GUROBI OPTIMIZATION

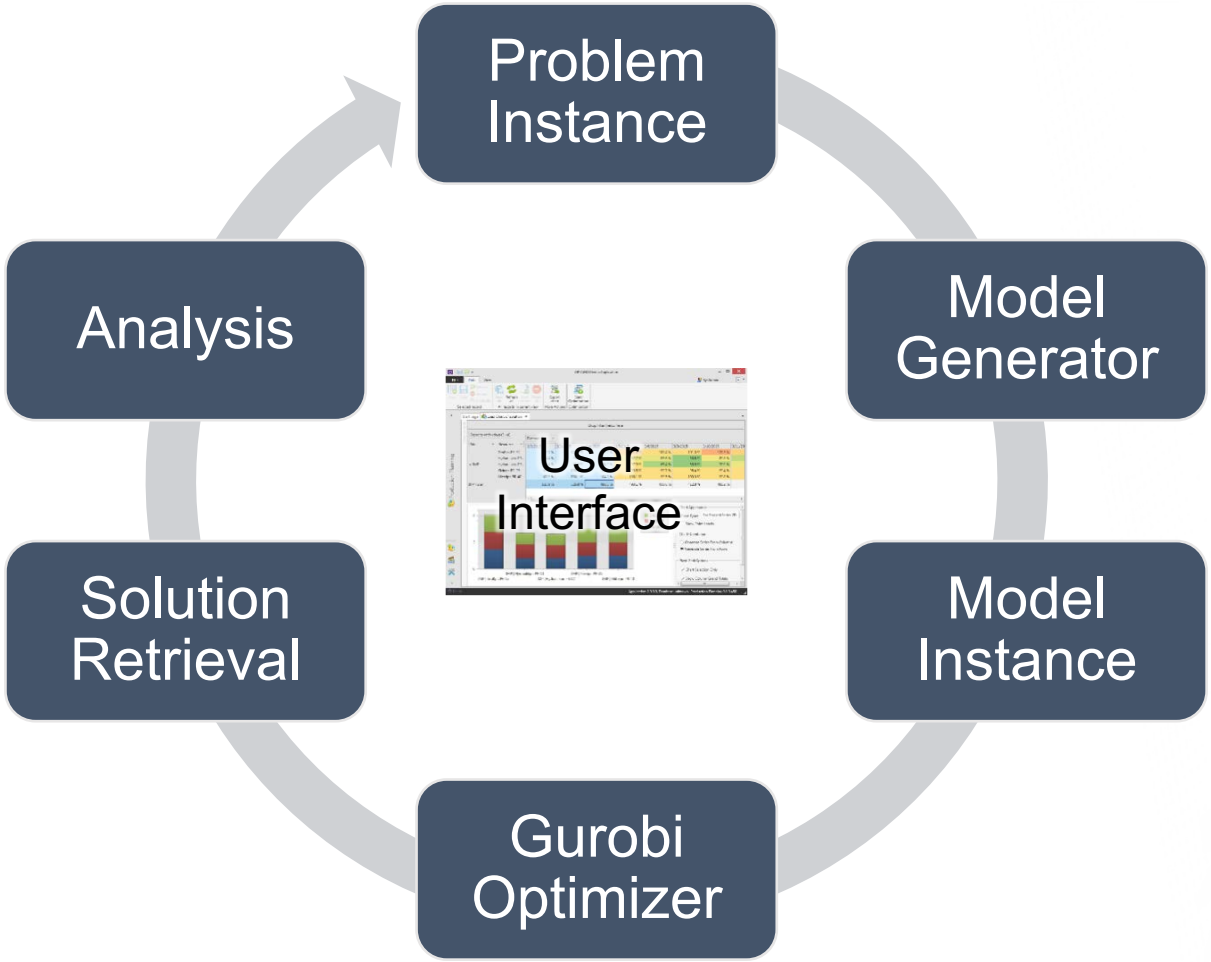
Introduction to Tuning for Optimization System

Webinar

Agenda

- Tuning Opportunities & Aspects
  - System Architecture & Implementation
    - Hardware
    - Data Access
    - Model Construction
    - Memory
    - Network Communication Overhead
  - Performance Variability
  - Numerical Issues
  - Profiling & Benchmarking
- Gurobi Parameters
  - Presolve
  - Continuous Models
  - Integer Models
- Automated Parameter Tuning

The Optimization Workflow



- The real-world problem instance is usually derived from a specific planning problem.
- A configurable model generator is used to build the model instance using data sources.
- Gurobi is used to find an optimal solution of the model instance.
- The solution is transferred back to the planning system for further analysis.
- The cycle repeats until a satisfying real-world solution has been found.

Image: http://optano.net/en/production-planning/

Tuning Opportunities & Aspects

System Architecture

- Performance problems can arise in many different parts of the system architecture

- Measure runtimes youself in every part of the process
  - Time to collect and process input data
  - Time to build the model
  - Time to solve the model

- Keep log files

- Optimization needs physical CPU and memory resources – be aware of concurrency
  - Other processes
  - Other virtual machines
  - Other Docker containers
  - …

Hardware

- People often contact Gurobi support with hardware questions
  - „I‘m going to invest into new hardware. What is the best machine for my optimization application?“
  - „Will my model solve faster with more RAM?“
  - „Which CPU models do you recommend?“
  - „How many models can I solve in parallel on this machine?“

- Answer: „It depends“

- If you are solving a large MIP model in parallel, you'll get the best performance from a system with
  - the fastest possible clock rate and
  - 4 channels per socket of the fastest available memory.

- There is no hardware recommendation for all models. You are welcome to support@gurobi.com to discuss your specific models.

Data Access

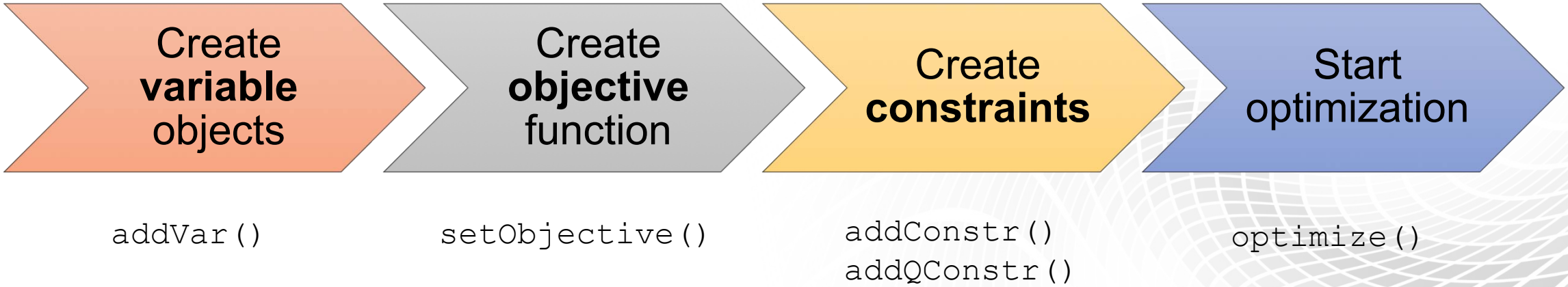
- Common support case
  - “Gurobi is soooo slow”
  - Reason: Model building outside of Gurobi takes 30 minutes
  - Model solving takes only a few seconds

- Inefficient data access is the most common reason for slow model construction
  - Long lookup times
  - Insufficient caching / redundant queries
  - Single elements instead of batch processing

- If you would remove all Gurobi API calls, how long would it take?

Model Construction

- Each matrix generator has its own pitfalls and best practices
  - Use iterators effectively for your API
  - Are you using the most efficient way to build expressions?
- Look for: bottleneck via a code profiler or measure yourself
- OO interfaces are thin layer upon C matrix interface
- With Gurobi OO interfaces, take advantage of lazy updates
  - Best to create variables all at once, then create the objective and constraints
  - Keep Gurobi model objects in memory, do not lookup elements by their name



Memory

- Insufficient memory can destroy performance
  - Virtual memory via disk is far slower than RAM
  - Parallel optimization requires more memory

- Look for: memory use via system monitor tools on computer
  - Ex: System Monitor, top, Activity Monitor

- Helpful parameters
  - Decrease `Threads`
    - High parallelism is not helpful for all models
  - Set `NodefileStart` to store MIP node info on disk
    - Only helpful when solving a MIP that requires many nodes!

- Memory is cheap; no need to skimp

Network Communication Overhead

- Be aware of potential network issues when retrieving data or accessing Gurobi Compute Server
  - High latency
  - Low bandwith
  - Stability

- Latency can become an issue when a lot of messages are sent over the network

- Compute Server statistics in the log file:

```
Compute Server communication statistics:
Sent: 8.3 MBytes in 244 msgs and 0.76s (10.92 MB/s)
Received: 7.2 MBytes in 304603 msgs and 1.53s (4.71 MB/s)
```

Performance Variability

- A MIP solver often needs to “flip a coin” to decide what to do next
- Gurobi uses a pseudo-random number generator that produces that same series of random numbers for the same random seed. The seed value can be set using a parameter (`Seed`).
- Playing with the seed parameters helps to determine the impact of random algorithmic choices (e.g. to break ties) on runtime. This is called “performance variability”.

- Performance variability is intrinsic to MIP (as a result of its NP-hardness).
  - Some models behave pretty stable and have only a small variance in their running time for different seed values.
  - There are also highly pathological models which solve in a fraction of a second for some seeds and cannot be solved within days for other seed values.

- Bottom line: Always pay attention to performance variability when benchmarking models
  - Use different instances of the samel model type
  - Run the same model instance with different random seeds

Numerical Issues

- Numerical problems within models can affects both
  - the solution time and
  - the solution quality.

It is often very helpful to reformulate and/or rescale the model.

We don't cover this topic in this presentation. You can find our numerics guide at http://files.gurobi.com/Numerics.pdf

We also had a webinar about this topic recently: http://www.gurobi.com/resources/seminars-and-videos/numerical-issues-webinar

Profiling & Benchmarking

- Model initialization and solution retrieval
  - Export and test MPS file using `gurobi_cl`
  - See if solution times are much faster

- What algorithmic part is the bottleneck?
  - Presolve
  - Solving (initial LP)
  - At node 0 of MIP
  - Other nodes of MIP
  - Log shows time spent in presolve, LP relaxation, MIP root, nodes

- Use the logs to identify the bottleneck

Gurobi Parameters

Gurobi Parameters

- Version 7.5 has 120 parameters
- Can be grouped by function:
  - **Termination**: control termination of the solvers
    - Example: `TimeLimit`
  - **Tolerances**: control solver tolerances
    - Example: `FeasibilityTol`
  - **Simplex**: affect the operation of the simplex algorithms
    - Example: `SimplexPricing`
  - **Barrier**: affect the operation of the barrier algorithm
    - Example: `Crossover`
  - **MIP**: affect the operation of the MIP algorithm
    - Example: `MIPFocus`
  - **MIP Cuts**: affect the MIP cutting planes
    - Example: `CutPasses`
  - **Tuning**: affect the operation of the automatic parameter tuner
    - Example: `TuneTimeLimit`
  - **Distributed**: affect the behavior of the distributed algorithms
    - Example: `WorkerPool`
  - **Other**

Gurobi Parameters

- Of these 120, 57 are related to performance
  - Multiple possible values for most
- We could tell you when each is appropriate…
  - Except that would take all day
  - And we often don't know ourselves
- Better to use the Gurobi parameter tuning tool…

Presolve

- Tradeoff: spend time up front with hope of simplifying model
- Look for: performance with different presolve parameters
- Primary control: `Presolve` parameter
  - Reduce if spending too much time up front
  - Increase to hope to get a simpler model
- Additional parameters for fine-grain control
  - `PrePasses`
  - `Aggregate`
  - `AggFill`
  - `PreSparsify`
  - `PreDual`
  - `PreDepRow`

Continuous algorithms

- Dual simplex
- Primal simplex
- Barrier

- Concurrent (LP)
  - Use multiple algorithms at the same time on multiple processor cores
  - Deterministic and non-deterministic versions available
  - Multiple algorithms makes it very robust
  - Requires more memory

Defaults for continuous optimization

- LP — Concurrent (non-deterministic)
- QP — Barrier
- MIP root — Dual simplex or (deterministic) concurrent, depending on model size
- MIP nodes — Dual simplex

- Parameters used to select the algorithm
  - `Method`: continuous models and root of MIPs
  - `NodeMethod`: nodes of MIPs

The fastest solver

- Interesting:
  - Concurrent LP performance worse than dual simplex performance
  - Reason?

- Concurrent **fastest on average, but never fastest for a specific model**
  - If barrier wins:
    - Concurrent wasted threads on simplex
  - If simplex wins:
    - Simplex had to fight for resources with barrier

- Concurrent uses much more memory than simplex
  - Concurrent invokes barrier
    - Barrier typically uses a lot more memory than simplex
  - Concurrent runs multiple solvers at once
    - Each needs a copy of the model

- If memory is tight, use dual simplex

LP Example

- First run

```
gurobi> m.optimize()
...
Solved with dual simplex
Solved in 11615 iterations and 3.72 seconds
Optimal objective  2.382165864e+10
gurobi> print m.getVars()[0].X
351.0
```

- Second run of same model

```
gurobi> m.optimize()
...
Solved with barrier
Solved in 53305 iterations and 3.70 seconds
Optimal objective  2.382165864e+10
gurobi> print m.getVars()[0].X
0.0
```

LP Example

- Default solver is non-deterministic concurrent
- Different optimal basis possible when dual and barrier runtimes are very close
  - Fortunately, this is rare

- If this is an issue, use a deterministic method
  - Deterministic concurrent (will be a bit slower)
  - Parallel barrier
  - Simplex

Notable continuous parameters

- `NormAdjust`
  - Select different simplex pricing norm variants
- `SimplexPricing`
  - Simplex variable pricing strategy
- `Crossover`
  - Determines strategy used to produce basis from initial crossover basis
- `CrossoverBasis`
  - Determines strategy used to produce initial basis from barrier result

Integer Problems – What makes it difficult?

- Time to solve LP/QP relaxations?
  - Adjust LP parameters
- Moving the bound?
  - Adjust cutting plane generation
  - Improve node throughput
- Finding feasible solutions?
  - Increase heuristics
  - Change branching strategy

Additional helpful MIP parameters

- Change branching strategy using
  - `VarBranch`

- Change settings for
  - `Cuts`
  - `Heuristics`

- If no solution is found after the root node try
  - `PumpPasses`
  - `ZeroObjNodes`
  - `MinRelNodes`

MIP – Example 1

- MIP log looks like this…

```
    Nodes    |    Current Node    |     Objective Bounds      |     Work
 Expl Unexpl |  Obj  Depth IntInf | Incumbent    BestBd   Gap | It/Node Time
     0     0 -264137.12    0   75          - -264137.12     -      -    0s
H    0     0                     -0.0000107 -264137.12     -      -    0s
H    0     0                    -162837.1173 -264137.12  62.2%     -    0s
     0     0 -251769.54    0   50 -162837.12 -251769.54  54.6%     -    0s
     0     0 -246602.68    0   83 -162837.12 -246602.68  51.4%     -    0s
     0     0 -241768.49    0   29 -162837.12 -241768.49  48.5%     -    0s
H    0     3                    -180084.9071 -241768.49  34.3%     -    0s
     0     3 -241768.49    0   29 -180084.91 -241768.49  34.3%     -    0s
* 2343  1391              49    -181346.8006 -214776.82  18.4%   6.4    2s
* 2440  1339              70    -183734.9437 -214277.60  16.6%   6.4    2s
* 3103  1527              59    -183933.1807 -213309.24  16.0%   6.4    2s
* 3710  1825              61    -184549.0628 -212449.79  15.1%   6.4    2s
H 5876  3164                    -188226.7818 -210900.89  12.0%   6.3    3s
  9281  5152 -192568.56   62   50 -188226.78 -209633.03  11.4%   6.2    5s
 24765 12393 -194237.85   42   50 -188226.78 -205750.35  9.31%   6.4   10s
H25267 11765                    -190619.4788 -205716.49  7.92%   6.4   10s
 39314 15223     cutoff   50      -190619.48 -203402.09  6.71%   6.4   15s
 51427 17115 -194173.45   47   25 -190619.48 -201649.93  5.79%   6.4   20s
 67287 17611     cutoff   36      -190619.48 -199597.77  4.71%   6.4   25s
 82144 15576     cutoff   42      -190619.48 -197694.67  3.71%   6.3   30s
...
```

MIP – Example 1

- Try changing the focus of the search…
  - `MIPFocus=1`: focus on finding feasible solutions
  - `MIPFocus=2`: focus on proving optimality
  - `MIPFocus=3`: focus on improving the bound

```
    Nodes    |    Current Node    |     Objective Bounds      |     Work
 Expl Unexpl |  Obj  Depth IntInf | Incumbent    BestBd   Gap | It/Node Time
...
     0     0 -209067.20    0   70 -162837.12 -209067.20  28.4%     -    1s
     0     5 -209067.20    0   70 -162837.12 -209067.20  28.4%     -    2s
H   79    66                     -165743.5947 -208071.05  25.5%   3.7    2s
H 1045   871                     -167148.0384 -208070.60  24.5%   5.2    2s
  1238  1014 -193529.37   37   55 -167148.04 -206993.76  23.8%   5.1    5s
H 1647  1140                     -168514.0578 -203113.35  20.5%   6.8    8s
* 1787   994              61     -173659.7799 -203113.35  17.0%   7.0    8s
* 2018   993              54     -175972.1389 -202292.18  15.0%   7.1    8s
H 2692   615                     -187397.9703 -200840.93  7.17%   6.9    8s
  2829   688 -187728.90   32   47 -187397.97 -200681.75  7.09%   7.4   10s
H 4228   578                     -190619.4788 -198075.82  3.91%   7.2   11s
```

MIP – Example 2

- MIP log looks like this…

```
  Nodes    |    Current Node    |     Objective Bounds      |     Work
 Expl Unexpl |  Obj  Depth IntInf | Incumbent    BestBd   Gap | It/Node Time

     0     0 164800.976    0 1096 1169768.62 164800.976  85.9%     -  186s
     0     0 165096.286    0 1607 1169768.62 165096.286  85.9%     -  526s
H    0     0                     481589.35817 165096.286  65.7%     - 1065s
     0     0 165452.400    0 1705 481589.358 165452.400  65.6%     - 1419s
     0     0 165566.561    0 1774 481589.358 165566.561  65.6%     - 1783s
     0     0 165719.831    0 1778 481589.358 165719.831  65.6%     - 2368s
     0     0 165849.075    0 1924 481589.358 165849.075  65.6%     - 2819s
…
```

- Issues:
  - Bound moving very slowly
  - “Stuck” at the root

MIP – Example 2

- In extreme cases, try turning off cuts (set `Cuts=0`)…

```
    Nodes    |    Current Node    |     Objective Bounds      |     Work
 Expl Unexpl |  Obj  Depth IntInf | Incumbent    BestBd   Gap | It/Node Time

     0     0 164800.976    0 1108 1169768.62 164800.976  85.9%     -  225s
     0     0 164800.976    0 1004 1169768.62 164800.976  85.9%     -  249s
H    0     0                    503233.56923 164800.976  67.3%     -  406s
     0     2 164800.976    0  805 503233.569 164800.976  67.3%     -  425s
     2     4 164888.973    2 1009 503233.569 164800.976  67.3%  4039  437s
     3     5 164800.976    2  889 503233.569 164800.976  67.3%  2861  440s
     5     7 164899.272    3  998 503233.569 164800.976  67.3%  2285  454s
     6     7 164998.929    4 1047 503233.569 164800.976  67.3%  2341  465s
     7    10 164837.100    4 1007 503233.569 164800.976  67.3%  2201  481s
     9    10 165308.611    5 1100 503233.569 164837.100  67.2%  2660  497s
    11    16 164837.100    5 1005 503233.569 164837.100  67.2%  2200  509s
    15    18 165265.546    6 1075 503233.569 164837.100  67.2%  2080  531s
    19    23 164837.100    6 1004 503233.569 164837.100  67.2%  1817  557s
    24    29 164864.772    7 1026 503233.569 164864.772  67.2%  1897  571s
H   28    30                    435540.73217 164864.772  62.1%  1722  874s
H   29    32                    371622.48999 164864.772  55.6%  1663  874s
...
```

MIP – Example 2

- Change MIP strategies (set `ImproveStartTime=3600`)…

```
…
   836   828 170012.416  113 1107 371622.490 164864.772  55.6%   901 3614s

Resetting heuristic parameters to focus on improving solution
(using Heuristics=0.5 and RINS=10)...

   913   905 170302.813  121 1073 371622.490 164864.772  55.6%   837 4349s
   917   906 170306.908  122 1073 371622.490 164864.772  55.6%   834 5120s
H  920   904                     236882.29084 164864.772  30.4%   833 5120s
H  924   908                     235325.82038 164864.772  29.9%   829 5120s
H  935   917                     235325.82033 164864.772  29.9%   823 6310s
H  941   921                     235325.81953 164864.772  29.9%   820 11402s
   945   930 173125.061  126 1055 235325.820 164864.772  29.9%   819 20667s
   949   936 172549.331  125 1077 235325.820 164864.772  29.9%   818 21413s
H  952   936                     223946.40672 164864.772  26.4%   816 21413s
H  956   938                     223050.75538 164864.772  26.1%   813 21413s
   957   942 172391.221  126 1052 223050.755 164864.772  26.1%   812 29280s
   961   940 173719.366  127 1048 223050.755 164864.772  26.1%   811 29963s
  1001   972 172401.160  130 1050 223050.755 164864.772  26.1%   794 30970s
H 1030   994                     218867.25201 164864.772  24.7%   784 30970s
…
```

MIP – Example 3

- Solve progress slows, ‘`top`’ (or Task Manager) shows Gurobi not making good progress…

```
top - 14:27:11 up 22 days, 22:07,  3 users,  load average: 4.15, 4.05, 3.99
Tasks:  73 total,   1 running,  71 sleeping,   1 stopped,   0 zombie
Cpu(s):  5.9%us,  0.5%sy,  0.0%ni, 32.4%id, 61.2%wa,  0.0%hi,  0.0%si,  0.0%st
Mem:   8193828k total,  8144716k used,    49112k free,      284k buffers
Swap: 19800072k total,  3337364k used, 16462708k free,     2108k cached

  PID USER      PR  NI  VIRT  RES  SHR S %CPU %MEM    TIME+  COMMAND
 3414 rothberg  20   0 10.1g 7.4g 1636 D   23 95.3 657:27.82 gurobi_cl
  207 root      15  -5     0    0    0 S    2  0.0   0:21.37 kswapd0
    1 root      20   0  4020  168  168 S    0  0.0   0:01.22 init
    2 root      15  -5     0    0    0 S    0  0.0   0:00.00 kthreadd
    3 root      RT  -5     0    0    0 S    0  0.0   0:00.51 migration/0
    4 root      15  -5     0    0    0 S    0  0.0   0:00.40 ksoftirqd/0
    5 root      RT  -5     0    0    0 S    0  0.0   0:00.13 watchdog/0
    6 root      RT  -5     0    0    0 S    0  0.0   0:00.54 migration/1
```

MIP – Example 3

- Use node files
  - `NodefileStart` parameter

- Performance penalty typically less than 10%

MIP – Example 4

- Progress stalls, not short on memory…

```
 Nodes    |    Current Node    |     Objective Bounds      |     Work
 Expl Unexpl |  Obj  Depth IntInf | Incumbent    BestBd   Gap | It/Node Time

     0     0 3817190.68    0  544          - 3817190.68     -      -    2s
H    0     0                    1.555212e+09 3817190.68   100%     -    2s
H    0     0                    1.540152e+09 3817190.68   100%     -    2s
H    0     0                    1.496837e+09 3817190.68   100%     -    2s
H    0     0                    1.639804e+08 3817190.68  97.7%     -    3s
...
  5468  4240 3830021.31  435   96 8247474.49 3822445.39  53.7%  36.7   73s
H 5724  3251                    7554727.8619 3822445.39  49.4%  35.7   73s
* 5724  3251             484    7554727.8619 3822445.39  49.4%  35.7   73s
H 6526   921                    3830692.9887 3822445.39  0.22%  33.0   74s
  6526   921             535    3830692.9887 3822445.39  0.22%  33.0   74s
...
 22945  6545 3824602.97  208   45 3824990.61 3824448.25  0.01%  27.5  175s
*23183  6450             348    3824966.4316 3824448.25  0.01%  27.3  175s
 27795 10035 3824931.45  147   52 3824966.43 3824448.56  0.01%  25.6  180s
...
3975393 2423369     cutoff  123      3824938.94 3824541.29  0.01%  23.0 7200s
...
```

MIP – Example 4

- Adjust your termination criteria
- Model data often have estimation errors
- Default `MIPGap` (0.01%) is overkill for many models

Parameter pitfalls

- Don’t over-tune parameters
  - Default values are carefully selected, based on thousands of models
  - Avoid setting parameters unless they make a big improvement across multiple test models
  - "Less is more"

- Don’t assume parameters that were effective for another solver are ideal for Gurobi

- If there is a new Gurobi major release try the defaults first

Automated Parameter Tuning

How to use the Parameter Tuning Tool

1) Use the command-line client (`grbtune`)

Example:

```
grbtune LogFile=tune.log TuneOutput=3 TuneTrials=5 TuneTimeLimit=432000
TimeLimit=600 TuneCriterion=2 Method=1 model1.mps model2.mps
```

Use two model files, run for 5 days, show the full log output of each run, test each model with 5 different random seeds, write output to tune.log, run each model for up to 10 minutes, always use dual simplex, tune for the best feasible solution within the time limit.

2) Use the API to tune a specific model

```
m=read('model.mps‘)
m.params.Method=1
m.params.TuneTimeLimit=86400
m.tune()
```

Distributed Tuning

Multiple computers can work together at the same time on the same set of models

- Start `grb_rs` on the worker nodes
- Set the `TuneJobs` and `WorkerPool` parameters on the master node

- Make sure all computers have a similar hardware configuration
- Tuning performance scales linearly with the number of distributed workers

Tuning Tool Output

- Tries multiple parameter combinations, looking for improving set...

```
Testing candidate parameter set 16...

        MIPFocus 2
        VarBranch 1

Solving with random seed #1 ... runtime 3.01s+

Progress so far: baseline runtime 3.91s, best runtime 1.83s
Total elapsed tuning time 99s (18s remaining)
```

- Reports best results found when it finishes…

```
Tested 20 parameter sets in 117.17s

Baseline parameter set: runtime 3.91s

…

Improved parameter set 3 (runtime 2.52s):

        VarBranch 1
```

Don‘t overtune…

Example output:

```
Improved parameter set 1 (MIP gap 2.33% ):

        SimplexPricing 3
        Heuristics 0.001
        MIPFocus 2
        RINS 2500
        VarBranch 0
        CutPasses 5
        PrePasses 1

Improved parameter set 2 (MIP gap 2.54% ):

        SimplexPricing 3
        Heuristics 0.001
        MIPFocus 2
        RINS 500
        VarBranch 0
        PrePasses 1

Improved parameter set 3 (MIP gap 2.70% ):

        SimplexPricing 3
        Heuristics 0.001
        MIPFocus 2
        VarBranch 0
        PrePasses 1

Improved parameter set 4 (MIP gap 3.94% ):

        Heuristics 0.001
        MIPFocus 2
        VarBranch 0
        PrePasses 1

Improved parameter set 5 (MIP gap 4.56% ):

        MIPFocus 2
        VarBranch 0
        PrePasses 1

Improved parameter set 6 (MIP gap 7.33% ):

        PumpPasses 10
        VarBranch 0

Improved parameter set 7 (MIP gap 12.4% ):

        Method 0
```

Tuning Metrics

- The primary tuning criterion is always to minimize the runtime (wall-clock time) required to find a proven optimal solution.
- For MIP models that don't solve to optimality within the specified time limit, a secondary criterion is needed. You can use the `TuneCriterion` parameter for this.

- `TuneCriterion=0` - ignore the secondary criterion and focus entirely on minimizing the time to find a proven optimal solution.
- `TuneCriterion=1` - use the optimality gap as the secondary criterion (current default).
- `TuneCriterion=2` - use the objective of the best feasible solution found.
- `TuneCriterion=3` - use the best objective bound.

Parameter Tuning Tool – Best Practices

- Pay attention to the value of the `TuneTrials` parameter.
  - Never only do a single trial
  - The default value (3) may not be large enough to produce good results
  - Models with high performance variability are likely to require more trials
  - Models that solve within fractions of a second may require more trials

- Try to understand why a certain set of parameters is helpful
  - Gurobi Support can help you!

- Only set parameters that really help performance
  - Look for 10-20% performance gain or more
  - Avoid random results

- Use enough time to explore the parameter space
  - Examine at least a few hundred parameter combinations