Gurobi Days Paris

Advanced Gurobi Algorithms

Roland Wunderling

October 2022

GUROBI OPTIMIZATION

GUROBI OPTIMIZATION

Agenda

- Problem Types
- Presolve
- Algorithms for Continuous Optimization
- Algorithms for Discrete Optimization

GUROBI OPTIMIZATION

Problem Types

Problem types and Algorithms

Presolve

Continuous

- **LP**
  - Simplex
  - Barrier (+ crossover)
- **Convex QP**
  - QP Simplex
  - Barrier
- **Convex QCP**
  - Barrier
- **Non-convex QP**
  - Same as for non-convex MIQCP

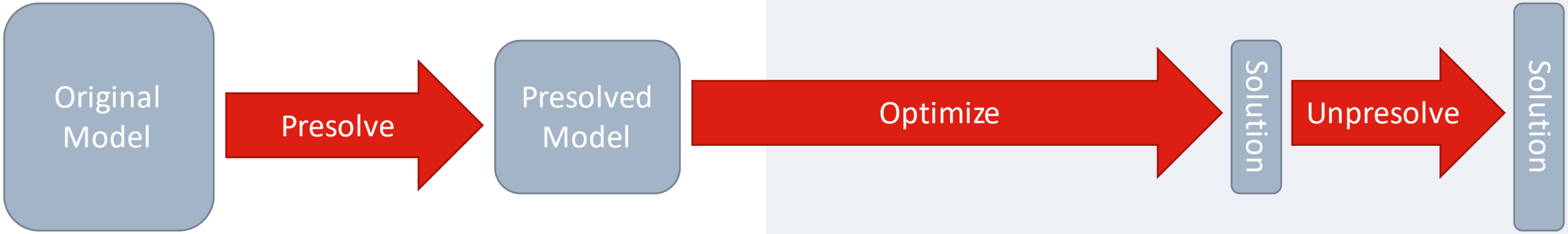
Mixed Integer

- **MILP**
  - Branch-and-Cut
- **MIQP (convex)**
- **MIQCP (convex)**
  - Outer approximation
- **MIQCP (non-convex)**
  - Spatial branching

GUROBI OPTIMIZATION

Presolve

Presolve



- **Purpose of Presolve**
  - Reduce the model size
  - Improve linear algebra during solve
  - Tighten the formulation (for MIP)
  - Identify problem sub-structure

- **Some Presolve Reduction**
  - Bound tightening
  - Aggregation
  - Coefficient strengthening
  - …

GUROBI OPTIMIZATION

Presolve – Coefficient Strengthening

Given a constraint

- $ax \leq b$,
  where $l \leq x \leq u$ and
  some x_j integral

Replace with $a'x \leq b'$, such that

- It is valid for all $x \in X$
- It dominates the original constraint

Example

- Knapsack constraint with binary variables:

$$10b_1 + 5b_2 + 10b_3 + 11b_4 \leq 23$$

- After strengthening:

$$b_1 + b_2 + b_3 + b_4 \leq 2$$

- Is it Valid?
  - Rewrite as
    $$10(b_1 + b_2 + b_3 + b_4) - 5b_2 + b_4 \leq 23$$
  - Infeasible for
    $$b_1 + b_2 + b_3 + b_4 \geq 3$$

- Is it stronger?
  - Consider
    $$(b_1, b_2, b_3, b_4) = (1, \frac{3}{5}, 1, 0)$$

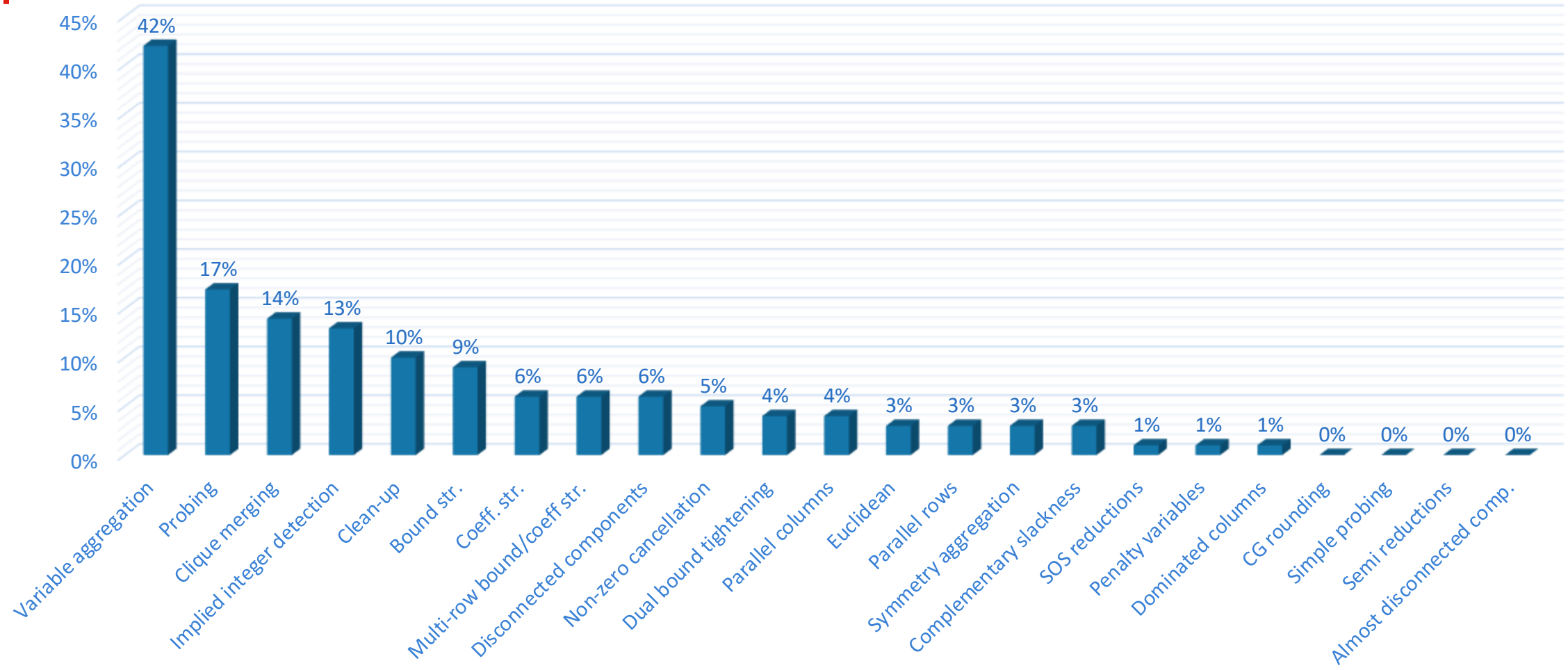
Impact of Presolve

GUROBI OPTIMIZATION

For MIP more powerful than just for LP:

- Exploit integrality
  - Round fractional bounds and right-hand sides
  - Lifting/coefficient strengthening
  - Probing
- Does not need to preserve duality
  - We only need to “uncrush” a primal solution
  - Neither a dual solution nor a basis needs to be “uncrushed”
- Larger work limits
  - For a given problem size solving a MIP takes much more time than solving an LP
  - We can spend more time in presolve

Presolve – Performance Impact on MIP



Time limit: 10000 sec.
Intel Xeon CPU E3-1240 v3 @ 3.40GHz
4 cores, 8 hyper-threads
32 GB RAM

Test set has 3182 models:
- degradation measured individually on >10s bracket: ~1200 models
Benchmark data based on Gurobi 6.5
Results from Achterberg, Bixby, Gu, Rothberg, Weninger (2020): "Presolve Reductions in Mixed Integer Programming"

Presolve Log

```
gfd-schedulen180f7d50m30k18: 457985 rows, 227535 columns, 1233372 nonzeros
Thread count: 8 physical cores, 16 logical processors, using up to 8 threads
Optimize a model with 457985 rows, 227535 columns and 1233372 nonzeros
Model fingerprint: 0x14c90069
Variable types: 33102 continuous, 194433 integer (0 binary)
Coefficient statistics:
  Matrix range     [1e+00, 2e+02]
  Objective range  [1e+00, 1e+00]
  Bounds range     [1e+00, 1e+05]
  RHS range        [1e+00, 3e+02]
Presolve removed 211497 rows and 111352 columns (presolve time = 7s) ...
Presolve removed 227114 rows and 113578 columns
Presolve time: 7.59s
Presolved: 230871 rows, 113957 columns, 661696 nonzeros
Variable types: 0 continuous, 113957 integer (113246 binary)
```

Model reduction: 50%
Solve time reduction:
8 minutes instead of 15 hours and counting

GUROBI OPTIMIZATION

Continuous Optimization

Continuous Algorithms

for LP / QP / QCP

Primal & dual simplex method

- Numerically stable (if you are careful)
- Easy to restart after a model modification
- Does not parallelize well
- Basic solutions are often desirable

Barrier method

- Numerically more challenging
- Restart remains open research question
- Can effectively exploit multiple cores
- Crossover to Simplex Solution

Concurrent optimization

- Run both simplex and barrier simultaneously
- Solution is reported by first one to finish
- Great use of multiple CPU cores
- Best mix of speed and robustness
- Deterministic and non-deterministic versions available

GUROBI OPTIMIZATION

Simplex Algorithms

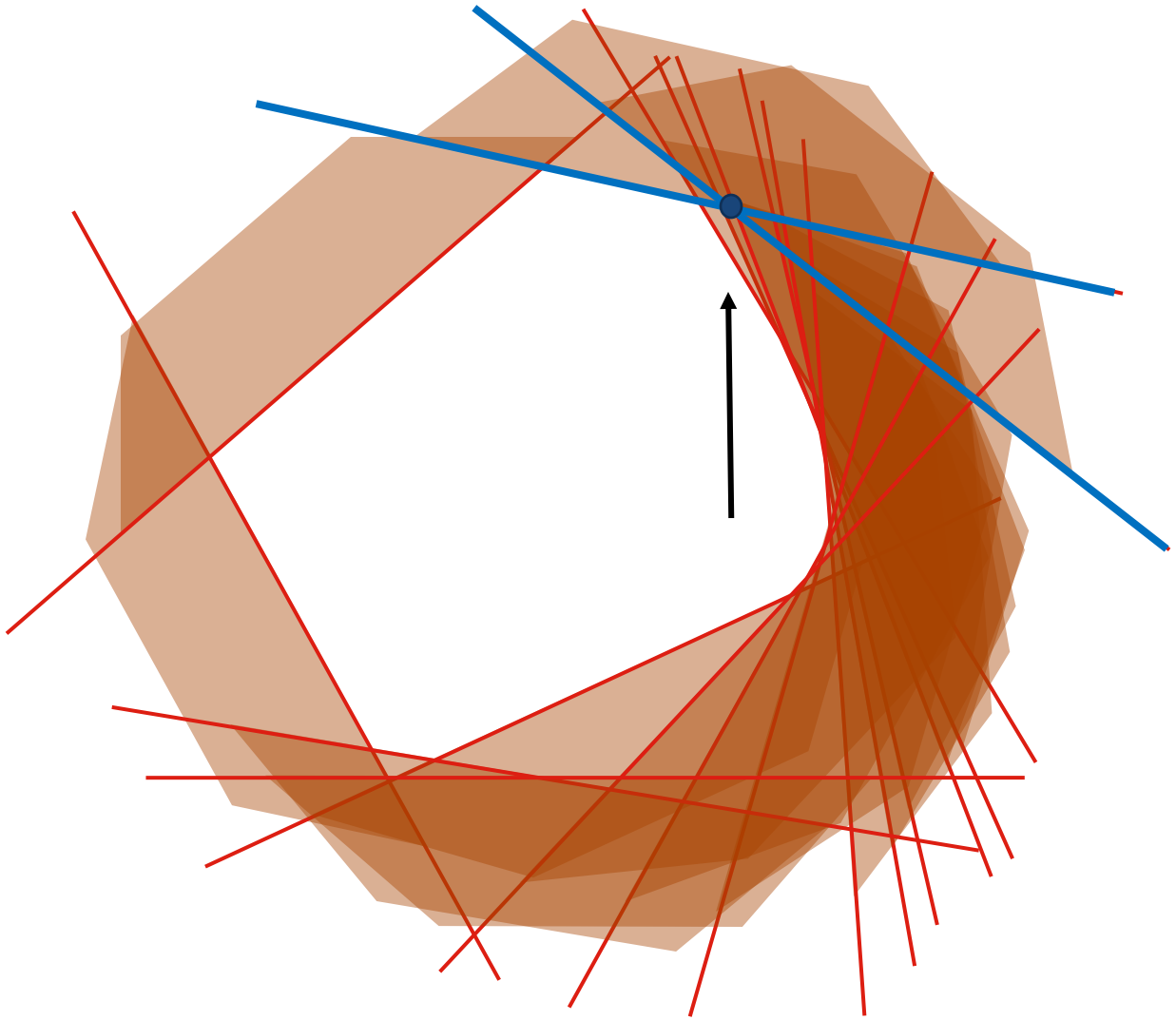
GUROBI OPTIMIZATION

Linear Program

- Problem statement

$$\begin{aligned} \boldsymbol{min} \;\; & \boldsymbol{c'x} \\ \boldsymbol{s.t.} \;\; & \boldsymbol{Ax} = \boldsymbol{b} \\ & \boldsymbol{x} \geq \boldsymbol{0} \end{aligned}$$

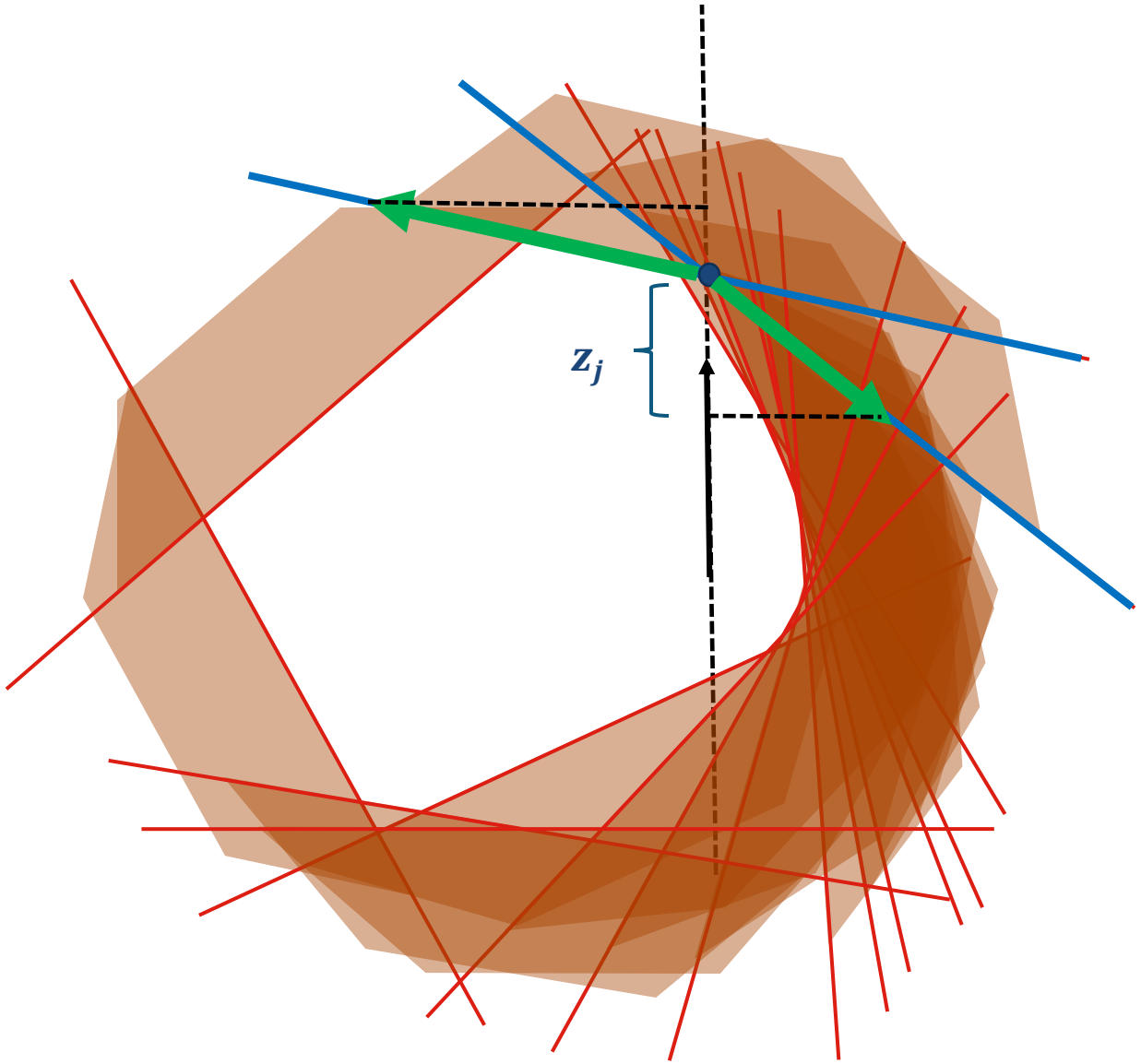
- Optimal solution can be found at a vertex
  - Intersection of n constraints satisfied with equality
  - Pick $\boldsymbol{Ax} = \boldsymbol{b}$ and $\boldsymbol{x_N} = \boldsymbol{0}$, $|\boldsymbol{N}| = \boldsymbol{n} - \boldsymbol{m}$
  - Then $\boldsymbol{x_B} = \boldsymbol{A_B}^{-1}(\boldsymbol{b} - \boldsymbol{N x_N})$, $\boldsymbol{B} = \{\boldsymbol{1}, \ldots, \boldsymbol{n}\} \backslash \boldsymbol{N}$
- Basis
  - Partition: $\{\boldsymbol{1}, \ldots, \boldsymbol{n}\} = \boldsymbol{B} \cup \boldsymbol{N}$, $\boldsymbol{B} \cap \boldsymbol{N} = \emptyset$
  - Such that $\boldsymbol{A_B}$ is non-singular



GUROBI OPTIMIZATION

Linear Program

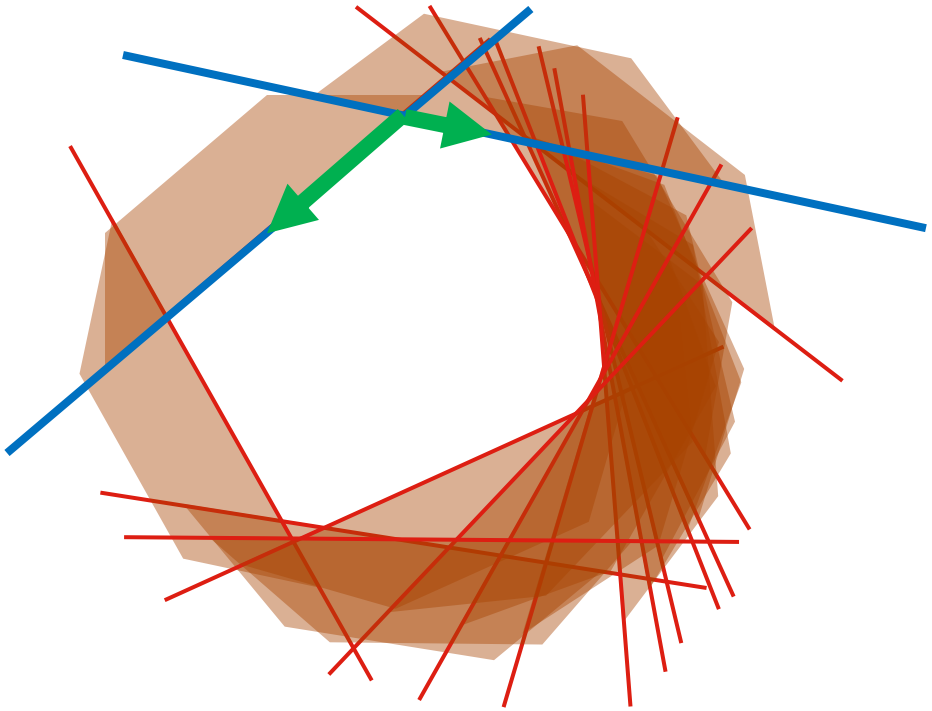
- Primal feasibility
  - All constraints must be satisfied
  - $\boldsymbol{x_B \geq 0}$
- Dual feasiblity (optimality)
  - Consider rays of the recession cone
  - Scalar product with objective function are the reduced cost $\boldsymbol{z_j = c_j - c_B^T A_B^{-1} A_j}$
  - Dual feasible if $z_N \geq \mathbf{0}$



Simplex Algorithm

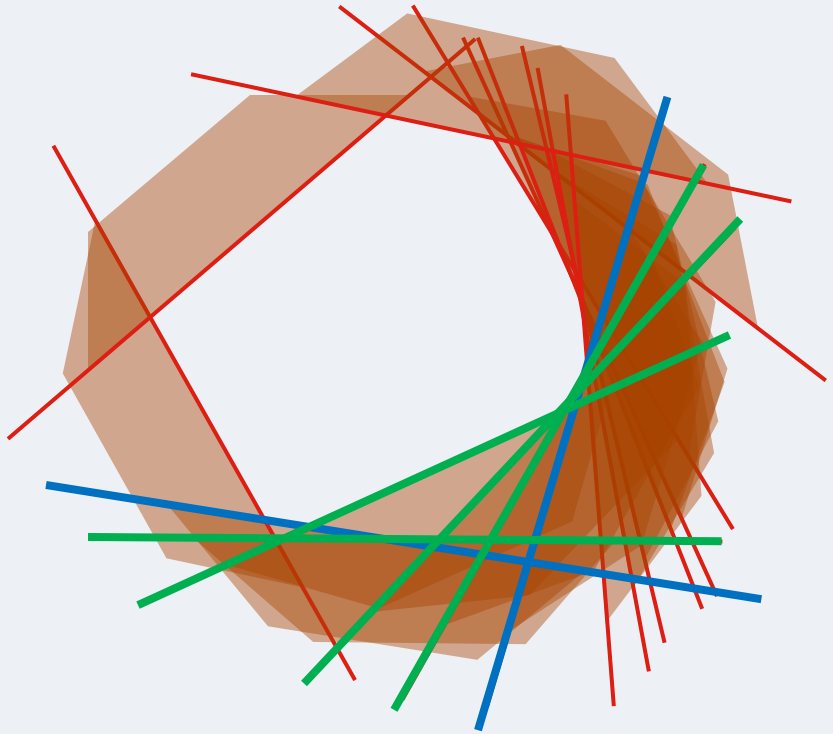
Primal Simplex

- Start with primal feasible basis
- But dual infeasible



Dual Simplex

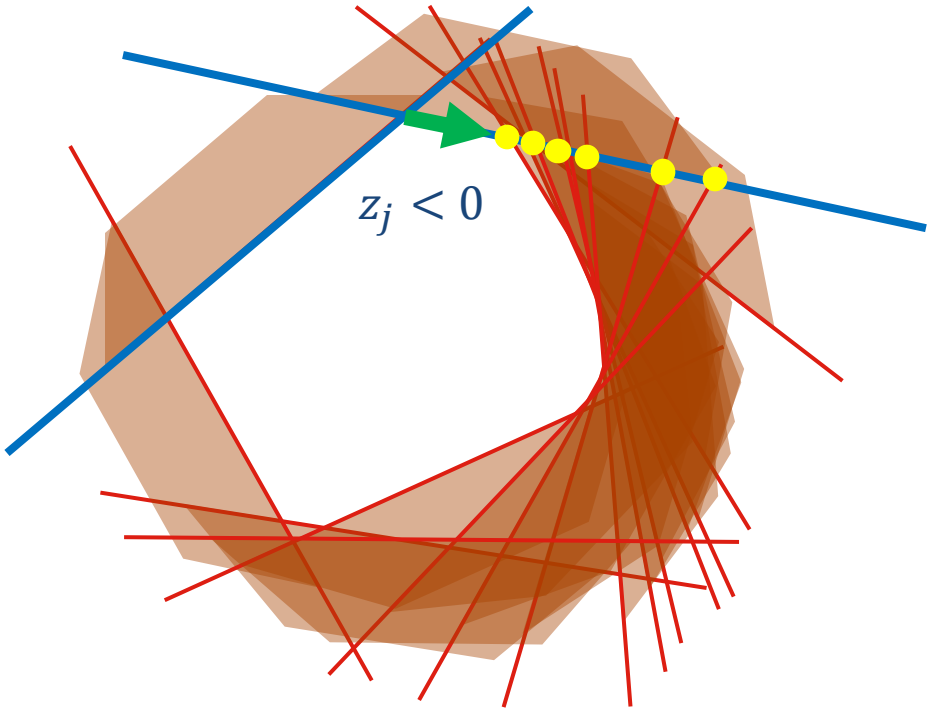
- Start with dual feasible basis
- But primal infeasible



Simplex Algorithm

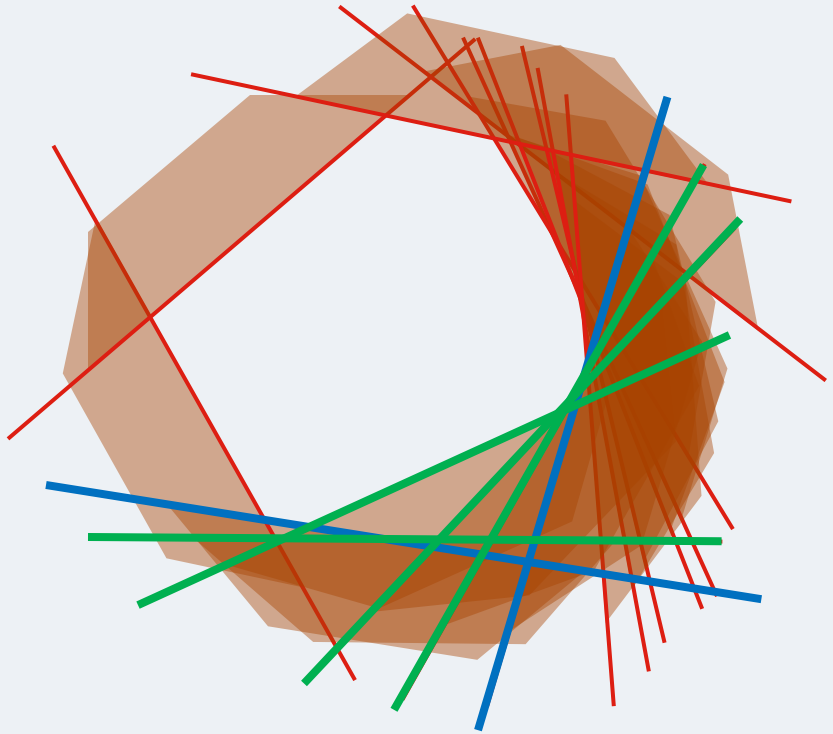
Primal Simplex

- Pricing: Pick an improving direction
- Ratio Test: Pick feasible intersection



Dual Simplex

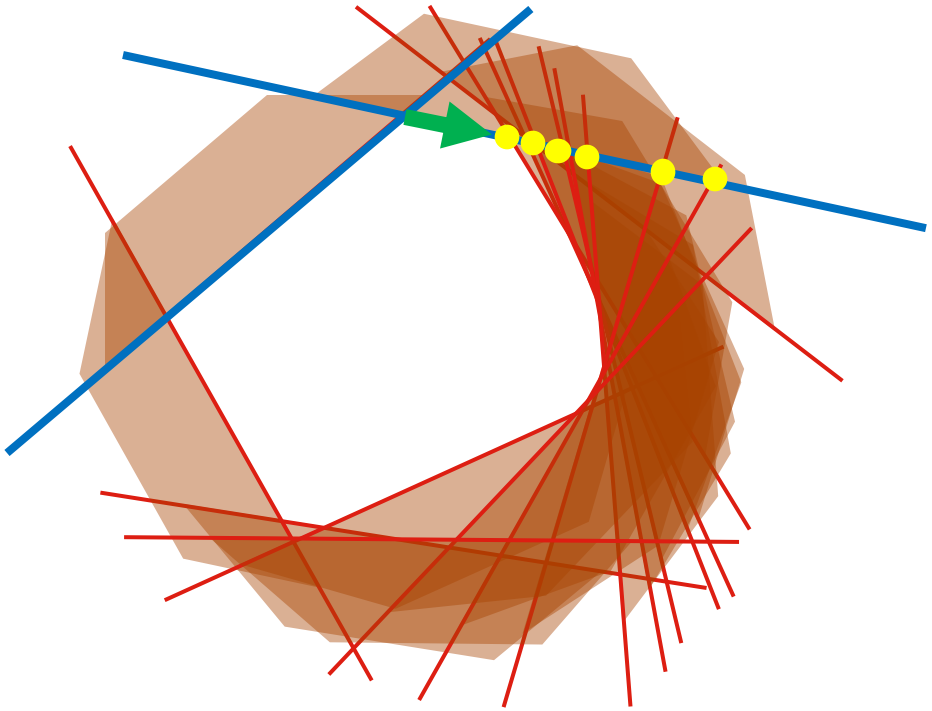
- Start with dual feasible basis
- But primal infeasible



Simplex Algorithm

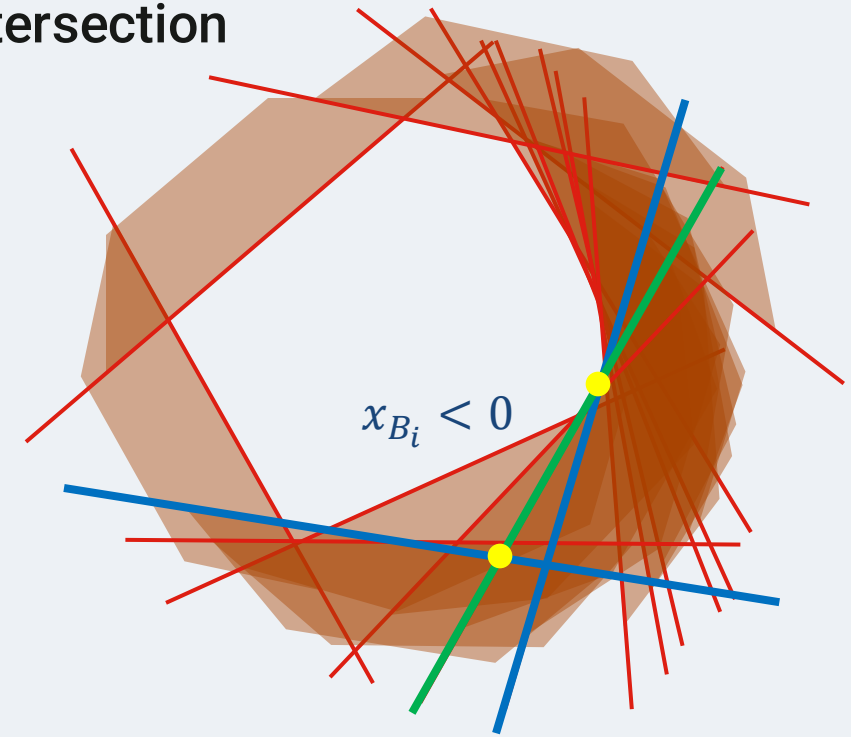
Primal Simplex

- Pricing: Pick an improving direction
- Ratio Test: Pick feasible intersection



Dual Simplex

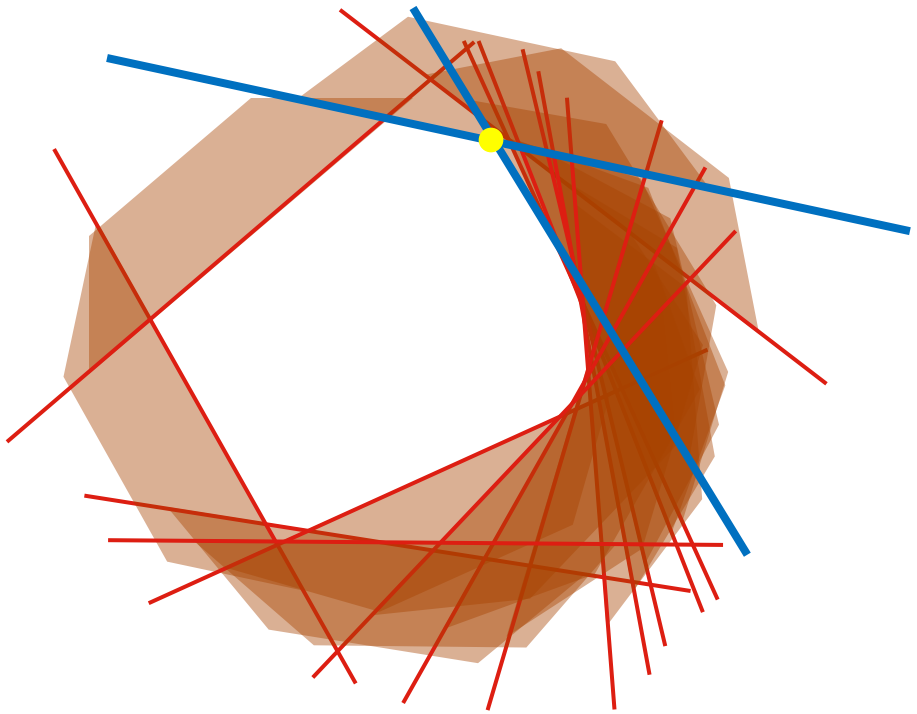
- Pricing: Pick a violated constraint
- Ratio Test: Pick dual feasible intersection



Simplex Algorithm

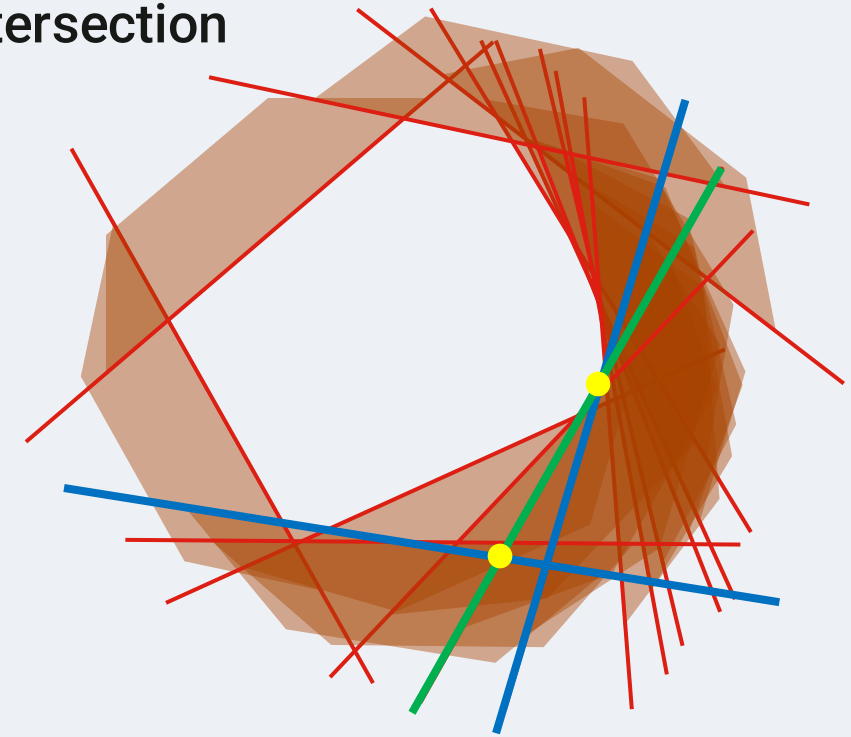
Primal Simplex

- Update Basis



Dual Simplex

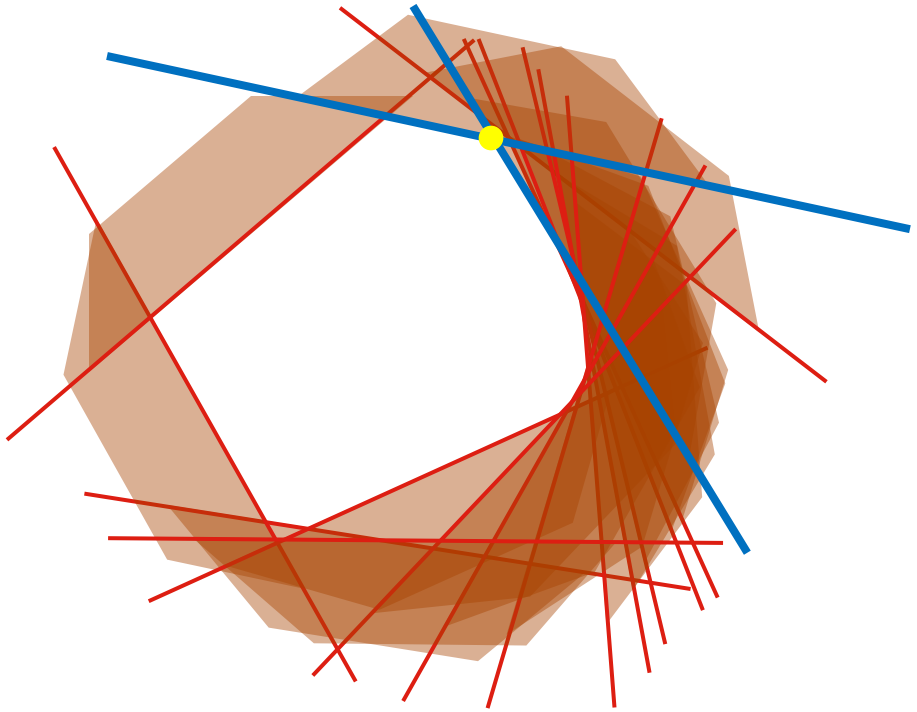
- Pricing: Pick a violated constraint
- Ratio Test: Pick dual feasible intersection



Simplex Algorithm

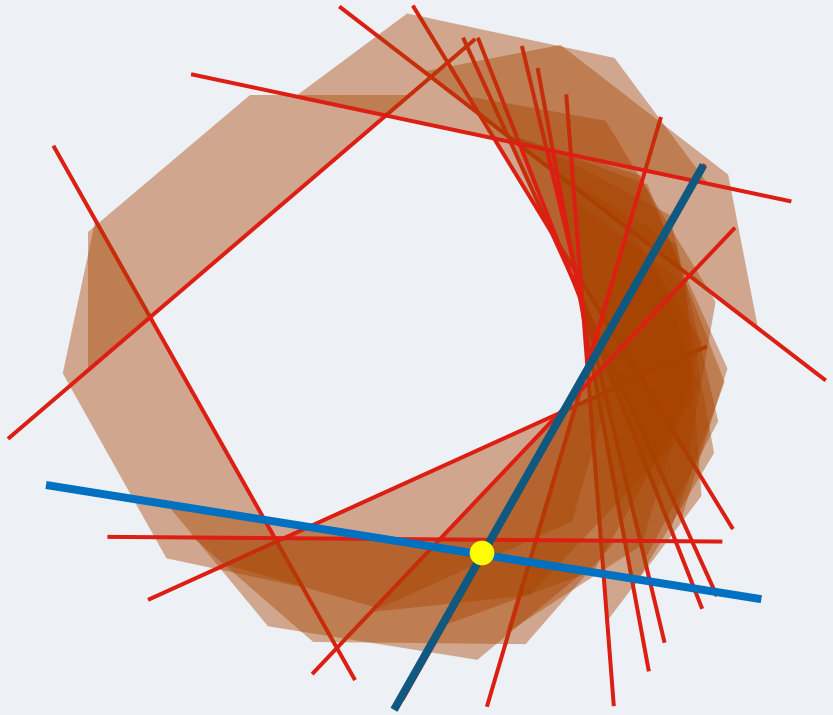
Primal Simplex

- Update Basis



Dual Simplex

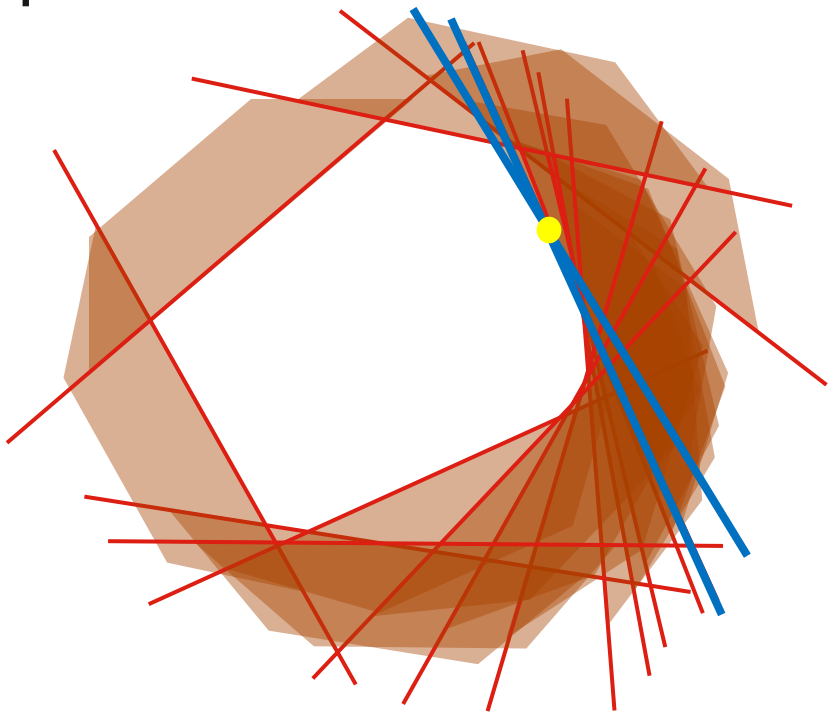
- Update Basis



Simplex Algorithm

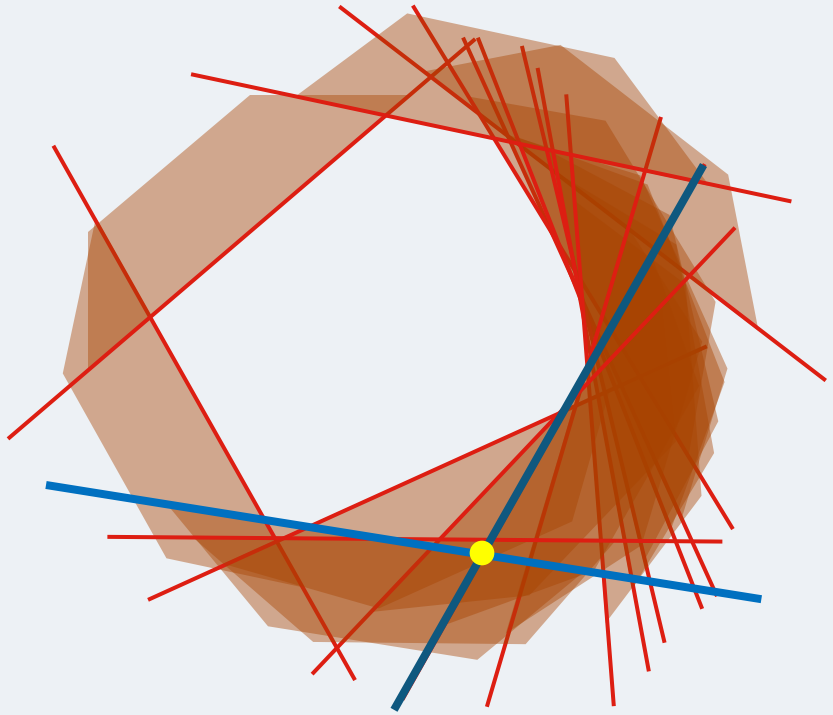
Primal Simplex

- Update Basis
- Repeat



Dual Simplex

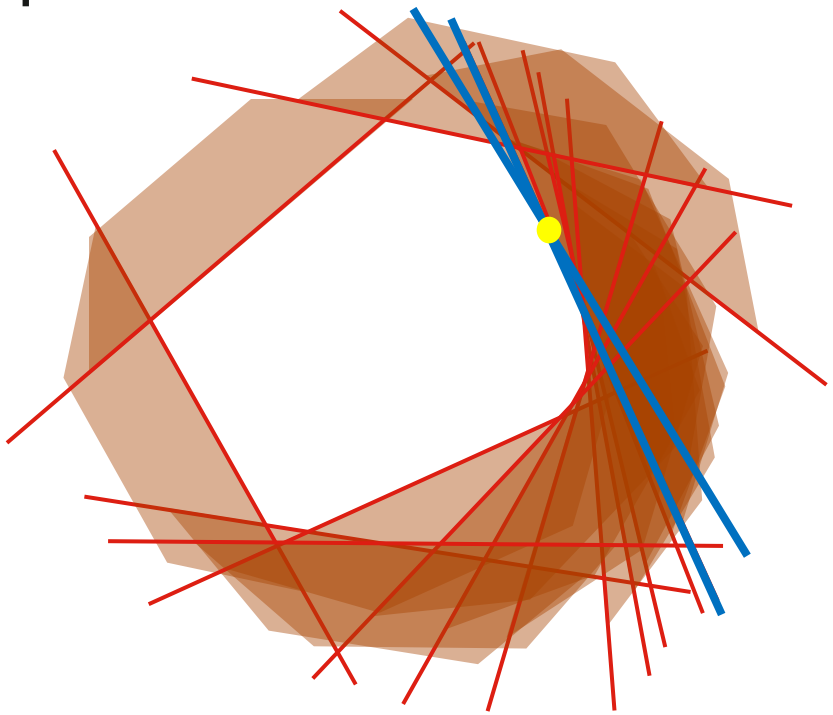
- Update Basis



Simplex Algorithm

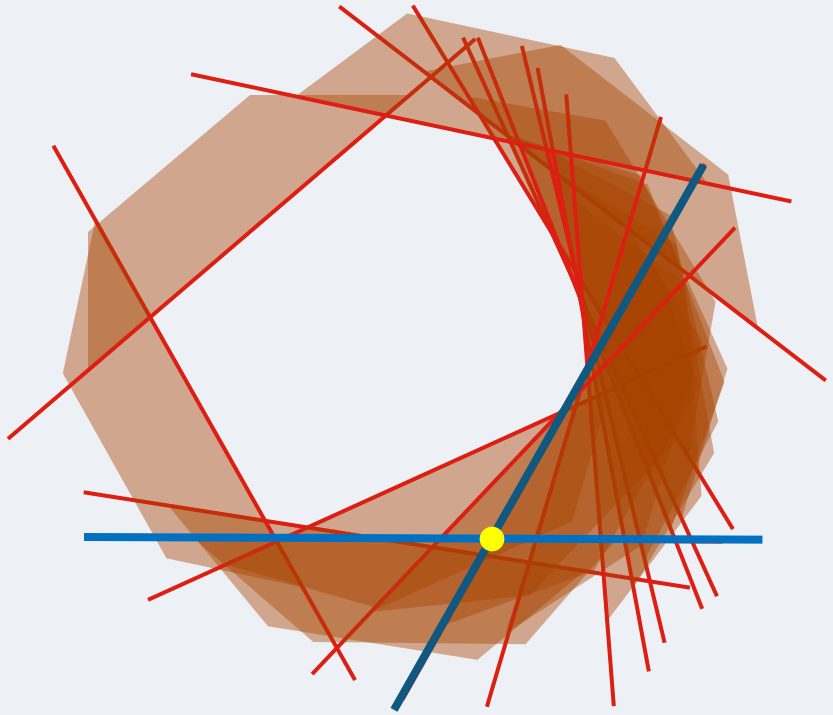
Primal Simplex

- Update Basis
- Repeat



Dual Simplex

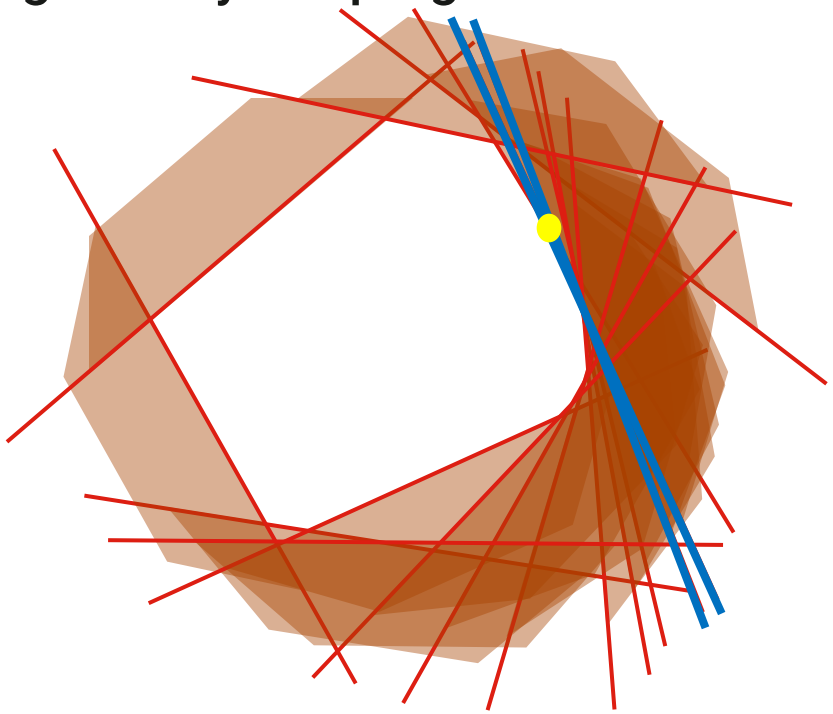
- Update Basis
- Repeat



Simplex Algorithm

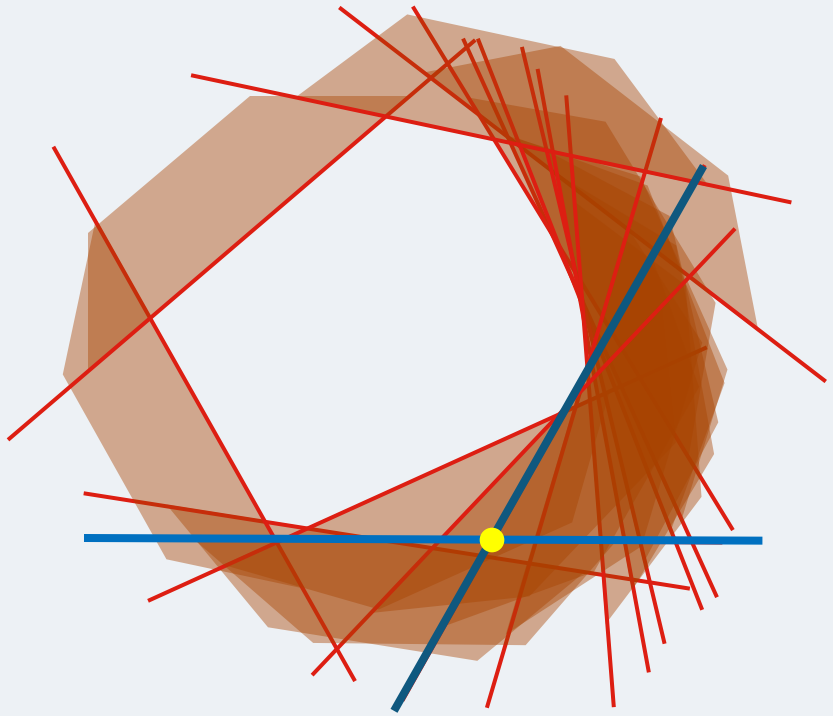
Primal Simplex

- Repeat?
- Degeneracy: No progress



Dual Simplex

- Update Basis
- Repeat



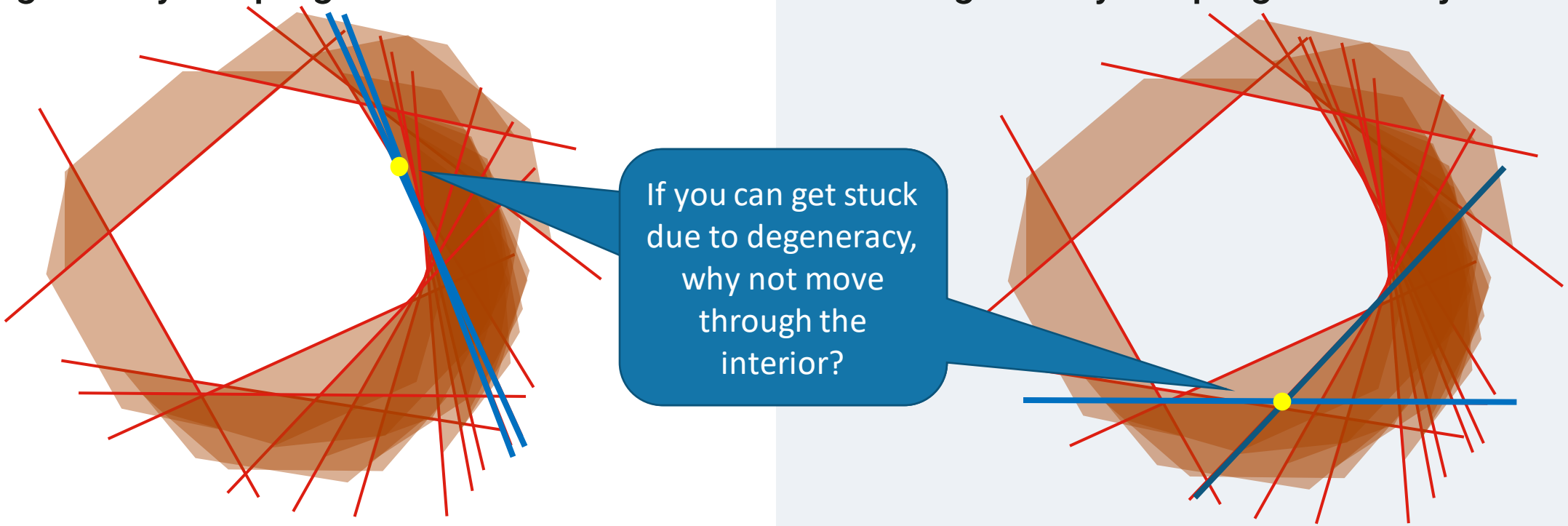
Simplex Algorithm

Primal Simplex

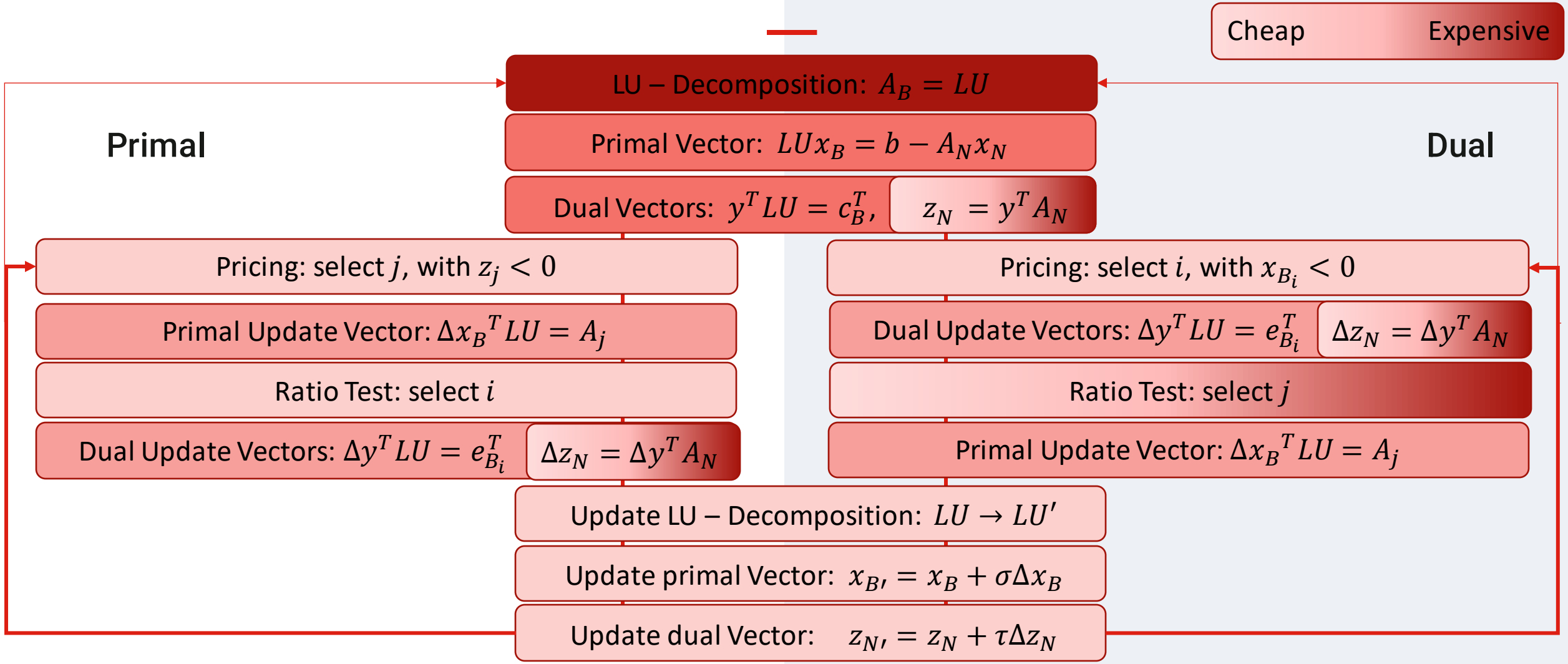
- Repeat?
- Degeneracy: No progress

Dual Simplex

- Repeat?
- Degeneracy: No progress in objective



Computational Steps of the Simplex Algorithms



GUROBI OPTIMIZATION

Simplex Log

- Dual feasible -> Dual Simplex
- Degeneracy is real
  - No more progress in objective
  - Gurobi removes degeneracy by perturbing
  - Gets out of degeneracy and solves the *perturbed* model
  - But needs to solve unperturbed model using primal Simplex
  - Primal also runs into degeneracy and perturbs the problem (but less)
  - Solves primal perturbed model
  - Final basis happens to be primal and dual feasible for unperturbed problem

```
Iteration    Objective       Primal Inf.    Dual Inf.      Time
       0    4.3000000e+01   7.750000e+01   0.000000e+00     37s
[...]
  412980    1.8708334e+03   1.382705e+02   0.000000e+00    516s
  413737    1.8710001e+03   1.087931e+03   0.000000e+00    521s
  414379    1.8710001e+03   1.228713e+02   0.000000e+00    527s
  415021    1.8710001e+03   2.513642e+01   0.000000e+00    530s
  415481    1.8710001e+03   4.105177e+01   0.000000e+00    538s
  415921    1.8710001e+03   1.100249e+02   0.000000e+00    545s
  416261    1.8710001e+03   9.163224e+02   0.000000e+00    553s
  416621    1.8710001e+03   5.824055e+00   0.000000e+00    560s
  416881    1.8710001e+03   5.413714e+00   0.000000e+00    568s
  417121    1.8710001e+03   1.704219e+01   0.000000e+00    577s
  417351    1.8710001e+03   3.007301e+00   0.000000e+00    585s
Perturb objective with value 0.0006 at iteration 417581
  417581    1.8710001e+03   3.461285e-01   0.000000e+00    594s
  417816    1.8710001e+03   0.000000e+00   0.000000e+00    601s
Perturb rhs with value 1e-05
  419742    1.8710000e+03   0.000000e+00   1.960654e-01    608s
  420384    1.8710000e+03   0.000000e+00   2.572093e-01    613s
[...]
  443092    1.8710000e+03   0.000000e+00   2.761620e-01    807s
Perturbation ends
  443913    1.8710000e+03   0.000000e+00   0.000000e+00    812s
Violations(dual): const 0.000000e+00, bound 0.000000e+00, rc 3.254286e-12
Scaled violations: const 0.000000e+00, bound 0.000000e+00, rc 5.206857e-11
```

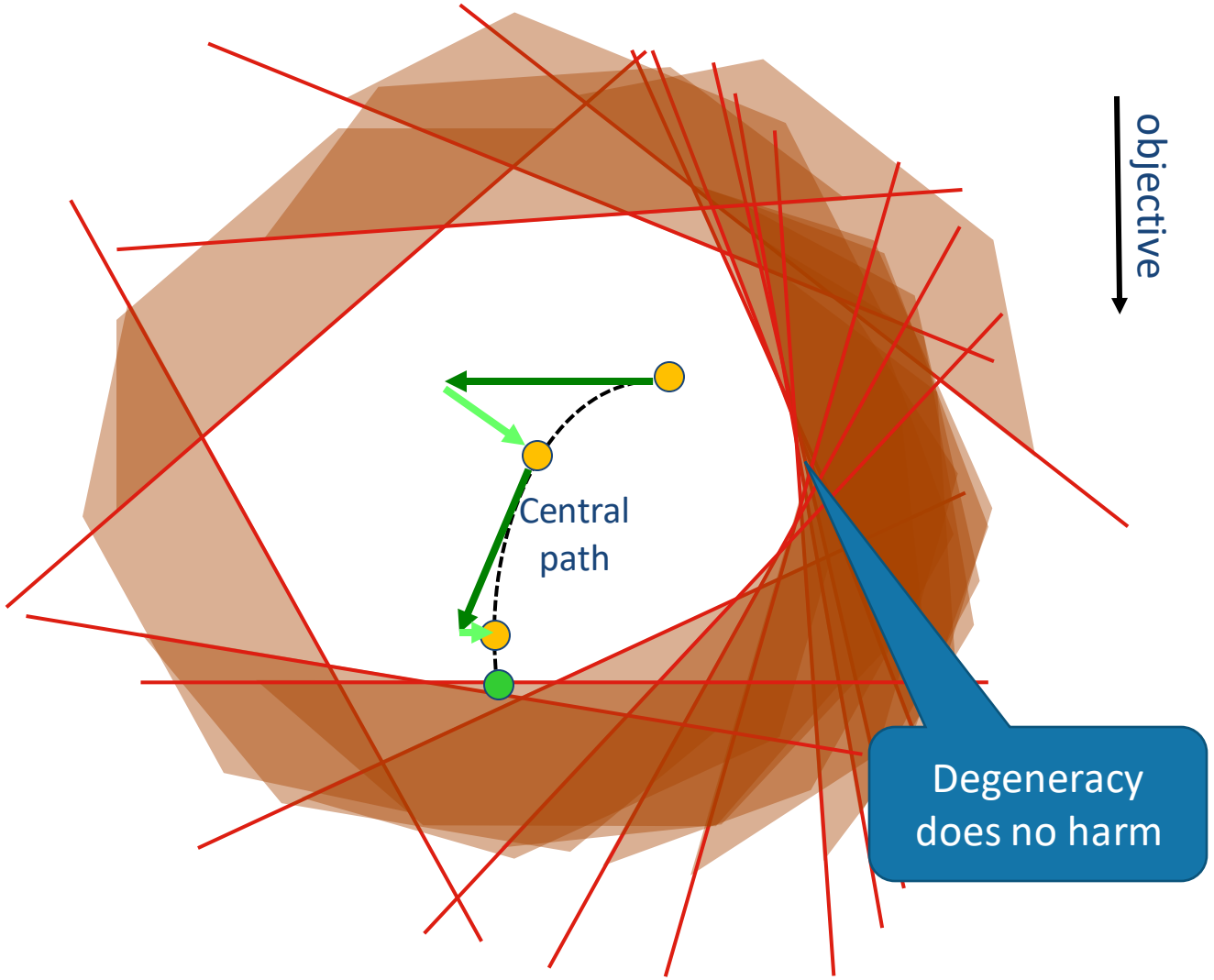

GUROBI OPTIMIZATION

Barrier Algorithm

GUROBI OPTIMIZATION

Barrier Algorithm

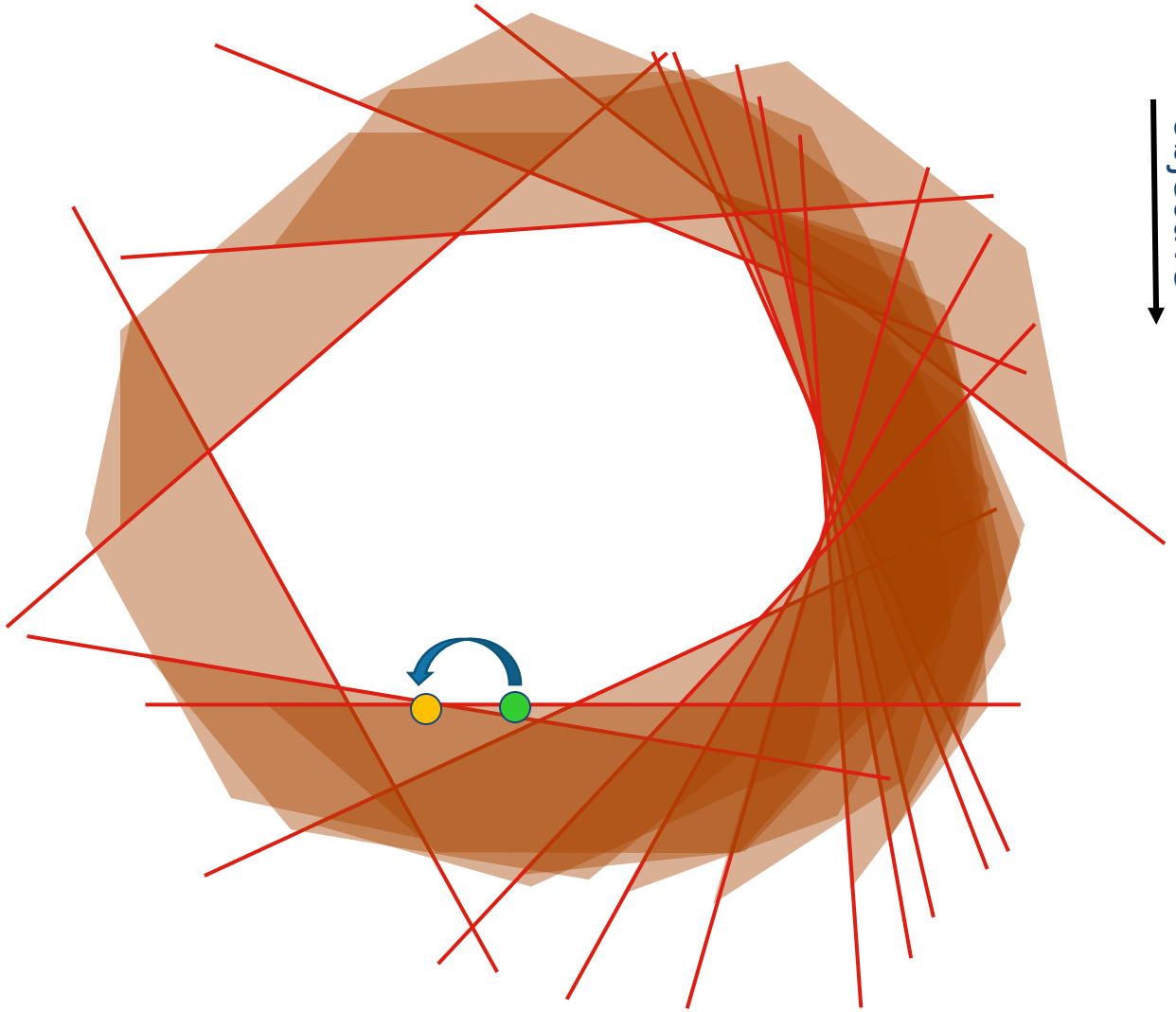
- Start with an interior point
- Move along central path:
  - Predictor: Take a step of a certain size along the tangent direction of the central path
  - Corrector: Move back to central path
  - Iterate until close enough
- Converges to analytic center of optimal face
- How do we know when we are done?



GUROBI OPTIMIZATION

Crossover

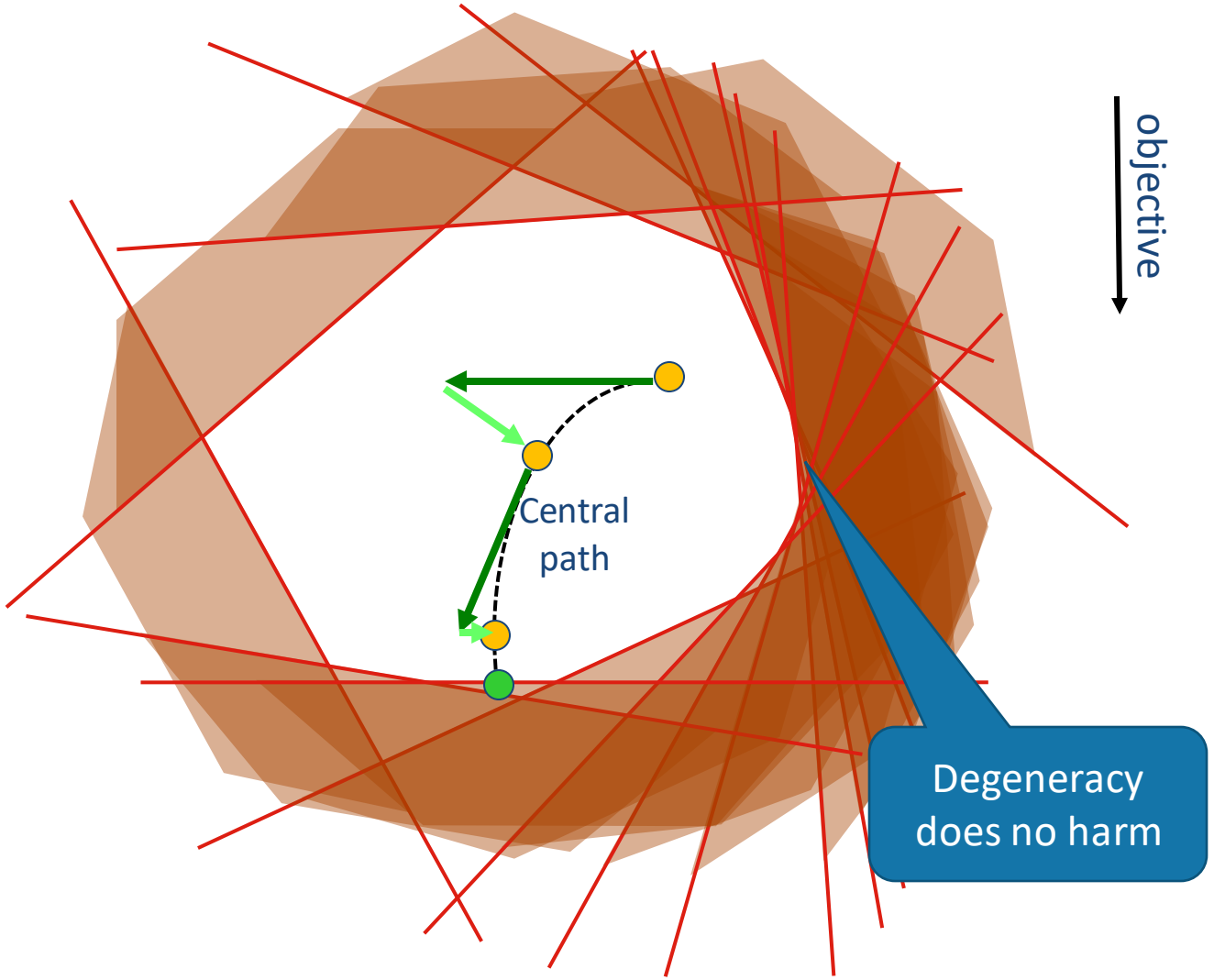
- Basic solutions are often desired since they are much sparser (more variables are at their bounds)
- Crossover:
  - Move from interior point solution to a vertex solution
  - In theory o(n) pivot operations
  - In practice numerical inaccuracies (of barrier solution) may require cleanup with Simplex



GUROBI OPTIMIZATION

Barrier Algorithm

- Dikin's Algorithm:
  apply affine transformation to stay away from the boundary at each iteration
- Karmarkar's Algorithm:
  apply projective transformation to re-center the solution at each iteration
- **Logarithmic Barrier Algorithm:**
  use a logarithmic penalty function on the variable bounds to stay centered



Foundation: Duality

- **Primal Linear Program:**

$$\begin{array}{lrcl} \min & c^T x & & \\ \text{s.t.} & Ax & = & b \\ & x & \geq & 0 \end{array}$$

- **Weighted combination of constraints (y) and bounds (z):**

$$y^T Ax + z^T x \geq y^T b \quad (\text{with } z \geq 0)$$
$$(y^T A + z^T)x \geq y^T b$$

- **Dual Linear Program:**

$$\begin{array}{lrcl} \max & y^T b & & \\ \text{s.t.} & y^T A + z^T & = & c^T \\ & z & \geq & 0 \end{array}$$

Obvious: Weak Duality

$$c^T x^* \geq y^{*T} b$$

(if primal and dual are both feasible)

Strong Duality Theorem:

$$c^T x^* = y^{*T} b$$

(if primal and dual are both feasible)

2 Interpretations of the Barrier Algorithm

#1: Interior Point Method that follows the central path

- Start with linear equations that define the optimality conditions for the primal and dual LPs

$$
\begin{aligned}
Ax &= b, \quad x \geq 0 && \text{(primal feasibility)} \\
A^T y + z &= c, \quad z \geq 0 && \text{(dual feasibility)} \\
c^T x - b^T y = 0 \iff c^T x - (Ax)^T y = 0 &\iff (c^T - y^T A)x \iff z^T x = 0
\end{aligned}
$$

strong duality

complementary slackness

$X = Diag(x);\ Z = Diag(z);\ e = (1, 1, \ldots, 1)$

$$F(x, y, z) = \begin{pmatrix} Ax - b \\ A^T y + z - c \\ XZe \end{pmatrix} = 0$$

Can be solved with Newton's method,
but iterates need not be interior points yet…

2 Interpretations of the Barrier Algorithm

#1: Interior Point Method that follows the central path

- Adjust complementary slackness conditions to consider only interior point solutions

$$F(x,y,z) = \begin{pmatrix} Ax - b \\ A^T y + z - c \\ XZe - \mu e \end{pmatrix} = \begin{pmatrix} 0 \\ 0 \\ 0 \end{pmatrix}$$

$x_j * z_j = \mu > 0,$
$j = 1, \dots, n$

- Start with $\mu > 0$, systematically reduce it to 0 to converge to optimal primal dual pair to of LP
- Now we have an interior point method, but what makes it a barrier method?

2 Interpretations of the Barrier Algorithm

#2: Logarithmic Barrier Algorithm

- Primal Barrier Algorithm

$$\begin{aligned} \min \quad & c^T x \\ s.t. \quad & Ax = b \\ & x \geq 0 \end{aligned}$$

Introduce barrier function to force Interior point

$$\begin{aligned} \min \quad & c^T x - \mu \sum_{j=1}^{n} \log x_j \\ s.t. \quad & Ax = b \end{aligned}$$

= -∞ if $x_j = 0$

Lagrange to create unconstrained optimization

$$\min \quad c^T x - \mu \sum_{j=1}^{n} \log x_j - y^T (Ax - b)$$

- Dual Barrier Algorithm

$$\begin{aligned} \max \quad & b^T y \\ s.t. \quad & A^T y + z = c \\ & z \geq 0 \end{aligned}$$

$$\begin{aligned} \max \quad & b^T y + \mu \sum_{j=1}^{n} \log z_j \\ s.t. \quad & A^T y + z = c \end{aligned}$$

$$\max \quad b^T y + \mu \sum_{j=1}^{n} \log z_j - x^T (A^T y + z - c)$$

- In both cases, differentiate the unconstrained optimization and apply Newton's method

2 Interpretations of the Barrier Algorithm

GUROBI OPTIMIZATION

#2: Logarithmic Barrier Algorithm

- Primal Dual Barrier Algorithm
  - Optimality conditions for $\min\ c^T x - \mu \sum_{j=1}^{n} \log x_j - y^T(Ax - b)$:

$$
\begin{aligned}
c - y^T A - \mu X^{-1} &= 0 && \nabla_x && (1) \Leftrightarrow (3+5)\\
Ax - b &= 0 && \nabla_y && (2)
\end{aligned}
$$

  - Optimality conditions for $\max\ b^T y + \mu \sum_{j=1}^{n} \log z_j - x^T(A^T y + z - c)$:

$$
\begin{aligned}
c - y^T A - z &= 0 && \nabla_x && (3)\\
x^T A^T - b^T &= 0 && \nabla_y && (4) \Leftrightarrow (2)\\
\mu Z^{-1} - X &= 0 && \nabla_z && (5)
\end{aligned}
$$

  - These are duals of each other. From either one can derive (e.g. multiply (5) by Z)

Look familiar?

$$F(x,y,z) = \begin{pmatrix} Ax - b \\ A^T y + z - c \\ XZe - \mu e \end{pmatrix} = \begin{pmatrix} 0 \\ 0 \\ 0 \end{pmatrix}$$

Does not matter how we got here; we can use Newton's Method

Applying Newton's Method

Newton's method:

$$x_{k+1} = x_k - \underbrace{[\mathrm{J}(x_k)]^{-1} f(x_k)}_{\Delta x}, \text{ where } J(x)_{ij} = \frac{\partial f_i}{\partial x_j \partial x_i}(x)$$

Apply to:

$$F(x,y,z) = \begin{pmatrix} Ax - b \\ A^T y + z - c \\ XZe - \mu e \end{pmatrix} = \begin{pmatrix} 0 \\ 0 \\ 0 \end{pmatrix}$$

$$\mathrm{J}(x,y,z) = \begin{pmatrix} A & 0 & 0 \\ 0 & A^T & I \\ Z & 0 & X \end{pmatrix}$$
$$\nabla_x \quad \nabla_y \quad \nabla_z$$

Starting point $(x_0, y_0, z_0) > 0$, $F(x_0, y_0, z_0) = \begin{pmatrix} Ax_0 - b \\ A^T y_0 + z_0 - c \\ X_0 Z_0 e - \mu e \end{pmatrix}$

At each iteration solve $\underbrace{\begin{pmatrix} A & 0 & 0 \\ 0 & A^T & I \\ Z_0 & 0 & X_0 \end{pmatrix}}_{\mathrm{J}(x_0)} \underbrace{\begin{pmatrix} \Delta_x \\ \Delta_y \\ \Delta_z \end{pmatrix}}_{\Delta x} = \underbrace{\begin{pmatrix} Ax_0 - b \\ A^T y_0 + z_0 - c \\ X_0 Z_0 e - \mu e \end{pmatrix}}_{f(x_0)}$

Applying Newton's Method

At each iteration solve $\begin{pmatrix} A & 0 & 0 \\ 0 & A^T & I \\ Z_0 & 0 & X_0 \end{pmatrix} \begin{pmatrix} \Delta_x \\ \Delta_y \\ \Delta_z \end{pmatrix} = \begin{pmatrix} Ax_0 - b \\ A^T y_0 + z_0 - c \\ X_0 Z_0 e - \mu e \end{pmatrix}$

Transform: $\begin{pmatrix} A & 0 & 0 \\ 0 & A^T & I \\ Z_0 & 0 & X_0 \end{pmatrix}$ → $\begin{pmatrix} 0 & 0 & -AZ_0^{-1}X_0 \\ 0 & A^T & I \\ Z_0 & 0 & X_0 \end{pmatrix}$ → $\begin{pmatrix} 0 & -AZ_0^{-1}X_0A^T & 0 \\ 0 & A^T & I \\ Z_0 & 0 & X_0 \end{pmatrix}$

Most of the work

At each iteration:

1. Solve $\boldsymbol{AZ_0^{-1}X_0A^T}\,\Delta_y = b - Z_0^{-1}\mu e - AZ_0^{-1}X_0(A^T y_0 + z_0 - c)$
2. Compute $\Delta_z = A^T y_0 + z_0 - c - A^T \Delta_y$
3. Compute $\Delta_x = Z_0^{-1}(X_0 Z_0 e - \mu e - X_0 \Delta_z)$
4. Update $x_1 = x_0 - \Delta_x,\ y_1 = x_0 - \Delta_y,\ z_1 = z_0 - \Delta_z$; reduce μ

Everything else cheap:
- Matrix vector mult.
- Vector additions
- Vector subtractions

Termination of the Barrier Algorithm

- Recall

$$\begin{aligned} Ax &= b, \quad x \geq 0 && \text{(primal feasibility)} \\ A^T y + z &= c, \quad z \geq 0 && \text{(dual feasibility)} \\ c^T x - b^T y &= 0 \iff z^T x = 0 && \text{(duality gap} \iff \text{complementary slackness)} \end{aligned}$$

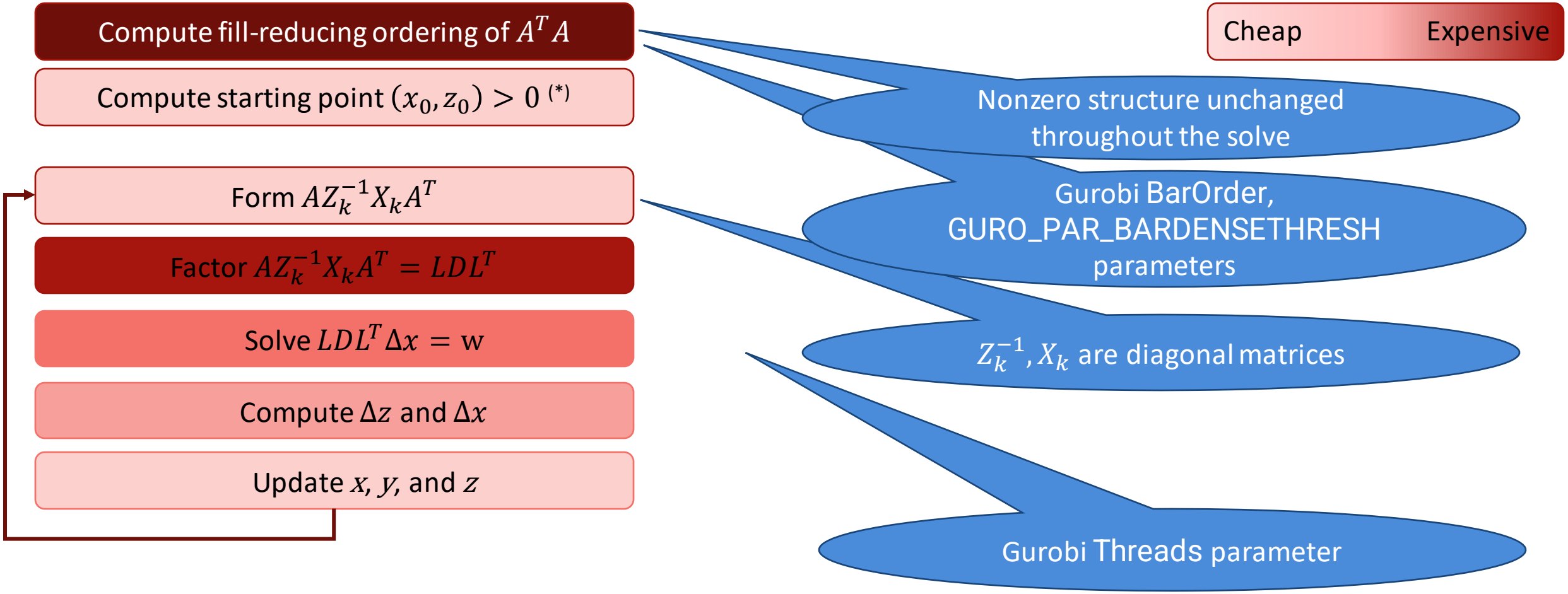
- At each iteration
  - Duality gap can be shown to reduce
  - Albeit neither primal nor dual objective needs change monotonicaly

- Terminate when normalized duality gap and complementary slackness within tolerance:
  - $\frac{\|Ax-b\|}{\|x\|} < \varepsilon \quad \text{and} \quad \frac{\|A^T y+z-c\|}{\|y\|} < \varepsilon$
  - $\frac{|c^T x-b^T y|}{|1+b^T y|} < \varepsilon \quad \text{and} \quad \frac{|z^T x|}{|1+b^T y|} < \varepsilon$

Gurobi BarConvTol parameter, default 1e-8

Qualitatively different than Simplex tolerances

Computational Steps of the Barrier Algorithm



* Lustig, I.J. (1990). "Feasibility issues in a primal-dual interior point method for linear programming," Mathematical Programming, 49(2), 145-162

GUROBI OPTIMIZATION

Barrier Log

- Statistics about most expensive operation
- Termination when complementarity is small enough

- Continue with crossover
- Finish up with Simplex

```
Ordering time: 0.00s

Barrier statistics:
 Dense cols : 1
 AA' NZ     : 1.056e+03
 Factor NZ  : 4.200e+03 (roughly 1 MB of memory)
 Factor Ops : 8.603e+04 (less than 1 second per iteration)
 Threads    : 1
```

	Objective		Residual			
	$c^T x$	$b^T y$	$\|\|Ax-b\|\|$	$\|\|A^T y+z-c\|\|$	$\|z^T x\|$	
Iter	Primal	Dual	Primal	Dual	Compl	Time
0	-2.88769591e+03	-0.00000000e+00	5.74e+03	0.00e+00	2.93e+01	0s
1	-2.93201530e+03	-2.32719862e+02	6.17e+02	2.33e-02	7.21e+00	0s
2	-1.20511641e+03	-2.09346815e+02	0.00e+00	2.08e-17	7.34e-01	0s
3	-3.12189924e+02	-2.58701130e+02	0.00e+00	5.55e-17	3.94e-02	0s
4	-3.00241585e+02	-2.99695549e+02	0.00e+00	5.55e-17	4.02e-04	0s
5	-3.00000246e+02	-2.99999695e+02	0.00e+00	2.08e-17	4.06e-07	0s
6	-3.00000000e+02	-3.00000000e+02	0.00e+00	1.14e-16	4.06e-10	0s
7	**-3.00000000e+02**	**-3.00000000e+02**	**7.11e-15**	**1.44e-15**	**4.06e-16**	**0s**

```
Barrier solved model in 7 iterations and 0.36 seconds (0.83 work units)
Optimal objective -3.00000000e+02

Crossover log...

      24 DPushes remaining with DInf 0.0000000e+00                 0s
       0 DPushes remaining with DInf 0.0000000e+00                 0s

       1 PPushes remaining with PInf 0.0000000e+00                 0s
       0 PPushes remaining with PInf 0.0000000e+00                 0s

  Push phase complete: Pinf 0.0000000e+00, Dinf 0.0000000e+00      0s

Iteration    Objective       Primal Inf.    Dual Inf.      Time
      28   -3.0000000e+02   0.000000e+00   0.000000e+00      0s
```


Essential Differences – Simplex vs Barrier

Simplex	Barrier
• Thousand/millions of iterations on extremely sparse matrices • Each iteration extremely cheap	• Dozens of iterations on denser matrices • Each iteration is expensive

• Few opportunities to exploit parallelism
  • Nonzero structure of matrices changes with every iteration
  • Computational effort spread out over several procedures, none of which typically dominate the work in an iteration
	• Multiple opportunities to exploit parallelism
	  • Nonzero structure of matrices is unchanged
	  • Computational effort focused in two or three procedures that dominate the overall run time

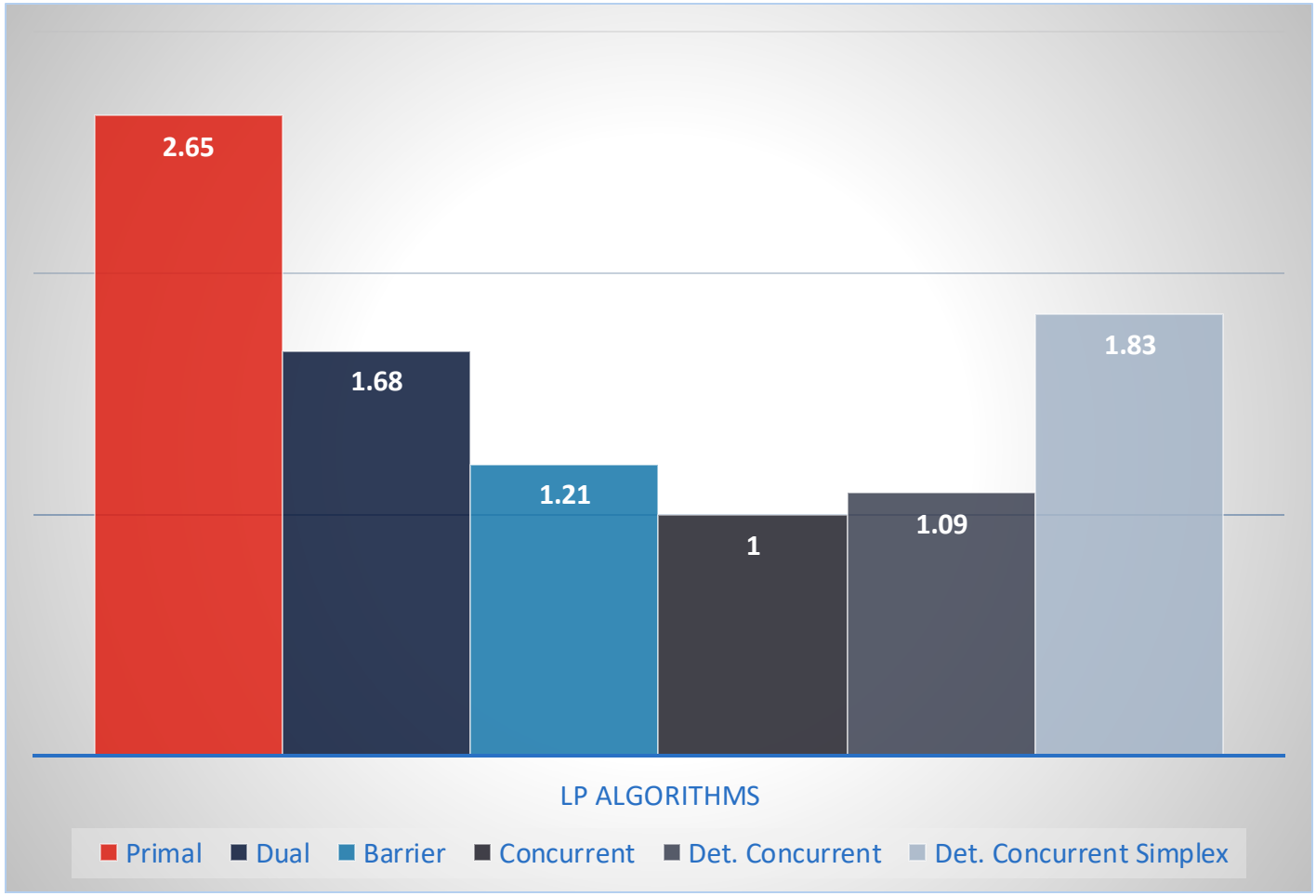
• Can be warm-started • Effectively handles problem modifications	• Barrier warm-start still an open research topic
• Primal or Dual degeneracy can be problematic	• No issues with degenerate extreme points

GUROBI OPTIMIZATION

LP Performance

Performance results:

- Gurobi 9.5, Intel(R) Xeon (R) E3-1240 v5 (4 core at 3.5GHz)
- Simplex on 1 core, Barrier on 4 cores
- Concurrent with 1 thread dual, 3 threads barrier
- Result for models that take >1s



GUROBI OPTIMIZATION

Improving Default Performance

How can I use my newfound knowledge about Gurobi's algorithmic features to get better LP performance?

- Typically, defaults are well suited
  - Uses concurrent LP if problem large enough
  - Using 8-16 threads is often the sweet spot for performance, but you may want to play with that
- If your models are reliably best solved by one of the concurrent algorithms only use that algorithm
  - Avoid contention memory bus
  - Avoid synchronization (for deterministic case)
- Mostly performance issues are due to degeneracy or numerical difficulties – consider your model!
- For problems with special structure:
  - Sifting when cols >> rows: Parameter **`Sifting=1,2`**
  - Network Algorithm (new in 10.0.0): Parameter **`NetworkAlg=1`**
  - Gurobi heuristically will chose those by default

© 2022 Gurobi Optimization, LLC. Confidential, All Rights Reserved | 43

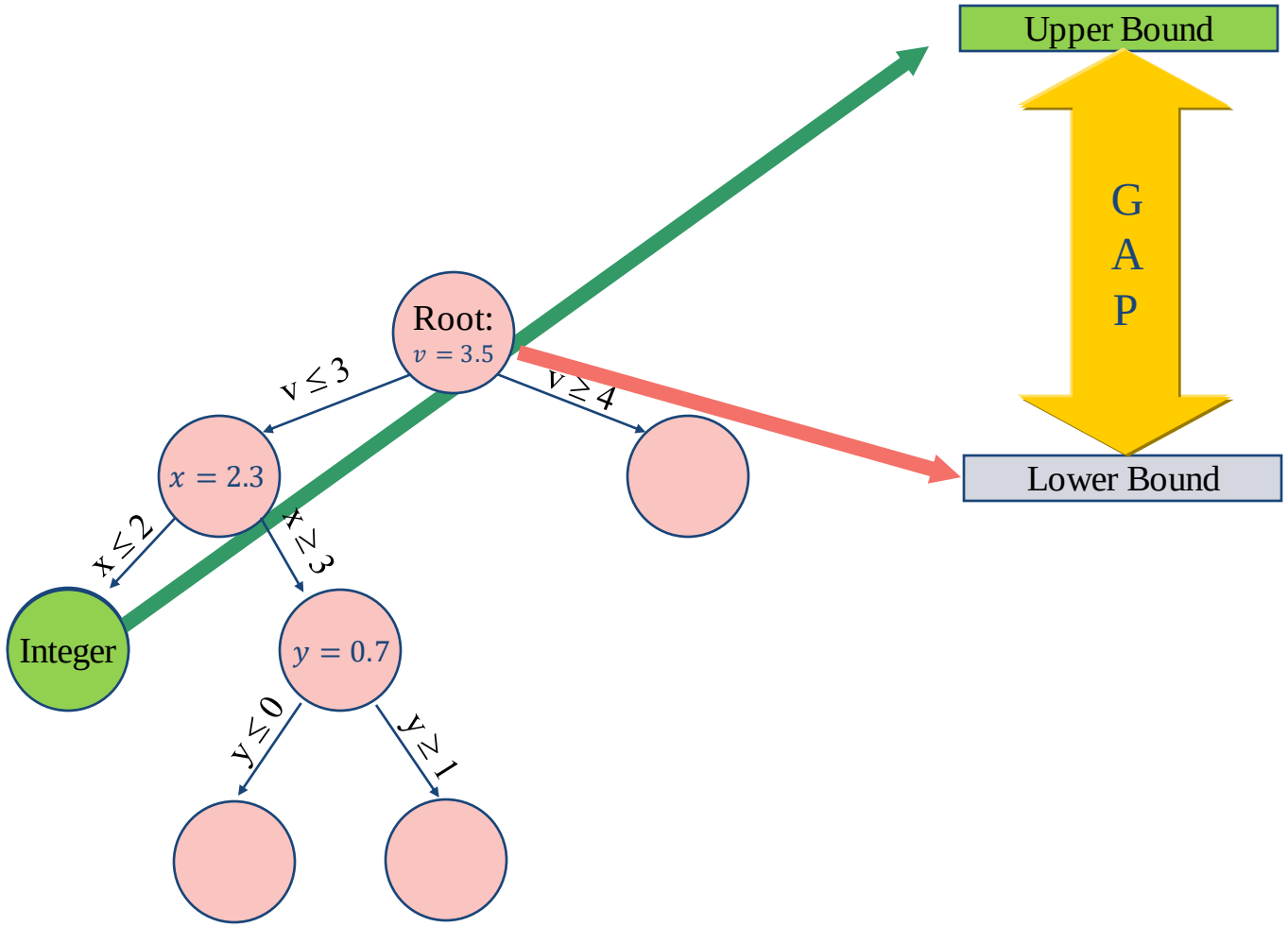
GUROBI OPTIMIZATION

Discrete Optimization

GUROBI OPTIMIZATION

LP based Branch-and-Bound

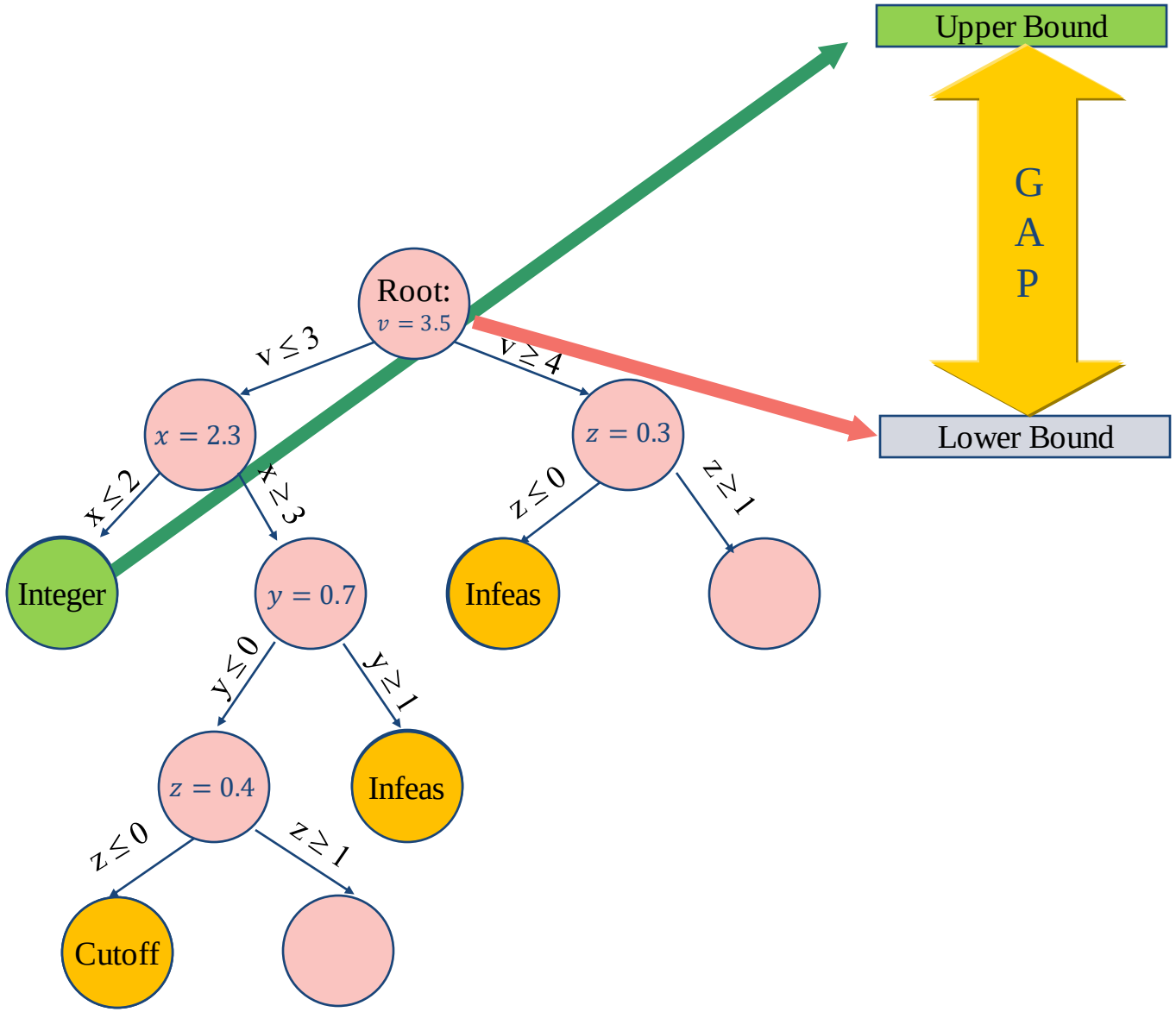
- Solve LP relaxation of the problem (root)
  - Gives rise to first lower bound
  - $v = 3.5$ (fractional)
  - Branch on v: $v \leq 3$ and $v \geq 4$
- Solve child node:
  - $x = 2.3$
  - Branch on x: $x \leq 2$ and $x \geq 3$
- Solve child node:
  - Solution is integer feasible
  - Gives rise to first upper bound
  - Gives rise to optimality gap
- Solve child node:
  - $y = 1.7$
  - Branch on y: $y \leq 0$ and $y \geq 1$



GUROBI OPTIMIZATION

LP based Branch-and-Bound

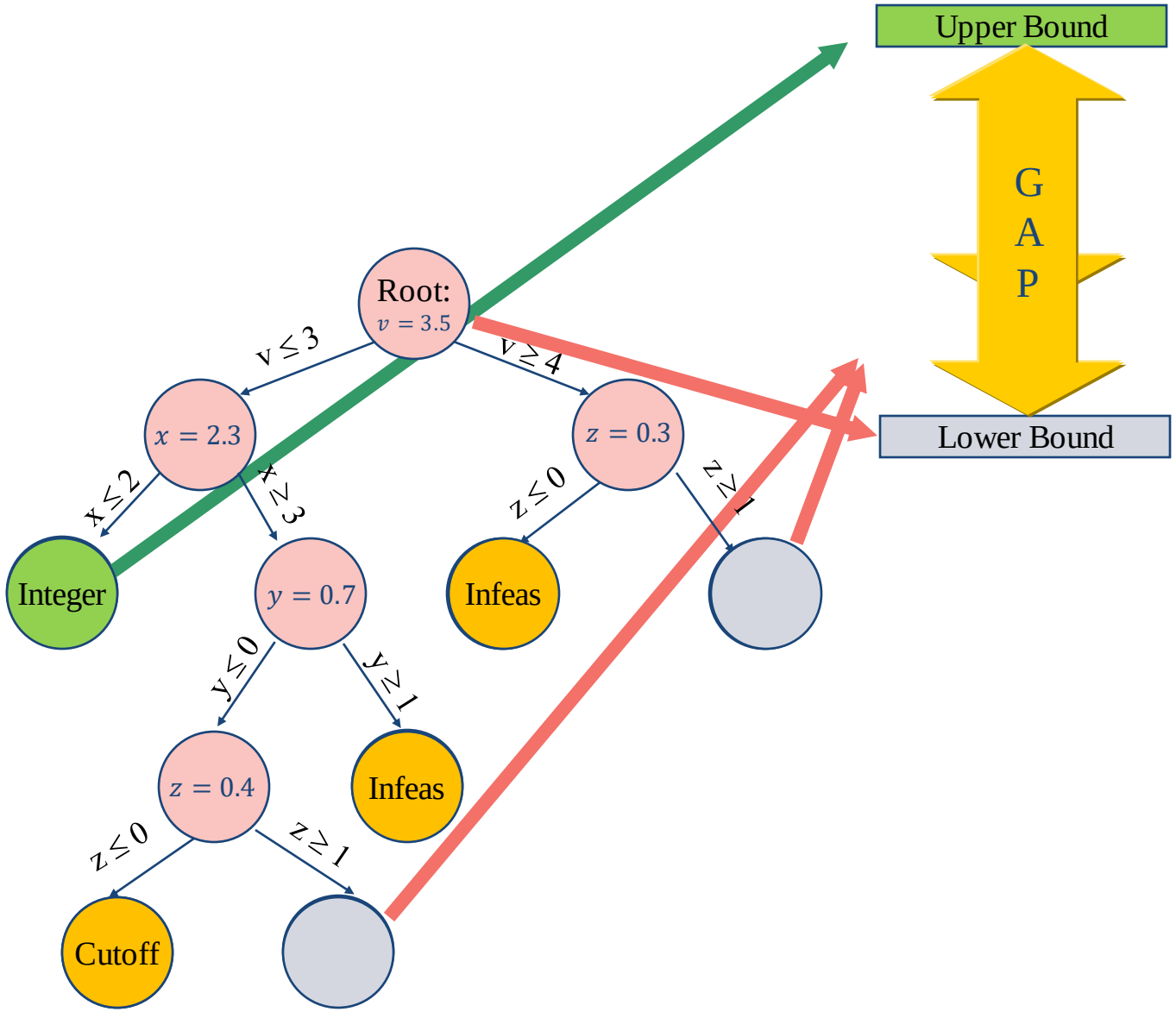
- Solve child node:
  - Node is infeasible
- Solve child node:
  - $z = 0.4$
  - Branch on z: $z \leq 0$ and $z \geq 1$
- Solve child node:
  - Cutoff the node:
    Node objective exceeds current upper bound
- Solve child node:
  - $z = 0.3$
  - Branch on z: $z \leq 0$ and $z \geq 1$
- Solve child node:
  - Node is infeasible



GUROBI OPTIMIZATION

LP based Branch-and-Bound

- Solve more nodes:
  - Minimum of the objective all active node relaxations gives rise to new lower bound
- When gap reaches 0 current incumbent solution is proven to be optimal
  - In practice, „good“ solutions are good enough
  - Since proving optimality may take a long time



MIP Building Blocks

Presolve ← Presolve, PrePasses, AggFill, Aggregate, DualReductions, PreSparsify, …
- Tighten formulation and reduce problem size

Node selection ← BranchDir, Var Hints
- Select next subproblem to process

Node presolve ← Symmetry
- Additional presolve for subproblem

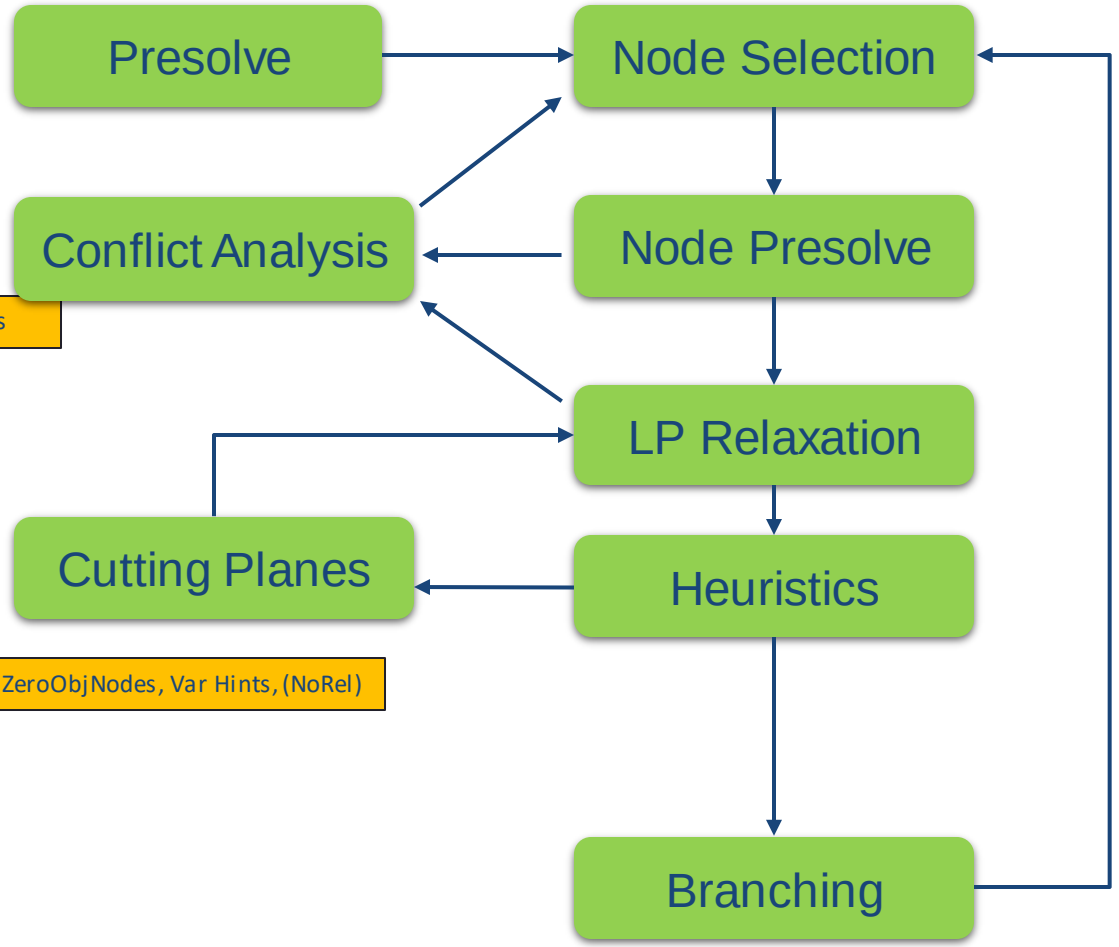
Solve continuous relaxations ← Method, NodeMethod, DegenMoves
- Gives a bound on the optimal integral objective

Conflict analysis
- Learn from infeasible subproblems

Cutting planes ← Cuts, CutPasses, GomoryPasses, CliqueCuts, …
- Tighten relaxation by cutting relaxation solutions

Primal heuristics ← Heuristics, MinRelNodes, PumpPasses, RINS, SubMIPNodes, ZeroObjNodes, Var Hints, (NoRel)
- Find integer feasible solutions

Branching variable selection ← VarBranch, BranchDir
- Crucial for limiting search tree size



MIP Building Blocks

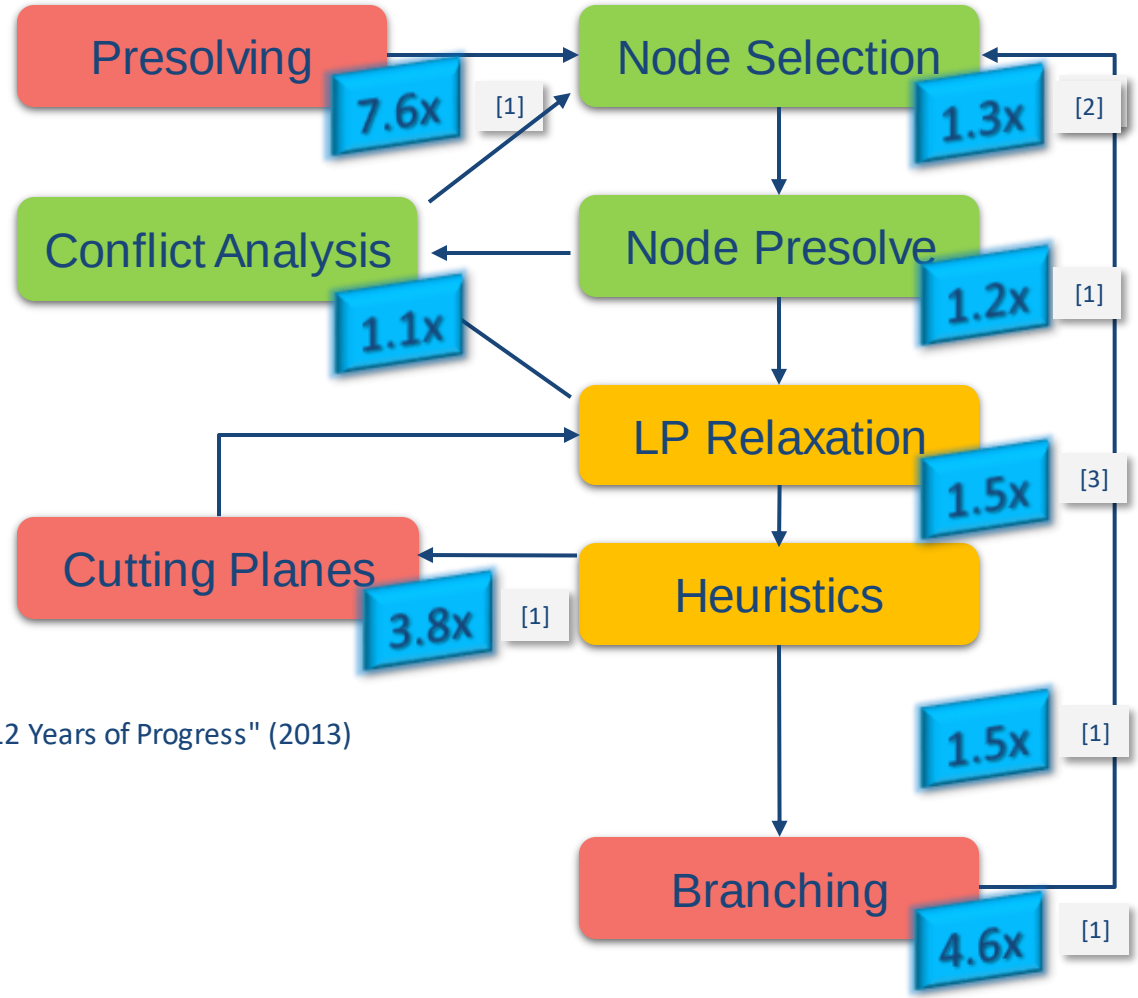
GUROBI OPTIMIZATION

Each box represents a giant bag of tricks

- To cover everything would take weeks

A sampling of techniques instead

- One from each of the most important boxes



[1] Achterberg and Wunderling: "Mixed Integer Programming: Analyzing 12 Years of Progress" (2013)

[2] Achterberg: "Constraint Integer Programming" (2007)

[3] http://plato.asu.edu/ftp/milpc.html

MIP Building Blocks

Presolve
- Tighten formulation and reduce problem size

Node selection
- Select next subproblem to process

Node presolve
- Additional presolve for subproblem

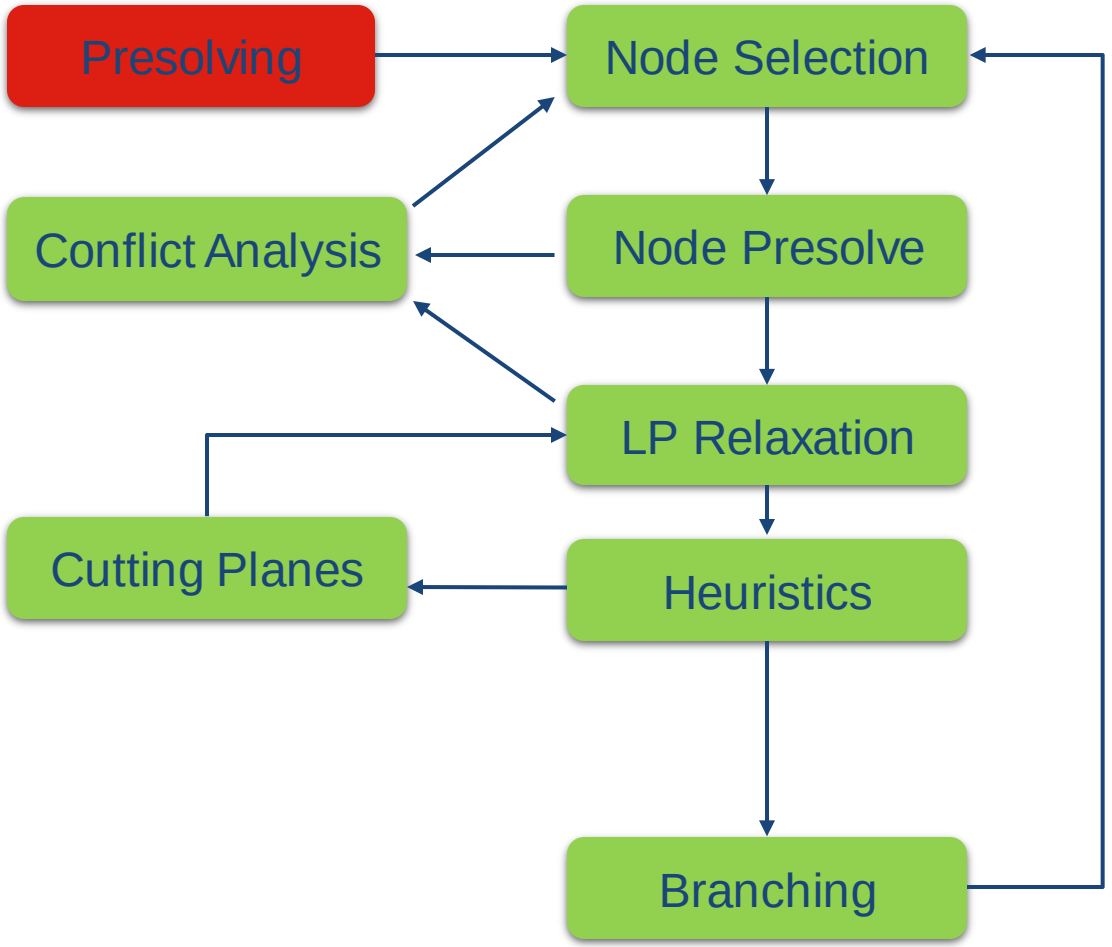
Solve continuous relaxations
- Gives a bound on the optimal integral objective

Conflict analysis
- Learn from infeasible subproblems

Cutting planes
- Cut off relaxation solutions

Primal heuristics
- Find integer feasible solutions

Branching variable selection
- Crucial for limiting search tree size



MIP Building Blocks

Presolve
- Tighten formulation and reduce problem size

Node selection
- Select next subproblem to process

Node presolve
- Additional presolve for subproblem

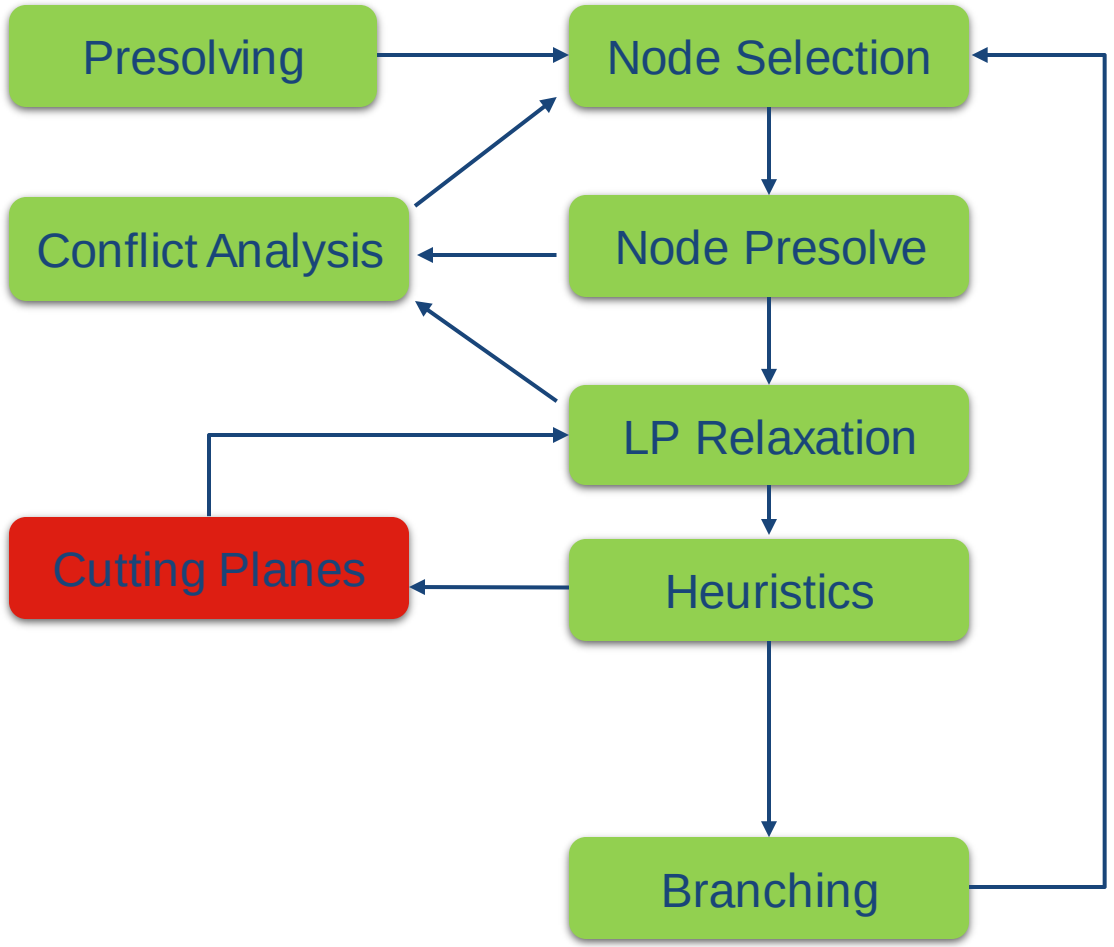
Solve continuous relaxations
- Gives a bound on the optimal integral objective

Conflict analysis
- Learn from infeasible subproblems

Cutting planes
- Cut off relaxation solutions

Primal heuristics
- Find integer feasible solutions

Branching variable selection
- Crucial for limiting search tree size



GUROBI OPTIMIZATION

Cutting Planes

- A cut (cutting plane) is a constraint that reduces the feasible region of the continuous relaxation but not its integer hull
- Separation of cuts:
  Given an x that is feasible for the relaxation, find a cut for which x is infeasible and add it to the relaxation
- Thus, the relaxation more closely approximates the integer hull

The cut menu

Gomory
Mixed Integer Rounding (MIR)
StrongCG cuts
Lift and Project
Infeasibility cuts
Flow cover
Flow path
Network
MIP separation cuts
Relax and Lift
User Cuts

Cover
Implied bound
Projected Implied bound
Clique
GUB Cover
Zero-half
Mod-K
RLT
BQP
SubMIP
Outer Approximation

Chvatal-Gomory Cuts

Given $A \in \mathbb{Q}^{m \times n}, b \in \mathbb{Q}^m$, consider the rational polyhedron

$$P = \{x \in \mathbb{R}^n | Ax \leq b, x \geq 0\}$$

We want to find the integer hull

$$P_I = \text{conv}\{x \in \mathbb{Z}^n | Ax \leq b, x \geq 0\}$$

Chvatal-Gomory procedure:

- **Choose non-negative multipliers $\boldsymbol{\lambda \in \mathbb{R}^m_{\geq 0}}$**
- Aggregated inequality $\lambda^T Ax \leq \lambda^T b$ is valid for P because $\lambda \geq 0$
- Relaxed inequality $\lfloor \lambda^T A \rfloor x \leq \lambda^T b$ is still valid for P because $x \geq 0$
- Rounded Inequality $\lfloor \lambda^T A \rfloor x \leq \lfloor \lambda^T b \rfloor$ is still valid for P_I because $x \in \mathbb{Z}^n$

CG procedure suffices to generate all non-dominated valid inequalities for P_I in a finite number of iterations!

- $P^{(0)} = P$, $P^{(k)} = P^{(k-1)} \cap \{\text{CG cuts for } P^{(k-1)}\}$: k-th CG closure of P - is a polyhedron!
- CG rank of a valid inequality for P_I: minimum k s.t. inequality is valid for $P^{(k)}$
- Higher rank cuts get more and more dense and numerically unstable

Knapsack Cover Cuts

A (binary) knapsack is a constraint $ax \leq b$ with

- $a_i \geq 0$ the weight of item i, $i = 1, \dots, n$
- $b \geq 0$ the capacity of the knapsack

An index set $C \subseteq \{1, \dots, n\}$ is called a cover, if $\sum_{i \in C} a_i > b$

- You can't fit all items from C in the knapsack

A cover C implies a cover inequality: $\sum_{i \in C} x_i \leq |C| - 1$

- You must leave out at least one of them

Interesting for cuts: minimal covers

- $\sum_{i \in C} a_i > b$ and $\sum_{i \in C'} a_i \leq b$ for all $C' \subset C, C' \neq C$

Knapsack Cover Cuts – Example

Consider knapsack $3x_1 + 5x_2 + 8x_3 + 10x_4 + 17x_5 \leq 24$, $x \in \{0,1\}^5$

A minimal cover is $C = \{1,2,3,4\}$

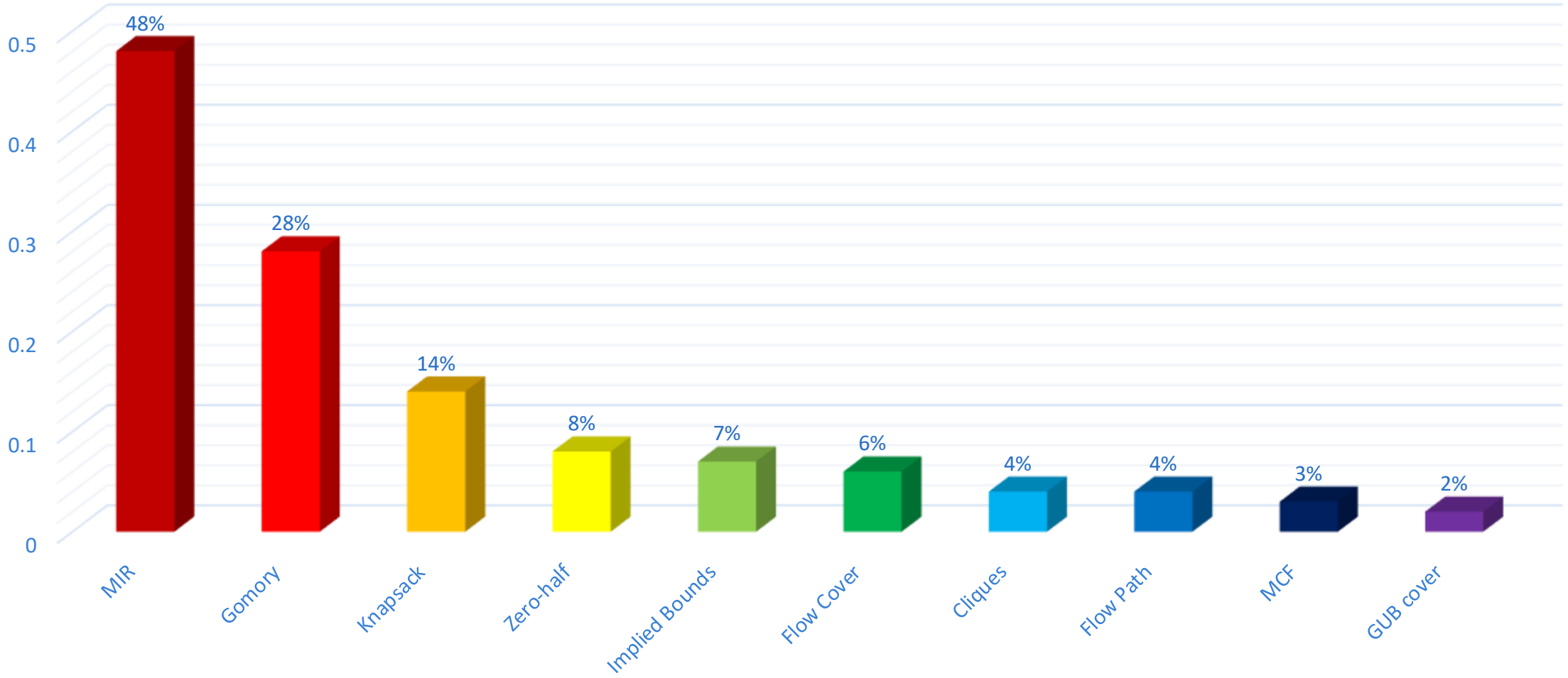
Resulting cover inequality: $x_1 + x_2 + x_3 + x_4 \leq 3$

Lifting

- If $x_5 = 1$, then $x_1 + x_2 + x_3 + x_4 \leq 1$
- Hence, $x_1 + x_2 + x_3 + x_4 + 2x_5 \leq 3$ is valid
- Need to solve knapsack problem $\alpha_j := d_0 - \max\{dx \mid ax \leq b - a_j\}$ to find lifting coefficient for variable x_j
  - Use dynamic programming to solve knapsack problem

Consider (1,1,1,0,1/17)

Cutting Planes – Performance



Achterberg and Wunderling: "Mixed Integer Programming: Analyzing 12 Years of Progress" (2013)
benchmark data based on CPLEX 12.5

Cuts Statistics Log

- At the end of a MIP log we usually find statistics about which cuts have been added to the LP relaxation:

```
 43287     0     cutoff   56       4.7096e+07 4.7092e+07  0.01%   375 1366s

Cutting planes:
  Gomory: 37
  Lift-and-project: 3
  Cover: 8
  Implied bound: 19
  MIR: 326
  StrongCG: 14
  Flow cover: 624
  Inf proof: 4
  Zero half: 19
  Mod-K: 1

Explored 44197 nodes (16447802 simplex iterations) in 1366.22 seconds
(2785.50 work units)
Thread count was 8 (of 16 available processors)
```

MIP Building Blocks

Presolve
- Tighten formulation and reduce problem size

Node selection
- Select next subproblem to process

Node presolve
- Additional presolve for subproblem

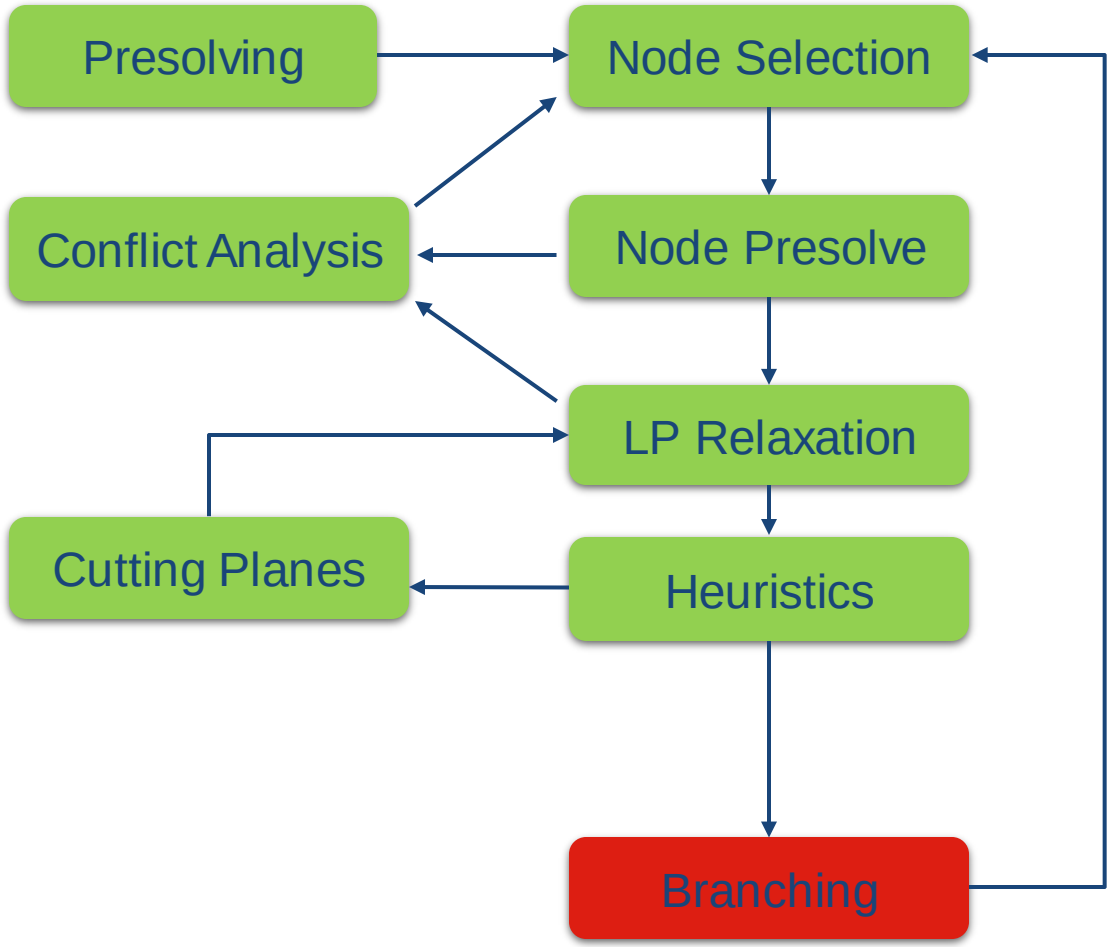
Solve continuous relaxations
- Gives a bound on the optimal integral objective

Conflict analysis
- Learn from infeasible subproblems

Cutting planes
- Cut off relaxation solutions

Primal heuristics
- Find integer feasible solutions

Branching variable selection
- Crucial for limiting search tree size



GUROBI OPTIMIZATION

Branching variable selection

Given node relaxation solution x^*

- Any integer variable with fractional x_j^* can be branched on
  - Branching on j creates two child nodes with
    - $x_j \leq \lfloor x_j^* \rfloor$
    - $x_j \geq \lceil x_j^* \rceil$
- How to choose j?
  - Choice has a dramatic impact on the size of the search tree

What's a good branching variable?

- Superb: fractional variable infeasible in both branch directions
  - Immediately prune the node as infeasible
- Great: infeasible in one direction
- Good: both directions move the objective

Expensive to predict which branches lead to infeasibility or big objective moves

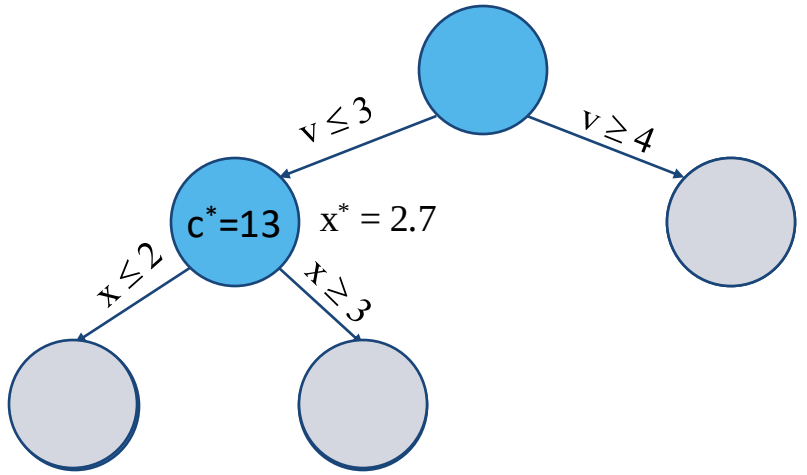
- Strong branching
  - Truncated LP solve for every possible branch at every node
  - Rarely cost effective
- Need a quick estimate

GUROBI OPTIMIZATION

Pseudo-Costs

Use historical data to predict impact of a branch:

- Record cost(x_j) = Δobj / Δx_j for each branch in a pseudo-cost table
  - Two entries per integer variable
  - Average down cost
  - Average up cost
- Use table to predict cost of a future branch

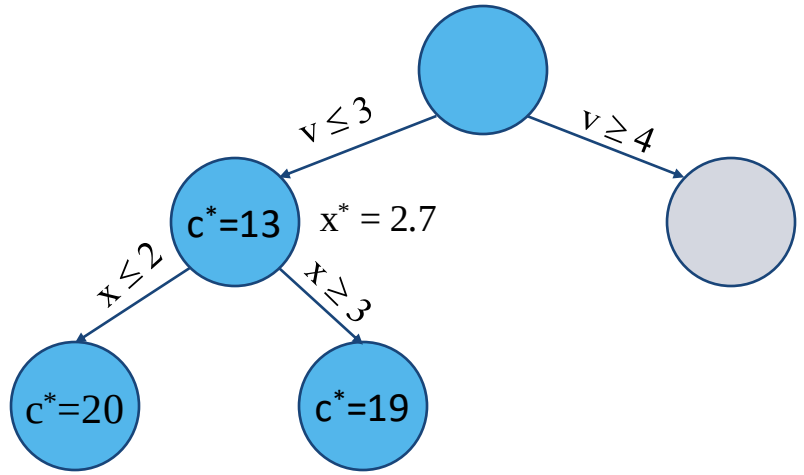


GUROBI OPTIMIZATION

Pseudo-Costs

Use historical data to predict impact of a branch:

- Record cost(x_j) = Δobj / Δx_j for each branch in a pseudo-cost table
  - Two entries per integer variable
  - Average down cost
  - Average up cost
- Use table to predict cost of a future branch



down pseudo-cost update:
Δobj/Δx = 7/0.7 = 10

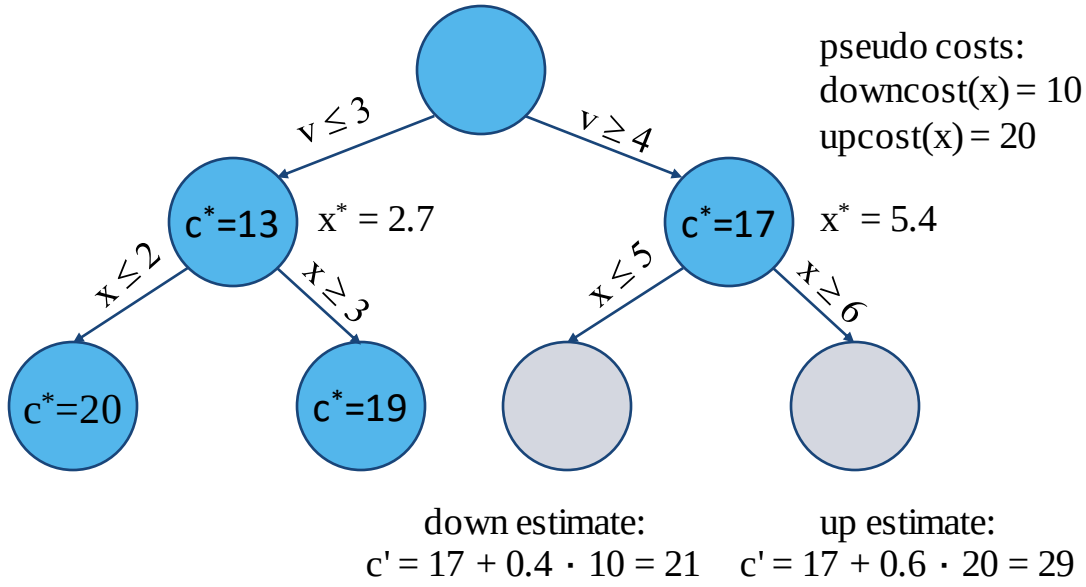
up pseudo-cost update:
Δobj/Δx = 6/0.3 = 20

GUROBI OPTIMIZATION

Pseudo-Costs

Use historical data to predict impact of a branch:

- Record cost(x_j) = Δobj / Δx_j for each branch in a pseudo-cost table
  - Two entries per integer variable
  - Average down cost
  - Average up cost
- Use table to predict cost of a future branch



Pseudo-Costs

GUROBI OPTIMIZATION

What do you do when there is no history?

- E.g., at the root node

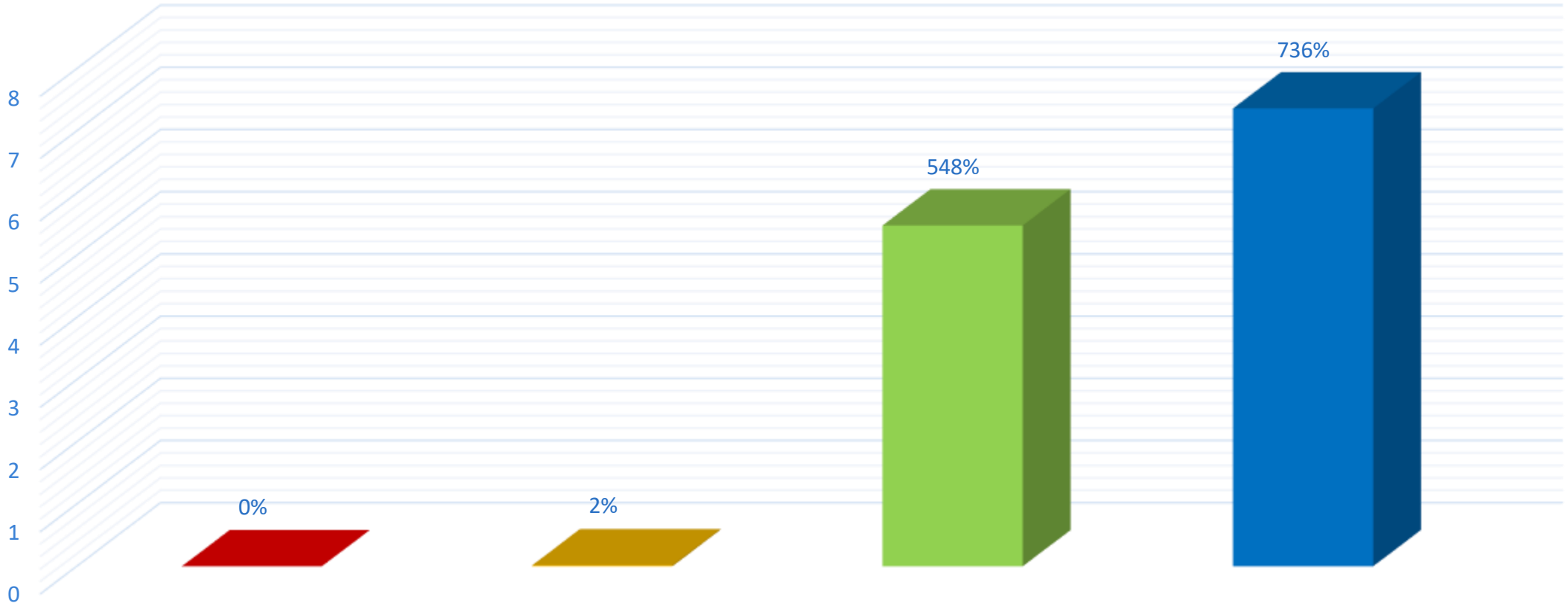
Initialize pseudo-costs [Linderoth & Savelsbergh, 1999]

- Always compute up/down cost (using strong branching) for new fractional variables
  - Initialize pseudo-costs for every fractional variable at root

Reliability branching [Achterberg, Koch & Martin, 2005]

- Do not rely on historical data until pseudo-cost for a variable has been recomputed r times

Branching Rules – Performance



Achterberg and Wunderling: "Mixed Integer Programming: Analyzing 12 Years of Progress" (2013)
benchmark data based on CPLEX 12.5
Achterberg, Koch, and Martin: "Branching Rules Revisited" (2005)

MIP Building Blocks

Presolve
- Tighten formulation and reduce problem size

Node selection
- Select next subproblem to process

Node presolve
- Additional presolve for subproblem

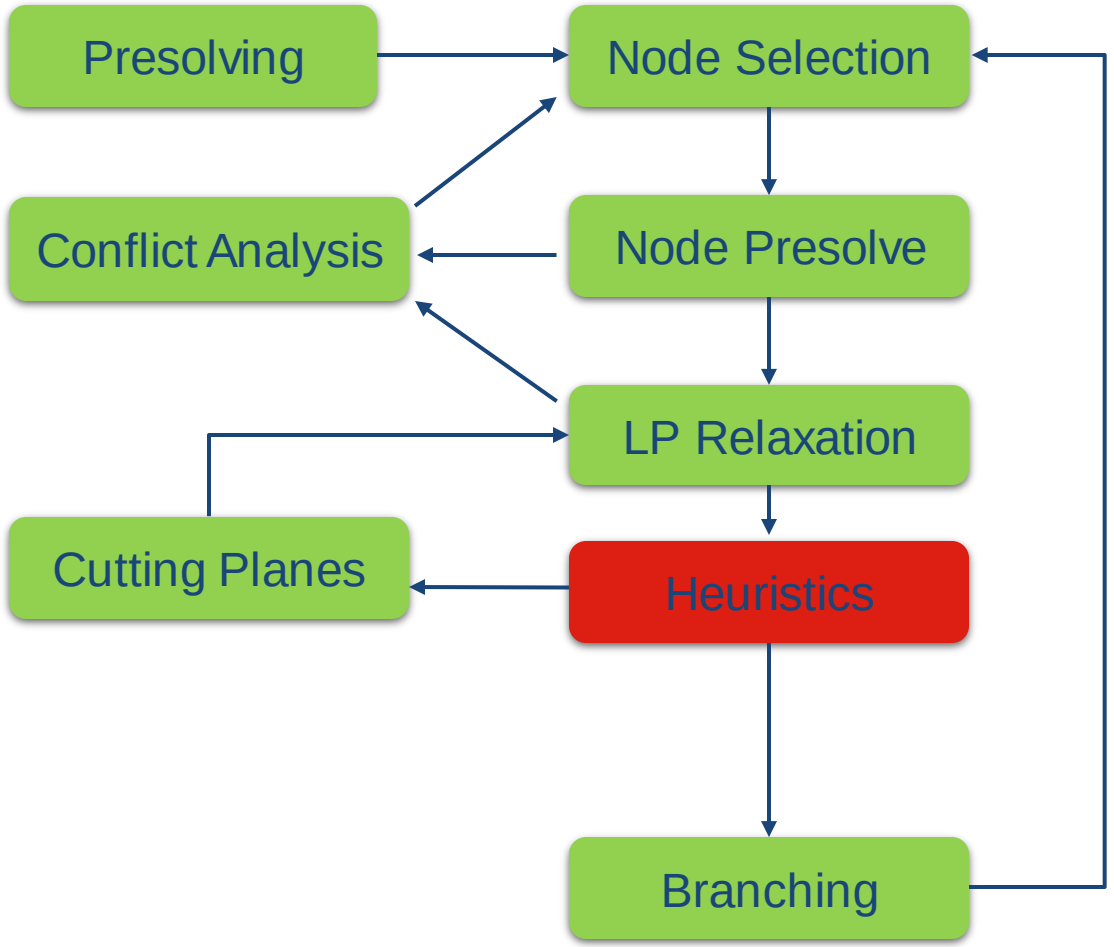
Solve continuous relaxations
- Gives a bound on the optimal integral objective

Conflict analysis
- Learn from infeasible subproblems

Cutting planes
- Cut off relaxation solutions

Primal heuristics
- Find integer feasible solutions

Branching variable selection
- Crucial for limiting search tree size



MIP Heuristics

GUROBI OPTIMIZATION

MIP solvers find new feasible solutions in two ways

- Branching
- Primal heuristics

Properties of a good heuristic

- Quick
  - Finds solutions earlier than branching
- Captures problem structure
  - Exploits structure more effectively than branching
- General
  - Finds solutions for lots of models

Gurobi has more than 30 heuristic types

- Adaptive strategies decide when to apply each

Types of MIP Heuristics

GUROBI OPTIMIZATION

Constructive heuristics

- No knowledge about other solutions needed
- Goal is to find solution early and to define “starting” point for improvement heuristics
- May produce poor quality solutions
- Typically fast (but not always, e.g. NoRel, ZeroObj)

Improvement heuristics

- Can be more expensive
- Need at least one known to solution to work on
- High quality solutions
- Provide better cutoff bound to prune tree
- Can be effective even on low quality solutions

Example MIP Heuristic - Rounding

Rounding Heuristic

- Start with:
  - Solution of relaxation
- Round integer variables

Quick?

- Very quick

Captures problem structure?

- No

General?

- Finds solutions to lots of easy models

Example MIP Heuristic - RINS

GUROBI OPTIMIZATION

Relaxation Induced Neighborhood Search

- Start with:
  - Node relaxation solution
  - Best known integer feasible solution
- Fix integer variables whose values agree in both
- Solve a MIP on the rest

Quick?

- No – solves a MIP
  - Integer infeasibilities at a node where RINS is called is the minimum number of unfixed variables

Captures problem structure?

- Yes – searches a neighborhood of the relaxation
- No – neighborhood considers mathematical calculations, not problem structure

General?

- Yes – effective on a variety of models

Example MIP Heuristic - NoRel

No Relaxation Heuristic

- Start from some (feasible or infeasible) vector
  - Constructed by quick heuristic
- Solve smaller sub-MIPs (with fixed variables) to decrease infeasibility or objective value
  - Use multiple threads to solve sub-MIPs in parallel
- Various neighborhood strategies
  - adaptive to spend more time on more successful ones

Quick?

- No – runs "forever", until work (`NoRelHeurWork`) or time (`NoRelHeurTime`) limit is reached

Captures problem structure?

- No – ad-hoc generation of sub-MIPs

General?

- No – main use when relaxations solve too slowly
- Yes – can be successful to find good solution in limited time

Heuristic Log Lines

```
Found heuristic solution: objective 1.377283e+07
Presolve removed 918 rows and 26705 columns
Presolve time: 2.50s
Presolved: 1269 rows, 21712 columns, 398388 nonzeros
Found heuristic solution: objective 2770423.8600
Variable types: 0 continuous, 21712 integer (21709 binary)
Found heuristic solution: objective 2762983.8600

Root relaxation: objective 1.010525e+06, 1504 iterations, 0.06 seconds (0.13 work units)

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

     0     0 1010524.73    0  269 2762983.86 1010524.73  63.4%     -    3s
H    0     0                     2736029.7100 1010524.73  63.1%     -    3s
H    0     0                     2727449.4800 1010524.73  62.9%     -    3s
H    0     0                     2693681.7100 1011189.12  62.5%     -    3s
     0     0 1038921.12    0  523 2693681.71 1038921.12  61.4%     -    3s
H    0     0                     2670522.0600 1038921.12  61.1%     -    4s
```

Heuristic found better incumbent solution

Lower bound on # unfixed integer variables in RINS heuristic

GUROBI OPTIMIZATION

Improving Default Performance

How can I use my newfound knowledge about Gurobi's algorithmic features to get better MIP performance?

Examine node logs to identify likely areas of performance bottlenecks

- Does the lack of progress involve the upper bound, lower bound or both?
- Assess whether node LP solve time is the primary bottleneck
  - Do not: adjust branching parameters if Gurobi is in the root cut loop
  - Do: consider using Gurobi's No Relaxation Heuristic (NoRelHeur parameter) if node LP solve times are highly problematic.

Distinguish parameters that primarily help upper bound from those that primarily help lower bound

- Example: Don't raise intensity of heuristics when log indicates lack of progress in the lower bound is the performance problem
  - Aggressive cuts, MIPFocus = 2 or 3 more likely to help

GUROBI OPTIMIZATION

Improving Default Performance

How can I use my newfound knowledge about Gurobi's algorithmic features to get better MIP performance?

Sometimes reducing or disabling completely default parameter settings can help

- Some features may not succeed, so don't spend time on them
- Some features may be better covered by others
  - Turning off heuristics (or reducing their intensity) even though node log indicates they are effectively finding good solutions.
  - Branching may be able to find equally good solutions, resulting faster node throughput will give branching more opportunity to succeed.
  - Turning off cuts (or reducing their intensity) when finding good solutions is important and progress in the lower bound is modest
- Gurobi's concurrentMIP feature can help
  - 2 or more runs in parallel on the same model with different parameter settings

Consider the fundamental tradeoff between node processing rate and computation in each node

GUROBI OPTIMIZATION

Thank You

For more information: www.gurobi.com

Roland Wunderling
Senior Developer

Parallelization

Parallelization opportunities

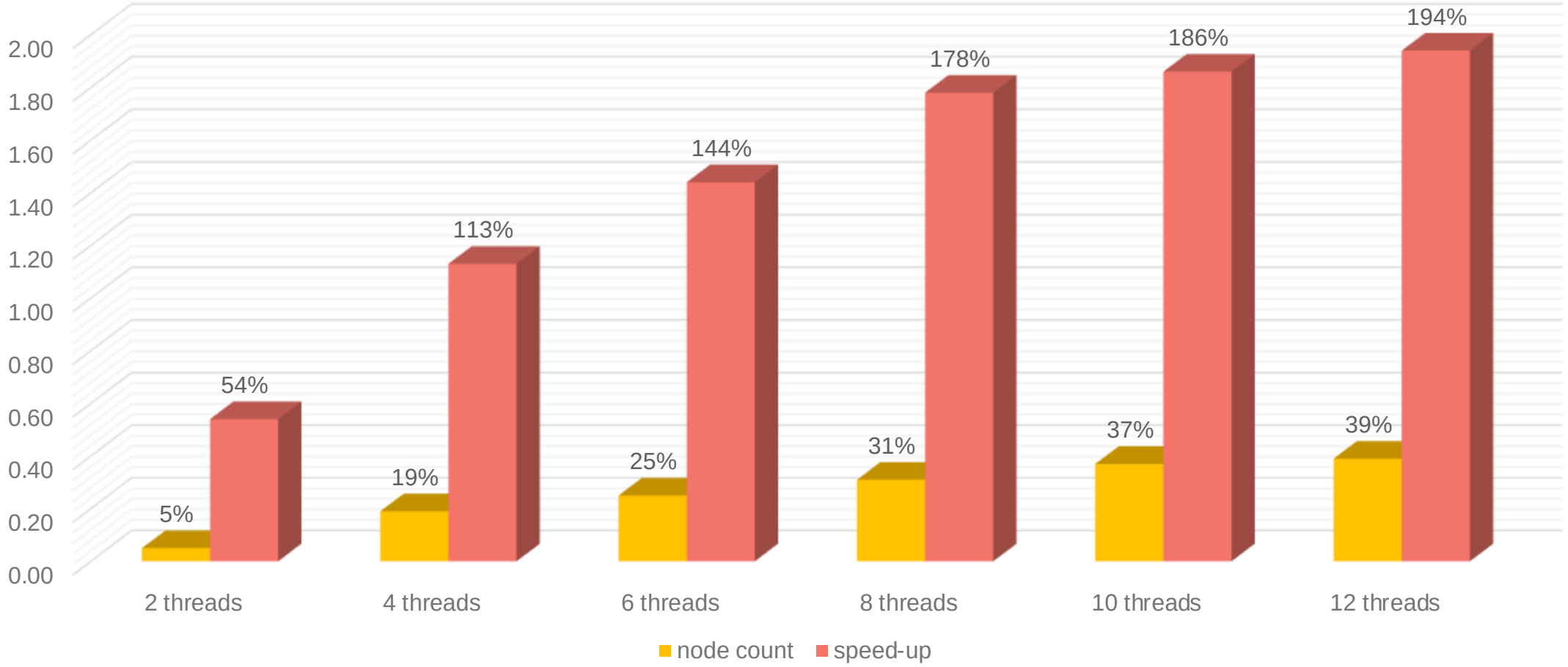
- Parallel probing during presolve
  - Almost no improvement
- Use barrier or concurrent LP for initial LP relaxation solve
  - Only helps for large models
- Run heuristics or other potentially useful algorithms in parallel to the root cutting plane loop
  - Moderate performance improvements: 20-25%
  - Does not scale beyond a few threads
- Solve branch-and-bound nodes in parallel
  - Main speed-up for parallel MIP
  - Performance improvement depends a lot on shape of search tree
  - Typically scales relatively well up to 8 to 16 threads

Parallelization

Parallelization issues

- Determinism
- Load balancing
- CPU heat and memory bandwidth
  - Additional threads slow down main thread
- Root node does not parallelize well
  - Sequential runtime of root node imposes limits on parallelization speed-up
  - Amdahl's law
- A dive in the search tree cannot be parallelized
  - Parallelization only helps if significant number of dives necessary to solve model

Parallel MIP – Performance



Achterberg and Wunderling: "Mixed Integer Programming: Analyzing 12 Years of Progress" (2013)
benchmark data based on CPLEX 12.5, models with ≥ 10 seconds solve time

Presolve - Coefficient Strengthening

Strictly Stronger?

Consider:

- $5\ b_1 + 3\ b_2 + 3\ b_3 + 3\ b_4 + 8\ b_5 \leq 8$

Stronger…?

- $4\ b_1 + 4\ b_2 + 4\ b_3 + 4\ b_4 + 8\ b_5 \leq 8$

Probably, but…

- Doesn't strictly dominate original
  - $(1, 0, 0, 0, 0.5)$ satisfies second, but not first
- Could weaken relaxation
  - No definitive metrics
- Hippocratic oath of presolve
  - "First, do no harm"
  - Lots of cases where it hurts

Presolve - Cost Versus Benefit

GUROBI OPTIMIZATION

Trade off cost of reduction vs benefit

Why worry about cost?

- Only one presolve for each MIP model

A few reasons:

- One presolve can still be expensive
- Aggressive use of sub-MIP heuristics (RINS)
  - Lots of “truncated” MIP solves
    - Multiple presolves, on smaller models
    - Presolve can be dominant cost on each
- Strengthening as a cut separator