj: %g' % m.ObjVal)
print('')

# Negate piecewise-linear objective function for x
for i in range(npts):
    ptf[i] = -ptf[i]

m.setPWLObj(x, ptu, ptf)

# Optimize model as a MIP
m.optimize()

print('IsMIP: %d' % m.IsMIP)
for v in m.getVars():
    print('%s %g' % (v.VarName, v.X))
print('Obj: %g' % m.ObjVal)

except gp.GurobiError as e:
    print('Error code ' + str(e.errno) + ': ' + str(e))
except AttributeError:
    print('Encountered an attribute error')

poolsearch.py

#!/usr/bin/env python3.7

# Copyright 2020, Gurobi Optimization, LLC

# We find alternative epsilon-optimal solutions to a given knapsack
# problem by using PoolSearchMode

from __future__ import print_function
import gurobipy as gp
from gurobipy import GRB
import sys

try:
    # Sample data
    Groundset = range(10)
    objCoef = [32, 32, 15, 15, 6, 6, 1, 1, 1, 1]
    knapsackCoef = [16, 16, 8, 8, 4, 4, 2, 2, 1, 1]
    Budget = 33

    # Create initial model
    model = gp.Model("poolsearch")

    # Create dicts for tupledict.prod() function
    objCoefDict = dict(zip(Groundset, objCoef))
    knapsackCoefDict = dict(zip(Groundset, knapsackCoef))
# Initialize decision variables for ground set:
# x[e] == 1 if element e is chosen
Elem = model.addVars(Groundset, vtype=GRB.BINARY, name='El')

# Set objective function
model.ModelSense = GRB.MAXIMIZE
model.setObjective(Elem.prod(objCoefDict))

# Constraint: limit total number of elements to be picked to be at most
# Budget
model.addConstr(Elem.prod(knapsackCoefDict) <= Budget, name='Budget')

# Limit how many solutions to collect
model.setParam(GRB.Param.PoolSolutions, 1024)
# Limit the search space by setting a gap for the worst possible solution
# that will be accepted
model.setParam(GRB.Param.PoolGap, 0.10)
# do a systematic search for the k-best solutions
model.setParam(GRB.Param.PoolSearchMode, 2)

# save problem
model.write('poolsearch.lp')

# Optimize
model.optimize()
model.setParam(GRB.Param.OutputFlag, 0)

# Status checking
status = model.Status
if status in (GRB.INF_OR_UNBD, GRB.INFEASIBLE, GRB.UNBOUNDED):
    print('The model cannot be solved because it is infeasible or ' +
          'unbounded')
sys.exit(1)

if status != GRB.OPTIMAL:
    print('Optimization was stopped with status ' + str(status))
sys.exit(1)

# Print best selected set
print('Selected elements in best solution:');
print('	', end='');
for e in Groundset:
    if Elem[e].X > .9:
        print(' El%d ' % e, end='');
print(')

# Print number of solutions stored
nSolutions = model.SolCount
print('Number of solutions found: ' + str(nSolutions))

# Print objective values of solutions
for e in range(nSolutions):
    model.setParam(GRB.Param.SolutionNumber, e)
    print('%g ' % model.PoolObjVal, end='');

495
if e % 15 == 14:
    print(''
print(''

# print fourth best set if available
if nSolutions >= 4):
    model.setParam(GRB.Param.SolutionNumber, 3)

    print('Selected elements in fourth best solution:'
    print('	', end='')
    for e in Groundset:
        if Elem[e].Xn > .9:
            print(' El%d' % e, end='')
    print(''
except gp.GurobiError as e:
    print('Gurobi error ' + str(e.errno) + ': "' + str(e.message))
except AttributeError as e:
    print('Encountered an attribute error: ' + str(e))

portfolio.py

# Copyright 2020, Gurobi Optimization, LLC

# Portfolio selection: given a sum of money to invest, one must decide how to
# spend it amongst a portfolio of financial securities. Our approach is due
# to Markowitz (1959) and looks to minimize the risk associated with the
# investment while realizing a target expected return. By varying the target,
# one can compute an 'efficient frontier', which defines the optimal portfolio
# for a given expected return.  
# Note that this example reads historical return data from a comma-separated
# file (../data/portfolio.csv). As a result, it must be run from the Gurobi
# examples/python directory.
# This example requires the pandas (>= 0.20.3), NumPy, and Matplotlib
# Python packages, which are part of the SciPy ecosystem for
# mathematics, science, and engineering (http://scipy.org). These
# packages aren't included in all Python distributions, but are
# included by default with Anaconda Python.

import gurobipy as gp
from gurobipy import GRB
from math import sqrt
import pandas as pd
import numpy as np
import matplotlib
matplotlib.use('Agg')
import matplotlib.pyplot as plt

# Import (normalized) historical return data using pandas
data = pd.read_csv('../data/portfolio.csv', index_col=0)
stocks = data.columns
# Calculate basic summary statistics for individual stocks
stock_volatility = data.std()
stock_return = data.mean()

# Create an empty model
m = gp.Model('portfolio')

# Add a variable for each stock
vars = pd.Series(m.addVars(stocks), index=stocks)

# Objective is to minimize risk (squared). This is modeled using the
# covariance matrix, which measures the historical correlation between stocks.
sigma = data.cov()
portfolio_risk = sigma.dot(vars).dot(vars)
m.setObjective(portfolio_risk, GRB.MINIMIZE)

# Fix budget with a constraint
m.addConstr(vars.sum() == 1, 'budget')

# Optimize model to find the minimum risk portfolio
m.setParam('OutputFlag', 0)
m.optimize()

# Create an expression representing the expected return for the portfolio
portfolio_return = stock_return.dot(vars)

# Display minimum risk portfolio
print('Minimum Risk Portfolio:
')
for v in vars:
    if v.x > 0:
        print('	%s	: %g' % (v.varname, v.x))
minrisk_volatility = sqrt(portfolio_risk.getValue())
print('Volatility = %g' % minrisk_volatility)
minrisk_return = portfolio_return.getValue()
print('Expected Return = %g' % minrisk_return)

# Add (redundant) target return constraint
target = m.addConstr(portfolio_return == minrisk_return, 'target')

# Solve for efficient frontier by varying target return
frontier = pd.Series()
for r in np.linspace(stock_return.min(), stock_return.max(), 100):
    target.rhs = r
    m.optimize()
    frontier.loc[sqrt(portfolio_risk.getValue())] = r

# Plot volatility versus expected return for individual stocks
ax = plt.gca()
ax.scatter(x=stock_volatility, y=stock_return,
           color='Blue', label='Individual Stocks')
for i, stock in enumerate(stocks):
    ax.annotate(stock, (stock_volatility[i], stock_return[i]))

# Plot volatility versus expected return for minimum risk portfolio
ax.scatter(x=minrisk_volatility, y=minrisk_return, color='DarkGreen')
# Plot efficient frontier
frontier.plot(color='DarkGreen', label='Efficient Frontier', ax=ax)

# Format and display the final plot
ax.axis([0.005, 0.06, -0.02, 0.025])
ax.set_xlabel('Volatility (standard deviation)')
ax.set_ylabel('Expected Return')
ax.legend()
ax.grid()
plt.savefig('portfolio.png')
print("Plotted efficient frontier to 'portfolio.png'")

qcp.py

#!/usr/bin/env python3.7

# Copyright 2020, Gurobi Optimization, LLC

# This example formulates and solves the following simple QCP model:
# maximize x
# subject to x + y + z = 1
#        x^2 + y^2 <= z^2 (second-order cone)
#        x^2 <= yz (rotated second-order cone)
#        x, y, z non-negative

import gurobipy as gp
from gurobipy import GRB

# Create a new model
m = gp.Model("qcp")

# Create variables
x = m.addVar(name="x")
y = m.addVar(name="y")
z = m.addVar(name="z")

# Set objective: x
obj = 1.0*x
m.setObjective(obj, GRB.MAXIMIZE)

# Add constraint: x + y + z = 1
m.addConstr(x + y + z == 1, "c0")

# Add second-order cone: x^2 + y^2 <= z^2
m.addConstr(x*x + y*y <= z*z, "qc0")

# Add rotated cone: x^2 <= yz
m.addConstr(x*x <= y*z, "qc1")

m.optimize()

for v in m.getVars():
    print("%s %g" % (v.varName, v.x))
print('Obj: %g' % obj.getValue())

qp.py

#!/usr/bin/env python3.7

# Copyright 2020, Gurobi Optimization, LLC

# This example formulates and solves the following simple QP model:
# minimize
# x^2 + x*y + y^2 + y*z + z^2 + 2 x
# subject to
# x + 2 y + 3 z >= 4
# x + y >= 1
# x, y, z non-negative
#
# It solves it once as a continuous model, and once as an integer model.

import gurobipy as gp
from gurobipy import GRB

# Create a new model
m = gp.Model("qp")

# Create variables
x = m.addVar(ub=1.0, name="x")
y = m.addVar(ub=1.0, name="y")
z = m.addVar(ub=1.0, name="z")

# Set objective: x^2 + x*y + y^2 + y*z + z^2 + 2*x
obj = x*x + x*y + y*y + y*z + z*z + 2*x
m.setObjective(obj)

# Add constraint: x + 2 y + 3 z <= 4
m.addConstr(x + 2 * y + 3 * z >= 4, "c0")

# Add constraint: x + y >= 1
m.addConstr(x + y >= 1, "c1")

m.optimize()

for v in m.getVars():
    print('%.s %g' % (v.varName, v.x))

print('Obj: %g' % obj.getValue())

x.vType = GRB.INTEGER
y.vType = GRB.INTEGER
z.vType = GRB.INTEGER

m.optimize()

for v in m.getVars():
    print('%.s %g' % (v.varName, v.x))
print('Obj: %g % obj.getValue())

sensitivity.py

#!/usr/bin/env python3.7

# Copyright 2020, Gurobi Optimization, LLC

# A simple sensitivity analysis example which reads a MIP model from a file
# and solves it. Then uses the scenario feature to analyze the impact
# w.r.t. the objective function of each binary variable if it is set to
# 1-X, where X is its value in the optimal solution.
#
# Usage:
#    sensitivity.py <model filename>
#
import sys
import gurobipy as gp
from gurobipy import GRB

# Maximum number of scenarios to be considered
maxScenarios = 100

if len(sys.argv) < 2:
    print('Usage: sensitivity.py filename')
    quit()

# Read model
model = gp.read(sys.argv[1])

if model.IsMIP != 0:
    print('Model is not a MIP')
    sys.exit(0)

# Solve model
model.optimize()

if model.status != GRB.OPTIMAL:
    print('Optimization ended with status %s' % model.status)
    sys.exit(0)

# Store the optimal solution
origObjVal = model ObjVal
for v in model.getVars():
    v._origX = v.X

scenarios = 0

# Count number of unfixed, binary variables in model. For each we create a
# scenario.
for v in model.getVars():
    if (v.LB == 0.0 and v.UB == 1.0 and v.VType in (GRB.BINARY, GRB.INTEGER)):
        scenarios += 1
if scenarios >= maxScenarios:
    break

# Set the number of scenarios in the model
model.NumScenarios = scenarios
scenarios = 0

print('### construct multi-scenario model with %d scenarios' % scenarios)

# Create a (single) scenario model by iterating through unfixed binary
# variables in the model and create for each of these variables a scenario
# by fixing the variable to 1-X, where X is its value in the computed
# optimal solution
for v in model.getVars():
    if (v.LB == 0.0 and v.UB == 1.0
        and v.VType in (GRB.BINARY, GRB.INTEGER)
        and scenarios < maxScenarios):
        # Set ScenarioNumber parameter to select the corresponding scenario
        # for adjustments
        model.params.ScenarioNumber = scenarios

        # Set variable to 1-X, where X is its value in the optimal solution
        if v._origX < 0.5:
            v.ScenNLB = 1.0
        else:
            v.ScenNUB = 0.0

        scenarios += 1
    else:
        # Add MIP start for all other variables using the optimal solution
        # of the base model
        v.Start = v._origX

# Solve multi-scenario model
model.optimize()

# In case we solved the scenario model to optimality capture the
# sensitivity information
if model.status == GRB.OPTIMAL:

    modelSense = model.ModelSense
    scenarios = 0

    # Capture sensitivity information from each scenario
    for v in model.getVars():
        if (v.LB == 0.0 and v.UB == 1.0 and v.VType in (GRB.BINARY, GRB.INTEGER)):

            # Set scenario parameter to collect the objective value of the
            # corresponding scenario
            model.params.ScenarioNumber = scenarios
# Collect objective value and bound for the scenario
scenarioObjVal = model.ScenNObjVal
scenarioObjBound = model.ScenNObjBound

# Check if we found a feasible solution for this scenario
if scenarioObjVal >= modelSense * GRB.INFINITY:
    # Check if the scenario is infeasible
    if scenarioObjBound >= modelSense * GRB.INFINITY:
        print('Objective sensitivity for variable %s is infeasible' %
              v.VarName)
    else:
        print('Objective sensitivity for variable %s is unknown (no solution available)
              v.VarName)
else:
    # Scenario is feasible and a solution is available
    print('Objective sensitivity for variable %s is %g' %
          (v.VarName, modelSense * (scenarioObjVal - origObjVal)))

scenarios += 1

if scenarios >= maxScenarios:
    break

---

sos.py

#!/usr/bin/env python3.7

# Copyright 2020, Gurobi Optimization, LLC

# This example creates a very simple Special Ordered Set (SOS) model.
# The model consists of 3 continuous variables, no linear constraints,
# and a pair of SOS constraints of type 1.

import gurobipy as gp
from gurobipy import GRB

try:
    # Create a new model
    model = gp.Model("sos")

    # Create variables
    x0 = model.addVar(ub=1.0, name="x0")
    x1 = model.addVar(ub=1.0, name="x1")
    x2 = model.addVar(ub=2.0, name="x2")

    # Set objective
    model.setObjective(2 * x0 + x1 + x2, GRB.MAXIMIZE)

    # Add first SOS: x0 = 0 or x1 = 0
    model.addSOS(GRB.SOS_TYPE1, [x0, x1], [1, 2])

    # Add second SOS: x0 = 0 or x2 = 0

model.addSOS(�GRB.SOS_TYPE1, [x0, x2], [1, 2])

model.optimize()

for v in model.getVars():
    print('％s %g' % (v.varName, v.x))

    print('Obj: %g % model.objVal)

except gp.GurobiError as e:
    print('Error code ' + str(e.errno) + ": " + str(e))

except AttributeError:
    print('Encountered an attribute error')

dkuku.py

#!/usr/bin/env python3.7

# Copyright 2020, Gurobi Optimization, LLC

# Sudoku example.

# The Sudoku board is a 9x9 grid, which is further divided into a 3x3 grid
# of 3x3 grids. Each cell in the grid must take a value from 0 to 9.
# No two grid cells in the same row, column, or 3x3 subgrid may take the
# same value.

# In the MIP formulation, binary variables x[i,j,v] indicate whether
# cell <i,j> takes value 'v'. The constraints are as follows:
# 1. Each cell must take exactly one value (sum_v x[i,j,v] = 1)
# 2. Each value is used exactly once per row (sum_i x[i,j,v] = 1)
# 3. Each value is used exactly once per column (sum_j x[i,j,v] = 1)
# 4. Each value is used exactly once per 3x3 subgrid (sum_grid x[i,j,v] = 1)

# Input datasets for this example can be found in examples/data/sudoku*.

import sys
import math
import gurobipy as gp
from gurobipy import GRB

if len(sys.argv) < 2:
    print('Usage: sudoku.py filename')
    quit()

f = open(sys.argv[1])

gird = f.read().split()

n = len(gird[0])
s = int(math.sqrt(n))

# Create our 3-D array of model variables
model = gp.Model('sudoku')

vars = model.addVars(n, n, n, vtype=GRB.BINARY, name='G')

# Fix variables associated with cells whose values are pre-specified
for i in range(n):
    for j in range(n):
        if grid[i][j] != '.':
            v = int(grid[i][j]) - 1
            vars[i, j, v].LB = 1

# Each cell must take one value
model.addConstrs((vars.sum(i, j, '*') == 1
                   for i in range(n)
                   for j in range(n)), name='V')

# Each value appears once per row
model.addConstrs((vars.sum(i, '*', v) == 1
                   for i in range(n)
                   for v in range(n)), name='R')

# Each value appears once per column
model.addConstrs((vars.sum('*', j, v) == 1
                   for j in range(n)
                   for v in range(n)), name='C')

# Each value appears once per subgrid
model.addConstrs((
    gp.quicksum(vars[i, j, v]
                 for i in range(i0*s, (i0+1)*s)
                 for j in range(j0*s, (j0+1)*s)) == 1
    for v in range(n)
    for i0 in range(s)
    for j0 in range(s)), name='Sub')

model.optimize()
model.write('sudoku.lp')

print('')
print('Solution:')
print('')

# Retrieve optimization result
solution = model.getAttr('X', vars)
for i in range(n):
    sol = ''
for j in range(n):
    for v in range(n):
        if solution[i, j, v] > 0.5:
            sol += str(v+1)

print(sol)

tsp.py

#!/usr/bin/env python3.7
# Copyright 2020, Gurobi Optimization, LLC

# Solve a traveling salesman problem on a randomly generated set of
# points using lazy constraints. The base MIP model only includes
# 'degree-2' constraints, requiring each node to have exactly
# two incident edges. Solutions to this model may contain subtours -
# tours that don't visit every city. The lazy constraint callback
# adds new constraints to cut them off.

import sys
import math
import random
from itertools import combinations
import gurobipy as gp
from gurobipy import GRB

# Callback - use lazy constraints to eliminate sub-tours
def subtourelim(model, where):
    if where == GRB.Callback.MIPSOL:
        # make a list of edges selected in the solution
        vals = model.cbGetSolution(model._vars)
        selected = gp.tuplelist((i, j) for i, j in model._vars.keys()
                       if vals[i, j] > 0.5)
        # find the shortest cycle in the selected edge list
        tour = subtour(selected)
        if len(tour) < n:
            # add subtour elimination constr. for every pair of cities in tour
            model.cbLazy(gp.quicksum(model._vars[i, j]
                                      for i, j in combinations(tour, 2))
                           <= len(tour)-1)

# Given a tuplelist of edges, find the shortest subtour
def subtour(edges):
    unvisited = list(range(n))
    cycle = range(n+1)  # initial length has 1 more city
    while unvisited:    # true if list is non-empty
        thiscycle = []
        neighbors = unvisited
        while neighbors:
            current = neighbors[0]
            neighbors = [j for i, j in edges.select(current, '*')
                         if i not in unvisited]
            thiscycle.append(current)
            unvisited.remove(current)
            neighbors = [j for i, j in edges.select(current, '*')
                         if i not in unvisited]
        cycle = cycle[:thiscycle.index(current)] +
                thiscycle + cycle[thiscycle.index(current):]
    return cycle

505
if j in unvisited
    if len(cycle) > len(thiscycle):
        cycle = thiscycle
    return cycle

# Parse argument
if len(sys.argv) < 2:
    print('Usage: tsp.py npoints')
    sys.exit(1)
n = int(sys.argv[1])

# Create n random points
random.seed(1)
points = [(random.randint(0, 100), random.randint(0, 100)) for i in range(n)]

# Dictionary of Euclidean distance between each pair of points
dist = {(i, j):
    math.sqrt(sum((points[i][k]-points[j][k])**2 for k in range(2)))
    for i in range(n) for j in range(i)}
m = gp.Model()

# Create variables
vars = m.addVars(dist.keys(), obj=dist, vtype=GRB.BINARY, name='e')
for i, j in vars.keys():
    vars[j, i] = vars[i, j] # edge in opposite direction

# You could use Python looping constructs and m.addVar() to create
# these decision variables instead. The following would be equivalent
# to the preceding m.addVars() call...
#
# vars = tupledict()
# for i,j in dist.keys():
#     vars[i,j] = m.addVar(obj=dist[i,j], vtype=GRB.BINARY,
#     # name='e[%d,%d]'%(i,j))

# Add degree-2 constraint
m.addConstrs(vars.sum(i, '*') == 2 for i in range(n))

# Using Python looping constructs, the preceding would be...
#
# for i in range(n):
#     m.addConstr(sum(vars[i,j] for j in range(n)) == 2)

# Optimize model
m._vars = vars
m.Params.lazyConstraints = 1

506
m.optimize(subtourelim)

vals = m.getAttr('x', vars)
selected = gp.tuplelist((i, j) for i, j in vals.keys() if vals[i, j] > 0.5)

tour = subtour(selected)
assert len(tour) == n

print('')
print('Optimal tour: %s' % str(tour))
print('Optimal cost: %g' % m.objVal)
print('')

tune.py

#!/usr/bin/env python3.7
#
# Copyright 2020, Gurobi Optimization, LLC
#
# This example reads a model from a file and tunes it.
# It then writes the best parameter settings to a file
# and solves the model using these parameters.

import sys
import gurobipy as gp

if len(sys.argv) < 2:
    print('Usage: tune.py filename')
    quit()

# Read the model
model = gp.read(sys.argv[1])

# Set the TuneResults parameter to 1
model.Params.tuneResults = 1

# Tune the model
model.tune()

if model.tuneResultCount > 0:
    # Load the best tuned parameters into the model
    model.getTuneResult(0)

    # Write tuned parameters to a file
    model.write('tune.prm')

    # Solve the model using the tuned parameters
    model.optimize()

workforce_batchmode.py

#!/usr/bin/env python3.7
#
# Copyright 2020, Gurobi Optimization, LLC


# Assign workers to shifts; each worker may or may not be available on a
# particular day. The optimization problem is solved as a batch, and
# the schedule constructed only from the meta data available in the solution
# JSON.
#
# NOTE: You’ll need a license file configured to use a Cluster Manager
# for this example to run.

import time
import json
import sys
import gurobipy as gp
from gurobipy import GRB
from collections import OrderedDict, defaultdict

# For later pretty printing names for the shifts
shiftname = OrderedDict(
    ("Mon1", "Monday 8:00"),
    ("Mon8", "Monday 14:00"),
    ("Tue2", "Tuesday 8:00"),
    ("Tue9", "Tuesday 14:00"),
    ("Wed3", "Wednesday 8:00"),
    ("Wed10", "Wednesday 14:00"),
    ("Thu4", "Thursday 8:00"),
    ("Thu11", "Thursday 14:00"),
    ("Fri5", "Friday 8:00"),
    ("Fri12", "Friday 14:00"),
    ("Sat6", "Saturday 8:00"),
    ("Sat13", "Saturday 14:00"),
    ("Sun7", "Sunday 9:00"),
    ("Sun14", "Sunday 12:00"),
)

# Build the assignment problem in a Model, and submit it for batch optimization
# Required input: A Cluster Manager environment setup for batch optimization
def submit_assigment_problem(env):
    # Number of workers required for each shift
    shifts, shiftRequirements = gp.multidict({
        "Mon1": 3,
        "Tue2": 2,
        "Wed3": 4,
        "Thu4": 4,
        "Fri5": 5,
        "Sat6": 5,
        "Sun7": 3,
        "Mon8": 2,
        "Tue9": 2,
        "Wed10": 3,
        "Thu11": 4,
        "Fri12": 5,
        "Sat13": 7,
        "Sun14": 5,
    })
# Amount each worker is paid to work one shift
workers, pay = gp.multidict({
    "Amy": 10,
    "Bob": 12,
    "Cathy": 10,
    "Dan": 8,
    "Ed": 8,
    "Fred": 9,
    "Gu": 11,
})

# Worker availability
availability = gp.tuplelist([
    ('Amy', 'Tue2'), ('Amy', 'Wed3'), ('Amy', 'Thu4'), ('Amy', 'Sun7'),
    ('Amy', 'Tue9'), ('Amy', 'Wed10'), ('Amy', 'Thu11'), ('Amy', 'Fri12'),
    ('Amy', 'Sat13'), ('Amy', 'Sun14'), ('Bob', 'Mon1'), ('Bob', 'Tue2'),
    ('Bob', 'Fri5'), ('Bob', 'Sat6'), ('Bob', 'Mon8'), ('Bob', 'Thu11'),
    ('Bob', 'Sat13'), ('Cathy', 'Wed3'), ('Cathy', 'Thu4'),
    ('Cathy', 'Fri5'), ('Cathy', 'Sun7'), ('Cathy', 'Mon8'),
    ('Cathy', 'Tue9'), ('Cathy', 'Wed10'), ('Cathy', 'Thu11'),
    ('Cathy', 'Fri12'), ('Cathy', 'Sat13'), ('Cathy', 'Sun14'),
    ('Dan', 'Tue2'), ('Dan', 'Thu4'), ('Dan', 'Fri5'), ('Dan', 'Sat6'),
    ('Dan', 'Mon8'), ('Dan', 'Tue9'), ('Dan', 'Wed10'), ('Dan', 'Thu11'),
    ('Dan', 'Fri12'), ('Dan', 'Sat13'), ('Dan', 'Sun14'), ('Ed', 'Mon1'),
    ('Ed', 'Tue2'), ('Ed', 'Wed3'), ('Ed', 'Thu4'), ('Ed', 'Fri5'),
    ('Ed', 'Sat6'), ('Ed', 'Mon8'), ('Ed', 'Tue9'), ('Ed', 'Thu11'),
    ('Ed', 'Sat13'), ('Ed', 'Sun14'), ('Fred', 'Mon1'), ('Fred', 'Tue2'),
    ('Fred', 'Wed3'), ('Fred', 'Sat6'), ('Fred', 'Mon8'), ('Fred', 'Tue9'),
    ('Fred', 'Fri12'), ('Fred', 'Sat13'), ('Fred', 'Sun14'),
    ('Gu', 'Mon1'), ('Gu', 'Tue2'), ('Gu', 'Wed3'), ('Gu', 'Fri5'),
    ('Gu', 'Sat6'), ('Gu', 'Sun7'), ('Gu', 'Mon8'), ('Gu', 'Tue9'),
    ('Gu', 'Wed10'), ('Gu', 'Thu11'), ('Gu', 'Fri12'), ('Gu', 'Sat13'),
    ('Gu', 'Sun14')
])

# Start environment, get model in this environment
with gp.Model("assignment", env=env) as m:
    # Assignment variables: x[w,s] == 1 if worker w is assigned to shift s.
    # Since an assignment model always produces integer solutions, we use
    # continuous variables and solve as an LP.
    x = m.addVars(availability, ub=1, name="x")
    # Set tags encoding the assignments for later retrieval of the schedule.
    # Each tag is a JSON string of the format
    # {
    #   "Worker": "<Name of the worker>",
    #   "Shift": "String representation of the shift"
    # }
    #
    # for k, v in x.items():
    #    name, timeslot = k
    #    d = {"Worker": name, "Shift": shiftname[timeslot]}
    #    v.VTag = json.dumps(d)
    # The objective is to minimize the total pay costs
m.setObjective(gp.quicksum(pay[w]*x[w, s] for w, s in availability),
GRB.MINIMIZE)

# Constraints: assign exactly shiftRequirements[s] workers to each shift
reqCts = m.addConstrs((x.sum('*', s) == shiftRequirements[s]
    for s in shifts), "=")

# Submit this model for batch optimization to the cluster manager
# and return its batch ID for later querying the solution
batchID = m.optimizeBatch()

return batchID

# Wait for the final status of the batch.
# Initially the status of a batch is "submitted"; the status will change
# once the batch has been processed (by a compute server).
def waitforfinalbatchstatus(batch):
    # Wait no longer than ten seconds
    maxwaittime = 10

    starttime = time.time()
    while batch.BatchStatus == GRB.BATCH_SUBMITTED:

        # Abort this batch if it is taking too long
        curtime = time.time()
        if curtime - starttime > maxwaittime:
            batch.abort()
            break

        # Wait for one second
        time.sleep(1)

        # Update the resident attribute cache of the Batch object with the
        # latest values from the cluster manager.
        batch.update()

    return batchID

# Print the schedule according to the solution in the given dict
def print_shift_schedule(soldict):
    schedule = defaultdict(list)

    # Iterate over the variables that take a non-zero value (i.e.,
    # an assignment), and collect them per day
    for v in soldict['Vars']:
        # There is only one VTag, the JSON dict of an assignment we passed
        # in as the VTag
        assignment = json.loads(v['VTag'][0])
        schedule[assignment['Shift']].append(assignment['Worker'])

    # Print the schedule
    for k in shiftname.values():
        day, time = k.split()
        workers = "", ".join(schedule[k])
        print(" - {:10} {: >5}: {}".format(day, time, workers))
if __name__ == '__main__':
    # Create Cluster Manager environment in batch mode.
    env = gp.Env(empty=True)
    env.setParam('CSBatchMode', 1)

    # env is a context manager; upon leaving, Env.dispose() is called
    with env.start():
        # Submit the assignment problem to the cluster manager, get batch ID
        batchID = submit_assignment_problem(env)

        # Create a batch object, wait for batch to complete, query solution JSON
        with gp.Batch(batchID, env) as batch:
            waitforfinalbatchstatus(batch)

            if batch.BatchStatus != GRB.BATCH_COMPLETED:
                print("Batch request couldn't be completed")
                sys.exit(0)

            jsonsol = batch.getJSONSolution()

            # Dump JSON solution string into a dict
            soldict = json.loads(jsonsol)

            # Has the assignment problem been solved as expected?
            if soldict['SolutionInfo']['Status'] != GRB.OPTIMAL:
                # Shouldn't happen...
                print("Assignment problem could not be solved to optimality")
                sys.exit(0)

            # Print shift schedule from solution JSON
            print_shift_schedule(soldict)

workforce1.py

#!/usr/bin/env python3.7

# Copyright 2020, Gurobi Optimization, LLC

# Assign workers to shifts; each worker may or may not be available on a
# particular day. If the problem cannot be solved, use IIS to find a set of
# conflicting constraints. Note that there may be additional conflicts besides
# what is reported via IIS.

import gurobipy as gp
from gurobipy import GRB
import sys

# Number of workers required for each shift
shifts, shiftRequirements = gp.multidict({
    "Mon1": 3,
    "Tue2": 2,
    "Wed3": 4,
    "Thu4": 4,
    "Fri5": 5,
    "Sat6": 6,

})
# Amount each worker is paid to work one shift

workers, pay = gp.multidict({
    "Amy": 10,
    "Bob": 12,
    "Cathy": 10,
    "Dan": 8,
    "Ed": 8,
    "Fred": 9,
    "Gu": 11,
})

# Worker availability
availability = gp.tuplelist([ ('Amy', 'Tue2'), ('Amy', 'Wed3'), ('Amy', 'Fri5'), ('Amy', 'Sun7'),
    ('Amy', 'Tue9'), ('Amy', 'Wed10'), ('Amy', 'Thu11'), ('Amy', 'Fri12'),
    ('Amy', 'Sat13'), ('Amy', 'Sun14'), ('Bob', 'Mon1'), ('Bob', 'Tue2'),
    ('Bob', 'Fri5'), ('Bob', 'Sat6'), ('Bob', 'Mon8'), ('Bob', 'Thu11'),
    ('Bob', 'Sat13'), ('Cathy', 'Wed3'), ('Cathy', 'Thu4'), ('Cathy', 'Fri5'),
    ('Cathy', 'Sun7'), ('Cathy', 'Mon8'), ('Cathy', 'Tue9'),
    ('Cathy', 'Wed10'), ('Cathy', 'Thu11'), ('Cathy', 'Fri12'),
    ('Cathy', 'Sat13'), ('Cathy', 'Sun14'), ('Dan', 'Tue2'), ('Dan', 'Wed3'),
    ('Dan', 'Fri5'), ('Dan', 'Sat6'), ('Dan', 'Mon8'), ('Dan', 'Tue9'),
    ('Dan', 'Wed10'), ('Dan', 'Thu11'), ('Dan', 'Fri12'), ('Dan', 'Sat13'),
    ('Dan', 'Sun14'), ('Ed', 'Mon1'), ('Ed', 'Tue2'), ('Ed', 'Wed3'),
    ('Ed', 'Thu4'), ('Ed', 'Fri5'), ('Ed', 'Sun7'), ('Ed', 'Mon8'),
    ('Ed', 'Tue9'), ('Ed', 'Thu11'), ('Ed', 'Sat13'), ('Ed', 'Sun14'),
    ('Fred', 'Mon1'), ('Fred', 'Tue2'), ('Fred', 'Wed3'), ('Fred', 'Sat6'),
    ('Fred', 'Mon8'), ('Fred', 'Fri5'), ('Fred', 'Sat13'),
    ('Fred', 'Sun14'), ('Gu', 'Mon1'), ('Gu', 'Tue2'), ('Gu', 'Wed3'),
    ('Gu', 'Fri5'), ('Gu', 'Sat6'), ('Gu', 'Sun7'), ('Gu', 'Mon8'),
    ('Gu', 'Tue9'), ('Gu', 'Wed10'), ('Gu', 'Thu11'), ('Gu', 'Fri12'),
    ('Gu', 'Sat13'), ('Gu', 'Sun14')
})

# Model
m = gp.Model("assignment")

# Assignment variables: x[w,s] == 1 if worker w is assigned to shift s.
# Since an assignment model always produces integer solutions, we use
# continuous variables and solve as an LP.
x = m.addVars(availability, ub=1, name="x")

# The objective is to minimize the total pay costs
m.setObjective(gp.quicksum(pay[w]*x[w, s] for w, s in availability), GRB.MINIMIZE)

# Constraints: assign exactly shiftRequirements[s] workers to each shift s
reqCts = m.addConstrs((x.sum('*', s) == shiftRequirements[s]
    for s in shifts), "")

# Using Python looping constructs, the preceding statement would be...
#
# reqCts = {}
# for s in shifts:
#    reqCts[s] = m.addConstr(
#        gp.quicksum(x[w,s] for w,s in availability.select('*', s)) ==
#        shiftRequirements[s], s)

# Save model
m.write('workforce1.lp')

# Optimize
m.optimize()
status = m.status
if status == GRB.UNBOUNDED:
    print('The model cannot be solved because it is unbounded')
    sys.exit(0)
if status == GRB.OPTIMAL:
    print('The optimal objective is %g' % m.objVal)
    sys.exit(0)
if status != GRB.INF_OR_UNBD and status != GRB.INFEASIBLE:
    print('Optimization was stopped with status %d' % status)
    sys.exit(0)

# do IIS
print('The model is infeasible; computing IIS')
m.computeIIS()
if m.IISMinimal:
    print('IIS is minimal
')
else:
    print('IIS is not minimal
')
print('The following constraint(s) cannot be satisfied: ')
for c in m.getConstrs():
    if c.IISConstr:
        print(' %s' % c.constrName)

workforce2.py

#!/usr/bin/env python3.7

# Copyright 2020, Gurobi Optimization, LLC

# Assign workers to shifts; each worker may or may not be available on a
# particular day. If the problem cannot be solved, use IIS iteratively to
# find all conflicting constraints.
import gurobipy as gp
from gurobipy import GRB
import sys

# Number of workers required for each shift
shifts, shiftRequirements = gp.multidict({
    "Mon1": 3,
    "Mon2": 2, ...
})
# Amount each worker is paid to work one shift
workers, pay = gp.multidict({
    "Amy": 10,
    "Bob": 12,
    "Cathy": 10,
    "Dan": 8,
    "Ed": 8,
    "Fred": 9,
    "Gu": 11,
})

# Worker availability
availability = gp.tuplelist([
    ('Amy', 'Tue2'), ('Amy', 'Wed3'), ('Amy', 'Fri5'), ('Amy', 'Sun7'),
    ('Amy', 'Tue9'), ('Amy', 'Wed10'), ('Amy', 'Thu11'), ('Amy', 'Fri12'),
    ('Amy', 'Sat13'), ('Amy', 'Sun14'), ('Bob', 'Mon1'), ('Bob', 'Tue2'),
    ('Bob', 'Fri5'), ('Bob', 'Sat6'), ('Bob', 'Mon8'), ('Bob', 'Thu11'),
    ('Bob', 'Sat13'), ('Cathy', 'Wed3'), ('Cathy', 'Thu4'), ('Cathy', 'Fri5'),
    ('Cathy', 'Sun7'), ('Cathy', 'Mon8'), ('Cathy', 'Tue9'),
    ('Cathy', 'Wed10'), ('Cathy', 'Thu11'), ('Cathy', 'Fri12'),
    ('Cathy', 'Sat13'), ('Cathy', 'Sun14'), ('Dan', 'Tue2'), ('Dan', 'Wed3'),
    ('Dan', 'Fri5'), ('Dan', 'Sat6'), ('Dan', 'Mon8'), ('Dan', 'Tue9'),
    ('Dan', 'Wed10'), ('Dan', 'Thu11'), ('Dan', 'Fri12'), ('Dan', 'Sat13'),
    ('Dan', 'Sun14'), ('Ed', 'Mon1'), ('Ed', 'Tue2'), ('Ed', 'Wed3'),
    ('Ed', 'Thu4'), ('Ed', 'Fri5'), ('Ed', 'Sun7'), ('Ed', 'Mon8'),
    ('Ed', 'Tue9'), ('Ed', 'Thu11'), ('Ed', 'Sat13'), ('Ed', 'Sun14'),
    ('Fred', 'Mon1'), ('Fred', 'Tue2'), ('Fred', 'Wed3'), ('Fred', 'Sat6'),
    ('Fred', 'Mon8'), ('Fred', 'Tue9'), ('Fred', 'Fri12'), ('Fred', 'Sat13'),
    ('Fred', 'Sun14'), ('Gu', 'Mon1'), ('Gu', 'Tue2'), ('Gu', 'Wed3'),
    ('Gu', 'Fri5'), ('Gu', 'Sat6'), ('Gu', 'Sun7'), ('Gu', 'Mon8'),
    ('Gu', 'Tue9'), ('Gu', 'Wed10'), ('Gu', 'Thu11'), ('Gu', 'Fri12'),
    ('Gu', 'Sat13'), ('Gu', 'Sun14')
])

# Model
m = gp.Model("assignment")

# Assignment variables: x[w,s] == 1 if worker w is assigned to shift s.
# Since an assignment model always produces integer solutions, we use
# continuous variables and solve as an LP.
x = m.addVars(availability, ub=1, name="x")
# The objective is to minimize the total pay costs
m.setObjective(gp.quicksum(pay[w]*x[w, s] for w, s in availability), GRB.MINIMIZE)

# Constraint: assign exactly shiftRequirements[s] workers to each shift s
reqCts = m.addConstrs((x.sum('*', s) == shiftRequirements[s] for s in shifts), "_")

# Optimize
m.optimize()
status = m.status
if status == GRB.UNBOUNDED:
    print('The model cannot be solved because it is unbounded')
    sys.exit(0)
if status == GRB.OPTIMAL:
    print('The optimal objective is %g' % m.objVal)
    sys.exit(0)
if status != GRB.INF_OR_UNBD and status != GRB.INFEASIBLE:
    print('Optimization was stopped with status %d' % status)
    sys.exit(0)

# do IIS
print('The model is infeasible; computing IIS')
removed = []

# Loop until we reduce to a model that can be solved
while True:
    m.computeIIS()
    print('
The following constraint cannot be satisfied:
')
    for c in m.getConstrs():
        if c.IISConstr:
            print('%s' % c.constrName)
            # Remove a single constraint from the model
            removed.append(str(c.constrName))
            m.remove(c)
            break
    print('')
    m.optimize()
    status = m.status
    if status == GRB.UNBOUNDED:
        print('The model cannot be solved because it is unbounded')
        sys.exit(0)
    if status == GRB.OPTIMAL:
        break
    if status != GRB.INF_OR_UNBD and status != GRB.INFEASIBLE:
        print('Optimization was stopped with status %d' % status)
        sys.exit(0)

print('
The following constraints were removed to get a feasible LP:')
print(removed)

workforce3.py
#!/usr/bin/env python3.7

# Copyright 2020, Gurobi Optimization, LLC

# Assign workers to shifts; each worker may or may not be available on a
# particular day. If the problem cannot be solved, relax the model
# to determine which constraints cannot be satisfied, and how much
# they need to be relaxed.

import gurobipy as gp
from gurobipy import GRB
import sys

# Number of workers required for each shift
shifts , shiftRequirements = gp.multidict({
    "Mon1": 3,
    "Tue2": 2,
    "Wed3": 4,
    "Thu4": 4,
    "Fri5": 5,
    "Sat6": 6,
    "Sun7": 5,
    "Mon8": 2,
    "Tue9": 2,
    "Wed10": 3,
    "Thu11": 4,
    "Fri12": 6,
    "Sat13": 7,
    "Sun14": 5
})

# Amount each worker is paid to work one shift
workers , pay = gp.multidict({
    "Amy": 10,
    "Bob": 12,
    "Cathy": 10,
    "Dan": 8,
    "Ed": 8,
    "Fred": 9,
    "Gu": 11
})

# Worker availability
availability = gp.tupledict([
    ("Amy", 'Tue2'), ("Amy", 'Wed3'), ("Amy", 'Fri5'), ("Amy", 'Sun7'),
    ("Amy", 'Tue9'), ("Amy", 'Wed10'), ("Amy", 'Thu11'), ("Amy", 'Fri12'),
    ("Amy", 'Sat13'), ("Amy", 'Sun14'), ("Bob", 'Mon1'), ("Bob", 'Tue2'),
    ("Bob", 'Fri5'), ("Bob", 'Sat6'), ("Bob", 'Mon8'), ("Bob", 'Thu11'),
    ("Bob", 'Sat13'), ("Cathy", 'Wed3'), ("Cathy", 'Thu4'), ("Cathy", 'Fri5'),
    ("Cathy", 'Sun7'), ("Cathy", 'Mon8'), ("Cathy", 'Tue9'),
    ("Cathy", 'Wed10'), ("Cathy", 'Thu11'), ("Cathy", 'Fri12'),
    ("Cathy", 'Sat13'), ("Cathy", 'Sun14'), ("Dan", 'Tue2'), ("Dan", 'Wed3'),
    ("Dan", 'Fri5'), ("Dan", 'Sat6'), ("Dan", 'Mon8'), ("Dan", 'Tue9'),
    ("Dan", 'Wed10'), ("Dan", 'Thu11'), ("Dan", 'Fri12'), ("Dan", 'Sat13'),
    ("Dan", 'Sun14'), ("Ed", 'Mon1'), ("Ed", 'Tue2'), ("Ed", 'Wed3'),
    ("Ed", 'Thu4'), ("Ed", 'Fri5'), ("Ed", 'Sun7'), ("Ed", 'Mon8')
])

516
# Model
m = gp.Model("assignment")

# Assignment variables: x[w,s] == 1 if worker w is assigned to shift s.
# Since an assignment model always produces integer solutions, we use
# continuous variables and solve as an LP.
# The objective is to minimize the total pay costs.
# Constraint: assign exactly shiftRequirements[s] workers to each shift s.
# Optimize
# Relax the constraints to make the model feasible
print('%s = %g' % (sv.VarName, sv.X))

workforce4.py

#!/usr/bin/env python3.7
# Copyright 2020, Gurobi Optimization, LLC

# Assign workers to shifts; each worker may or may not be available on a
# particular day. We use lexicographic optimization to solve the model:
# first, we minimize the linear sum of the slacks. Then, we constrain
# the sum of the slacks, and we minimize a quadratic objective that
# tries to balance the workload among the workers.

import gurobipy as gp
from gurobipy import GRB
import sys

# Number of workers required for each shift
shifts, shiftRequirements = gp.multidict({
    "Mon1": 3,
    "Tue2": 2,
    "Wed3": 4,
    "Thu4": 4,
    "Fri5": 5,
    "Sat6": 6,
    "Sun7": 5,
    "Mon8": 2,
    "Tue9": 2,
    "Wed10": 3,
    "Thu11": 4,
    "Fri12": 6,
    "Sat13": 7,
    "Sun14": 5,
})

# Amount each worker is paid to work one shift
workers, pay = gp.multidict({
    "Amy": 10,
    "Bob": 12,
    "Cathy": 10,
    "Dan": 8,
    "Ed": 8,
    "Fred": 9,
    "Gu": 11,
})

# Worker availability
availability = gp.tuplelist([
    ('Amy', 'Tue2'), ('Amy', 'Wed3'), ('Amy', 'Fri5'), ('Amy', 'Sun7'),
    ('Amy', 'Tue9'), ('Amy', 'Wed10'), ('Amy', 'Thu11'), ('Amy', 'Fri12'),
    ('Amy', 'Sat13'), ('Amy', 'Sun14'), ('Bob', 'Mon1'), ('Bob', 'Tue2'),
    ('Bob', 'Fri5'), ('Bob', 'Sat6'), ('Bob', 'Mon8'), ('Bob', 'Thu11'),
    ('Bob', 'Sat13'), ('Cathy', 'Wed3'), ('Cathy', 'Thu4'), ('Cathy', 'Fri5'),
    ('Cathy', 'Sun7'), ('Cathy', 'Mon8'), ('Cathy', 'Tue9'),
    ('Cathy', 'Wed10'), ('Cathy', 'Thu11'), ('Cathy', 'Fri12'),
])
(('Cathy', 'Sat13'), ('Cathy', 'Sun14'), ('Dan', 'Tue2'), ('Dan', 'Wed3'), ('Dan', 'Fri5'), ('Dan', 'Sat6'), ('Dan', 'Mon8'), ('Dan', 'Tue9'), ('Dan', 'Wed10'), ('Dan', 'Thu11'), ('Dan', 'Fri12'), ('Dan', 'Sat13'), ('Dan', 'Sun14'), ('Ed', 'Mon1'), ('Ed', 'Tue2'), ('Ed', 'Wed3'), ('Ed', 'Thu4'), ('Ed', 'Fri5'), ('Ed', 'Sun7'), ('Ed', 'Mon8'), ('Ed', 'Tue9'), ('Ed', 'Thu11'), ('Ed', 'Sat13'), ('Ed', 'Sun14'), ('Fred', 'Mon1'), ('Fred', 'Tue2'), ('Fred', 'Wed3'), ('Fred', 'Sat6'), ('Fred', 'Mon8'), ('Fred', 'Tue9'), ('Fred', 'Fri12'), ('Fred', 'Sat13'), ('Fred', 'Sun14'), ('Gu', 'Mon1'), ('Gu', 'Tue2'), ('Gu', 'Wed3'), ('Gu', 'Fri5'), ('Gu', 'Sat6'), ('Gu', 'Sun7'), ('Gu', 'Mon8'), ('Gu', 'Tue9'), ('Gu', 'Wed10'), ('Gu', 'Thu11'), ('Gu', 'Fri12'), ('Gu', 'Sat13'), ('Gu', 'Sun14'))

# Model
m = gp.Model("assignment")

# Assignment variables: x[w,s] == 1 if worker w is assigned to shift s.
# This is no longer a pure assignment model, so we must use binary variables.
x = m.addVars(availability, vtype=GRB.BINARY, name="x")

# Slack variables for each shift constraint so that the shifts can
# be satisfied
slacks = m.addVars(shifts, name="Slack")

totSlack = m.addVar(name="totSlack")

# Variables to count the total shifts worked by each worker
totShifts = m.addVars(workers, name="TotShifts")

# Constraint: assign exactly shiftRequirements[s] workers to each shift s,
# plus the slack
reqCts = m.addConstrs((slacks[s] + x.sum('*', s) == shiftRequirements[s]
                       for s in shifts), "_")

# Constraint: set totSlack equal to the total slack
m.addConstr(totSlack == slacks.sum(), "totSlack")

# Constraint: compute the total number of shifts for each worker
m.addConstrs((totShifts[w] == x.sum(w) for w in workers), "totShifts")

# Objective: minimize the total slack
# Note that this replaces the previous 'pay' objective coefficients
m.setObjective(totSlack)

# Optimize
def solveAndPrint():
    m.optimize()
    status = m.status
    if status in (GRB.INF_OR_UNBD, GRB.INFEASIBLE, GRB.UNBOUNDED):
        print('The model cannot be solved because it is infeasible or \
unbounded')
sysexit(1)
if status != GRB.OPTIMAL:
    print('Optimization was stopped with status %d' % status)
sys.exit(0)

# Print total slack and the number of shifts worked for each worker
print('')
print('Total slack required: %g % totSlack.x)
for w in workers:
    print('%s worked %g shifts' % (w, totShifts[w].x))
print('')
solveAndPrint()

# Constrain the slack by setting its upper and lower bounds
totSlack.ub = totSlack.x
totSlack.lb = totSlack.x

# Variable to count the average number of shifts worked
avgShifts = m.addVar(name="avgShifts")

diffShifts = m.addVars(workers, lb=-GRB.INFINITY, name="Diff")

# Constraint: compute the average number of shifts worked
m.addConstr(len(workers) * avgShifts == totShifts.sum(), "avgShifts")

# Constraint: compute the difference from the average number of shifts
m.addConstrs((diffShifts[w] == totShifts[w] - avgShifts for w in workers), "Diff")

# Objective: minimize the sum of the square of the difference from the
# average number of shifts worked
m.setObjective(gp.quicksum(diffShifts[w]*diffShifts[w] for w in workers))

# Optimize
solveAndPrint()

workforce5.py

#!/usr/bin/env python3.7

# Copyright 2020, Gurobi Optimization, LLC

# Assign workers to shifts; each worker may or may not be available on a
# particular day. We use multi-objective optimization to solve the model.
# The highest-priority objective minimizes the sum of the slacks
# (i.e., the total number of uncovered shifts). The secondary objective
# minimizes the difference between the maximum and minimum number of
# shifts worked among all workers. The second optimization is allowed
# to degrade the first objective by up to the smaller value of 10% and 2 */

import gurobipy as gp
from gurobipy import GRB
import sys
# Sample data
# Sets of days and workers
Shifts = [
    "Mon1", "Tue2", "Wed3", "Thu4", "Fri5", "Sat6", "Sun7", "Mon8", "Tue9",
    "Wed10", "Thu11", "Fri12", "Sat13", "Sun14"
]

# Number of workers required for each shift
S = [3, 2, 4, 4, 5, 6, 5, 2, 3, 4, 6, 7, 5]
shiftRequirements = {s: S[i] for i, s in enumerate(Shifts)}

# Worker availability: 0 if the worker is unavailable for a shift
A = [
    [0, 1, 1, 0, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1],
    [1, 1, 0, 0, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1],
    [0, 0, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1],
    [1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1],
    [1, 1, 1, 0, 0, 1, 0, 1, 1, 1, 1, 1, 1, 1],
    [1, 1, 1, 0, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1],
    [0, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1],
    [1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1],
]
availability = {(w, s): A[j][i] for i, s in enumerate(Shifts) for j, w in enumerate(Workers)}

try:
    # Create model with a context manager. Upon exit from this block,
    # model.dispose is called automatically, and memory consumed by the model
    # is released.
    # The model is created in the default environment, which will be created
    # automatically upon model construction. For safe release of resources
    # tied to the default environment, disposeDefaultEnv is called below.
    with gp.Model("workforce5") as model:
        # Initialize assignment decision variables:
        # x[w][s] == 1 if worker w is assigned to shift s.
        # This is no longer a pure assignment model, so we must
        # use binary variables.
        x = model.addVars(availability.keys(), ub=availability, vtype=GRB.BINARY, name='x')

        # Slack variables for each shift constraint so that the shifts can
        # be satisfied
        slacks = model.addVars(Shifts, name='Slack')

        # Variable to represent the total slack
        totSlack = model.addVar(name='totSlack')

        # Variables to count the total shifts worked by each worker
        totShifts = model.addVars(Workers, name='TotShifts')
# Constraint: assign exactly shiftRequirements[s] workers to each shift s, plus the slack
model.addConstrs(
    (x.sum('*', s) + slacks[s] == shiftRequirements[s] for s in Shifts),
    name='shiftRequirement')

# Constraint: set totSlack equal to the total slack
model.addConstr(totSlack == slacks.sum(), name='totSlack')

# Constraint: compute the total number of shifts for each worker
model.addConstrs((totShifts[w] == x.sum(w, '*') for w in Workers),
                   name='totShifts')

# Constraint: set minShift/maxShift variable to less/greater than the number of shifts among all workers
minShift = model.addVar(name='minShift')
maxShift = model.addVar(name='maxShift')
model.addGenConstrMin(minShift, totShifts, name='minShift')
model.addGenConstrMax(maxShift, totShifts, name='maxShift')

# Set global sense for ALL objectives
model.ModelSense = GRB.MINIMIZE

# Set up primary objective
model.setObjectiveN(totSlack, index=0, priority=2, abstol=2.0, reltol=0.1, name='TotalSlack')

# Set up secondary objective
model.setObjectiveN(maxShift - minShift, index=1, priority=1, name='Fairness')

# Set global sense for ALL objectives
model.ModelSense = GRB.MINIMIZE

# Set up primary objective
model.setObjectiveN(totSlack, index=0, priority=2, abstol=2.0, reltol=0.1, name='TotalSlack')

# Set up secondary objective
model.setObjectiveN(maxShift - minShift, index=1, priority=1, name='Fairness')

# Set global sense for ALL objectives
model.ModelSense = GRB.MINIMIZE

# Set up primary objective
model.setObjectiveN(totSlack, index=0, priority=2, abstol=2.0, reltol=0.1, name='TotalSlack')

# Set up secondary objective
model.setObjectiveN(maxShift - minShift, index=1, priority=1, name='Fairness')

# Save problem
model.write('workforce5.lp')

# Optimize
model.optimize()

status = model.Status

if status in (GRB.INF_OR_UNBD, GRB.INFEASIBLE, GRB.UNBOUNDED):
    print('Model cannot be solved because it is infeasible or unbounded')
    sys.exit(0)

if status != GRB.OPTIMAL:
    print('Optimization was stopped with status ' + str(status))
    sys.exit(0)

# Print total slack and the number of shifts worked for each worker
print('')
print('Total slack required: ' + str(totSlack.X))
for w in Workers:
    print(w + ' worked ' + str(totShifts[w].X) + ' shifts')
print('')

except gp.GurobiError as e:
print('Error code ' + str(e.errno) + ": " + str(e))
except AttributeError as e:
    print('Encountered an attribute error: ' + str(e))
finally:
    # Safely release memory and/or server side resources consumed by
    # the default environment.
    gp.disposeDefaultEnv()

3.7 MATLAB Examples

This section includes source code for all of the Gurobi MATLAB examples. The same source code can be found in the examples/matlab directory of the Gurobi distribution.

bilinear.m

function bilinear
% This example formulates and solves the following simple bilinear model:
% maximize x
% subject to x + y + z <= 10
% x * y <= 2 (bilinear inequality)
% x * z + y * z = 1 (bilinear equality)
% x, y, z non-negative (x integral in second version)

% Linear constraint matrix
m.A = sparse([1, 1, 1]);
m.sense = '<';
m.rhs = 10;

% Variable names
m.varnames = {'x', 'y', 'z'};

% Objective function max 1.0 * x
m.obj = [1; 0; 0];
m.modelsense = 'max';

% Bilinear inequality constraint: x * y <= 2
m.quadcon(1).Qrow = 1;
m.quadcon(1).Qcol = 2;
m.quadcon(1).Qval = 1.0;
m.quadcon(1).q = sparse(3,1);
m.quadcon(1).rhs = 2.0;
m.quadcon(1).sense = '<';
m.quadcon(1).name = 'bilinear0';

% Bilinear equality constraint: x * z + y * z = 1
m.quadcon(2).Qrow = [1, 2];
m.quadcon(2).Qcol = [3, 3];
m.quadcon(2).Qval = [1.0, 1.0];
m.quadcon(2).q = sparse(3,1);
m.quadcon(2).rhs = 1.0;
m.quadcon(2).sense = '=';
m.quadcon(2).name = 'bilinear1';
function diet()
% Copyright 2020, Gurobi Optimization, LLC
%
% Solve the classic diet model
%
% Nutrition guidelines, based on
% USDA Dietary Guidelines for Americans, 2005
ncategories = 4;
categories = {'calories'; 'protein'; 'fat'; 'sodium'};
%
minNutrition maxNutrition
categorynutrition = [ 1800 2200; % calories
91 inf; % protein
0 65; % fat
0 1779]; % sodium

nfoods = 9;
foods = {'hamburger';
'chicken';
'hot dog';
'fries';
'macaroni';
'pizza';
'salad';
'milk';
'ice cream'};

foodcost = [2.49; % hamburger
2.89; % chicken
1.50; % hot dog
1.89; % fries
2.09; % macaroni
1.99; % pizza
2.49; % salad
0.89; % milk
1.59]; % ice cream

% calories protein fat sodium
nutritionValues = [ 410 24 26 730; % hamburger
420 32 10 1190; % chicken
560 20 32 1800; % hot dog
524

nutritionValues = sparse(nutritionValues);
model.modelName = 'diet';

% The variables are layed out as [ buy ; nutrition]
model.obj = [ foodcost; zeros(ncategories, 1)];
model.lb = [ zeros(nfoods, 1); categorynutrition(:, 1)];
model.ub = [ inf(nfoods, 1); categorynutrition(:, 2)];
model.A = [ nutritionValues' -speye(ncategories)];
model.rhs = zeros(ncategories, 1);
model.sense = repmat('=', ncategories, 1);

function printSolution(result)
    if strcmp(result.status, 'OPTIMAL')
        buy = result.x(1:nfoods);
        nutrition = result.x(nfoods+1:nfoods+ncategories);
        fprintf('
Cost: %f
', result.objval);
        fprintf('
Buy:
')
        for f =1:nfoods
            if buy(f) > 0.0001
                fprintf('%10 s %g
', foods{f}, buy(f));
            end
        end
        fprintf('
Nutrition:
')
        for c =1:ncategories
            fprintf('%10 s %g
', categories{c}, nutrition(c));
        end
    else
        fprintf('No solution
');
    end
end

% Solve
results = gurobi(model);
printSolution(results);

fprintf('
Adding constraint at most 6 servings of dairy
')
milk = find(strcmp('milk', foods));
icecream = find(strcmp('ice cream', foods));
model.A(end+1,:) = sparse([1; 1], [milk; icecream], 1, ... 
1, nfoods + ncategories);
model.rhs(end+1) = 6;
model.sense(end+1) = '<';

% Solve
results = gurobi(model);
printSolution(results)
function facility()

% Copyright 2020, Gurobi Optimization, LLC
%
% Facility location: a company currently ships its product from 5 plants
to 4 warehouses. It is considering closing some plants to reduce
costs. What plant(s) should the company close, in order to minimize
transportation and fixed costs?
%
% Note that this example uses lists instead of dictionaries. Since
it does not work with sparse data, lists are a reasonable option.
%
% Based on an example from Frontline Systems:
% http://www.solver.com/disfacility.htm
% Used with permission.

% define primitive data
nPlants = 5;
nWarehouses = 4;
% Warehouse demand in thousands of units
Demand = [15; 18; 14; 20];
% Plant capacity in thousands of units
Capacity = [20; 22; 17; 19; 18];
% Fixed costs for each plant
FixedCosts = [12000; 15000; 17000; 13000; 16000];
% Transportation costs per thousand units
TransCosts = [
    4000; 2000; 3000; 2500; 4500;
    2500; 2600; 3400; 3000; 4000;
    1200; 1800; 2600; 4100; 3000;
    2200; 2600; 3100; 3700; 3200];

% Index helper function
flowidx = @(w, p) nPlants * w + p;

% Build model
model.modelname = 'facility';
model.modelsense = 'min';

% Set data for variables
ncol = nPlants + nPlants * nWarehouses;
model.lb = zeros(ncol, 1);
model.ub = [ones(nPlants, 1); inf(nPlants * nWarehouses, 1)];
model.obj = [FixedCosts; TransCosts];
model.vtype = [repmat('B', nPlants, 1); repmat('C', nPlants * nWarehouses, 1)];

for p = 1:nPlants
    model.varnames{p} = sprintf('Open%d', p);
end

for w = 1:nWarehouses
    for p = 1:nPlants
        v = flowidx(w, p);
        model.varnames{v} = sprintf('Trans%d,%d', w, p);
% Set data for constraints and matrix
nrow = nPlants + nWarehouses;
model.A = sparse(nrow, ncol);
model.rhs = [zeros(nPlants, 1); Demand];
model.sense = [repmat('<', nPlants, 1); repmat('=', nWarehouses, 1)];

% Production constraints
for p = 1:nPlants
    for w = 1:nWarehouses
        model.A(p, p) = -Capacity(p);
        model.A(p, flowidx(w, p)) = 1.0;
    end
    model.constrnames{p} = sprintf('Capacity%d', p);
end

% Demand constraints
for w = 1:nWarehouses
    for p = 1:nPlants
        model.A(nPlants+w, flowidx(w, p)) = 1.0;
    end
    model.constrnames{nPlants+w} = sprintf('Demand%d', w);
end

% Save model
gurobi_write(model,'facility_m.lp');

% Guess at the starting point: close the plant with the highest fixed costs; open all others first open all plants
model.start = [ones(nPlants, 1); inf(nPlants * nWarehouses, 1)];
[-, idx] = max(FixedCosts);
model.start(idx) = 0;

% Set parameters
params.method = 2;

% Optimize
res = gurobi(model, params);

% Print solution
if strcmp(res.status, 'OPTIMAL')
    fprintf('
Total Costs: %g
', res.objval);
    fprintf('solution:
');
    for p = 1:nPlants
        if res.x(p) > 0.99
            fprintf('Plant %d open:
', p);
        end
        for w = 1:nWarehouses
            if res.x(flowidx(w, p)) > 0.0001
                fprintf( ' Transport %g units to warehouse %d
', res.x(flowidx(w, p)), w);
            end
        end
    end
else
end
fprintf('\n No solution\n');
end

function feasopt(filename)
% % Copyright 2020, Gurobi Optimization, LLC % % This example reads a MIP model from a file, adds artificial % variables to each constraint, and then minimizes the sum of the % artificial variables. A solution with objective zero corresponds % to a feasible solution to the input model. % We can also use FeasRelax feature to do it. In this example, we % use minrelax=1, i.e. optimizing the returned model finds a solution % that minimizes the original objective, but only from among those % solutions that minimize the sum of the artificial variables.

% Read model
fprintf('Reading model %s\n', filename);
model = gurobi_read(filename);

params.logfile = 'feasopt.log';
result1 = gurobi(model, params);

[rows, cols] = size(model.A);

% Create penalties, only linear constraints are allowed to be relaxed
penalties.rhs = ones(rows, 1);
result = gurobi_feasrelax(model, 0, true, penalties, params);
gurobi_write(result.model, 'feasopt1.lp');

% Clear objective
model.obj = zeros(cols, 1);

nvar = cols;
for c = 1:rows
    if model.sense(c) ~= '>
        nvar = nvar + 1;
        model.A(c, nvar) = -1;
        model.obj(nvar) = 1;
        model.vtype(nvar) = 'C';
        model.varnames(nvar) = strcat('ArtN_ ', model.constrnames(c));
        model.lb(nvar) = 0;
        model.ub(nvar) = inf;
    end
    if model.sense(c) ~= '<
        nvar = nvar + 1;
        model.A(c, nvar) = 1;
        model.obj(nvar) = 1;
        model.vtype(nvar) = 'C';
        model.varnames(nvar) = strcat('ArtP_ ', model.constrnames(c));
        model.lb(nvar) = 0;
    end
model.ub(nvar) = inf;
end
end
gurobi_write(model, 'feasopt2.lp');
result2 = gurobi(model, params);
end

fixanddive.m

function fixanddive(filename)
%
% Copyright 2020, Gurobi Optimization, LLC
%
% Implement a simple MIP heuristic. Relax the model,
% sort variables based on fractionality, and fix the 25% of
% the fractional variables that are closest to integer variables.
% Repeat until either the relaxation is integer feasible or
% linearly infeasible.
%
% Read model
fprintf(' Reading model %s
', filename);

model = gurobi_read(filename);
cols = size(model.A, 2);
ivars = find(model.vtype ~= 'C');

if length(ivars) <= 0
    fprintf('All variables of the model are continuous, nothing to do\n');
    return;
end

% save vtype and set all variables to continuous
vtype = model.vtype;
model.vtype = repmat('C', cols, 1);

params.OutputFlag = 0;
result = gurobi(model, params);

% Perform multiple iterations. In each iteration, identify the first
% quartile of integer variables that are closest to an integer value
% in the relaxation, fix them to the nearest integer, and repeat.
frac = zeros(cols, 1);
for iter = 1:1000
    % See if status is optimal
    if ~strcmp(result.status, 'OPTIMAL')
        fprintf('Model status is %s\n', result.status);
        fprintf('Can not keep fixing variables\n');
        break;
    end
    % collect fractionality of integer variables
    fracs = 0;
    for j = 1:cols
        % Collect fractionality of integer variables
        fracs(j) = model.vbrd(j);
    end
    % Find the first quartile of integer variables that satisfy the
    % condition that their vbrd is in the first quartile of fractionality
    idx = find(fracs <= quantile(frac, 0.25));
    % Fix the variables identified in the above loop
    for j = 1:length(idx)
        model.vbrd(idx(j)) = fixedvbrd(idx(j));
    end
    % Solve the relaxation
    result = gurobi(model, params);
    % Check if the solution is optimal
    if strcmp(result.status, 'OPTIMAL')
        fprintf('Solution is optimal\n');
        break;
    end
end

fprintf('Solution is linearly infeasible\n');
return;
if vtype(j) == 'C'
    frac(j) = 1; % indicating not integer variable
else
    t = result.x(j);
    t = t - floor(t);
    if t > 0.5
        t = t - 0.5;
    end
    if t > 1e-5
        frac(j) = t;
        fracs = fracs + 1;
    else
        frac(j) = 1; % indicating not fractional
    end
end
end
fprintf('Iteration %d, obj %g, fractional %d\n', iter, result.objval, fracs);
if fracs == 0
    fprintf('Found feasible solution - objective %g\n', result.objval);
    break;
end
% sort variables based on fractionality
[-, I] = sort(frac);
% fix the first quartile to the nearest integer value
nfix = max(fracs/4, 1);
for i = 1:nfix
    j = I(i);
    t = floor(result.x(j) + 0.5);
    model.lb(j) = t;
    model.ub(j) = t;
end
% use warm start basis and reoptimize
model.vbasis = result.vbasis;
model.cbasis = result.cbasis;
result = gurobi(model, params);
end

function gc_pwl
% Copyright 2020, Gurobi Optimization, LLC
%
% This example formulates and solves the following simple model
% with PWL constraints:
% maximize
%       sum c(j) * x(j)
% subject to
%       sum A(i,j) * x(j) <= 0, for i = 1, ..., m
%       sum y(j) <= 3
%       y(j) = pwl(x(j)), for j = 1, ..., n
\textbf{x(j)} free, \textbf{y(j)} \geq 0, \text{ for } j = 1, \ldots, n

where \text{pwl}(x) = 0, \text{ if } x = 0
= 1 + |x|, \text{ if } x \neq 0

\textbf{Note}
1. \text{sum pwl(x(j))} \leq b \text{ is to bound } x \text{ vector and also to favor sparse } x \text{ vector.}
   \text{Here } b = 3 \text{ means that at most two } x(j) \text{ can be nonzero and if two, then}
   \text{sum } x(j) \leq 1
2. \text{pwl}(x) \text{ jumps from 1 to 0 and from 0 to 1, if } x \text{ moves from negative 0 to 0,}
   \text{then to positive 0, so we need three points at } x = 0. \text{ x has infinite bounds}
   \text{on both sides, the piece defined with two points } (-1, 2) \text{ and } (0, 1) \text{ can}
   \text{extend x to } -\infty. \text{ Overall we can use five points } (-1, 2), (0, 1),
   (0, 0), (0, 1) \text{ and } (1, 2) \text{ to define } y = \text{pwl}(x)

n = 5;

% A x <= 0
A1 = [
    0, 0, 0, 1, -1;
    0, 0, 1, 1, -1;
    1, 1, 0, 0, -1;
    1, 0, 1, 0, -1;
    1, 0, 0, 1, -1;
    ];

% sum y(j) <= 3
A2 = [1, 1, 1, 1, 1];

% Constraint matrix altogether
model.A = sparse(blkdiag(A1, A2));

% Right-hand-side coefficient vector
model.rhs = [zeros(n,1); 3];

% Objective function (x coefficients arbitrarily chosen)
model.obj = [0.5, 0.8, 0.5, 0.1, -1, zeros(1, n)];

% It's a maximization model
model.modelsense = 'max';

% Lower bounds for x and y
model.lb = [-inf*ones(n,1); zeros(n,1)];

% PWL constraints
for k = 1:n
    model.genconpwl(k).xvar = k;
    model.genconpwl(k).yvar = n + k;
    model.genconpwl(k).xpts = [-1, 0, 0, 0, 1];
    model.genconpwl(k).ypts = [2, 1, 0, 1, 2];
end

result = gurobi(model);

for k = 1:n
    fprintf( '{\textbf{x}}(%d) = %g
', k, result.x(k));
fprintf('Objective value: %g
', result.objval);

function gc_pwl_func

% Copyright 2020, Gurobi Optimization, LLC
% This example considers the following nonconvex nonlinear problem
% maximize 2 x + y
% subject to exp(x) + 4 sqrt(y) <= 9
% x, y >= 0

% We show you two approaches to solve this:
% 1) Use a piecewise-linear approach to handle general function constraints (such as exp and sqrt).
% a) Add two variables
% u = exp(x)
% v = sqrt(y)
% b) Compute points (x, u) of u = exp(x) for some step length (e.g., x = 0, 1e-3, 2e-3, ..., xmax) and points (y, v) of v = sqrt(y) for some step length (e.g., y = 0, 1e-3, 2e-3, ..., ymax). We need to compute xmax and ymax (which is easy for this example, but this does not hold in general).
% c) Use the points to add two general constraints of type piecewise-linear.
% 2) Use the Gurobi built-in general function constraints directly (EXP and POW). Here, we do not need to compute the points and the maximal possible values, which will be done internally by Gurobi. In this approach, we show how to "zoom in" on the optimal solution and tighten tolerances to improve the solution quality.

% Four nonneg. variables x, y, u, v, one linear constraint u + 4*v <= 9
m.varnames = {'x', 'y', 'u', 'v'};
m.lb = zeros(4, 1);
m.ub = +inf(4, 1);
m.A = sparse([0, 0, 1, 4]);
m.rhs = 9;

% Objective
m.modelsense = 'max';
m.obj = [2; 1; 0; 0];

% First approach: PWL constraints
% Approximate u \approx exp(x), equispaced points in [0, xmax], xmax = log(9)
m.genconpwl(1).xvar = 1;
m.genconpwl(1).yvar = 3;
m.genconpwl(1).xpts = 0:1e-3:log(9);
m.genconpwl(1).ypts = exp(m.genconpwl(1).xpts);

% Approximate v \approx \sqrt{y}, equispaced points in [0, ymax], ymax = (9/4)^2
m.genconpwl(2).xvar = 2;
m.genconpwl(2).yvar = 4;
m.genconpwl(2).xpts = 0:1e-3:(9/4)^2;
m.genconpwl(2).ypts = sqrt(m.genconpwl(2).xpts);

% Solve and print solution
result = gurobi(m);
printsol(result.objval, result.x(1), result.x(2), result.x(3), result.x(4));

% Second approach: General function constraint approach with auto PWL
% translation by Gurobi

% Delete explicit PWL approximations from model
m = rmfield(m, 'genconpwl');

% Set u \approx \exp(x)
m.genconexp.xvar = 1;
m.genconexp.yvar = 3;
m.genconexp.name = 'gcf1';

% Set v \approx \sqrt{y} = y^{0.5}
m.genconpow.xvar = 2;
m.genconpow.yvar = 4;
m.genconpow.a = 0.5;
m.genconpow.name = 'gcf2';

% Parameters for discretization: use equal piece length with length = 1e-3
params.FuncPieces = 1;
params.FuncPieceLength = 1e-3;

% Solve and print solution
result = gurobi(m, params);
printsol(result.objval, result.x(1), result.x(2), result.x(3), result.x(4));

% Zoom in, use optimal solution to reduce the ranges and use a smaller
% pclen=1e-5 to resolve
m.lb(1) = max(m.lb(1), result.x(1) - 0.01);
m.ub(1) = min(m.ub(1), result.x(1) + 0.01);
m.lb(2) = max(m.lb(2), result.x(2) - 0.01);
m.ub(2) = min(m.ub(2), result.x(2) + 0.01);
params.FuncPieceLength = 1e-5;

% Solve and print solution
result = gurobi(m, params);
printsol(result.objval, result.x(1), result.x(2), result.x(3), result.x(4));
end

function printsol(objval, x, y, u, v)
    fprintf('x = %g, u = %g\n', x, u);
    fprintf('y = %g, v = %g\n', y, v);
    fprintf('Obj = %g\n', objval);
Calculate violation of \( \exp(x) + 4 \sqrt{y} \leq 9 \)

\[
vio = \exp(x) + 4 \times \sqrt{y} - 9;
\]

\[\text{if } vio < 0 \]
\[vio = 0;\]
\[\text{end}\]
\[\text{fprintf('Vio = \%g\n', vio);}\]
\[\text{end}\]

**genconstr.m**

```matlab
function genconstr()

% Copyright 2020, Gurobi Optimization, LLC
% %
% % In this example we show the use of general constraints for modeling
% % some common expressions. We use as an example a SAT-problem where we
% % want to see if it is possible to satisfy at least four (or all) clauses
% % of the logical for
% %
% % L = (x1 or \(\sim x2\) or \(x3\)) and (\(x2\) or \(\sim x3\) or \(x4\)) and
% % (\(x3\) or \(\sim x4\) or \(x1\)) and (\(x4\) or \(\sim x1\) or \(x2\)) and
% % (\(\sim x1\) or \(\sim x2\) or \(x3\)) and (\(\sim x2\) or \(\sim x3\) or \(x4\)) and
% % (\(\sim x3\) or \(\sim x4\) or \(x1\)) and (\(\sim x4\) or \(\sim x1\) or \(x2\))
% %
% % We do this by introducing two variables for each literal (itself and its
% % negated value), a variable for each clause, and then two
% % variables for indicating if we can satisfy four, and another to identify
% % the minimum of the clauses (so if it one, we can satisfy all clauses)
% % and put these two variables in the objective.
% % i.e. the Objective function will be
% %
% % maximize \(Obj1 + Obj2\)
% %
% % \(Obj1 = \min(\text{Clause2}, \ldots, \text{Clause8})\)
% % \(Obj2 = 2 \rightarrow \text{Clause2} + \ldots + \text{Clause8} \geq 4\)
% %
% % thus, the objective value will be two if and only if we can satisfy all
% % clauses; one if and only if at least four clauses can be satisfied, and
% % zero otherwise.
% %
% % define primitive data
n = 4;
nLiterals = 4;
nClauses = 8;
nObj = 2;
nVars = 2 * nLiterals + nClauses + nObj;
Clauses = [
    1, n+2, 3; 2, n+3, 4;
    3, n+4, 1; 4, n+1, 2;
    n+1, n+2, 3; n+2, n+3, 4;
    n+3, n+4, 1; n+4, n+1, 2
];

% Create model
```
model.modelname = 'genconstr';
model.modelsense = 'max';

% Set-up data for variables and constraints
model.vtype = repmat('B', nVars, 1);
model.ub = ones(nVars, 1);
model.obj = [zeros(2*nLiterals + nClauses, 1); ones(nObj, 1)];
model.A = sparse(nLiterals, nVars);
model.rhs = ones(nLiterals, 1);
model.sense = repmat('=', nLiterals, 1);

for j = 1:nLiterals
    model.varnames{j} = sprintf('X%d', j);
    model.varnames{ nLiterals+j} = sprintf('notX%d', j);
end
for j = 1:nClauses
    model.varnames{2*nLiterals+j} = sprintf('Clause%d', j);
end
for j = 1:nObj
    model.varnames{2*nLiterals+nClauses+j} = sprintf('Obj%d', j);
end

% Link Xi and notXi
for i = 1:nLiterals
    model.A(i, i) = 1;
    model.A(i, nLiterals+i) = 1;
    model.constrnames{i} = sprintf('CNSTR_X%d', i);
end

% Link clauses and literals
for i = 1:nClauses
    model.genconor(i).resvar = 2 * nLiterals + i;
    model.genconor(i).vars = Clauses(i:i,1:3);
    model.genconor(i).name = sprintf('CNSTR_Clause%d', i);
end

% Link objs with clauses
model.genconmin.resvar = 2 * nLiterals + nClauses + 1;
for i = 1:nClauses
    model.genconmin.vars(i) = 2 * nLiterals + i;
end
model.genconmin.name = 'CNSTR_Obj1';

model.genconind.binvar = 2 * nLiterals + nClauses + 2;
model.genconind.binval = 1;
model.genconind.a = [zeros(2*nLiterals,1); ones(nClauses,1); zeros(nObj,1)];
model.genconind.sense = '>';  
model.genconind.rhs = 4;
model.genconind.name = 'CNSTR_Obj2';

% Save model
fprintf(model, 'genconstr_m.lp');

% Optimize
params.logfile = 'genconstr.log';
result = gurobi(model, params);
if strcmp(result.status, 'OPTIMAL')
    if result.objval > 1.9
        fprintf('Logical expression is satisfiable\n');
    else
        if result.objval > 0.9
            fprintf('At least four clauses are satisfiable\n');
        else
            fprintf('At most three clauses may be satisfiable\n');
        end
    end
else
    fprintf('Optimization failed\n');
end

intlinprog.m

function [x,fval,exitflag,output] = intlinprog(f,intcon,A,b,Aeq,beq,lb,ub,x0,options)
% INTLINPROG A mixed integer programming (MIP) example using the
% Gurobi MATLAB interface
% This example is based on the intlinprog interface defined in the
% MATLAB Optimization Toolbox. The Optimization Toolbox
% is a registered trademark of The Math Works, Inc.
% x = INTLINPROG(f,intcon,A,b) solves the MIP problem:
% minimize f'*x
% subject to A*x <= b,
%        x(j) integer, where j is in the vector
%        intcon of integer constraints.
% For large problems, you can pass A as a sparse matrix and b as a
% sparse vector.
% x = INTLINPROG(f,intcon,A,b,Aeq,beq) solves the MIP problem:
% minimize f'*x
% subject to A*x <= b,
%        Aeq*x == beq,
%        x(j) integer, where j is in the vector
%        intcon of integer constraints.
% For large problems, you can pass Aeq as a sparse matrix and beq as a
% sparse vector. You can set A=[] and b=[] if no inequalities exist.
% x = INTLINPROG(f,intcon,A,b,Aeq,beq,lb,ub) solves the MIP problem:
% minimize f'*x
% subject to A*x <= b,
%        Aeq*x == beq,
%        lb <= x <= ub,
%        x(j) integer, where j is in the vector
%        intcon of integer constraints.
% intcon of integer constraints.
% You can set lb(j) = -inf, if x(j) has no lower bound, and ub(j) = inf,
% if x(j) has no upper bound. You can set Aeq=[] and beq=[] if no
% equalities exist.
% x = INTLINPROG(f, intcon, A, b, Aeq, beq, lb, ub, X0) solves the problem above
% with MIP start set to X0.
% You can set lb=[] or ub=[] if no bounds exist.
% x = INTLINPROG(f, intcon, A, b, Aeq, beq, lb, ub, x0, OPTIONS) solves the
% problem above given the specified OPTIONS. Only a subset of possible
% options have any effect:
% OPTIONS.Display 'off' or 'none' disables output,
% OPTIONS.MaxTime time limit in seconds,
% OPTIONS.MaxFeasiblePoints MIP feasible solution limit,
% OPTIONS.RelativeGapTolerance relative MIP optimality gap,
% OPTIONS.AbsoluteGapTolerance absolute MIP optimality gap.
% x = INTLINPROG(PROBLEM) solves PROBLEM, which is a structure that must
% have solver name 'intlinprog' in PROBLEM.solver. You can also specify
% any of the input arguments above using fields PROBLEM.f, PROBLEM.A, ...
% [x,fval] = INTLINPROG(f, intcon, A, b) returns the objective value at the
% solution. That is, fval = f'*x.
% [x,fval,exitflag] = INTLINPROG(f, intcon, A, b) returns an exitflag
% containing the status of the optimization. The values for exitflag and
% the corresponding status codes are:
% 2 stopped prematurely, integer feasible point found
% 1 converged to a solution
% 0 stopped prematurely, no integer feasible point found
% -2 no feasible point found
% -3 problem is unbounded
% [x,fval,exitflag,OUTPUT] = INTLINPROG(f, intcon, A, b) returns information
% about the optimization. OUTPUT is a structure with the following fields:
% OUTPUT.message Gurobi status code
% OUTPUT.relativegap relative MIP optimality gap
% OUTPUT.absolategap absolute MIP optimality gap
% OUTPUT.numnodes number of branch-and-cut nodes explored
% OUTPUT.constrviolation maximum violation for constraints and bounds
%
% Initialize missing arguments
if nargin == 1
    if isa(f,'struct') && isfield(f,'solver') && strcmpi(f.solver,'intlinprog')
        [f, intcon, A, b, Aeq, beq, lb, ub, x0, options] = probstruct2args(f);
    else
        error('PROBLEM should be a structure with valid fields');
    end
elseif nargin < 4 || nargin > 10
error('INTLINPROG: the number of input arguments is wrong');
elseif nargin < 10
    options = struct();
    if nargin == 9
        if isa(x0,'struct') || isa(x0,'optim.options.SolverOptions')
            options = x0; % x0 was omitted and options were passed instead
            x0 = [];% x0 was omitted and options were passed instead
        end
    else
        x0 = [];% x0 was omitted and options were passed instead
        if nargin < 8
            ub = [];
            if nargin < 7
                lb = [];
                if nargin < 6
                    beq = [];
                    if nargin < 5
                        Aeq = [];
                        end
                    end
                end
            end
        end
    end
end

% build Gurobi model
model.obj = f;
model.A = [sparse(A); sparse(Aeq)]; % A must be sparse
n = size(model.A, 2);
model.vtype = repmat('C', n, 1);
model.vtype(intcon) = 'I';
model.sense = [repmat('<',size(A,1),1); repmat('=',size(Aeq,1),1)];
model.rhs = full([b(:); beq(:)]); % rhs must be dense
if ~isempty(x0)
    model.start = x0;
end
if ~isempty(lb)
    model.lb = lb;
else
    model.lb = -inf(n,1); % default lb for MATLAB is -inf
end
if ~isempty(ub)
    model.ub = ub;
end

% Extract relevant Gurobi parameters from (subset of) options
params = struct();

if isfield(options,'Display') || isa(options,'optim.options.SolverOptions')
    if any(strcmp(options.Display,{'off','none'}))
        params.OutputFlag = 0;
    end
end

if isfield(options,'MaxTime') || isa(options,'optim.options.SolverOptions')
    params.TimeLimit = options.MaxTime;

end

if isfield(options,'MaxFeasiblePoints') || isa(options,'optim.options.SolverOptions')
    params.SolutionLimit = options.MaxFeasiblePoints;
end

if isfield(options,'RelativeGapTolerance') || isa(options,'optim.options.SolverOptions')
    params.MIPGap = options.RelativeGapTolerance;
end

if isfield(options,'AbsoluteGapTolerance') || isa(options,'optim.options.SolverOptions')
    params.MIPGapAbs = options.AbsoluteGapTolerance;
end

% Solve model with Gurobi
result = gurobi(model, params);

% Resolve model if status is INF_OR_UNBD
if strcmp(result.status,'INF_OR_UNBD')
    params.DualReductions = 0;
    warning('Infeasible or unbounded, resolve without dual reductions to determine...');
    result = gurobi(model, params);
end

% Collect results
x = [];
output.message = result.status;
output.relativegap = [];
output.absolutegap = [];
output.numnodes = result.nodecount;
output.constrviolation = [];

if isfield(result,'x')
    x = result.x;
    if nargout > 3
        slack = model.A*x-model.rhs;
        violA = slack(1:size(A,1));
        violAeq = norm(slack((size(A,1)+1):end),inf);
        viollb = model.lb()-x;
        violub = 0;
        if isfield(model,'ub')
            violub = x-model.ub();
        end
        output.constrviolation = max([0; violA; violAeq; viollb; violub]);
    end
end
fval = [];

if isfield(result,'objval')
    fval = result.objval;
    if nargout > 3 && numel(intcon) > 0
        U = fval;
end

539
\[ L = \text{result}.\text{objbound}; \]
\[ \text{output}.\text{relativegap} = 100\times(U-L)/(\text{abs}(U)+1); \]
\[ \text{output}.\text{absolutegap} = U-L; \]

```
if strcmp(result.status, 'OPTIMAL')
    exitflag = 1;
elseif strcmp(result.status, 'INFEASIBLE') || strcmp(result.status, 'CUTOFF')
    exitflag = -2;
elseif strcmp(result.status, 'UNBOUNDED')
    exitflag = -3;
else if isfield(result, 'x')
    exitflag = 2;
else
    exitflag = 0;
end
```

% Local Functions

```
function [f, intcon, A, b, Aeq, beq, lb, ub, x0, options] = probstruct2args(s)
% PROBSTRUCT2ARGS Get problem structure fields ([] is returned when missing)

f = getstructfield(s,'f');
intcon = getstructfield(s,'intcon');
A = getstructfield(s,'Aineq');
b = getstructfield(s,'bineq');
Aeq = getstructfield(s,'Aeq');
beq = getstructfield(s,'beq');
lb = getstructfield(s,'lb');
ub = getstructfield(s,'ub');
x0 = getstructfield(s,'x0');
options = getstructfield(s,'options');
```

```
function f = getstructfield(s, field)
% GETSTRUCTFIELD Get structure field ([] is returned when missing)

if isfield(s, field)
    f = getfield(s, field);
else
    f = [];
end
```

```
linprog.m
```

```
function [x, fval, exitflag, output, lambda] = linprog(f, A, b, Aeq, beq, lb, ub, x0, options)
% Copyright 2020, Gurobi Optimization, LLC
%
% LINPROG A linear programming example using the Gurobi MATLAB interface
%
% This example is based on the linprog interface defined in the
% MATLAB Optimization Toolbox. The Optimization Toolbox
% is a registered trademark of The Math Works, Inc.
%
% x = LINPROG(f,A,b) solves the linear programming problem:
```
minimize \ f^t \ x
subject to \ A \ x \leq \ b.

For large problems, you can pass A as a sparse matrix and b as a sparse vector.

x = Linprog(f,A,b) solves the problem:

minimize \ f^t \ x
subject to \ A \ x \leq \ b.

For large problems, you can pass A as a sparse matrix and b as a sparse vector. You can set A=[] and b=[] if no inequalities exist.

x = Linprog(f,A,b,Aeq,beq) solves the problem:

minimize \ f^t \ x
subject to \ A \ x \leq \ b,
   \ Aeq \ x = beq.

For large problems, you can pass Aeq as a sparse matrix and beq as a sparse vector. You can set A=[] and beq=[] if no equalities exist.

x = Linprog(f,A,b,Aeq,beq,lb,ub) solves the problem:

minimize \ f^t \ x
subject to \ A \ x \leq \ b,
   \ Aeq \ x = beq,
   \ lb \leq \ x \leq \ ub.

You can set lb(j) = -inf, if x(j) has no lower bound, and ub(j) = inf, if x(j) has no upper bound. You can set Aeq=[] and beq=[] if no equalities exist.

x = Linprog(f,A,b,Aeq,beq,lb,ub,OPTIONS) solves the problem above given the specified OPTIONS. Only a subset of possible options have any effect:

OPTIONS.Display 'off' or 'none' disables output,
OPTIONS.MaxTime time limit in seconds.

You can set lb=[] or ub=[] if no bounds exist.

x = Linprog(PROBLEM) solves PROBLEM, which is a structure that must have solver name 'linprog' in PROBLEM.solver. You can also specify any of the input arguments above using fields PROBLEM.f, PROBLEM.A, ...

[x,fval] = Linprog(f,A,b) returns the objective value at the solution. That is, fval = f^t x.

[x,fval,exitflag] = Linprog(f,A,b) returns an exitflag containing the status of the optimization. The values for exitflag and the corresponding status codes are:

1 converged to a solution (OPTIMAL),
0 maximum number of iterations reached (ITERATION_LIMIT),
-2 no feasible point found (INFEASIBLE, NUMERIC, ...),
-3 problem is unbounded (UNBOUNDED).

[x,fval,exitflag,OUTPUT] = Linprog(f,A,b) returns information about the optimization. OUTPUT is a structure with the following fields:

OUTPUT.message Gurobi status code
% OUTPUT.constrviolation maximum violation for constraints and bounds
% [x,fval,exitflag,OUTPUT,LAMBDA] = LINPROG(f,A,b) returns the
% Lagrangian multipliers at the solution. LAMBDA is a structure with
% the following fields:
% LAMBDA.lower multipliers corresponding to x >= lb
% LAMBDA.upper multipliers corresponding to x <= ub
% LAMBDA.ineqlin multipliers corresponding to A*x <= b
% LAMBDA.eqlin multipliers corresponding to Aeq*x == beq
%
% Initialize missing arguments
if nargin == 1
    if isa(f,'struct') && isfield(f,'solver') && strcmpi(f.solver,'linprog')
        [f,A,b,Aeq,beq,lb,ub,x0,options] = probstruct2args(f);
    else
        error('PROBLEM should be a structure with valid fields');
    end
elseif nargin < 3 || nargin > 9
    error('LINPROG: the number of input arguments is wrong');
elseif nargin < 9
    options = struct();
    if nargin == 8
        if isa(x0,'struct') || isa(x0,'optim.options.SolverOptions')
            options = x0; % x0 was omitted and options were passed instead
            x0 = [];
        else
            x0 = [];
            if nargin < 7
                ub = [];
                if nargin < 6
                    lb = [];
                    if nargin < 5
                        beq = [];
                        if nargin < 4
                            Aeq = [];
                        end
                    end
                end
            end
        end
    end
end
%
% Warn user if x0 argument is ignored
if ~isempty(x0)
    warning('LINPROG will ignore non-empty starting point X0');
end
%
% Build Gurobi model
model.obj = f;
model.A = [sparse(A); sparse(Aeq)]; % A must be sparse
model.sense = [repmat('<',size(A,1),1); repmat('=',size(Aeq,1),1)];
model.rhs = full([b(:); beq(:)]); % rhs must be dense
if ~isempty(lb)
model.lb = lb;
else
    model.lb = -inf(size(model.A,2),1); % default lb for MATLAB is -inf
end
if ~isempty(ub)
    model.ub = ub;
end

% Extract relevant Gurobi parameters from (subset of) options
params = struct();
if isfield(options,'Display') || isa(options,'optim.options.SolverOptions')
    if any(strcmp(options.Display,{ 'off','none'}))
        params.OutputFlag = 0;
    end
end
if isfield(options,'MaxTime') || isa(options,'optim.options.SolverOptions')
    params.TimeLimit = options.MaxTime;
end

% Solve model with Gurobi
result = gurobi(model, params);

% Resolve model if status is INF_OR_UNBD
if strcmp(result.status,'INF_OR_UNBD')
    params.DualReductions = 0;
    warning('Infeasible or unbounded, resolve without dual reductions to determine...');
    result = gurobi(model, params);
end

% Collect results
x = [];
output.message = result.status;
output.constrviolation = [];
if isfield(result,'x')
    x = result.x;
    if nargout > 3
        slack = model.A*x-model.rhs;
        violA = slack(1:size(A,1));
        violAeq = norm(slack((size(A,1)+1):end),inf);
        viollb = model.lb(:)-x;
        violub = 0;
        if isfield(model,'ub')
            violub = x-model.ub(:);
        end
        output.constrviolation = max([0; violA; violAeq; viollb; violub]);
    end
end
fval = [];
if isfield(result,'objval')
    fval = result.objval;
end
if strcmp(result.status,'OPTIMAL')
    exitflag = 1; % converged to a solution
elseif strcmp(result.status,'UNBOUNDED')
    exitflag = -3; % problem is unbounded
elseif strcmp(result.status,'ITERATION_LIMIT')
    exitflag = 0; % maximum number of iterations reached
else
    exitflag = -2; % no feasible point found
end

lambda.lower = [];
lambda.upper = [];
lambda.ineqlin = [];
lambda.eqlin = [];

if nargout > 4
    if isfield(result,'rc')
        lambda.lower = max(0,result.rc);
        lambda.upper = -min(0,result.rc);
    end
    if isfield(result,'pi')
        if ~isempty(A)
            lambda.ineqlin = -result.pi(1:size(A,1));
        end
        if ~isempty(Aeq)
            lambda.eqlin = -result.pi((size(A,1)+1):end);
        end
    end
end

% Local Functions =========================================================

function [f,A,b,Aeq,beq,lb,ub,x0,options] = probstruct2args(s)
%PROBSTRUCT2ARGS Get problem structure fields ([] is returned when missing)

f = getstructfield(s,'f');
A = getstructfield(s,'Aineq');
b = getstructfield(s,'bineq');
Aeq = getstructfield(s,'Aeq');
beq = getstructfield(s,'beq');
lb = getstructfield(s,'lb');
ub = getstructfield(s,'ub');
x0 = getstructfield(s,'x0');
options = getstructfield(s,'options');

function f = getstructfield(s,field)
%GETSTRUCTFIELD Get structure field ([] is returned when missing)

if isfield(s,field)
    f = getfield(s,field);
else
    f = [];
end
function lp()
% Copyright 2020, Gurobi Optimization, LLC
%
% This example formulates and solves the following simple LP model:
% maximize
% x + 2 y + 3 z
% subject to
% x + y <= 1
% y + z <= 1
%
    model.A = sparse([1, 1, 0; 0, 1, 1]);
    model.obj = [1, 2, 3];
    model.modelsense = 'Max';
    model.rhs = [1, 1];
    model.sense = ['<', '<'];

    result = gurobi(model);
    disp(result.objval);
    disp(result.x);

% Alternative representation of A - as sparse triplet matrix
    i = [1; 1; 2; 2];
    j = [1; 2; 2; 3];
    x = [1; 1; 1; 1];
    model.A = sparse(i, j, x, 2, 3);

% Set some parameters
    params.method = 2;
    params.timelimit = 100;

    result = gurobi(model, params);
    disp(result.objval);
    disp(result.x)
end

function lp2()
% Copyright 2020, Gurobi Optimization, LLC
%
% Formulate a simple linear program, solve it, and then solve it
% again using the optimal basis.

    model.A = sparse([1, 3, 4; 8, 2, 3]);
    model.obj = [1, 2, 3];
    model.rhs = [4, 7];
    model.sense = ['>', '>'];

% First solve requires a few simplex iterations
    result = gurobi(model)
Second solve - start from an optimal basis, so no iterations
model.vbasis = result.vbasis;
model.cbasis = result.cbasis;
result = gurobi(model)
end

lpmethod.m

function lpmethod(filename)
    % Copyright 2020, Gurobi Optimization, LLC
    %
    % Solve a model with different values of the Method parameter;
    % show which value gives the shortest solve time.
    %
    % Read model
    fprintf('Reading model %s\n', filename);
    model = gurobi_read(filename);
    bestTime = inf;
    bestMethod = -1;
    for i = 0:4
        params.method = i;
        res = gurobi(model, params);
        if strcmp(res.status, 'OPTIMAL')
            bestMethod = i;
            bestTime = res.runtime;
            params.TimeLimit = bestTime;
        end
    end
    % Report which method was fastest
    if bestMethod == -1
        fprintf('Unable to solve this model\n');
    else
        fprintf('Solved in %g seconds with Method %d\n', bestTime, bestMethod);
    end

lpmod.m

function lpmod(filename)
    % Copyright 2020, Gurobi Optimization, LLC
    %
    % This example reads an LP model from a file and solves it.
    % If the model can be solved, then it finds the smallest positive variable,
    % sets its upper bound to zero, and resolves the model two ways:
    % first with an advanced start, then without an advanced start
    % (i.e. 'from scratch').
    %
    % Read model
    fprintf('Reading model %s\n', filename);
    model = gurobi_read(filename);
    if (isfield(model, 'multiobj') && ~isempty(model.multiobj)) || ...
(isfield(model, 'sos') && ~isempty(model.sos)) || ... 
(isfield(model, 'pwlobj') && ~isempty(model.pwlobj)) || ... 
(isfield(model, 'quadcon') && ~isempty(model.quadcon)) || ... 
(isfield(model, 'genconstr') && ~isempty(model.genconstr)) || ... 
isfield(model, 'Q') 
fprintf('The model is not a linear program, quit\n');  
return;
end

ivars = find(model.vtype ~= 'C');
ints = length(ivars);
if ints > 0
    fprintf('problem is a MIP, quit\n');
    return;
end

result = gurobi(model);
if ~strcmp(result.status, 'OPTIMAL')
    fprintf('This model cannot be solved because its optimization status is %s\n', result.status);
    return;
end

cols = size(model.A,2);

%c create lb if they do not exists, and set them to default values
if ~isfield(model, 'lb') || isempty(model.lb)
    model.lb = zeros(cols, 1);
end

% Find the smallest variable value
minVal = inf;
minVar = -1;
for j = 1:cols
    if result.x(j) > 0.0001 && result.x(j) < minVal && model.lb(j) == 0.0
        minVal = result.x(j);
        minVar = j;
    end
end

fprintf('\n*** Setting %s from %d to zero ***\n', model.varnames{minVar}, minVar);
model.ub(minVar) = 0;

model.vbasis = result.vbasis;
model.cbasis = result.cbasis;

% Solve from this starting point
result = gurobi(model);

% Save iteration & time info
warmCount = result.itercount;
warmTime = result.runtime;

% Remove warm start basis and resolve
model = rmfield(model, 'vbasis');
model = rmfield(model, 'cbasis');
result = gurobi(model);

coldCount = result.itercount;
coldTime = result.runtime;

fprintf('\n);
fprintf('*** Warm start: %g iterations, %g seconds\n', warmCount, warmTime);
fprintf('*** Cold start: %g iterations, %g seconds\n', coldCount, coldTime);

mip1.m

function mip1()

% Copyright 2020, Gurobi Optimization, LLC
% This example formulates and solves the following simple MIP model:
% maximize
% x + y + 2 z
% subject to
% x + 2 y + 3 z <= 4
% x + y >= 1
% x, y, z binary

names = {'x'; 'y'; 'z'};

model.A = sparse([1 2 3; 1 1 0]);
model.obj = [1 1 2];
model.rhs = [4; 1];
model.sense = '<>';
model.vtype = 'B';
model.modelsense = 'max';
model.varnames = names;

gurobi_write(model, 'mip1.lp');

params.outputflag = 0;

result = gurobi(model, params);

disp(result);

for v = 1:length(names)
    fprintf('%s %d\n', names{v}, result.x(v));
end

fprintf('Obj: %e\n', result.objval);
end

mip2.m

function mip2(filename)

% Copyright 2020, Gurobi Optimization, LLC
% This example reads a MIP model from a file, solves it and prints
% the objective values from all feasible solutions generated while
% solving the MIP. Then it creates the associated fixed model and
% solves that model.
% Read model
fprintf('Reading model %s\n', filename);

model = gurobi_read(filename);

cols = size(model.A, 2);

ivars = find(model.vtype ~= 'C');
ints = length(ivars);

if ints <= 0
    fprintf('All variables of the model are continuous, nothing to do\n');
    return;
end

% Optimize
params.poolsolutions = 20;
result = gurobi(model, params);

% Capture solution information
if ~strcmp(result.status, 'OPTIMAL')
    fprintf('This model cannot be solved because its optimization status is %s\n', ...
            result.status);
    return;
end

% Iterate over the solutions
if isfield(result, 'pool') && ~isempty(result.pool)
    solcount = length(result.pool);
    for k = 1:solcount
        fprintf('Solution %d has objective %g\n', k, result.pool(k).objval);
    end
else
    fprintf('Solution 1 has objective %g\n', result.objval);
end

% Convert to fixed model
for j = 1:cols
    if model.vtype(j) ~= 'C'
        t = floor(result.x(j) + 0.5);
        model.lb(j) = t;
        model.ub(j) = t;
    end
end

% Solve the fixed model
result2 = gurobi(model, params);
if ~strcmp(result.status, 'OPTIMAL')
    fprintf('Error: fixed model is not optimal\n');
    return;
end
if abs(result.objval - result2.objval) > 1e-6 * (1 + abs(result.objval))
    fprintf('Error: Objective values differ\n');
end
% Print values of non-zero variables
for j = 1:cols
    if abs(result2.x(j)) > 1e-6
        fprintf('%s %g\n', model.varnames{j}, result2.x(j));
    end
end

multiobj.m

function multiobj()

% Copyright 2020, Gurobi Optimization, LLC
%
% Want to cover three different sets but subject to a common budget of
% elements allowed to be used. However, the sets have different priorities to
% be covered; and we tackle this by using multi-objective optimization.

% define primitive data
groundSetSize = 20;
nSubSets = 4;
Budget = 12;
Set = [
    1 1 1 1 1 1 1 1 0 0 0 0 0 0 0 0 0 0 0 0;
    0 0 0 0 0 1 1 1 1 0 0 0 0 0 1 1 1 1 1;
    0 0 0 1 1 0 1 1 0 0 0 0 0 1 1 0 1 1 0 0;
    0 0 0 1 1 0 0 0 1 1 1 0 0 0 1 1 1 0 0 0
];
SetObjPriority = [3; 2; 2; 1];
SetObjWeight = [1.0; 0.25; 1.25; 1.0];

% Initialize model
model.modelsense = 'max';
model.modelname = 'multiobj';

% Set variables and constraints
model.vtype = repmat('B', groundSetSize, 1);
model.lb = zeros(groundSetSize, 1);
model.ub = ones(groundSetSize, 1);
model.A = sparse(1, groundSetSize);
model.rhs = Budget;
model.sense = '<';
model.constrnames = {'Budget'};
for j = 1:groundSetSize
    model.varnames{j} = sprintf('El%d', j);
    model.A(1, j) = 1;
end

% Set multi-objectives
for m = 1:nSubSets
    model.multiobj(m).objn = Set(m, :);
    model.multiobj(m).priority = SetObjPriority(m);
    model.multiobj(m).weight = SetObjWeight(m);
    model.multiobj(m).abstol = m;
    model.multiobj(m).reltol = 0.01;
    model.multiobj(m).name = sprintf('Set%d', m);
end
model.multiobj(m).con = 0.0;
end

% Save model
gurobi_write(model,'multiobj_m.lp')

% Set parameters
params.PoolSolutions = 100;

% Optimize
result = gurobi(model, params);

% Capture solution information
if ~strcmp(result.status, 'OPTIMAL')
    fprintf('Optimization finished with status %d, quit now\n', result.status);
    return;
end

% Print best solution
fprintf('Selected elements in best solution:\n');
for j = 1:groundSetSize
    if result.x(j) >= 0.9
        fprintf('%s ', model.varnames{j});
    end
end
fprintf('\n);

% Print all solution objectives and best furth solution
if isfield(result, 'pool') && ~isempty(result.pool)
    solcount = length(result.pool);
    fprintf('Number of solutions found: %d\n', solcount);
    fprintf('Objective values for first %d solutions:\n', solcount);
    for m = 1:nSubSets
        fprintf(' %s:', model.multiobj(m).name);
        for k = 1:solcount
            fprintf(' %3g', result.pool(k).objval(m));
        end
        fprintf('\n');
    end
    fprintf('\n');
else
    fprintf('Number of solutions found: 1\n');
    fprintf('Solution 1 has objective values:');
    for k = 1:nSubSets
        fprintf(' %g', result.objval(k));
    end
    fprintf('\n');
end

opttoolbox_lp.m

function opttoolbox_lp()
% Copyright 2020, Gurobi Optimization, LLC
% % This example uses Matlab 2017b problem based modeling feature, which
% requires Optimization Toolbox, to formulate and solve the following
% simple LP model, the same model as for lp.m
% maximize
% \[ \begin{align*}
& x + 2y + 3z \\
& \text{subject to} \\
& x + y \leq 1 \\
& y + z \leq 1
\end{align*} \]
% To use Gurobi with this example, linprog.m must be in the current
% directory or added to Matlab path

x = optimvar('x', 'LowerBound',0);
y = optimvar('y', 'LowerBound',0);
z = optimvar('z', 'LowerBound',0);

prob = optimproblem('ObjectiveSense','maximize');
prob.Objective = x + 2 * y + 3 * z;
prob.Constraints.cons1 = x + y <= 1;
prob.Constraints.cons2 = y + z <= 1;
options = optimoptions('linprog');

% For Matlab R2017b use the following
% sol = solve(prob, options)
% Syntax for R2018a and later
% sol = solve(prob, 'Options', options);
end

opttoolbox_mip1.m

function opttoolbox_mip1()
% Copyright 2020, Gurobi Optimization, LLC
% This example uses Matlab 2017b problem based modeling feature, which
% requires Optimization Toolbox, to formulate and solve the following
% simple MIP model, the same model as for mip1.m
% maximize
% \[ \begin{align*}
& x + y + 2z \\
& \text{subject to} \\
& x + 2y + 3z \leq 4 \\
& x + y \geq 1
\end{align*} \]
% To use Gurobi with this example, intlinprog.m must be in the current
% directory or added to Matlab path

x = optimvar('x', 'Type','integer','LowerBound',0,'UpperBound',1);
y = optimvar('y', 'Type','integer','LowerBound',0,'UpperBound',1);
z = optimvar('z', 'Type','integer','LowerBound',0,'UpperBound',1);

prob = optimproblem('ObjectiveSense','maximize');
prob.Objective = x + y + 2 * z;
prob.Constraints.cons1 = x + 2 * y + 3 * z <= 4;
prob.Constraints.cons2 = x + y >= 1;

options = optimoptions('intlinprog');

% For Matlab R2017b use the following
% sol = solve(prob, options)

% Syntax for R2018a and later
sol = solve(prob, 'Options', options);
end

params.m

function params(filename)
% Copyright 2020, Gurobi Optimization, LLC
%
% Use parameters that are associated with a model.
%%
% A MIP is solved for a few seconds with different sets of parameters.
% The one with the smallest MIP gap is selected, and the optimization
% is resumed until the optimal solution is found.
%
% Read model
fprintf('Reading model %s\n', filename);

model = gurobi_read(filename);

ivars = find(model.vtype ~= 'C');

if length(ivars) <= 0
    fprintf('All variables of the model are continuous, nothing to do\n');
    return;
end

% Set a 2 second time limit
params.TimeLimit = 2;

% Now solve the model with different values of MIPFocus

params.MIPFocus = 0;
result = gurobi(model, params);
bestgap = result.mipgap;
bestparams = params;
for i = 1:3
    params.MIPFocus = i;
    result = gurobi(model, params);
    if result.mipgap < bestgap
        bestgap = result.mipgap;
        bestparams = params;
    end
end

end
Finally, reset the time limit and Re-solve model to optimality
bestparams.TimeLimit = Inf;
result = gurobi(model, bestparams);
fprintf('Solution status: %s, objective value %g\n', ...
    result.status, result.objval);
end

function piecewise()
% Copyright 2020, Gurobi Optimization, LLC
%
% This example considers the following separable, convex problem:
% minimize f(x) - y + g(z)
% subject to x + 2 y + 3 z <= 4
% x + y >= 1
% x, y, z <= 1
% where f(u) = exp(-u) and g(u) = 2 u^2 - 4 u, for all real u. It
% formulates and solves a simpler LP model by approximating f and
% g with piecewise-linear functions. Then it transforms the model
% into a MIP by negating the approximation for f, which corresponds
% to a non-convex piecewise-linear function, and solves it again.

names = {'x'; 'y'; 'z'};

model.A = sparse([1 2 3; 1 1 0]);
model.obj = [0; -1; 0];
model.rhs = [4; 1];
model.sense = '<>l';
model.vtype = 'C';
model.lb = [0; 0; 0];
model.ub = [1; 1; 1];
model.varnames = names;

% Compute f and g on 101 points in [0,1]
u = linspace(0.0, 1.0, 101);
f = exp(-u);
g = 2*u.^2 - 4*u;

% Set piecewise-linear objective f(x)
model.pwlobj(1).var = 1;
model.pwlobj(1).x = u;
model.pwlobj(1).y = f;

% Set piecewise-linear objective g(z)
model.pwlobj(2).var = 3;
model.pwlobj(2).x = u;
model.pwlobj(2).y = g;

% Optimize model as LP
result = gurobi(model);

disp(result);
for v=1:length(names)
    fprintf('%s %d
', names{v}, result.x(v));
end

fprintf('Obj: %e
', result.objval);

model.pwlobj(1).y = f;
gurobi_write(model, 'pwl.lp')

result = gurobi(model);
disp(result);

for v=1:length(names)
    fprintf('%s %d
', names{v}, result.x(v));
end

fprintf('Obj: %e
', result.objval);
end

function poolsearch()

% Copyright 2020, Gurobi Optimization, LLC
%
% We find alternative epsilon-optimal solutions to a given knapsack problem by using PoolSearchMode
%
% define primitive data
groundSetSize = 10;
objCoef = [32; 32; 15; 15; 6; 6; 1; 1; 1; 1];
knapsackCoef = [16, 16, 8, 8, 4, 4, 2, 2, 1, 1];
Budget = 33;

% Initialize model
model.modelsense = 'max';
model.modelname = 'poolsearch';

% Set variables
model.obj = objCoef;
model.vtype = repmat('B', groundSetSize, 1);
model.lb = zeros(groundSetSize, 1);
model.ub = ones(groundSetSize, 1);
for j = 1:groundSetSize
    model.varnames{j} = sprintf('El%d', j);
end

% Build constraint matrix
model.A = sparse(knapsackCoef);
model.rhs = Budget;
model.sense = '<';
model.constrnames = {'Budget'};

% Set poolsearch parameters
params.PoolSolutions = 1024;
params.PoolGap = 0.10;
params.PoolSearchMode = 2;

% Save problem
gurobi_write(model, 'poolsearch_m.lp');

% Optimize
result = gurobi(model, params);

% Capture solution information
if ~strcmp(result.status, 'OPTIMAL')
    fprintf('Optimization finished with status %s, quit now\n', result.status);
    return;
end

% Print best solution
fprintf('Selected elements in best solution:\n');
for j = 1:groundSetSize
    if result.x(j) >= 0.9
        fprintf('%s ', model.varnames{j});
    end
end
fprintf('\n');

% Print all solution objectives and best further solution
if isfield(result, 'pool') && ~isempty(result.pool)
    solcount = length(result.pool);
    fprintf('Number of solutions found: %d\n', solcount);
    fprintf('Objective values for all %d solutions:\n', solcount);
    for k = 1:solcount
        fprintf('%g ', result.pool(k).objval);
    end
    fprintf('\n');
    if solcount >= 4
        fprintf('Selected elements in fourth best solution:\n');
        for k = 1:groundSetSize
            if result.pool(4).xn(k) >= 0.9
                fprintf(' %s', model.varnames{k});
            end
        end
        fprintf('\n');
    end
else
    fprintf('Number of solutions found: 1\n');
    fprintf('Solution 1 has objective: %g \n', result.objval);
end

qcp.m

function qcp()
% Copyright 2020, Gurobi Optimization, LLC
%
This example formulates and solves the following simple QCP model:

maximize

subject to

\[ x + y + z = 1 \]
\[ x^2 + y^2 \leq z^2 \quad \text{(second-order cone)} \]
\[ x^2 \leq yz \quad \text{(rotated second-order cone)} \]
\[ x, y, z \text{ non-negative} \]

names = {'x', 'y', 'z'};
model.varnames = names;

Set objective: x
model.obj = [1 0 0];
model.modelsense = 'max';

Add constraint: x + y + z = 1
model.A = sparse([1 1 1]);
model.rhs = 1;
model.sense = '=';

Add second-order cone: \( x^2 + y^2 \leq z^2 \) using a sparse matrix
model.quadcon(1).Qc = sparse([1 0 0; 0 1 0; 0 0 -1]);
model.quadcon(1).q = zeros(3,1);
model.quadcon(1).rhs = 0.0;
model.quadcon(1).name = 'std_cone';

Add rotated cone: \( x^2 \leq yz \) using sparse triplet representation
% Equivalent sparse matrix data:
% model.quadcon(2).Qc = sparse([1 0 0; 0 0 -1; 0 0 0]);
model.quadcon(2).Qrow = [1, 2]
model.quadcon(2).Qcol = [1, 3]
model.quadcon(2).Qval = [1, -1]
% All-zero sparse 3-by-1 vector
model.quadcon(2).q = sparse(3,1);
model.quadcon(2).rhs = 0.0;
model.quadcon(2).name = 'rot_cone';

gurobi_write(model, 'qcp.lp');
result = gurobi(model);

for j = 1:3
    fprintf('%s %e
', names{j}, result.x(j))
end

fprintf('Obj: %e
', result.objval);
end
qp.m

function qp()
% Copyright 2020, Gurobi Optimization, LLC
%
% This example formulates and solves the following simple QP model:
% minimize
%   x^2 + x*y + y^2 + y*z + z^2 + 2 x
% subject to
%   x + 2 y + 3 z >= 4
%   x + y >= 1
%   x, y, z non-negative
%
% It solves it once as a continuous model, and once as an integer model.

names = {'x', 'y', 'z'};
model.varnames = names;
model.Q = sparse([1 0.5 0; 0.5 1 0.5; 0 0.5 1]);
model.A = sparse([1 2 3; 1 1 0]);
model.obj = [2 0 0];
model.rhs = [4 1];
model.sense = '>';   
gurobi_write(model, 'qp.lp');
results = gurobi(model);

for v = 1:length(names)
    fprintf('%s %e
', names{v}, results.x(v));
end

fprintf('Obj : %e
', results.objval);
model.vtype = 'B';
results = gurobi(model);

for v = 1:length(names)
    fprintf('%s %e
', names{v}, results.x(v));
end

fprintf('Obj : %e
', results.objval);
end

sensitivity.m

function sensitivity(filename)
% Copyright 2020, Gurobi Optimization, LLC
%
% A simple sensitivity analysis example which reads a MIP model from a file
% and solves it. Then each binary variable is set to 1-X, where X is its
% value in the optimal solution, and the impact on the objective function
% value is reported.


% Read model
fprintf('Reading model %s\n', filename);

model = gurobi_read(filename);
cols = size(model.A, 2);

ivars = find(model.vtype == 'C');
if length(ivars) <= 0
    fprintf('All variables of the model are continuous, nothing to do\n');
    return;
end

% Optimize
result = gurobi(model);

% Capture solution information
if result.status ~= 'OPTIMAL'
    fprintf('Model status is %d, quit now\n', result.status);
end

origx = result.x;
origobjval = result.objval;

params.OutputFlag = 0;

% Iterate through unfixed binary variables in the model
for j = 1:cols
    if model.vtype(j) == 'B' || model.vtype(j) == 'I'
        continue;
    end
    if model.vtype(j) == 'I'
        if model.lb(j) ~= 0.0 || model.ub(j) ~= 1.0
            continue;
        end
        if model.lb(j) > 0.0  || model.ub(j) < 1.0
            continue;
        end
    else
        if model.lb(j) > 0.0 || model.ub(j) < 1.0
            continue;
        end
    end
end

% Update MIP start for all variables
model.start = origx;

% Set variable to 1-X, where X is its value in optimal solution
if origx(j) < 0.5
    model.start(j) = 1;
    model.lb(j) = 1;
else
    model.start(j) = 0;
    model.ub(j) = 0;
end

% Optimize
result = gurobi(model, params);

% Display result
if ~strcmp(result.status, 'OPTIMAL')
gap = inf;
else
gap = result.objval - origobjval;
end
fprintf('Objective sensitivity for variable %s is %g\n', ...model.varnames{j}, gap);

% Restore original bounds
model.lb(j) = 0;
model.ub(j) = 1;
end

sos.m

function sos()
% Copyright 2020, Gurobi Optimization, LLC
% This example creates a very simple Special Ordered Set (SOS) model. The model consists of 3 continuous variables, no linear constraints, and a pair of SOS constraints of type 1.
model.ub = [1 1 2];
model.obj = [2 1 1];
model.modelsense = 'Max';
model.A = sparse(1,3);
model.rhs = 0;
model.sense = '=';

% Add first SOS: x1 = 0 or x2 = 0
model.sos(1).type = 1;
model.sos(1).index = [1 2];
model.sos(1).weight = [1 2];

% Add second SOS: x1 = 0 or x3 = 0
model.sos(2).type = 1;
model.sos(2).index = [1 3];
model.sos(2).weight = [1 2];

% Write model to file
gurobi_write(model, 'sos.lp');
result = gurobi(model);
for i=1:3
fprintf('x%d %e\n', i, result.x(i))
end
fprintf('Obj: %e\n', result.objval);
end

sudoku.m

function sudoku(filename)
% Copyright 2020, Gurobi Optimization, LLC */
Sudoku example.

The Sudoku board is a 9x9 grid, which is further divided into a 3x3 grid of 3x3 grids. Each cell in the grid must take a value from 0 to 9. No two grid cells in the same row, column, or 3x3 subgrid may take the same value.

In the MIP formulation, binary variables x[i,j,v] indicate whether cell <i,j> takes value 'v'. The constraints are as follows:

1. Each cell must take exactly one value (\(\sum_v x[i,j,v] = 1\))
2. Each value is used exactly once per row (\(\sum_i x[i,j,v] = 1\))
3. Each value is used exactly once per column (\(\sum_j x[i,j,v] = 1\))
4. Each value is used exactly once per 3x3 subgrid (\(\sum_{grid} x[i,j,v] = 1\))

Input datasets for this example can be found in examples/data/sudoku*.

SUBDIM = 3;
DIM = SUBDIM*SUBDIM;

fileID = fopen(filename);
if fileID == -1
  fprintf('Could not read file %s, quit
', filename);
  return;
end

board = repmat(-1, DIM, DIM);
for i = 1: DIM
  s = fgets(fileID, 100);
  if length(s) <= DIM
    fprintf('Error: not enough board positions specified, quit
');
    return;
  end
  for j = 1: DIM
    if s(j) ~= '.
      board(i, j) = str2double(s(j));
      if board(i,j) < 1 || board(i,j) > DIM
        fprintf('Error: Unexpected character in Input line %d, quit
', i);
        return;
      end
    end
  end
end

% Map X(i,j,k) into an index variable in the model
nVars = DIM * DIM * DIM;

% Build model
model.vtype = repmat('B', nVars, 1);
model.lb = zeros(nVars, 1);
model.ub = ones(nVars, 1);
for v = 1:DIM
    var = (i-1)*DIM*DIM + (j-1)*DIM + v;
    model.varnames{var} = sprintf('x[%d,%d,%d]', i, j, v);
end
end

% Create constraints:  
nRows = 4 * DIM * DIM;  
model.A = sparse(nRows, nVars);  
model.rhs = ones(nRows, 1);  
model.sense = repmat('=', nRows, 1);

Row = 1;

% Each cell gets a value */
for i = 1:DIM
    for j = 1:DIM
        for v = 1:DIM
            if board(i,j) == v
                model.lb((i-1)*DIM*DIM + (j-1)*DIM + v) = 1;
            end
            model.A(Row, (i-1)*DIM*DIM + (j-1)*DIM + v) = 1;
        end
        Row = Row + 1;
    end
end

% Each value must appear once in each row
for v = 1:DIM
    for j = 1:DIM
        for i = 1:DIM
            model.A(Row, (i-1)*DIM*DIM + (j-1)*DIM + v) = 1;
        end
        Row = Row + 1;
    end
end

% Each value must appear once in each column
for v = 1:DIM
    for i = 1:DIM
        for j = 1:DIM
            model.A(Row, (i-1)*DIM*DIM + (j-1)*DIM + v) = 1;
        end
        Row = Row + 1;
    end
end

% Each value must appear once in each subgrid
for v = 1:DIM
    for ig = 0: SUBDIM-1
        for jg = 0: SUBDIM-1
            for i = ig*SUBDIM+1:(ig+1)*SUBDIM-1
                for j = jg*SUBDIM+1:(jg+1)*SUBDIM
                    model.A(Row, (i-1)*DIM*DIM + (j-1)*DIM + v) = 1;
                end
            end
        end
    end
end
Row = Row + 1;
end

end

% Save model
gurobi_write (model, 'sudoku_m.lp');

% Optimize model
params.logfile = 'sudoku_m.log';
result = gurobi (model, params);

if strcmp(result.status, 'OPTIMAL')
    fprintf('Solution:
');
    for i = 1:DIM
        for j = 1:DIM
            for v = 1:DIM
                var = (i-1)*DIM*DIM + (j-1)*DIM + v;
                if result.x(var) > 0.99
                    fprintf('%d', v);
                end
            end
        end
        fprintf('
');
    end
else
    fprintf('Problem was infeasible
')
end

workforce1.m

function workforce1()

% Copyright 2020, Gurobi Optimization, LLC
% Assign workers to shifts; each worker may or may not be available on a
% particular day. If the problem cannot be solved, use IIS to find a set of
% conflicting constraints. Note that there may be additional conflicts
% besides what is reported via IIS.

% define data
nShifts = 14;
nWorkers = 7;
nVars = nShifts * nWorkers;
Shifts = {'Mon1'; 'Tue2'; 'Wed3'; 'Thu4'; 'Fri5'; 'Sat6'; 'Sun7';
            'Mon8'; 'Tue9'; 'Wed10'; 'Thu11'; 'Fri12'; 'Sat13'; 'Sun14'};
Workers = {'Amy'; 'Bob'; 'Cathy'; 'Dan'; 'Ed'; 'Fred'; 'Gu'};
pay = [10; 12; 10; 8; 8; 9; 11];
shiftRequirements = [3; 2; 4; 4; 5; 6; 5; 2; 2; 3; 4; 6; 7; 5];
availability = [
    0 1 1 0 1 0 1 1 1 1 1 1 1;]
% Build model
model.modelname = 'workforce1';
model.modesense = 'min';

% Initialize assignment decision variables:
% x[w][s] == 1 if worker w is assigned
to shift s. Since an assignment model always produces integer
solutions, we use continuous variables and solve as an LP.
model.ub = ones(nVars, 1);
model.obj = zeros(nVars, 1);

for w = 1:nWorkers
    for s = 1:nShifts
        model.varnames{s+(w-1)*nShifts} = sprintf('%s.%s', Workers{w}, Shifts{s});
        model.obj(s+(w-1)*nShifts) = pay(w);
        if availability(w, s) == 0
            model.ub(s+(w-1)*nShifts) = 0;
        end
    end
end

% Set-up shift-requirements constraints
model.sense = repmat('=', nShifts, 1);
model.rhs = shiftRequirements;
model.constrnames = Shifts;
model.A = sparse(nShifts, nVars);
for s = 1:nShifts
    for w = 1:nWorkers
        model.A(s, s+(w-1)*nShifts) = 1;
    end
end

% Save model
gurobi_write(model,'workforce1_m.lp');

% Optimize
paramslogfile = 'workforce1_m.log';
result = gurobi(model, params);

% Display results
if strcmp(result.status, 'OPTIMAL')
    fprintf('The optimal objective is %g\n', result.objval);
    fprintf('Schedule:\n');
    for s = 1:nShifts
        fprintf('\%s:\n', Shifts{s});
        for w = 1:nWorkers
            fprintf('	%g\n', model.x{s+(w-1)*nShifts});
        end
    end
end
if result.x(s+(w-1)*nShifts) > 0.9
    fprintf('%s ', Workers{w});
end
end
fprintf('
);
end
else
    if strcmp(result.status, 'INFEASIBLE')
        fprintf('Problem is infeasible.... computing IIS
');
        iis = gurobi_iis(model, params);
        if iis.minimal
            fprintf('IIS is minimal
');
        else
            fprintf('IIS is not minimal
');
        end
        if any(iis.Arows)
            fprintf('Rows in IIS: ');
            disp(strjoin(model.constrnames(iis.Arows)));
        end
        if any(iis.lb)
            fprintf('LB in IIS: ');
            disp(strjoin(model.varnames(iis.lb)));
        end
        if any(iis.ub)
            fprintf('UB in IIS: ');
            disp(strjoin(model.varnames(iis.ub)));
        end
    else
        % Just to handle user interruptions or other problems
        fprintf('Unexpected status %s
', result.status);
    end
end
workforce2.m

function workforce2()

% Copyright 2020, Gurobi Optimization, LLC
%
% Assign workers to shifts; each worker may or may not be available on a
% particular day. If the problem cannot be solved, use IIS iteratively to
% find all conflicting constraints.
%
% define data
nShifts = 14;
nWorkers = 7;
nVars = nShifts * nWorkers;
Shifts = {'Mon1'; 'Tue2'; 'Wed3'; 'Thu4'; 'Fri5'; 'Sat6'; 'Sun7';
    'Mon8'; 'Tue9'; 'Wed10'; 'Thu11'; 'Fri12'; 'Sat13'; 'Sun14'};
Workers = {'Amy'; 'Bob'; 'Cathy'; 'Dan'; 'Ed'; 'Fred'; 'Gu'};
pay = [10; 12; 10; 8; 8; 9; 11];
shiftRequirements = [3; 2; 4; 4; 5; 6; 5; 2; 2; 3; 4; 6; 7; 5];
availability = [
    0 1 1 0 1 0 1 1 1 1 1 1;
    1 1 0 0 1 1 0 1 0 0 1 0;
    0 0 1 1 1 0 1 1 1 1 1 1;
    0 1 1 0 1 1 0 1 1 1 1 1;
    1 1 1 1 1 0 1 1 1 0 1 1;
    1 1 1 0 1 0 1 1 0 0 1 1;
    1 1 1 0 1 1 1 1 1 1 1 1
];

% Build model
model.modelname = 'workforce2';
model.modelsense = 'min';

% Initialize assignment decision variables:
% x[w][s] == 1 if worker w is assigned to shift s. Since an assignment model always produces integer solutions, we use continuous variables and solve as an LP.
model.ub = ones(nVars, 1);
model.obj = zeros(nVars, 1);
for w = 1:nWorkers
    for s = 1:nShifts
        model.varnames{s+(w -1)*nShifts} = sprintf('%s.%s', Workers{w}, Shifts{s});
        model.obj(s+(w -1)*nShifts) = pay(w);
        if availability(w, s) == 0
            model.ub(s+(w -1)*nShifts) = 0;
        end
    end
end

% Set-up shift-requirements constraints
model.sense = repmat('=', nShifts, 1);
model.rhs = shiftRequirements;
model.constrnames = Shifts;
model.A = sparse(nShifts, nVars);
for s = 1:nShifts
    for w = 1:nWorkers
        model.A(s, s+(w -1)*nShifts) = 1;
    end
end

% Save model
gurobi_write(model, 'workforce2_m.lp');

% Optimize
params.logfile = 'workforce2_m.log';
result = gurobi(model, params);

% If infeasible, remove IIS rows until it becomes feasible
numremoved = 0;
if strcmp(result.status, 'INFEASIBLE')
    numremoved = 0;
    while strcmp(result.status, 'INFEASIBLE')
        numremoved = numremoved + 1;
        model.A(model.constrnames{result.IIS(rows(numremoved))}) = 0;
        gurobi_write(model, 'workforce2_m.lp');
        result = gurobi(model, params);
    end
end

566
iis = gurobi_iis(model, params);
keep = find(~iis.Arows);
fprintf('Removing rows: ');
disp(model.constrnames{iis.Arows})
model.A = model.A(keep,:);
model.sense = model.sense(keep,:);
model.rhs = model.rhs(keep,:);
model.constrnames = model.constrnames(keep,:);
numremoved = numremoved + 1;
result = gurobi(model, params);
end

% Display results
if strcmp(result.status,'OPTIMAL')
  if numremoved > 0
    fprintf('It becomes feasible after %d rounds of IIS row removing\n', numremoved);
  end
  printsolution(result, Shifts, Workers)
else
  % Just to handle user interruptions or other problems
  fprintf('Unexpected status %s\n', result.status)
end

function printsolution(result, Shifts, Workers)
% Helper function to display results
nShifts = length(Shifts);
nWorkers = length(Workers);

fprintf('The optimal objective is %g\n', result.objval);
fprintf('Schedule:\n');
for s = 1:nShifts
  fprintf('\t%s:', Shifts(s));
  for w = 1:nWorkers
    if result.x(s+(w-1)*nShifts) > 0.9
      fprintf(' %s ', Workers{w});
    end
  end
  fprintf('\n');
end
end

workforce3.m

function workforce3()

% Copyright 2020, Gurobi Optimization, LLC
%
% Assign workers to shifts; each worker may or may not be available on a
% particular day. If the problem cannot be solved, relax the model
% to determine which constraints cannot be satisfied, and how much
% they need to be relaxed.
%
% define data
nShifts = 14;
nWorkers = 7;
nVars = nShifts * nWorkers;

Shifts = {'Mon1'; 'Tue2'; 'Wed3'; 'Thu4'; 'Fri5'; 'Sat6'; 'Sun7'; ...
'Mon8'; 'Tue9'; 'Wed10'; 'Thu11'; 'Fri12'; 'Sat13'; 'Sun14'};

Workers = {'Amy'; 'Bob'; 'Cathy'; 'Dan'; 'Ed'; 'Fred'; 'Gu'};

pay = [10; 12; 10; 8; 8; 9; 11];

shiftRequirements = [3; 2; 4; 4; 5; 6; 5; 2; 2; 3; 4; 6; 7; 5];

availability = [
    0 1 1 0 1 0 1 1 1 1 1 1 1 1 1 ;
    1 1 0 0 1 1 0 1 0 1 0 1 0 ;
    0 0 1 1 1 0 1 1 1 1 1 1 1 1 ;
    0 1 1 0 1 1 0 1 1 1 1 1 1 1 ;
    1 1 1 1 0 1 1 0 1 0 1 1 ;
    1 1 0 1 0 1 1 0 0 1 1 1 ;
    1 1 1 1 0 1 1 1 1 1 1 1 1 1 ];

% Build model
model.modelname = 'workforce3';
model.modelsense = 'min';

% Initialize assignment decision variables:
% x[w][s] == 1 if worker w is assigned to shift s. Since an assignment model always produces integer solutions, we use continuous variables and solve as an LP.
model.ub = ones(nVars, 1);
model.obj = zeros(nVars, 1);

for w = 1:nWorkers
    for s = 1:nShifts
        model.varnames{s+(w-1)*nShifts} = sprintf('%s.%s', Workers{w}, Shifts{s});
        model.obj(s+(w-1)*nShifts) = pay(w);
        if availability(w, s) == 0
            model.ub(s+(w-1)*nShifts) = 0;
        end
    end
end

% Set-up shift-requirements constraints
model.sense = repmat('=', nShifts, 1);
model.rhs = shiftRequirements;
model.constrnames = Shifts;
model.A = sparse(nShifts, nVars);
for s = 1:nShifts
    for w = 1:nWorkers
        model.A(s, s+(w-1)*nShifts) = 1;
    end
end

% Save model
gurobi_write(model,'workforce3_m.lp');
% Optimize
params.logfile = 'workforce3_m.log';
result = gurobi(model, params);

% Display results
if strcmp(result.status, 'OPTIMAL')
    % The code may enter here if you change some of the data... otherwise
    % this will never be executed.
    printsolution(result, Shifts, Workers)
else
    if strcmp(result.status, 'INFEASIBLE')
        penalties.lb = inf(nVars, 1);
        penalties.ub = inf(nVars, 1);
        penalties.rhs = ones(nShifts, 1);
        feasrelax = gurobi_feasrelax(model, 0, false, penalties, params);
        result = gurobi(feasrelax.model, params);
        if strcmp(result.status, 'OPTIMAL')
            printsolution(result, Shifts, Workers);
            fprintf('Slack value:
');
            for j = nVars+1:length(result.x)
                if result.x(j) > 0.1
                    fprintf('	%s, %g
', feasrelax.model.varnames{j}, result.x(j));
                end
            end
        else
            fprintf('Unexpected status %s
', result.status);
        end
    else
        % Just to handle user interruptions or other problems
        fprintf('Unexpected status %s
', result.status);
    end
end

function printsolution(result, Shifts, Workers)
% Helper function to display results
nShifts = length(Shifts);
nWorkers = length(Workers);
fprintf('The optimal objective is %g
', result.objval);
fprintf('Schedule:
');
for s = 1:nShifts
    fprintf('	%s:', Shifts{s});
    for w = 1:nWorkers
        if result.x(s+(w-1)*nShifts) > 0.9
            fprintf(' %s ', Workers{w});
        end
    end
    fprintf('
');
end
end

workforce4.m

function workforce4()
% Assign workers to shifts; each worker may or may not be available on a
% particular day. We use Pareto optimization to solve the model:
% first, we minimize the linear sum of the slacks. Then, we constrain
% the sum of the slacks, and we minimize a quadratic objective that
% tries to balance the workload among the workers.

% define data
nShifts = 14;
nWorkers = 7;
nVars = (nShifts + 1) * (nWorkers + 1) + nWorkers + 1;
avgShiftIdx = (nShifts+1)*(nWorkers+1);
totalSlackIdx = nVars;
Shifts = {'Mon1'; 'Tue2'; 'Wed3'; 'Thu4'; 'Fri5'; 'Sat6'; 'Sun7';
          'Mon8'; 'Tue9'; 'Wed10'; 'Thu11'; 'Fri12'; 'Sat13'; 'Sun14'};
Workers = {'Amy'; 'Bob'; 'Cathy'; 'Dan'; 'Ed'; 'Fred'; 'Gu'};
shiftRequirements = [3; 2; 4; 4; 5; 6; 5; 2; 2; 3; 4; 6; 7; 5];
availability = [
    0 1 1 0 1 0 1 0 1 1 1 1 1 1;
    1 1 0 0 1 1 0 1 0 0 1 0 1 0;
    0 0 1 1 1 0 1 1 1 1 1 1 1 1;
    0 1 1 0 1 1 0 1 1 1 1 1 1 1;
    1 1 1 1 0 1 1 0 1 0 1 1 1;
    1 1 1 0 0 1 0 1 1 0 0 1 1 1;
    1 1 1 0 1 1 1 1 1 1 1 1 1 1
];

% Build model
model.modelname = 'workforce4';
model.modelsense = 'min';

% Initialize assignment decision variables:
% x[w][s] == 1 if worker w is assigned
% to shift s. Since an assignment model always produces integer
% solutions, we use continuous variables and solve as an LP.
model.vtype = repmat('C', nVars, 1);
model.lb = zeros(nVars, 1);
model.ub = ones(nVars, 1);
model.obj = zeros(nVars, 1);
for w = 1:nWorkers
    for s = 1:nShifts
        model.varnames{s+(w-1)*nShifts} = sprintf('%s.%s', Workers{w}, Shifts{s});
        if availability(w, s) == 0
            model.ub(s+(w-1)*nShifts) = 0;
        end
    end
end

% Initialize shift slack variables
for s = 1:nShifts
% Initialize worker slack and diff variables
for w = 1:nWorkers
    model.varnames{w + nShifts * (nWorkers +1)} = sprintf('TotalShifts_%s', Workers{w});
    model.ub(w + nShifts * (nWorkers +1)) = inf;
    model.varnames(w + avgShiftIdx) = sprintf('DiffShifts_%s', Workers{w});
    model.ub(w + avgShiftIdx) = inf;
    model.lb(w + avgShiftIdx) = -inf;
end

% Initialize average shift variable
model.ub((nShifts +1)*(nWorkers +1)) = inf;
model.varnames((nShifts +1)*(nWorkers +1)) = 'AvgShift';

% Initialize total slack variable
model.ub(totalSlackIdx) = inf;
model.varnames(totalSlackIdx) = 'TotalSlack';
model.obj(totalSlackIdx) = 1;

% Set-up shift-requirements constraints with shift slack
model.sense = repmat('=', nShifts +1+ nWorkers , 1);
model.rhs = [shiftRequirements; zeros(1+nWorkers, 1)];
model.constrnames = Shifts;
model.A = sparse(nShifts +1+nWorkers, nVars);
for s = 1:nShifts
    for w = 1:nWorkers
        model.A(s, s+(w -1)*nShifts) = 1;
        model.A(s, s + nShifts * nWorkers) = 1;
    end
    model.A(s, nShifts +1 , s+ nShifts * nWorkers) = -1;
    model.A(s, (nShifts +1)*nWorkers +1) = 1;
    model.constrnames{nShifts +1} = 'TotalSlack';
end

% Set TotalSlack equal to the sum of each shift slack
for s = 1:nShifts
    model.A(nShifts +1, s+ nShifts * nWorkers) = -1;
end
model.A(nShifts +1, totalSlackIdx) = 1;
model.constrnames{nShifts +1} = 'TotalSlack';

% Set total number of shifts for each worker
for w = 1:nWorkers
    for s = 1:nShifts
        model.A(w + nShifts +1, s+(w -1)*nShifts) = -1;
        model.A(w + nShifts +1, w + nShifts * (nWorkers +1)) = 1;
    end
    model.constrnames{nShifts +1+w} = sprintf('totShifts_%s', Workers{w});
end

% Save model
gurobi_write(model,’workforce4a_m.lp’);

% Optimize
params.logfile = ’workforce4_m.log’;
result = solveandprint(model, params, Shifts, Workers);
if ~strcmp(result.status, 'OPTIMAL')
    fprintf('Quit now\n');
    return;
end

% Constraint the slack by setting its upper and lower bounds
totalSlack = result.x(totalSlackIdx);
model.lb(totalSlackIdx) = totalSlack;
model.ub(totalSlackIdx) = totalSlack;

Rows = nShifts+1+nWorkers;
for w = 1:nWorkers
    model.A(Rows+w, w + nShifts * (nWorkers+1)) = 1;
    model.A(Rows+w, w + avgShiftIdx) = -1;
    model.A(Rows+w, avgShiftIdx) = -1;
    model.A(Rows+w+nWorkers, w + nShifts * (nWorkers+1)) = 1;
    model.rhs(Rows+w) = 0;
    model.sense(Rows+w) = '=';
    model.constrnames{Rows+w} = sprintf('DiffShifts_%s_AvgShift', Workers{w});
end
model.A(Rows+nWorkers, avgShiftIdx) = -nWorkers;
model.rhs(Rows+nWorkers) = 0;
model.sense(Rows+nWorkers) = '=';
model.constrnames{Rows+nWorkers} = 'AvgShift';

% Objective: minimize the sum of the square of the difference from the
% average number of shifts worked
model.obj = zeros(nVars, 1);
model.Q = sparse(nVars, nVars);
for w = 1:nWorkers
    model.Q(avgShiftIdx + w, avgShiftIdx + w) = 1;
end

% model is no longer an assignment problem, enforce binary constraints
% on shift decision variables.
model.vtype(1:(nWorkers * nShifts), 1) = 'B';
model.vtype((nWorkers * nShifts + 1):nVars, 1) = 'C';

% Save modified model
gurobi_write(model,'workforce4b_m.lp');

% Optimize
result = solveandprint(model, params, Shifts, Workers);
if ~strcmp(result.status, 'OPTIMAL')
    fprintf('Not optimal\n');
end

end

function result = solveandprint(model, params, Shifts, Workers)
% Helper function to solve and display results

nShifts = length(Shifts);
NWorkers = length(Workers);
result = gurobi(model, params);
if strcmp(result.status, 'OPTIMAL')
fprintf('The optimal objective is %g\n', result.objval);
fprintf('Schedule:\n');
for s = 1:nShifts
    fprintf('\t%s:', Shifts{s});
    for w = 1:nWorkers
        if result.x(s+(w-1)*nShifts) > 0.9
            fprintf('%s ', Workers{w});
        end
    end
    fprintf('\n');
end
fprintf('Workload:\n');
for w = 1:nWorkers
    fprintf('\t%s: %g\n', Workers{w}, result.x(w + nShifts * (nWorkers+1)));
end
else
    fprintf('Optimization finished with status %s\n', result.status);
end

workforce5.m

function workforce5()

% Copyright 2020, Gurobi Optimization, LLC
%
% Assign workers to shifts; each worker may or may not be available on a
% particular day. We use multi-objective optimization to solve the model.
% The highest-priority objective minimizes the sum of the slacks
% (i.e., the total number of uncovered shifts). The secondary objective
% minimizes the difference between the maximum and minimum number of
% shifts worked among all workers. The second optimization is allowed
% to degrade the first objective by up to the smaller value of 10% and 2
%
% define data
nShifts = 14;
nWorkers = 8;
nVars = (nShifts + 1) * (nWorkers + 1) + 2;
minShiftIdx = (nShifts+1)*(nWorkers+1);
maxShiftIdx = minShiftIdx + 1;
totalSlackIdx = nVars;

Shifts = {'Mon1'; 'Tue2'; 'Wed3'; 'Thu4'; 'Fri5'; 'Sat6'; 'Sun7';
          'Mon8'; 'Tue9'; 'Wed10'; 'Thu11'; 'Fri12'; 'Sat13'; 'Sun14'};
Workers = {'Amy'; 'Bob'; 'Cathy'; 'Dan'; 'Ed'; 'Fred'; 'Gu'; 'Tobi'};
shiftRequirements = [3; 2; 4; 4; 5; 6; 5; 2; 2; 3; 4; 6; 7; 5];
availability = [0 1 1 0 1 0 1 1 1 1 1 1 1 1 1;
                1 1 0 0 1 1 0 1 0 0 1 0 1 0;
                0 0 1 1 1 0 1 1 1 1 1 1 1 1 1;
                0 1 1 0 1 1 0 1 1 1 1 1 1 1 1;
                1 1 1 1 1 0 1 1 1 0 1 0 1 1;
                1 1 1 0 0 1 0 1 1 0 0 1 1 1;
                0 1 1 1 0 1 1 0 1 1 1 0 1 1;...];
% Build model
model.modelname = 'workforce5';
model.modelsense = 'min';

% Initialize assignment decision variables:
% x[w][s] == 1 if worker w is assigned
% to shift s. Since an assignment model always produces integer
% solutions, we use continuous variables and solve as an LP.
model.vtype = repmat('C', nVars, 1);
model.lb = zeros(nVars, 1);
model.ub = ones(nVars, 1);
for w = 1:nWorkers
    for s = 1:nShifts
        model.vtype(s+(w-1)*nShifts) = 'B';
        model.varnames{s+(w-1)*nShifts} = sprintf('%s.%s', Workers{w}, Shifts{s});
        if availability(w, s) == 0
            model.ub(s+(w-1)*nShifts) = 0;
        end
    end
end

% Initialize shift slack variables
for s = 1:nShifts
    model.varnames{s+nShifts*nWorkers} = sprintf('ShiftSlack_%s', Shifts{s});
    model.ub(s+nShifts*nWorkers) = inf;
end

% Initialize worker slack and diff variables
for w = 1:nWorkers
    model.varnames{w + nShifts * (nWorkers+1)} = sprintf('TotalShifts_%s', Workers{w});
    model.ub(w + nShifts * (nWorkers+1)) = inf;
end

% Initialize min/max shift variables
model.ub(minShiftIdx) = inf;
model.varnames{minShiftIdx} = 'MinShift';
model.ub(maxShiftIdx) = inf;
model.varnames{maxShiftIdx} = 'MaxShift';

% Initialize total slack variable
model.ub(totalSlackIdx) = inf;
model.varnames{totalSlackIdx} = 'TotalSlack';

% Set-up shift-requirements constraints with shift slack
model.sense = repmat('=', nShifts+1+nWorkers, 1);
model.rhs = [shiftRequirements; zeros(1+nWorkers, 1)];
model.constrnames = Shifts;
model.A = sparse(nShifts+1+nWorkers, nVars);
for s = 1:nShifts
    for w = 1:n Workers
        model.A(s, s+(w-1)*nShifts) = 1;
    end
end
model.A(s, s + nShifts*nWorkers) = 1;
end

% Set TotalSlack equal to the sum of each shift slack
for s = 1:nShifts
    model.A(nShifts+1, s+nShifts*nWorkers) = -1;
end
model.A(nShifts+1, totalSlackIdx) = 1;
model.constrnames{nShifts+1} = 'TotalSlack';

% Set total number of shifts for each worker
for w = 1:nWorkers
    for s = 1:nShifts
        model.A(w + nShifts+1, s+(w-1)*nShifts) = -1;
    end
    model.A(w + nShifts+1, w + nShifts * (nWorkers+1)) = 1;
    model.constrnames{nShifts+1+w} = sprintf('totShifts_%s', Workers{w});
end

% Set minShift / maxShift general constraints
model.genconmin.resvar = minShiftIdx;
model.genconmin.name = 'MinShift';
model.genconmax.resvar = maxShiftIdx;
model.genconmax.name = 'MaxShift';
for w = 1:nWorkers
    model.genconmin.vars(w) = w + nShifts * (nWorkers+1);
    model.genconmax.vars(w) = w + nShifts * (nWorkers+1);
end

% Set multiobjective
model.multiobj(1).objn = zeros(nVars, 1);
model.multiobj(1).objn(totalSlackIdx) = 1;
model.multiobj(1).priority = 2;
model.multiobj(1).abstol = 2;
model.multiobj(1).reltol = 0.1;
model.multiobj(1).name = 'TotalSlack';
model.multiobj(1).con = 0.0;
model.multiobj(2).objn = zeros(nVars, 1);
model.multiobj(2).objn(minShiftIdx) = -1;
model.multiobj(2).objn(maxShiftIdx) = 1;
model.multiobj(2).priority = 1;
model.multiobj(2).weight = 1;
model.multiobj(2).abstol = 0;
model.multiobj(2).reltol = 0;
model.multiobj(2).name = 'Fairness';
model.multiobj(2).con = 0.0;

% Save initial model
gurobi_write(model,'workforce5_m.lp');

% Optimize
params.logfile = 'workforce5_m.log';
result = solveandprint(model, params, Shifts, Workers);
if ~strcmp(result.status, 'OPTIMAL')
    fprintf('Not optimal
');
function result = solveandprint(model, params, Shifts, Workers)
% Helper function to solve and display results

nShifts = length(Shifts);
nWorkers = length(Workers);
result = gurobi(model, params);
if strcmp(result.status, 'OPTIMAL')
    fprintf('The optimal objective is %g
', result.objval);
    fprintf(' Schedule :
');
    for s = 1:nShifts
        fprintf('%s:', Shifts{s});
        for w = 1:nWorkers
            if result.x(s+(w-1)*nShifts) > 0.9
                fprintf('%s ', Workers{w});
            end
        end
        fprintf('
');
    end
    fprintf(' Workload :
');
    for w = 1:nWorkers
        fprintf('%s: %g
', Workers{w}, result.x(w + nShifts * (nWorkers+1)));
    end
else
    fprintf('Optimization finished with status %s
', result.status);
end
end

3.8 R Examples

This section includes source code for all of the Gurobi R examples. The same source code can be found in the examples/R directory of the Gurobi distribution.

bilinear.R

# Copyright 2020, Gurobi Optimization, LLC
# # This example formulates and solves the following simple bilinear model:
# # maximize
# # x
# # subject to
# # x + y + z <= 10
# # x * y <= 2 (bilinear inequality)
# # x * z + y * z = 1 (bilinear equality)
# # x, y, z non-negative (x integral in second version)

library(gurobi)
library(Matrix)

model <- list()

# Linear constraint matrix
model$A <- matrix(c(1, 1, 1), nrow=1, byrow=T)
model$rhs <- c(10.0)
model$sense <- c('<')

# Variable names
model$varnames <- c('x', 'y', 'z')

# Objective function max 1.0 * x
model$obj <- c(1, 0, 0)
model$modelsense <- 'max'

# Bilinear inequality constraint: x * y <= 2
qc1 <- list()
qc1$Qc <- spMatrix(3, 3, c(1), c(2), c(1.0))
qc1$rhs <- 2.0
qc1$sense <- c('<')
qc1$name <- 'bilinear0'

# Bilinear equality constraint: x * z + y * z == 1
qc2 <- list()
qc2$Qc <- spMatrix(3, 3, c(1, 2), c(3, 3), c(1.0, 1.0))
qc2$rhs <- 1.0
qc2$sense <- c('=')
qc2$name <- 'bilinear1'

model$quadcon <- list(qc1, qc2)

# Solve bilinear model, display solution. The problem is non-convex,
# we need to set the parameter 'NonConvex' in order to solve it.
params <- list()
params$NonConvex <- 2
result <- gurobi(model, params)
print(result$x)

# Constrain 'x' to be integral and solve again
model$vtype <- c('I', 'C', 'C')
result <- gurobi(model, params)
print(result$x)

# Copyright 2020, Gurobi Optimization, LLC
# Solve the classic diet model, showing how to add constraints
# to an existing model.
library(Matrix)
library(gurobi)

# display results
printSolution <- function(model, res, nCategories, nFoods) {
  if (res$status == 'OPTIMAL') {
    cat('
Cost : ',res$objval, '
Buy :
')
    for (j in nCategories + 1: nFoods) {
      if (res$x[j] > 1e-4) {
        cat(format(model$varnames[j], justify='left', width=10),', ')
      }
    }
  }
}

# Example usage
printSolution(model, result, 3, 4)
### Nutrition

```r
for (j in 1:nCategories) {
cat(format(model$varnames[j], justify='left', width=10), ': ',
    format(res$x[j], justify='right', width=10, nsmall=2), '\n')
}

else {
cat('No solution
')
}
```

# Define primitive data

```r
categories <- c('calories', 'protein', 'fat', 'sodium')
nCategories <- length(categories)
minNutrition <- c( 1800, 91, 0, 0 )
maxNutrition <- c( 2200, Inf, 65, 1779 )

foods <- c('hamburger', 'chicken', 'hot dog', 'fries', 'macaroni',
            'pizza', 'salad', 'milk', 'ice cream')
nFoods <- length(foods)
cost <- c(2.49, 2.89, 1.50, 1.89, 2.09, 1.99, 2.49, 0.89, 1.59)
nutritionValues <- c( 410, 24, 26, 730,
                      420, 32, 10, 1190,
                      560, 20, 32, 1800,
                      380, 4, 19, 270,
                      320, 12, 10, 930,
                      320, 15, 12, 820,
                      320, 31, 12, 1230,
                      100, 8, 2.5, 125,
                      330, 8, 10, 180 )
```

# Build model

```r
model <- list()
model$A <- spMatrix(nCategories, nCategories + nFoods,
    i = c(mapply(rep,1:4,1:nFoods)),
    j = c(1, (nCategories+1):(nCategories+nFoods),
            2, (nCategories+1):(nCategories+nFoods),
            3, (nCategories+1):(nCategories+nFoods),
            4, (nCategories+1):(nCategories+nFoods) ),
    x = c(-1.0, nutritionValues[1 + nCategories*0:(nFoods-1)]),
    -1.0, nutritionValues[2 + nCategories*0:(nFoods-1)]),
    -1.0, nutritionValues[3 + nCategories*0:(nFoods-1)]),
    -1.0, nutritionValues[4 + nCategories*0:(nFoods-1)] ))
model$obj <- c(rep(0, nCategories), cost)
model$lb <- c(minNutrition, rep(0, nFoods))
model$ub <- c(maxNutrition, rep(Inf, nFoods))
model$varnames <- c(categories, foods)
model$rhs <- rep(0,nCategories)
model$constrnames <- rep('=',nCategories)
model$sense <- rep('=',nCategories)
model$modelname <- 'diet'
model$modelsense <- 'min'

# Optimize
res <- gurobi(model)
```
printSolution(model, res, nCategories, nFoods)

# Adding constraint: at most 6 servings of dairy
# this is the matrix part of the constraint
B <- spMatrix(1, nCategories + nFoods,
  i = rep(1,2),
  j = (nCategories+c(8,9)),
  x = rep(1,2))

# append B to A
model$A <- rbind(model$A, B)

# extend row-related vectors
model$constrnames <- c(model$constrnames, 'limit_dairy')
model$rhs <- c(model$rhs, 6)
model$sense <- c(model$sense, '<')

# Optimize
res <- gurobi(model)
printSolution(model, res, nCategories, nFoods)

# Clear space
rm(res, model)

---

Copyright 2020, Gurobi Optimization, LLC

Facility location: a company currently ships its product from 5 plants
to 4 warehouses. It is considering closing some plants to reduce
costs. What plant(s) should the company close, in order to minimize
transportation and fixed costs?

Note that this example uses lists instead of dictionaries. Since
it does not work with sparse data, lists are a reasonable option.

Based on an example from Frontline Systems:
http://www.solver.com/disfacility.htm
Used with permission.

library(Matrix)
library(gurobi)

# define primitive data
nPlants <- 5
nWarehouses <- 4

# Warehouse demand in thousands of units
Demand <- c(15, 18, 14, 20)

# Plant capacity in thousands of units
Capacity <- c(20, 22, 17, 19, 18)

# Fixed costs for each plant
FixedCosts <- c(12000, 15000, 17000, 13000, 16000)

# Transportation costs per thousand units
TransCosts <- c(4000, 2000, 3000, 2500, 4500, 2500, 2600, 3400, 3000, 4000, 1200, 1800, 2600, 4100, 3000, 2200, 2600, 3100, 3700, 3200)
flowidx <- function(w, p) { nPlants * (w - 1) + p }

# Build model
model <- list()
model$modelname <- 'facility'
model$modelsense <- 'min'

# initialize data for variables
model$lb <- 0
model$ub <- c(rep(1, nPlants), rep(Inf, nPlants * nWarehouses))
model$vtype <- c(rep('B', nPlants), rep('C', nPlants * nWarehouses))
model$obj <- c(FixedCosts, TransCosts)
model$varnames <- c(paste0(rep('Open', nPlants), 1:nPlants),
              sprintf('Trans%d,%d',
              c(mapply(rep, 1:nWarehouses, nPlants)),
              1:nPlants))

# build production constraint matrix
A1 <- spMatrix(nPlants, nPlants, i = c(1:nPlants), j = (1:nPlants), x = -Capacity)
A2 <- spMatrix(nPlants, nPlants * nWarehouses,
              i = c(mapply(rep, 1:nPlants, nWarehouses)),
              j = mapply(flowidx, 1:nWarehouses, c(mapply(rep, 1:nPlants, nWarehouses)) ),
              x = rep(1, nWarehouses * nPlants))
A3 <- spMatrix(nWarehouses, nPlants)
A4 <- spMatrix(nWarehouses, nPlants * nWarehouses,
              i = c(mapply(rep, 1:nWarehouses, nPlants)),
              j = mapply(flowidx, c(mapply(rep, 1:nWarehouses, nPlants)), 1:nPlants),
              x = rep(1, nPlants * nWarehouses))
model$A <- rbind(cbind(A1, A2), cbind(A3, A4))
model$rhs <- c(rep(0, nPlants), Demand)
model$sense <- c(rep('<', nPlants), rep('=', nWarehouses))
model$constrnames <- c(sprintf('Capacity%d', 1:nPlants),
              sprintf('Demand%d', 1:nWarehouses))

# Save model
gurobi_write(model, 'facilityR.lp')

# Guess at the starting point: close the plant with the highest fixed
# costs; open all others first open all plants
model$start <- c(rep(1, nPlants), rep(NA, nPlants * nWarehouses))

# find most expensive plant, and close it in mipstart
cat('Initial guess:
')
worstidx <- which.max(FixedCosts)
model$start[worstidx] <- 0
cat('Closing plant', worstidx, '
')

# set parameters
params <- list()
params$method <- 2

# Optimize
res <- gurobi(model, params)

# Print solution
if (res$status == 'OPTIMAL') {
Total Costs:

Facilities:

Flows:

No solution

# Clear space
rm(res, model, params, A1, A2, A3, A4)

# Copyright 2020, Gurobi Optimization, LLC
#
# This example reads a MIP model from a file, adds artificial
# variables to each constraint, and then minimizes the sum of the
# artificial variables. A solution with objective zero corresponds
# to a feasible solution to the input model.
# We can also use FeasRelax feature to do it. In this example, we
# use minrelax=1, i.e. optimizing the returned model finds a solution
# that minimizes the original objective, but only from among those
# solutions that minimize the sum of the artificial variables.

library(Matrix)
library(gurobi)

args <- commandArgs(trailingOnly = TRUE)
if (length(args) < 1) {
  stop('Usage: Rscript feasopt.R filename
')
}

# Set-up environment
env <- list()
env$logfile <- 'feasopt.log'

# Read model
cat('Reading model', args[1], '...
')
model <- gurobi_read(args[1], env)
cat('... done
')

# Create penalties
penalties <- list()
penalties$lb <- Inf
penalties$ub <- Inf
penalties$rhs <- rep(1, length(model$rhs))

result <- gurobi_feasrelax(model, 0, TRUE, penalties, env = env)

# Display results
if (result$feasobj > 1e-6) {
  cat('Model', args[1], 'is infeasible within variable bounds
')
} else {
  cat('Model', args[1], 'is feasible
')
# Clear space
rm(env, model, penalties, result)

fixanddive.R

# Copyright 2020, Gurobi Optimization, LLC
#
# Implement a simple MIP heuristic. Relax the model,
# sort variables based on fractionality, and fix the 25% of
# the fractional variables that are closest to integer variables.
# Repeat until either the relaxation is integer feasible or
# linearly infeasible.

library(Matrix)
library(gurobi)

args <- commandArgs(trailingOnly = TRUE)
if (length(args) < 1) {
  stop('Usage: Rscript fixanddive.R filename\n')
}

# Read model
cat('Reading model', args[1], '...')
model <- gurobi_read(args[1])
cat('... done')

# Detect set of non-continuous variables
numvars <- ncol(model$A)
intvars <- which(model$vtype != 'C')
numintvars <- length(intvars)
if (numintvars < 1) {
  stop('All model\’s variables are continuous, nothing to do')
}

# create lb and ub if they do not exists, and set them to default values
if (!('lb' %in% model)) {
  model$lb <- numeric(numvars)
}
if (!('ub' %in% model)) {
  model$ub <- Inf + numeric(numvars)
}

# set all variables to continuous
ovtype <- model$vtype
model$vtype[1:numvars] <- 'C'

# parameters
params <- list()
params$OutputFlag <- 0
result <- gurobi(model, params)

# Perform multiple iterations. In each iteration, identify the first
# quartile of integer variables that are closest to an integer value
# in the relaxation, fix them to the nearest integer, and repeat.
for (iter in 1:1000) {
    # See if status is optimal
    if (result$status != 'OPTIMAL') {
        cat('Model status is', result$status, '\n')
        stop('Can not keep fixing variables\n')
    }
    # collect fractionality of integer variables
    fractional <- abs(result$x - floor(result$x + 0.5))
    fractional <- replace(fractional, fractional < 1e-5, 1)
    fractional <- replace(fractional, ovtype == 'C', 1)
    fractional <- replace(fractional, ovtype == 'S', 1)
    nfractional <- length(which(fractional < 0.51))
    cat('Iteration:', iter, 'Obj:', result$objval, '
', 'Fractional:', nfractional, '\n')
    if (nfractional == 0) {
        cat('Found feasible solution - objective', result$objval, '\n')
        break
    }
    # order the set of fractional index
    select <- order(fractional, na.last = TRUE, decreasing = FALSE)
    # fix 25% of variables
    nfix <- as.integer(ceiling(nfractional / 4))
    # cat('Will fix', nfix, 'variables, out of', numvars, '\n')
    if (nfix < 10)
        cat('Fixing ')
    else
        cat('Fixing ', nfix, 'variables, fractionality threshold:', fractional[select[nfix]], '\n')
    for (k in 1:nfix) {
        j <- select[k]
        val <- floor(result$x[j] + 0.5)
        model$lb[j] <- val
        model$ub[j] <- val
        if (nfix < 10)
            cat(model$varname[j], 'x*=', result$x[j], 'to', val, ' ')
    }
    if (nfix < 10)
        cat('\n')
    # reoptimize
    result <- gurobi(model, params)
}
# Clear space
rm(model, params, result)

gc_pwl.R

# Copyright 2020, Gurobi Optimization, LLC
#
# This example formulates and solves the following simple model
# with PWL constraints:
#
# maximize
# \[ \text{sum} \ c(j) \cdot x(j) \]
# subject to
# \[ \text{sum} \ A(i,j) \cdot x(j) \leq 0, \ \text{for} \ i = 1, \ldots, m \]
# \[ \text{sum} \ y(j) \leq 3 \]
# \[ y(j) = \text{pwl}(x(j)), \ \text{for} \ j = 1, \ldots, n \]
# \[ x(j) \text{ free}, \ y(j) \geq 0, \ \text{for} \ j = 1, \ldots, n \]
# where \( \text{pwl}(x) = 0, \ \text{if} \ x = 0 \)
# \[ = 1+|x|, \ \text{if} \ x \neq 0 \]
#
# Note
# 1. \( \text{sum} \ \text{pwl}(x(j)) \leq b \) is to bound \( x \) vector and also to favor sparse \( x \) vector.
# Here \( b = 3 \) means that at most two \( x(j) \) can be nonzero and if two, then
# \( \text{sum} \ x(j) \leq 1 \)
# 2. \( \text{pwl}(x) \) jumps from 1 to 0 and from 0 to 1, if \( x \) moves from negative 0 to 0,
# then to positive 0, so we need three points at \( x = 0 \). \( x \) has infinite bounds
# on both sides, the piece defined with two points ((-1, 2) and (0, 1)) can
# extend \( x \) to -\( \infty \). Overall we can use five points (-1, 2), (0, 1),
# (0, 0), (0, 1) and (1, 2) to define \( y = \text{pwl}(x) \)

library(gurobi)
library(Matrix)

n = 5

# A x <= 0
A <- rbind(c(0, 0, 0, 1, -1),
           c(0, 0, 1, 1, -1),
           c(1, 1, 0, 0, -1),
           c(1, 0, 1, 0, -1),
           c(1, 0, 0, 1, -1))

# sum y(j) <= 3
y <- rbind(c(1, 1, 1, 1, 1))

# Initialize model
model <- list()

# Constraint matrix
model$A <- bdiag(A, y)

# Right-hand-side coefficient vector
model$rhs <- c(rep(0, n), 3)

# Objective function (x coefficients arbitrarily chosen)
model$obj <- c(0.5, 0.8, 0.5, 0.1, -1, rep(0, n))

# It’s a maximization model
model$modelsense <- "max"

# Lower bounds for x and y
model$lb <- c(rep(-Inf, n), rep(0, n))

# PWL constraints
model$genconpwl <- list()
for (k in 1:n) {
    model$genconpwl[[k]] <- list()
    model$genconpwl[[k]]$xvar <- k
    model$genconpwl[[k]]$yvar <- n + k
    model$genconpwl[[k]]$xpts <- c(-1, 0, 0, 0, 1)
    model$genconpwl[[k]]$ypts <- c(2, 1, 0, 1, 2)
}

# Solve the model and collect the results
result <- gurobi(model)

# Display solution values for x
for (k in 1:n)
    print(sprintf('x(%d) = %g', k, result$x[k]))

print(sprintf('Objective value: %g', result$objval))

# Clear space
rm(model, result)

gc_pwl_func.R

# Copyright 2020, Gurobi Optimization, LLC
#
# This example considers the following nonconvex nonlinear problem
#
# maximize 2 x + y
# subject to exp (x) + 4 sqrt(y) <= 9
#          x, y >= 0
#
# We show you two approaches to solve this:
#
# 1) Use a piecewise-linear approach to handle general function
#    constraints (such as exp and sqrt).
#    a) Add two variables
#       u = exp(x)
#       v = sqrt(y)
#    b) Compute points (x, u) of u = exp(x) for some step length (e.g., x
#       = 0, 1e-3, 2e-3, ..., xmax) and points (y, v) of v = sqrt(y) for
#       some step length (e.g., y = 0, 1e-3, 2e-3, ..., ymax). We need to
#       compute xmax and ymax (which is easy for this example, but this
#       does not hold in general).
#    c) Use the points to add two general constraints of type
#       piecewise-linear.
#
# 2) Use the Gurobi's built-in general function constraints directly (EXP
#    and POW). Here, we do not need to compute the points and the maximal
#    possible values, which will be done internally by Gurobi. In this
#    approach, we show how to "zoom in" on the optimal solution and
#    tighten tolerances to improve the solution quality.

library(gurobi)

printsol <- function(model, result) {
    print(sprintf('%s = %g, %s = %g',
                  model$varnames[1], result$x[1],}
```r
model$varnames[3], result$x[3])
print(sprintf('%s = %g, %s = %g',
    model$varnames[2], result$x[2],
    model$varnames[4], result$x[4]))
print(sprintf('Obj = %g', + result$objval))

# Calculate violation of exp(x) + 4 sqrt(y) <= 9
vio <- exp(result$x[1]) + 4 * sqrt(result$x[2]) - 9
if (vio < 0.0)
    vio <- 0.0
print(sprintf('Vio = %g', vio))
}

model <- list()

# Four nonneg. variables x, y, u, v, one linear constraint u + 4*v <= 9
model$varnames <- c('x', 'y', 'u', 'v')
model$lb    <- c(rep(0, 4))
model$ub    <- c(rep(Inf, 4))
model$A     <- matrix(c(0, 0, 1, 4), nrow = 1)
model$rhs   <- 9

# Objective
model$modelsense <- 'max'
model$obj    <- c(2, 1, 0, 0)

# First approach: PWL constraints
model$genconpwl[1] <- list()
intv <- 1e-3

# Approximate u \approx exp(x), equispaced points in [0, xmax], xmax = log(9)
model$genconpwl[1]$xvar <- 1L
model$genconpwl[1]$yvar <- 3L
xmax <- log(9)
point <- 0
for (t <- 0:1)
    while (t < xmax + intv) {
        point <- point + 1
        model$genconpwl[1]$xpts[point] <- t
        model$genconpwl[1]$ypts[point] <- exp(t)
    }

# Approximate v \approx sqrt(y), equispaced points in [0, ymax], ymax = (9/4)^2
model$genconpwl[2] <- list()
model$genconpwl[2]$xvar <- 2L
model$genconpwl[2]$yvar <- 4L
ymax <- (9/4)^2
point <- 0
for (t <- 0:1)
    while (t < ymax + intv) {
        point <- point + 1
```

586
model$genconpwl[[2]]$xpts[point] <- t
model$genconpwl[[2]]$ypts[point] <- sqrt(t)
t <- t + intv

# Solve and print solution
result = gurobi(model)
printsol(model, result)

# Second approach: General function constraint approach with auto PWL translation by Gurobi
# Delete explicit PWL approximations from model
model$genconpwl <- NULL

# Set u \approx \exp(x)
model$genconexp <- list()
model$genconexp[[1]] <- list()
model$genconexp[[1]]$xvar <- 1L
model$genconexp[[1]]$yvar <- 3L
model$genconexp[[1]]$name <- 'gcf1'

# Set v \approx \sqrt(y) = y^{0.5}
model$genconpow <- list()
model$genconpow[[1]] <- list()
model$genconpow[[1]]$xvar <- 2L
model$genconpow[[1]]$yvar <- 4L
model$genconpow[[1]]$a <- 0.5
model$genconpow[[1]]$name <- 'gcf2'

# Parameters for discretization: use equal piece length with length = 1e-3
params <- list()
params$FuncPieces <- 1
params$FuncPieceLength <- 1e-3

# Solve and print solution
result = gurobi(model, params)
printsol(model, result)

# Zoom in, use optimal solution to reduce the ranges and use a smaller # pclen=1-5 to resolve
model$lb[1] <- max(model$lb[1], result$x[1] - 0.01)
model$ub[1] <- min(model$ub[1], result$x[1] + 0.01)
model$lb[2] <- max(model$lb[2], result$x[2] - 0.01)
model$ub[2] <- min(model$ub[2], result$x[2] + 0.01)
params$FuncPieceLength <- 1e-5

# Solve and print solution
result = gurobi(model, params)
printsol(model, result)

# Clear space
rm(model, result)

genconstr.R
In this example we show the use of general constraints for modeling some common expressions. We use as an example a SAT-problem where we want to see if it is possible to satisfy at least four (or all) clauses of the logical for

\[ L = (x_0 \lor \neg x_1 \lor x_2) \land (x_1 \lor \neg x_2 \lor x_3) \land (x_2 \lor \neg x_3 \lor x_0) \land (x_3 \lor \neg x_0 \lor x_1) \land (\neg x_0 \lor \neg x_1 \lor x_2) \land (\neg x_1 \lor \neg x_2 \lor x_3) \land (\neg x_2 \lor \neg x_3 \lor x_0) \land (\neg x_3 \lor \neg x_0 \lor x_1) \]

We do this by introducing two variables for each literal (itself and its negated value), a variable for each clause, and then two variables for indicating if we can satisfy four, and another to identify the minimum of the clauses (so if it one, we can satisfy all clauses) and put these two variables in the objective.

i.e. the Objective function will be

\[ \text{maximize } Obj_0 + Obj_1 \]

\[ \text{Obj}_0 = \text{MIN} (\text{Clause}_1, \ldots, \text{Clause}_8) \]

\[ \text{Obj}_1 = \text{i} \rightarrow \text{Clause}_1 + \ldots + \text{Clause}_8 \geq 4 \]

thus, the objective value will be two if and only if we can satisfy all clauses; one if and only if at least four clauses can be satisfied, and zero otherwise.

library(Matrix)
library(gurobi)

# Set-up environment
env <- list()
env$logfile <- 'genconstr.log'

# define primitive data
nLiterals <- 4
nClauses <- 8
nObj <- 2
nVars <- 2 * nLiterals + nClauses + nObj
Literal <- function(n) {n}
NotLit <- function(n) {n + nLiterals}
Clause <- function(n) {2 * nLiterals + n}
Obj <- function(n) {2 * nLiterals + nClauses + n}

Clauses <- list(c(Literal(1), NotLit(2), Literal(3)),
c(Literal(2), NotLit(3), Literal(4)),
c(Literal(3), NotLit(4), Literal(1)),
c(Literal(4), NotLit(1), Literal(2)),
c(NotLit(1), NotLit(2), Literal(3)),
c(NotLit(2), NotLit(3), Literal(4)),
c(NotLit(3), NotLit(4), Literal(1)),
c(NotLit(4), NotLit(1), Literal(2)))
# Create model
model <- list
model$ modeloame <- 'genconstr'
model$ modelsenset <- 'max'

# Create objective function, variable names, and variable types
model$ vtype <- rep('B', nVars)
model$ ub <- rep(1, nVars)
model$ varnames <- c(sprintf('X%d', seq(1, nLiterals)),
  sprintf('notX%d', seq(1, nLiterals)),
  sprintf('Clause%d', seq(1, nClauses)),
  sprintf('Obj%d', seq(1, nObj)))
model$ obj <- c(rep(0, 2*nLiterals + nClauses), rep(1, nObj))

# Create linear constraints linking literals and not literals
model$ A <- spMatrix(nLiterals, nVars,
  i = c(seq(1, nLiterals),
    seq(1, nLiterals)),
  j = c(seq(1, nLiterals),
    seq(1+nLiterals, 2*nLiterals)),
  x = rep(1, 2*nLiterals))
model$ constrnames <- c(sprintf('CNSTR_X%d', seq(1, nLiterals)))
model$ rhs <- rep(1, nLiterals)
model$ sense <- rep('=', nLiterals)

# Create OR general constraints linking clauses and literals
model$ genconor <- lapply(1:nClauses,
  function(i) list(resvar = Clause(i),
    var = Clauses[[i]],
    name = sprintf('CNSTR_Clause%d', i))
)

# Set-up Obj1 = Min(Clause[1:nClauses])
model$ genconmin <- list(resvar = Obj(1),
  vars = c(seq(Clause(1), Clause(nClauses))),
  name = 'CNSTR_Obj1')

# the indicator constraint excepts the following vector types for the
# linear sum:
#
# - a dense vector, for example
#  c(rep(0, 2*nLiterals), rep(1, nClauses), rep(0, nObj))
# - a sparse vector, for example
#  sparseVector(x = rep(1, nClauses), i = (2*nLiterals+1):(2*nLiterals+nClauses), length=nVars)
# - In case all coefficients are the same, you can pass a vector of length
#  one with the corresponding value which gets expanded to a dense vector
#  with all values being the given one
#
# We use a sparse vector in this example
a <- sparseVector(x = rep(1, nClauses), i = (2*nLiterals+1):(2*nLiterals+nClauses), length=nVars)

# Set-up Obj2 to signal if any clause is satisfied, i.e.
# we use an indicator constraint of the form:
# (Obj2 = 1) -> sum(Clause[1:nClauses]) >= 4
model$ genconind <- list(resvar = Obj(2),
  binvar = Obj(2),
  binval = 1,
a = a,
sense = '>',
rhs = 4,
name = 'CNSTR_Obj2'))

# Save model

gurobi_write(model, 'genconstr.lp', env)

# Optimize
result <- gurobi(model, env = env)

# Check optimization status
if (result$status == 'OPTIMAL') {
  if (result$objval > 1.9) {
    cat('Logical expression is satisfiable\n')
  } else if(result$objval > 0.9) {
    cat('At least four clauses are satisfiable\n')
  } else {
    cat('At most three clauses may be satisfiable\n')
  }
} else {
  cat('Optimization failed\n')
}

# Clear space
rm(model, result, env, Clauses)

lp.R

# Copyright 2020, Gurobi Optimization, LLC
#
# This example formulates and solves the following simple LP model:
# maximize
# x + 2 y + 3 z
# subject to
# x + y <= 1
# y + z <= 1

library(Matrix)
library(gurobi)

model <- list()

model$A <- matrix(c(1,1,0,0,1,1), nrow=2, byrow=T)
model$obj <- c(1,2,3)
model$modelsense <- 'max'
model$rhs <- c(1,1)
model$sense <- c('<', '<')

result <- gurobi(model)

print(result$objval)
print(result$x)

# Second option for A - as a sparseMatrix (using the Matrix package)...

590
model$A <- spMatrix(2, 3, c(1, 1, 2, 2), c(1, 2, 2, 3), c(1, 1, 1, 1))

params <- list(Method=2, TimeLimit=100)
result <- gurobi(model, params)

print(result$objval)
print(result$x)

# Third option for A - as a sparse triplet matrix (using the slam package)...
model$A <- simple_triplet_matrix(c(1, 1, 2, 2), c(1, 2, 2, 3), c(1, 1, 1, 1))
params <- list(Method=2, TimeLimit=100)
result <- gurobi(model, params)

print(result$objval)
print(result$x)

# Clear space
rm(result, params, model)

# Copyright 2020, Gurobi Optimization, LLC
# Formulate a simple linear program, solve it, and then solve it again using the optimal basis.
library(gurobi)

model <- list()

model$A <- matrix(c(1,3,4,8,2,3), nrow=2, byrow=T)
model$obj <- c(1,2,3)
model$rhs <- c(4,7)
model$sense <- c(‘>’, ‘>’)

# First solve - requires a few simplex iterations
result <- gurobi(model)

model$vbasis <- result$vbasis
model$cbasis <- result$cbasis

# Second solve - start from optimal basis, so no iterations
result <- gurobi(model)

# Clear space
rm(model, result)

# Copyright 2020, Gurobi Optimization, LLC
# Solve a model with different values of the Method parameter; # show which value gives the shortest solve time.

library(Matrix)
library(gurobi)

args <- commandArgs(trailingOnly = TRUE)
if (length(args) < 1) {
  stop('Usage: Rscript lpmod.R filename\n')
}

# Read model
cat('Reading model', args[1], '...')
model <- gurobi_read(args[1])
cat('... done\n')

# Solve the model with different values of Method
params <- list()
bestTime <- Inf
bestMethod <- -1
for (i in 0:4) {
  params$method <- i
  res <- gurobi(model, params)
  if (res$status == 'OPTIMAL') {
    bestMethod <- i
    bestTime <- res$runtime
    params$TimeLimit <- bestTime
  }
}

# Report which method was fastest
if (bestMethod == -1) {
  cat('Unable to solve this model\n')
} else {
  cat('Solved in ', bestTime, ' seconds with Method:', bestMethod, '\n')
}

rm(params, model)

lpmod.R

# Copyright 2020, Gurobi Optimization, LLC
#
# This example reads an LP model from a file and solves it.
# If the model can be solved, then it finds the smallest positive variable,
# sets its upper bound to zero, and resulfolves the model two ways:
# first with an advanced start, then without an advanced start
# (i.e. 'from scratch').

library(Matrix)
library(gurobi)

args <- commandArgs(trailingOnly = TRUE)
if (length(args) < 1) {
  stop('Usage: Rscript lpmod.R filename\n')
}
# Read model
cat('Reading model, args[1], ...
')
model <- gurobi_read(args[1])
cat('... done
')

# Determine whether it is an LP
if ('multiobj' %in% names(model) ||
    'sos' %in% names(model) ||
    'pvlobj' %in% names(model) ||
    'cones' %in% names(model) ||
    'quadcon' %in% names(model) ||
    'genconstr' %in% names(model)) {
  stop('The model is not a linear program
')
}

# Detect set of non-continuous variables
intvars <- which(model$vtype != 'C')
umintvars <- length(intvars)
if (numintvars > 0) {
  stop('problem is a MIP, nothing to do
')
}

# Optimize
result <- gurobi(model)
if (result$status != 'OPTIMAL') {
  cat('This model cannot be solved because its optimization status is',
      result$status, '
')
  stop('Stop now
')
}

# Recover number of variables in model
numvars <- ncol(model$A)

# Ensure bounds array is initialized
if (is.null(model$lb)) {
  model$lb <- rep(0, numvars)
}
if (is.null(model$ub)) {
  model$ub <- rep(Inf, numvars)
}

# Find smallest (non-zero) variable value with zero lower bound
x <- replace(result$x, result$x < 1e-4, Inf)
x <- replace(x, model$lb > 1e-6, Inf)
minVar <- which.min(x)
minVal <- x[minVar]

# Get variable name
varname <- ''
if (is.null(model$varnames)) {
  varname <- sprintf('C%d', minVar)
} else {
  varname <- model$varnames[minVar]
}
cat('\n*** Setting', varname, 'from', minVal, 'to zero ***\n\n')
model$ub[minVar] <- 0

# Set advance start basis information
model$vbasis <- result$vbasis
model$cbasis <- result$cbasis

result2 <- gurobi(model)
warmCount <- result2$itercount
warmTime <- result2$runtime

# Reset - advance start information
model$vbasis <- NULL
model$cbasis <- NULL

result2 <- gurobi(model)
coldCount <- result2$itercount
coldTime <- result2$runtime

cat('\n*** Warm start:', warmCount, 'iterations,', warmTime, 'seconds\n')
cat('\n*** Cold start:', coldCount, 'iterations,', coldTime, 'seconds\n')

# Clear space
rm(model, result, result2)

mip.R

# Copyright 2020, Gurobi Optimization, LLC
#
# This example formulates and solves the following simple MIP model:
# maximize
# x + y + 2 z
# subject to
# x + 2 y + 3 z <= 4
# x + y >= 1
# x, y, z binary

library(gurobi)

model <- list()

model$A <- matrix(c(1,2,3,1,1,0), nrow=2, ncol=3, byrow=T)
model$obj <- c(1,1,2)
model$modelsense <- 'max'
model$rhs <- c(4,1)
model$sense <- c('<', '>')
model$vtype <- 'B'

params <- list(OutputFlag=0)

result <- gurobi(model, params)

print('Solution:')
print(result$objval)
print(result$x)
# Clear space
rm(model, result, params)

mip2.R

# Copyright 2020, Gurobi Optimization, LLC
#
# This example reads a MIP model from a file, solves it and
# prints the objective values from all feasible solutions
# generated while solving the MIP. Then it creates the fixed
# model and solves that model.

library(Matrix)
library(gurobi)

args <- commandArgs(trailingOnly = TRUE)
if (length(args) < 1) {
  stop('Usage: Rscript mip2.R filename\n')
}

# Read model
cat(' Reading model ', args[1], '... ')
model <- gurobi_read(args[1])
cat('... done\n')

# Detect set of non-continuous variables
numvars <- dim(model$A)[2]
intvars <- which(model$vtype != 'C')
umintvars <- length(intvars)
if (numintvars < 1) {
  stop('All model\'s variables are continuous, nothing to do\n')
}

# Optimize
params <- list()
params$poolsolutions <- 20
result <- gurobi(model, params)

# Capture solution information
if (result$status != 'OPTIMAL') {
  cat('Optimization finished with status', result$status, '\n')
  stop('Stop now\n')
}

# Iterate over the solutions
if ('pool' %in% names(result)) {
  solcount <- length(result$pool)
  for (k in 1:solcount) {
    cat('Solution', k, 'has objective:', result$pool[[k]]$objval, '\n')
  }
} else {
  solcount <- 1
  cat('Solution 1 has objective:', result$objval, '\n')
}
# Convert to fixed model
for (j in 1:numvars) {
  if (model$vtype[j] != 'C') {
    t <- floor(result$x[j]+0.5)
    model$lb[j] <- t
    model$ub[j] <- t
  }
}

# Solve the fixed model
result2 <- gurobi(model, params)
if (result2$status != 'OPTIMAL') {
  stop('Error: fixed model isn\'t optimal\n')
}
if (abs(result$objval - result2$objval) > 1e-6 * (1 + abs(result$objval))) {
  stop('Error: Objective values differ\n')
}

# Print values of non-zero variables
for (j in 1:numvars) {
  if (abs(result2$x[j]) < 1e-6) next
  varnames <- ''
  if ('varnames' %in% names(model)) {
    varnames <- model$varnames[j]
  } else {
    varnames <- sprintf('X%d', j)
  }
  cat(format(varnames, justify='left', width=10), ':',
       format(result2$x[j], justify='right', digits=2, width=10), ', \n')
}

# Clear space
rm(model, params, result, result2)

multiobj.R

# Copyright 2020, Gurobi Optimization, LLC
#
# Want to cover three different sets but subject to a common budget of
# elements allowed to be used. However, the sets have different priorities to
# be covered; and we tackle this by using multi-objective optimization.

library(Matrix)
library(gurobi)

# define primitive data
groundSetSize <- 20
nSubSets <- 4
Budget <- 12
Set <- list(
c( 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0 ),
c( 0, 0, 0, 0, 0, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1 ),
c( 0, 0, 0, 1, 1, 0, 1, 1, 0, 0, 0, 0, 0, 0, 1, 1, 0, 1, 0 ),
c( 0, 0, 0, 1, 1, 0, 0, 0, 0, 1, 1, 0, 0, 0, 1, 1, 0, 0, 0 )
)
SetObjPriority <- c(3, 2, 2, 1)
SetObjWeight <- c(1.0, 0.25, 1.25, 1.0)

# Initialize model
model <- list()
model$modelsense <- 'max'
model$modelname <- 'multiobj'

# Set variables, all of them are binary, with 0,1 bounds.
model$vtype <- 'B'
model$lb <- 0
model$ub <- 1
model$varnames <- paste(rep('El', groundSetSize), 1:groundSetSize, sep='')

# Build constraint matrix
model$A <- spMatrix(1, groundSetSize, 
i = rep(1,groundSetSize),
j = 1:groundSetSize,
x = rep(1,groundSetSize))
model$rhs <- c(Budget)
model$sense <- c('<')
model$constrnames <- c('Budget')

# Set multi-objectives
model$multiobj <- list()
for (m in 1:nSubSets) {
  model$multiobj[[m]] <- list()
  model$multiobj[[m]]$objn <- Set[[m]]
  model$multiobj[[m]]$priority <- SetObjPriority[m]
  model$multiobj[[m]]$weight <- SetObjWeight[m]
  model$multiobj[[m]]$abstol <- m
  model$multiobj[[m]]$reltol <- 0.01
  model$multiobj[[m]]$name <- sprintf('Set%d', m)
  model$multiobj[[m]]$con <- 0.0
}

# Save model
gurobi_write(model,'multiobj_R.lp')

# Set parameters
params <- list()
params$PoolSolutions <- 100

# Optimize
result <- gurobi(model, params)

# Capture solution information
if (result$status != 'OPTIMAL') {
  cat('Optimization finished with status', result$status, '
')
  stop('Stop now
')
}

# Print best solution
for (e in 1:groundSetSize) {
  if(result$x[e] < 0.9) next
  cat(' El',e,sep='')
}
# Iterate over the best 10 solutions
if ('pool' %in% names(result)) {
  solcount <- length(result$pool)
  cat('Number of solutions found:', solcount, '\n'
  if (solcount > 10) {
    solcount <- 10
  }
  cat('Objective values for first', solcount, 'solutions:\n'
  for (k in 1:solcount) {
    cat('Solution', k, 'has objective:', result$pool[[k]]$objval[1], '\n'
  }
  } else {
    solcount <- 1
    cat('Number of solutions found:', solcount, '\n'
    cat('Solution 1 has objective:', result$objval, '\n'
  }

# Clean up
rm(model, params, result)

params.R

# Copyright 2020, Gurobi Optimization, LLC
#
# Use parameters that are associated with a model.
#
# A MIP is solved for a few seconds with different sets of parameters.
# The one with the smallest MIP gap is selected, and the optimization
# is resumed until the optimal solution is found.

library(Matrix)
library(gurobi)

args <- commandArgs(trailingOnly = TRUE)
if (length(args) < 1) {
  stop('Usage: Rscript params.R filename
')
}

# Read model
cat('Reading model', args[1], '...')
model <- gurobi_read(args[1])
cat('... done\n')

# Detect set of non-continuous variables
intvars <- which(model$vtype != 'C')
umintvars <- length(intvars)
if (numintvars < 1) {
  stop('All model\'s variables are continuous, nothing to do\n')
}

# Set a 2 second time limit
params <- list()
params$TimeLimit <- 2
# Now solve the model with different values of MIPFocus
params$MIPFocus <- 0
result <- gurobi(model, params)
bestgap <- result$mipgap
bestparams <- params
for (i in 1:3) {
  params$MIPFocus <- i
  result <- gurobi(model, params)
  if (result$mipgap < bestgap) {
    bestparams <- params
    bestgap <- result$mipgap
  }
}
# Finally, reset the time limit and Re-solve model to optimality
bestparams$TimeLimit <- Inf
result <- gurobi(model, bestparams)
cat('Solved with MIPFocus:', bestparams$MIPFocus, '\n')

# Clear space
rm(model, params, result, bestparams)

piecewise.R

# Copyright 2020, Gurobi Optimization, LLC
#
# This example considers the following separable, convex problem:
# # minimize # f(x) - y + g(z) # subject to # # x + 2 y + 3 z <= 4 # x + y >= 1 # x, y, z <= 1 # # where f(u) = exp(-u) and g(u) = 2 u^2 - 4u, for all real u. It # formulates and solves a simpler LP model by approximating f and # g with piecewise-linear functions. Then it transforms the model # into a MIP by negating the approximation for f, which gives # a non-convex piecewise-linear function, and solves it again.

library(gurobi)
model <- list()

model$A <- matrix(c(1,2,3,1,1,0), nrow=2, byrow=T)
model$obj <- c(0,-1,0)
model$ub <- c(1,1,1)
model$rhs <- c(4,1)
model$sense <- c('<', '>', ')

# Uniformly spaced points in [0.0, 1.0]
u <- seq(from=0, to=1, by=0.01)

# First piecewise-linear function: f(x) = exp(-x)
pwl1 <- list()
pwl1$var <- 1
pwl1$x <- u
pwl1$y <- sapply(u, function(x) exp(-x))

# Second piecewise-linear function: g(z) = 2 z^2 - 4 z
pwl2 <- list()
pwl2$var <- 3
pwl2$x <- u
pwl2$y <- sapply(u, function(z) 2 * z * z - 4 * z)

model$pwlobj <- list(pwl1, pwl2)

result <- gurobi(model)

print(result$objval)
print(result$x)

# Negate piecewise-linear function on x, making it non-convex
model$pwlobj[[1]]$y <- sapply(u, function(x) -exp(-x))

result <- gurobi(model)
gurobi_write(model, "pwl.lp")

print(result$objval)
print(result$x)

# Clear space
rm(model, pwl1, pwl2, result)

poolsearch.R

# Copyright 2020, Gurobi Optimization, LLC
#
# We find alternative epsilon-optimal solutions to a given knapsack
# problem by using PoolSearchMode

library(Matrix)
library(gurobi)

# define primitive data
groundSetSize <- 10
objCoef <- c(32, 32, 15, 15, 6, 6, 1, 1, 1, 1)
knapsackCoef <- c(16, 16, 8, 8, 4, 4, 2, 2, 1, 1)
Budget <- 33

# Initialize model
model <- list()
model$modelsense <- 'max'
model$modelname <- 'poolsearch'

# Set variables
model$obj <- objCoef
model$vtype <- 'B'
model$lb <- 0
model$ub <- 1
model$varnames <- sprintf('El%d', seq(1, groundSetSize))

# Build constraint matrix
model$A <- spMatrix(1, groundSetSize,
  i = rep(1, groundSetSize),
  j = 1:groundSetSize,
  x = knapsackCoef)
modelrhs <- c(Budget)
modelsense <- c('<')
model$constrnames <- c('Budget')

# Set poolsearch parameters
params <- list()
params$PoolSolutions <- 1024
params$PoolGap <- 0.10
params$PoolSearchMode <- 2

# Save problem
gurobi_write(model, 'poolsearch_R.lp')

# Optimize
result <- gurobi(model, params)

# Capture solution information
if (result$status != 'OPTIMAL') {
  cat('Optimization finished with status ', result$status, '\n')
  stop('Stop now')
}

# Print best solution
cat('Selected elements in best solution: '\n')
cat(model$varnames[which(result$x >= 0.9)], '\n')

# Print all solution objectives and best further solution
if ('pool' %in% names(result)) {
  solcount <- length(result$pool)
  cat('Number of solutions found: ', solcount, '\n')
  cat('Objective values for first', solcount, 'solutions: '\n')
  for (k in 1:solcount) {
    cat(result$pool[[k]]$objval, ',', sep=',')
  }
  cat('\n')
  if (solcount >= 4) {
    cat('Selected elements in fourth best solution: '\n')
    cat(model$varnames[which(result$pool[[4]]$xn >= 0.9)], '\n')
  } else {
    solcount <- 1
    cat('Solution 1 has objective: ', result$objval, '\n')
  }
}

# Clean up
rm(model, params, result)
# This example formulates and solves the following simple QCP model:
# maximize
# x
# subject to
#  x + y + z  = 1
#  x^2 + y^2 <= z^2  (second-order cone)
#  x^2 <= yz  (rotated second-order cone)
#  x, y, z non-negative

library(gurobi)
library(Matrix)

model <- list()

model$A <- matrix(c(1,1,1), nrow=1, byrow=T)
model$modelsense <- 'max'
model$obj <- c(1,0,0)
model$rhs <- c(1)
model$sense <- c('=')

# First quadratic constraint: x^2 + y^2 - z^2 <= 0
qc1 <- list()
qc1$Qc <- spMatrix(3, 3, c(1, 2, 3), c(1, 2, 3), c(1.0, 1.0, -1.0))
qc1$rhs <- 0.0

# Second quadratic constraint: x^2 - yz <= 0
qc2 <- list()
qc2$Qc <- spMatrix(3, 3, c(1, 2), c(1, 3), c(1.0, -1.0))
qc2$rhs <- 0.0

model$quadcon <- list(qc1, qc2)

result <- gurobi(model)

print(result$objval)
print(result$x)

# Clear space
rm(model, result)
library(gurobi)

model <- list()

model$A <- matrix(c(1,2,3,1,1,0), nrow=2, byrow=T)
model$Q <- matrix(c(1,0.5,0,0.5,1,0.5,0,0.5,1), nrow=3, byrow=T)
model$obj <- c(2,0,0)
model$rhs <- c(4,1)
model$sense <- c('>', '>

result <- gurobi(model)

print(result$objval)
print(result$x)

model$vtype <- c('I', 'I', 'I')

result <- gurobi(model)

print(result$objval)
print(result$x)

# Clear space
rm(model, result)

sensitivity.R

# Copyright 2020, Gurobi Optimization, LLC
#
# A simple sensitivity analysis example which reads a MIP model
# from a file and solves it. Then each binary variable is set
# to 1-X, where X is its value in the optimal solution, and
# the impact on the objective function value is reported.

library(Matrix)
library(gurobi)

args <- commandArgs(trailingOnly = TRUE)
if (length(args) < 1) {
  stop('Usage: Rscript sensitivity.R filename
}

# Read model
cat('Reading model', args[1], '...
')
model <- gurobi_read(args[1])
cat('... done
')

# Detect set of non-continous variables
numvars <- ncol(model$A)
intvars <- which(model$vtype != 'C')
numintvars <- length(intvars)
if (numintvars < 1) {
  stop('All model\'s variables are continuous, nothing to do
')
}

# Optimize
result <- gurobi(model)

# Capture solution information
if (result$status != 'OPTIMAL') {
  cat('Optimization finished with status', result$status, '
')
  stop('Stop now
')
}
origx <- result$x
origobjval <- result$objval

# create lb and ub if they do not exists, and set them to default values
if (!('lb' %in% names(model))) {
  model$lb <- numeric(numvars)
}
if (!('ub' %in% names(model))) {
  # This line is not needed, as we must have ub defined
  model$ub <- Inf + numeric(numvars)
}

# Disable output for subsequent solves
params <- list()
params$OutputFlag <- 0

# Iterate through unfixed binary variables in the model
for (j in 1:numvars) {
  if (model$vtype[j] != 'B' &&
      model$vtype[j] != 'I') next
  if (model$vtype[j] == 'I') {
    if (model$lb[j] != 0.0) next
    if (model$ub[j] != 1.0) next
  } else {
    if (model$lb[j] > 0.0) next
    if (model$ub[j] < 1.0) next
  }
}

# Update MIP start for all variables
model$start <- origx

# Set variable to 1-X, where X is its value in optimal solution
if (origx[j] < 0.5) {
  model$start[j] <- 1
  model$lb[j] <- 1
} else {
  model$start[j] <- 0
  model$ub[j] <- 0
}

# Optimize
result <- gurobi(model, params)

# Display result
varnames <- ''
if ('varnames' %in% names(model)) {
  varnames <- model$varnames[j]
} else {
  varnames <- sprintf('%s%d', model$vtype[j], j)

if (result$status != 'OPTIMAL') {
    gap <- Inf
} else {
    gap <- result$objval - origobjval
}
cat('Objective sensitivity for variable', varnames, 'is', gap, '\n')

# Restore original bounds
model$lb[j] <- 0
model$ub[j] <- 1

# Clear space
rm(model, params, result, origx)

sos.R

# Copyright 2020, Gurobi Optimization, LLC
#
# This example formulates and solves the following simple SOS model:
# maximize
# 2 x + y + z
# subject to
#  x1 = 0 or x2 = 0 (SOS1 constraint)
#  x1 = 0 or x3 = 0 (SOS1 constraint)
#  x1 <= 1, x2 <= 1, x3 <= 2

library(gurobi)

model <- list()

model$A <- matrix(c(0,0,0), nrow=1, byrow=T)
model$obj <- c(2,1,1)
model$modelsense <- 'max'
model$ub <- c(1,1,2)
model$rhs <- c(0)
model$sense <- c('=')

# First SOS: x1 = 0 or x2 = 0
sos1 <- list()
sos1$type <- 1
sos1$index <- c(1, 2)
sos1$weight <- c(1, 2)

# Second SOS: x1 = 0 or x3 = 0
sos2 <- list()
sos2$type <- 1
sos2$index <- c(1, 3)
sos2$weight <- c(1, 2)

model$sos <- list(sos1, sos2)

result <- gurobi(model)
print(result$objval)
print(result$x)

# Clear space
rm(model, result)

sudoku.R

# Copyright 2020, Gurobi Optimization, LLC */
#
# Sudoku example.
#
# The Sudoku board is a 9x9 grid, which is further divided into a 3x3 grid
# of 3x3 grids. Each cell in the grid must take a value from 0 to 9.
# No two grid cells in the same row, column, or 3x3 subgrid may take the
# same value.
#
# In the MIP formulation, binary variables x[i,j,v] indicate whether
# cell <i,j> takes value 'v'. The constraints are as follows:
# 1. Each cell must take exactly one value (sum_v x[i,j,v] = 1)
# 2. Each value is used exactly once per row (sum_i x[i,j,v] = 1)
# 3. Each value is used exactly once per column (sum_j x[i,j,v] = 1)
# 4. Each value is used exactly once per 3x3 subgrid (sum_grid x[i,j,v] = 1)
#
# Input datasets for this example can be found in examples/data/sudoku*.
#
library(Matrix)
library(gurobi)

args <- commandArgs(trailingOnly = TRUE)
if (length(args) < 1) {
  stop('Usage: Rscript sudoku.R filename
')
}

# Read input file
conn <- file(args[1], open='r')
if(!isOpen(conn)) {
  cat('Could not read file',args[1],'
')
  stop('Stop now
')
}
linn <- readLines(conn)
close(conn)

# Ensure that all lines have the same length as the number of lines, and
# that the character set is the correct one.
# Load fixed positions in board
Dim <- length(linn)
board <- matrix(0, Dim, Dim, byrow = TRUE)
if (Dim != 9) {
  cat('Input file',args[1],'has',Dim,'lines instead of 9
')
  stop('Stop now
')
}
for (i in 1:Dim) {
  line <- strsplit(linn[[i]],split='')[[1]]
  if (length(line) != Dim) {
    cat('Line',i,'does not have',Dim,'characters
')
    stop('Stop now
')
  }
}
cat('Input line ',i,' has ',length(line),' characters, expected ',Dim,' \n')
stop('Stop now\n')
}
for (j in 1:Dim) {
  if (line[[j]] != '.') {
    k <- as.numeric(line[[j]])
    if (k < 1 || k > Dim) {
      cat('Unexpected character in Input line',i,' character',j,' \n')
      stop('Stop now\n')
    } else {
      board[i,j] = k
    }
  }
}

# Map X[i,j,k] into an index variable in the model
nVars <- Dim * Dim * Dim
varIdx <- function(i,j,k) {i + (j - 1) * Dim + (k - 1) * Dim * Dim}

for (i in 1:Dim) {
  for (j in 1:Dim) {
    for (k in 1:Dim) {
      if (board[i,j] == k) model$lb[varIdx(i,j,k)] = 1
      model$varnames[varIdx(i,j,k)] = paste0('X',i,j,k)
    }
  }
}

# Create (empty) constraints:
model$A <- spMatrix(0,nVars)
model$rhs <- c()
model$sense <- c()
model$constrnames <- c()

# Each cell gets a value:
for (i in 1:Dim) {
  for (j in 1:Dim) {
    B <- spMatrix(1, nVars, 
      i = rep(1,Dim),
\[
\begin{align*}
    j &= \text{varIdx}(i,j,1:Dim), \\
    x &= \text{rep}(1,Dim)) \\
    \text{model}\$A &= \text{rbind}(&\text{model}\$A, B) \\
    \text{model}\$rhs &= \text{c}(\text{model}\$rhs, 1) \\
    \text{model}\$sense &= \text{c}(\text{model}\$sense, '=' ) \\
    \text{model}\$constrnames &= \text{c}(\text{model}\$constrnames, paste0('OneValInCell',i,j)) \\
\end{align*}
\]

```r
# Each value must appear once in each column
for (i in 1:Dim) {
    for (k in 1:Dim) {
        B <- spMatrix(1, nVars, 
            i = rep(1,Dim), 
            j = varIdx(i,1:Dim,k), 
            x = rep(1,Dim)) 
        \text{model}\$A &= \text{rbind}(&\text{model}\$A, B) \\
        \text{model}\$rhs &= \text{c}(\text{model}\$rhs, 1) \\
        \text{model}\$sense &= \text{c}(\text{model}\$sense, '=' ) \\
        \text{model}\$constrnames &= \text{c}(\text{model}\$constrnames, paste0('OnceValueInRow',i,k)) \\
    }
}

# Each value must appear once in each row
for (j in 1:Dim) {
    for (k in 1:Dim) {
        B <- spMatrix(1, nVars, 
            i = rep(1,Dim), 
            j = varIdx(1:Dim,j,k), 
            x = rep(1,Dim)) 
        \text{model}\$A &= \text{rbind}(&\text{model}\$A, B) \\
        \text{model}\$rhs &= \text{c}(\text{model}\$rhs, 1) \\
        \text{model}\$sense &= \text{c}(\text{model}\$sense, '=' ) \\
        \text{model}\$constrnames &= \text{c}(\text{model}\$constrnames, paste0('OnceValueInColumn',j,k)) \\
    }
}

# Each value must appear once in each subgrid
SubDim <- 3
for (k in 1:Dim) {
    for (g1 in 1:SubDim) {
        for (g2 in 1:SubDim) {
            B <- spMatrix(1, nVars, 
                i = rep(1,Dim), 
                j = \text{varIdx}(1+(g1-1)*SubDim,(g2-1)*SubDim + 1:SubDim, k), 
                varIdx(2+(g1-1)*SubDim,(g2-1)*SubDim + 1:SubDim, k), 
                varIdx(3+(g1-1)*SubDim,(g2-1)*SubDim + 1:SubDim, k)), 
                x = rep(1,Dim)) 
            \text{model}\$A &= \text{rbind}(&\text{model}\$A, B) \\
            \text{model}\$rhs &= \text{c}(\text{model}\$rhs, 1) \\
            \text{model}\$sense &= \text{c}(\text{model}\$sense, '=' ) \\
            \text{model}\$constrnames &= \text{c}(\text{model}\$constrnames, 
                paste0('OnceValueInSubGrid',g1,g2,k)) \\
        }
    }
}
```
# Save model
```
gurobi_write(model, 'sudoku.lp', env)
```

# Optimize model
```
result <- gurobi(model, env = env)
if (result$status == 'OPTIMAL') {
  cat('Solution:
')
  for (i in 1:Dim) {
    for (j in 1:Dim) {
      if (j %% SubDim == 1) cat('| ',
        for (k in 1:Dim) {
          if (result$x[varIdx(i,j,k)] > 0.99) {
            cat(k, ' ', )
          }
        }
      }
    }
  }
  cat('
')
  if (i %% SubDim == 0) cat('----------------------------------
')
} else {
  cat('Problem was infeasible
')
}
```

# Clear space
```
rm(result, model, board, linn, env)
```

# Copyright 2020, Gurobi Optimization, LLC

# Assign workers to shifts; each worker may or may not be available on a
# particular day. If the problem cannot be solved, use IIS to find a set of
# conflicting constraints. Note that there may be additional conflicts
# besides what is reported via IIS.

library(Matrix)
library(gurobi)

# define data
nShifts <- 14
nWorkers <- 7
nVars <- nShifts * nWorkers
varIdx <- function(w,s) {s*(w-1)+nShifts}
Shifs <- c('Mon1', 'Tue2', 'Wed3', 'Thu4', 'Fri5', 'Sat6', 'Sun7',
  'Mon8', 'Tue9', 'Wed10', 'Thu11', 'Fri12', 'Sat13', 'Sun14')
Workers <- c('Amy', 'Bob', 'Cathy', 'Dan', 'Ed', 'Fred', 'Gu')
pay <- c(10, 12, 10, 8, 8, 9, 11)
shiftRequirements <- c(3, 2, 4, 4, 5, 6, 5, 2, 2, 3, 4, 6, 7, 5)
availability <- list(c(0, 1, 1, 0, 1, 0, 1, 0, 0, 1, 1, 1, 1, 1, 1),
  c(0, 1, 1, 0, 1, 0, 1, 0, 0, 1, 1, 1, 1, 1),
  c(0, 1, 1, 0, 1, 0, 1, 0, 0, 1, 1, 1, 1, 1),
  c(0, 1, 1, 0, 1, 0, 1, 0, 0, 1, 1, 1, 1, 1),
  c(0, 1, 1, 0, 1, 0, 1, 0, 0, 1, 1, 1, 1, 1),
  c(0, 1, 1, 0, 1, 0, 1, 0, 0, 1, 1, 1, 1, 1),
  c(0, 1, 1, 0, 1, 0, 1, 0, 0, 1, 1, 1, 1, 1),
  c(0, 1, 1, 0, 1, 0, 1, 0, 0, 1, 1, 1, 1, 1),
  c(0, 1, 1, 0, 1, 0, 1, 0, 0, 1, 1, 1, 1, 1),
  c(0, 1, 1, 0, 1, 0, 1, 0, 0, 1, 1, 1, 1, 1),
  c(0, 1, 1, 0, 1, 0, 1, 0, 0, 1, 1, 1, 1, 1),
  c(0, 1, 1, 0, 1, 0, 1, 0, 0, 1, 1, 1, 1, 1),
  c(0, 1, 1, 0, 1, 0, 1, 0, 0, 1, 1, 1, 1, 1),
  c(0, 1, 1, 0, 1, 0, 1, 0, 0, 1, 1, 1, 1, 1),
  c(0, 1, 1, 0, 1, 0, 1, 0, 0, 1, 1, 1, 1, 1))
# Set-up environment
env <- list()
env$logfile <- 'workforce1.log'

# Build model
model <- list()
model$modelname <- 'workforce1'
model$modelsense <- 'min'

# Initialize assignment decision variables:
# x[w][s] == 1 if worker w is assigned
to shift s. Since an assignment model always produces integer
solutions, we use continuous variables and solve as an LP.
model$lb <- 0
model$ub <- rep(1, nVars)
model$obj <- rep(0, nVars)
model$varnames <- rep('',nVars)
for (w in 1:nWorkers) {
  for (s in 1:nShifts) {
    model$varnames[varIdx(w,s)] = paste0(Workers[w],'.',Shifts[s])
    model$obj[varIdx(w,s)] = pay[w]
    if (availability[[w]][s] == 0) model$ub[varIdx(w,s)] = 0
  }
}

# Set-up shift-requirements constraints
model$A <- spMatrix(nShifts,nVars,
i = c(mapply(rep,1:nShifts,nWorkers)),
j = mapply(varIdx,1:nWorkers,
mapply(rep,1:nShifts,nWorkers)),
x = rep(1,nShifts * nWorkers))
model$sense <- rep('=',nShifts)
model$rhs <- shiftRequirements
model$constrnames <- Shifts

# Save model
gurobi_write(model,'workforce1.lp', env)

# Optimize
result <- gurobi(model, env = env)

# Display results
if (result$status == 'OPTIMAL') {
# The code may enter here if you change some of the data... otherwise
# this will never be executed.
cat('The optimal objective is ',result$objval,'
')
cat('Schedule:
')
for (s in 1:nShifts) {
  cat('\t',Shifts[s],':
')
}
for (w in 1:nWorkers) {
    if (result$x[varIdx(w,s)] > 0.9) cat(Workers[w], ' ')
} cat('
')
}
else if (result$status == 'INFEASIBLE') {
  # Find ONE IIS
  cat('Problem is infeasible.... computing IIS
')
  iis <- gurobi_iis(model, env = env)
  if (iis$minimal) cat('IIS is minimal
')
  else cat('IIS is not minimal
')
  cat('Rows in IIS: ', model$constrnames[iis$Arows])
  cat('
LB in IIS: ', model$varnames[iis$lb])
  cat('
UB in IIS: ', model$varnames[iis$ub])
  cat('
')
  rm(iis)
} else {
  # Just to handle user interruptions or other problems
  cat('Unexpected status', result$status,'
Ending now
')
}

# Clear space
rm(model, env, availability, Shifts, Workers, pay, shiftRequirements, result)

workforce2.R

# Copyright 2020, Gurobi Optimization, LLC
#
# Assign workers to shifts; each worker may or may not be available on a
# particular day. If the problem cannot be solved, use IIS iteratively to
# find all conflicting constraints.

library(Matrix)
library(gurobi)

# Function to display results
printsolution <- function(result) {
if (result$status == 'OPTIMAL') {
  cat('The optimal objective is', result$objval,'
')
  cat('Schedule:
')
  for (s in 1:nShifts) {
    cat('	', Shifts[s],':
')
    for (w in 1:nWorkers) {
      if (result$x[varIdx(w,s)] > 0.9) cat(Workers[w], ' ')
    }
  }
  cat('
')
}
}

# define data
nShifts <- 14
nWorkers <- 7
nVars <- nShifts * nWorkers
varIdx <- function(w,s) {s+(w-1)*nShifts}
Shifts <- c('Mon1', 'Tue2', 'Wed3', 'Thu4', 'Fri5', 'Sat6', 'Sun7', 'Mon8', 'Tue9', 'Wed10', 'Thu11', 'Fri12', 'Sat13', 'Sun14')
Workers <- c('Amy', 'Bob', 'Cathy', 'Dan', 'Ed', 'Fred', 'Gu')
pay <- c(10, 12, 10, 8, 8, 9, 11)
shiftRequirements <- c(3, 2, 4, 4, 5, 6, 5, 2, 2, 3, 4, 6, 7, 5)
availability <- list(c(0, 1, 1, 0, 1, 0, 1, 1, 1, 1, 1, 1),
                     c(1, 1, 0, 0, 1, 0, 1, 0, 1, 0, 1, 0),
                     c(0, 0, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1),
                     c(0, 1, 1, 0, 1, 0, 1, 1, 1, 1, 1, 1),
                     c(1, 1, 1, 1, 1, 0, 1, 1, 0, 1, 0, 1),
                     c(1, 1, 1, 0, 0, 1, 0, 1, 1, 0, 1, 1),
                     c(1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1))

# Set-up environment
env <- list()
env$logfile <- 'workforce2.log'

# Build model
model <- list()
model$modelname <- 'workforce2'
model$modelsense <- 'min'

# Initialize assignment decision variables:
# x[w][s] == 1 if worker w is assigned
to shift s. Since an assignment model always produces integer
solutions, we use continuous variables and solve as an LP.
model$lb <- 0
model$ub <- rep(1, nVars)
model$obj <- rep(0, nVars)
model$varnames <- rep('\', nVars)
for (w in 1:nWorkers) {
  for (s in 1:nShifts) {
    model$varnames[varIdx(w,s)] = paste0(Workers[w],'.',Shifts[s])
    model$obj[varIdx(w,s)] = pay[w]
    if (availability[[w]][s] == 0) model$ub[varIdx(w,s)] = 0
  }
}

# Set-up shift-requirements constraints
model$A <- spMatrix(nShifts, nVars,
i = c(mapply(rep,1:nShifts,nWorkers)),
j = mapply(varIdx,1:nWorkers,
mapply(rep,1:nShifts,nWorkers)),
x = rep(1,nShifts*nWorkers))
model$sense <- rep('=', nShifts)
model$rhs <- shiftRequirements
model$constrnames <- Shifts

# Save model
gurobi_write(model, 'workforce2.lp', env)

# Optimize
result <- gurobi(model, env = env)
if (result$status == 'OPTIMAL') {
  # The code may enter here if you change some of the data... otherwise
  # this will never be executed.
  printsolution(result);
} else if (result$status == 'INFEASIBLE') {
  # We will loop until we reduce a model that can be solved
  numremoved <- 0
  while(result$status == 'INFEASIBLE') {
    iis <- gurobi_iis(model, env = env)
    keep <- (!iis$Arows)
    cat('Removing rows',model$constrnames[iis$Arows],'
')
    model$A <- model$A[keep,,drop = FALSE]
    model$sense <- model$sense[keep]
    model$rhs <- model$rhs[keep]
    model$constrnames <- model$constrnames[keep]
    numremoved <- numremoved + 1
    gurobi_write(model, paste0('workforce2-',numremoved,'.lp'), env)
    result <- gurobi(model, env = env)
  }
  printsolution(result)
  rm(iis)
} else {
  # Just to handle user interruptions or other problems
  cat('Unexpected status',result$status,'
Ending now
')
}

# Clear space
rm(model, env, availability, Shifts, Workers, pay, shiftRequirements, result)

workforce3.R

# Copyright 2020, Gurobi Optimization, LLC
# Assign workers to shifts; each worker may or may not be available on a
# particular day. If the problem cannot be solved, relax the model
# to determine which constraints cannot be satisfied, and how much
# they need to be relaxed.

library(Matrix)
library(gurobi)

# Function to display results
printsolution <- function(result) {
  if(result$status == 'OPTIMAL') {
    cat('The optimal objective is',result$objval,'
')
    cat('Schedule:
')
    for (s in 1:nShifts) {
      cat('	',Shifts[s],':
')
      for (w in 1:nWorkers) {
        if (result$x[varIdx(w,s)] > 0.9) cat(Workers[w],', ')
      }
    }
    cat('
')
  }
}
# define data
nShifts <- 14
nWorkers <- 7
nVars <- nShifts * nWorkers
varIdx <- function(w,s) {s+(w-1)*nShifts}
Shifts <- c('Mon1', 'Tue2', 'Wed3', 'Thu4', 'Fri5', 'Sat6', 'Sun7', 'Mon8', 'Tue9', 'Wed10', 'Thu11', 'Fri12', 'Sat13', 'Sun14')
Workers <- c('Amy', 'Bob', 'Cathy', 'Dan', 'Ed', 'Fred', 'Gu')
pay <- c(10, 12, 10, 8, 8, 9, 11)
shiftRequirements <- c(3, 2, 4, 4, 5, 6, 5, 2, 2, 3, 4, 6, 7, 5)
availability <- list(c(0, 1, 1, 0, 1, 0, 1, 1, 1, 1, 1, 1),
                     c(1, 1, 0, 1, 1, 0, 1, 0, 1, 0, 1, 0),
                     c(0, 0, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1),
                     c(0, 1, 1, 1, 0, 1, 0, 1, 1, 1, 1, 1),
                     c(1, 1, 1, 1, 1, 0, 1, 1, 1, 0, 1, 1),
                     c(1, 1, 1, 0, 0, 1, 1, 1, 0, 1, 1, 1),
                     c(1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1))

# Set-up environment
env <- list()
env$logfile <- 'workforce3.log'

# Build model
model <- list()
model$modelname <- 'workforce3'
model$modelsense <- 'min'

# Initialize assignment decision variables:
# x[w][s] == 1 if worker w is assigned
# to shift s. Since an assignment model always produces integer
# solutions, we use continuous variables and solve as an LP.
model$lb <- 0
model$ub <- rep(1, nVars)
model$obj <- rep(0, nVars)
model$varnames <- rep('',nVars)
for (w in 1:nWorkers) {
  for (s in 1:nShifts) {
    model$varnames[varIdx(w,s)] = paste0(Workers[w],'.',Shifts[s])
    model$obj[varIdx(w,s)] = pay[w]
    if (availability[[w]][s] == 0) model$ub[varIdx(w,s)] = 0
  }
}

# Set-up shift-requirements constraints
model$A <- spMatrix(nShifts,nVars,
  i = c(mapply(rep,1:nShifts,nWorkers)),
  j = mapply(varIdx,1:nWorkers,
              mapply(rep,1:nShifts,nWorkers)),
  x = rep(1,nShifts * nWorkers))
model$sense <- rep('==',nShifts)
model$rhs  <- shiftRequirements
model$constrnames  <- Shifts

# Save model
gurobi_write(model,'workforce3.lp', env)

# Optimize
result  <- gurobi(model, env = env)

# Display results
if (result$status  == 'OPTIMAL') {
  # The code may enter here if you change some of the data... otherwise
  # this will never be executed.
  printsolution(result);
} else if (result$status  == 'INFEASIBLE') {
  # Use gurobi_feasrelax to find out which contraints should be relaxed
  # and by how much to make the problem feasible.
  penalties  <- list()
  penalties$lb  <- Inf
  penalties$ub  <- Inf
  penalties$rhs  <- rep(1,length(model$rhs))
  feasrelax  <- gurobi_feasrelax(model, 0, FALSE, penalties, env = env)
  result  <- gurobi(feasrelax$model, env = env)
  if (result$status  == 'OPTIMAL') {
    printsolution(result)
    cat('Slack values :
    for (j in (nVars+1):length(result$x)) {
      if(result$x[j] > 0.1)
        cat('	', feasrelax$model$varnames[j],result$x[j],'
    )
    } else {
      cat('Unexpected status ',result$status ,'
Ending now
')
    }
  else {
    # Just to handle user interruptions or other problems
    cat('Unexpected status ',result$status ,'
Ending now
')
  }
  }
  # Clear space
  rm(penalties, feasrelax)
} else {
  cat('Unexpected status ',result$status ,'
Ending now
')
}

workforce4.R

# Copyright 2020, Gurobi Optimization, LLC
#
# Assign workers to shifts; each worker may or may not be available on a
# particular day. We use Pareto optimization to solve the model:
# first, we minimize the linear sum of the slacks. Then, we constrain
# the sum of the slacks, and we minimize a quadratic objective that
# tries to balance the workload among the workers.

library(Matrix)
library(gurobi)

# define data
nShifts <- 14
nWorkers <- 7
nVars <- (nShifts + 1) * (nWorkers + 1)
varIdx <- function(w, s) {s + (w - 1) * nShifts}
shiftSlackIdx <- function(s) {s + nShifts * nWorkers}
totShiftIdx <- function(w) {w + nShifts * (nWorkers + 1)}
avgShiftIdx <- ((nShifts + 1) * (nWorkers + 1))
diffShiftIdx <- function(w) {w + avgShiftIdx}
totalSlackIdx <- nVars

Shifts <- c('Mon1', 'Tue2', 'Wed3', 'Thu4', 'Fri5', 'Sat6', 'Sun7',
            'Mon8', 'Tue9', 'Wed10', 'Thu11', 'Fri12', 'Sat13', 'Sun14')
Workers <- c('Amy', 'Bob', 'Cathy', 'Dan', 'Ed', 'Fred', 'Gu')
shiftRequirements <- c(3, 2, 4, 4, 5, 6, 5, 2, 2, 3, 4, 6, 7, 5)
availability <- list(c(0, 1, 1, 0, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1),
                      c(0, 0, 1, 0, 1, 0, 1, 1, 1, 1, 1, 1, 0, 0),
                      c(0, 0, 1, 1, 0, 1, 0, 1, 1, 1, 1, 1, 1, 1),
                      c(0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1),
                      c(1, 1, 1, 0, 0, 1, 0, 1, 0, 0, 1, 1, 1, 1),
                      c(1, 1, 1, 0, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1),
                      c(1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1))

# Function to display results
solveandprint <- function(model, env) {
  result <- gurobi(model, env = env)
  if(result$status == 'OPTIMAL') {
    cat('The optimal objective is', result$objval, '\n')
    cat('Schedule:\n')
    for (s in 1:nShifts) {
      cat(' ', Shifts[s], ':')
      for (w in 1:nWorkers) {
        if (result$x[varIdx(w, s)] > 0.9) cat(Workers[w], ' ')}
    }
  }
  cat('Workload:\n')
  for (w in 1:nWorkers) {
    cat(' ', Workers[w], ':', result$x[totShiftIdx(w)], ',')
  }
  else {
    cat('Optimization finished with status', result$status)
  }
  result
}

# Set-up environment
env <- list()
env$logfile <- 'workforce4.log'

# Build model
model <- list()
model$modelname <- 'workforce4'
model$modelsense <- 'min'
# Initialize assignment decision variables:
# x[w][s] == 1 if worker w is assigned to shift s.
# This is no longer a pure assignment model, so we must
# use binary variables.
model$vtype <- rep('C', nVars)
model$lb <- rep(0, nVars)
model$ub <- rep(1, nVars)
model$obj <- rep(0, nVars)
model$varnames <- rep('',nVars)
for (w in 1:nWorkers) {
  for (s in 1:nShifts) {
    model$vtype[varIdx(w,s)] = 'B'
    model$varnames[varIdx(w,s)] = paste0(Workers[w],'.',Shifts[s])
    if (availability[[w]][s] == 0) model$ub[varIdx(w,s)] = 0
  }
}

# Initialize shift slack variables
for (s in 1:nShifts) {
  model$varnames[shiftSlackIdx(s)] = paste0('ShiftSlack',Shifts[s])
  model$ub[shiftSlackIdx(s)] = Inf
}

# Initialize worker slack and diff variables
for (w in 1:nWorkers) {
  model$varnames[totShiftIdx(w)] = paste0('TotalShifts',Workers[w])
  model$ub[totShiftIdx(w)] = Inf
  model$varnames[diffShiftIdx(w)] = paste0('DiffShifts',Workers[w])
  model$ub[diffShiftIdx(w)] = Inf
  model$lb[diffShiftIdx(w)] = -Inf
}

#Initialize average shift variable
model$ub[avgShiftIdx] = Inf
model$varnames[avgShiftIdx] = 'AvgShift'

#Initialize total slack variable
model$ub[totalSlackIdx] = Inf
model$varnames[totalSlackIdx] = 'TotalSlack'
model$obj[totalSlackIdx] = 1

# Set-up shift-requirements constraints
model$A <- spMatrix(nShifts,nVars,
  i = c(c(mapply(rep,1:nShifts,nWorkers)),
        c(1:nShifts)),
  j = c(mapply(varIdx,1:nWorkers,
                mapply(rep,1:nShifts,nWorkers)),
        shiftSlackIdx(1:nShifts)),
  x = rep(1,nShifts * (nWorkers+1)))
model$sense <- rep('=',nShifts)
model$rhs <- shiftRequirements
model$constrnames <- Shifts

# Set TotalSlack equal to the sum of each shift slack
B <- spMatrix(1, nVars,
\[ i = \text{rep}(1, n\text{Shifts}+1), \]
\[ j = c(\text{shiftSlackIdx}(1: n\text{Shifts}), \text{totalSlackIdx}), \]
\[ x = c(\text{rep}(1, n\text{Shifts}), -1)) \]

\[
\text{model}\$A \leftarrow \text{rbind(model}\$A, B)\\
\text{model}\$rhs \leftarrow c(\text{model}\$rhs, 0)\\
\text{model}\$sense \leftarrow c(\text{model}\$sense, '=')\\
\text{model}\$constrnames \leftarrow c(\text{model}\$constrnames, 'TotalSlack')
\]

# Set total number of shifts for each worker
\[
\text{B} \leftarrow \text{spMatrix}(n\text{Workers}, n\text{Vars},)\\
\text{i} = c(\text{mapply(\text{rep},1:n\text{Workers},n\text{Shifts}),} 1:n\text{Workers}),\\
\text{j} = c(\text{mapply(\text{varIdx}, c(\text{mapply(\text{rep},1:n\text{Workers},n\text{Shifts})),} 1:n\text{Shifts}),} \text{totShiftIdx}(1:n\text{Workers})),\\
\text{x} = c(\text{rep}(1, n\text{Shifts} * n\text{Workers}), \text{rep}(-1, n\text{Workers}))
\]

\[
\text{model}\$A \leftarrow \text{rbind(model}\$A, B)\\
\text{model}\$rhs \leftarrow c(\text{model}\$rhs, \text{rep}(0, n\text{Workers}))\\
\text{model}\$sense \leftarrow c(\text{model}\$sense, \text{rep}('=', n\text{Workers}))\\
\text{model}\$constrnames \leftarrow c(\text{model}\$constrnames, \text{sprintf('TotalShifts%s', Workers[1:n\text{Workers}]})
\]

# Save initial model
\[
\text{gurobi}\_\text{write(model, 'workforce4.lp', env)}
\]

# Optimize
\[
\text{result} \leftarrow \text{solveandprint(model, env)}\\
\text{if } (\text{result}\$\text{status} \neq \text{OPTIMAL}) \text{ stop('Stop now\n'})
\]

# Constraint the slack by setting its upper and lower bounds
\[
\text{totalSlack} \leftarrow \text{result}\$\text{x[totalSlackIdx]}\\
\text{model}\$\text{lb[totalSlackIdx]} = \text{totalSlack}\\
\text{model}\$\text{ub[totalSlackIdx]} = \text{totalSlack}
\]

# Link average number of shifts worked and difference with average
\[
\text{B} \leftarrow \text{spMatrix}(n\text{Workers}+1, n\text{Vars},)\\
\text{i} = c(1:n\text{Workers}, 1:n\text{Workers},)\\
\text{j} = c(\text{totShiftIdx}(1:n\text{Workers}), \text{diffShiftIdx}(1:n\text{Workers}),)\\
\text{x} = c(\text{rep}(1, n\text{Workers}), \text{rep}(-1, n\text{Workers}), \text{rep}(-1, n\text{Workers}), \text{rep}(1, n\text{Workers}), -n\text{Workers})
\]

\[
\text{model}\$A \leftarrow \text{rbind(model}\$A, B)\\
\text{model}\$rhs \leftarrow c(\text{model}\$rhs, \text{rep}(0, n\text{Workers}+1))\\
\text{model}\$sense \leftarrow c(\text{model}\$sense, \text{rep}('=', n\text{Workers}+1))\\
\text{model}\$constrnames \leftarrow c(\text{model}\$constrnames, \text{sprintf('DiffShifts%s', Workers[1:n\text{Workers}]}, 'AvgShift'))
\]

# Objective: minimize the sum of the square of the difference from the
# average number of shifts worked
\[
\text{model}\$\text{obj} \leftarrow 0
\]
```r
model$Q <- spMatrix(nVars, nVars,
    i = c(diffShiftIdx(1:nWorkers)),
    j = c(diffShiftIdx(1:nWorkers)),
    x = rep(1, nWorkers))

# Save modified model
gurobi_write(model, 'workforce4b.lp', env)

# Optimize
result <- solveandprint(model, env)
if (result$status != 'OPTIMAL') stop('Stop now
')

# Clear space
rm(model, env, availability, Shifts, Workers, shiftRequirements, result)

workforce5.R

# Copyright 2020, Gurobi Optimization, LLC
#
# Assign workers to shifts; each worker may or may not be available on a
# particular day. We use multi-objective optimization to solve the model.
# The highest-priority objective minimizes the sum of the slacks
# (i.e., the total number of uncovered shifts). The secondary objective
# minimizes the difference between the maximum and minimum number of
# shifts worked among all workers. The second optimization is allowed
# to degrade the first objective by up to the smaller value of 10% and 2

library('Matrix')
library('gurobi')

# define data
nShifts <- 14
nWorkers <- 8
nVars <- (nShifts + 1) * (nWorkers + 1) + 2
varIdx <- function(w, s) {s + (w - 1) * nShifts}
shiftSlackIdx <- function(s) {s + nShifts * nWorkers}
totShiftIdx <- function(w) {w + nShifts * (nWorkers + 1)}
minShiftIdx <- ((nShifts + 1) * (nWorkers + 1))
maxShiftIdx <- (minShiftIdx + 1)
totalSlackIdx <- nVars

Shifts <- c('Mon1', 'Tue2', 'Wed3', 'Thu4', 'Fri5', 'Sat6', 'Sun7',
            'Mon8', 'Tue9', 'Wed10', 'Thu11', 'Fri12', 'Sat13', 'Sun14')
Workers <- c('Amy', 'Bob', 'Cathy', 'Dan', 'Ed', 'Fred', 'Gu', 'Tobi')

shiftRequirements <- c(3, 2, 4, 4, 5, 6, 5, 2, 2, 3, 4, 6, 7, 5)

availability <- list(
    c(0, 1, 1, 0, 1, 0, 1, 1, 1, 1, 1, 1),
    c(1, 1, 0, 0, 1, 0, 1, 1, 1, 1, 1, 0),
    c(0, 0, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1),
    c(0, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1),
    c(1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1),
    c(0, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1),
    c(1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1))
```

619
# Function to display results
solveandprint <- function(model, env) {
  result <- gurobi(model, env = env)
  if(result$status == 'OPTIMAL') {
    cat('The optimal objective is',result$objval,'
')
    cat('Schedule:
')
    for (s in 1:nShifts) {
      cat('	',Shifts[s],':
')
      for (w in 1:nWorkers) {
        if (result$x[varIdx(w,s)] > 0.9)
          cat(Workers[w],',
')
      }
      cat('
')
    }
    cat('Workload:
')
    for (w in 1:nWorkers) {
      cat('	',Workers[w],':',result$x[totShiftIdx(w)],'
')
    }
  } else {
    cat('Optimization finished with status',result$status)
  }
  result
}

# Set-up environment
env <- list()
env$logfile <- 'workforce5.log'

# Build model
model <- list()
model$modelname <- 'workforce5'
model$modelsense <- 'min'

# Initialize assignment decision variables:
# x[w][s] == 1 if worker w is assigned to shift s.
# This is no longer a pure assignment model, so we must
# use binary variables.
model$vtype <- rep('C', nVars)
model$lb <- rep(0, nVars)
model$ub <- rep(1, nVars)
model$varnames <- rep('',nVars)
for (w in 1:nWorkers) {
  for (s in 1:nShifts) {
    model$vtype[varIdx(w,s)] = 'B'
    model$varnames[varIdx(w,s)] = paste0(Workers[w],'.',Shifts[s])
    if (availability[[w]][s] == 0) model$ub[varIdx(w,s)] = 0
  }
}

# Initialize shift slack variables
for (s in 1:nShifts) {
  model$varnames[shiftSlackIdx(s)] = paste0('ShiftSlack',Shifts[s])
  model$ub[shiftSlackIdx(s)] = Inf
}

# Initialize worker slack and diff variables
for (w in 1:nWorkers) {
    model$varnames[totShiftIdx(w)] = paste0('TotalShifts',Workers[w])
    model$ub[totShiftIdx(w)] = Inf
}

# Initialize min/max shift variables
model$ub[minShiftIdx] = Inf
model$varnames[minShiftIdx] = 'MinShift'
model$ub[maxShiftIdx] = Inf
model$varnames[maxShiftIdx] = 'MaxShift'

# Initialize total slack variable
model$ub[totalSlackIdx] = Inf
model$varnames[totalSlackIdx] = 'TotalSlack'

# Set-up shift-requirements constraints
model$A <- spMatrix(nShifts, nVars, 
    i = c(mapply(rep,1:nShifts,nWorkers), 
          c(1:nShifts)),
    j = c(mapply(varIdx,1:nWorkers, 
                    mapply(rep,1:nShifts,nWorkers)), 
          shiftSlackIdx(1:nShifts)),
    x = rep(1,nShifts * (nWorkers+1)))
model$sense <- rep('=', nShifts)
model$rhs <- shiftRequirements
model$constrnames <- Shifts

# Set TotalSlack equal to the sum of each shift slack
B <- spMatrix(1, nVars, 
    i = rep(1,nShifts+1),
    j = c(shiftSlackIdx(1:nShifts),totalSlackIdx),
    x = c(rep(1,nShifts),-1))
model$A <- rbind(model$A, B)
model$rhs <- c(model$rhs,0)
model$sense <- c(model$sense,'=')
model$constrnames <- c(model$constrnames, 'TotalSlack')

# Set total number of shifts for each worker
B <- spMatrix(nWorkers, nVars, 
    i = c(mapply(rep,1:nWorkers,1:nWorkers),
          1:nWorkers),
    j = c(mapply(varIdx, c(mapply(rep,1:nWorkers,nShifts))),1:nShifts),
          totShiftIdx(1:nWorkers)),
    x = c(rep(1,nShifts*nWorkers),rep(-1,nWorkers)))
model$A <- rbind(model$A, B)
model$rhs <- c(model$rhs,rep(0,nWorkers))
model$sense <- c(model$sense,rep('=',nWorkers))
model$constrnames <- c(model$constrnames, sprintf('TotalShifts%s',Workers[1:nWorkers]))

# Set minShift / maxShift general constraints
model$genconmin <- list(list(resvar = minShiftIdx, 
                            vars = c(totShiftIdx(1:nWorkers)), 
                            name = 'MinShift'))
model$genconmax <- list(list(resvar = maxShiftIdx, 
                             vars = c(totShiftIdx(1:nWorkers)), 
                             name = 'MaxShift'))
# Set multiobjective
model$multiobj <- list(1:2)
model$multiobj[[1]] <- list()
model$multiobj[[1]]$objn <- c(rep(0, nVars))
model$multiobj[[1]]$objc[totalSlackIdx] = 1
model$multiobj[[1]]$priority <- 2
model$multiobj[[1]]$weight <- 1
model$multiobj[[1]]$abstol <- 2
model$multiobj[[1]]$reltol <- 0.1
model$multiobj[[1]]$name <- 'TotalSlack'
model$multiobj[[1]]$con <- 0.0
model$multiobj[[2]] <- list()
model$multiobj[[2]]$objn <- c(rep(0, nVars))
model$multiobj[[2]]$objc[minShiftIdx] = -1
model$multiobj[[2]]$objc[maxShiftIdx] = 1
model$multiobj[[2]]$priority <- 1
model$multiobj[[2]]$weight <- 1
model$multiobj[[2]]$abstol <- 0
model$multiobj[[2]]$reltol <- 0
model$multiobj[[2]]$name <- 'Fairness'
model$multiobj[[2]]$con <- 0.0

# Save initial model
gurobi_write(model, 'workforce5.lp', env)

# Optimize
result <- solveandprint(model, env)
if (result$status != 'OPTIMAL') stop('Stop now\n')

# Clear space
rm(model, env, availability, Shifts, Workers, shiftRequirements, result)