Lecture 16 MIP Solvers

Can Li

ChE 597: Computational Optimization
Purdue University

History of MIP solvers

- The CPLEX Optimizer was named for the simplex method implemented in the C programming language. It was originally developed by Robert Bixby and sold commercially from 1988 by CPLEX Optimization Inc. This was acquired by ILOG in 1997 and ILOG was subsequently acquired by IBM in January 2009.
- Zonghao Gu, Edward Rothberg, and Robert Bixby left CPLEX and founded Gurobi in 2008. Gurobi is similar to CPLEX by design. It was the best-performing MILP solver in the Mittelmann benchmark. Now it has stopped participating in the Mittelmann benchmark.
- Gurobi can now solve nonconvex MIQCQP. Starting from Gurobi 11, simple MINLP capability is added. Starting with Gurobi 12, general MINLP is added.

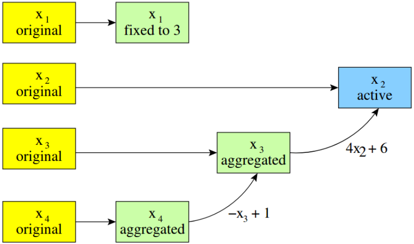
What's inside the box?

- Preprocessing
- Cuts
- Heuristics
- Branch and bound
- Callbacks

There are hundreds of parameters in CPLEX/Gurobi that control these procedures.

Preprocessing/presolve

Preprocessing routines can be run before the solving starts (usually several, sequentially), to simplify and / or tighten the problem formulation.




```
Presolve removed 83494 rows and 41814 columns
Presolve time: 0.35s
```

One can control how much time/rounds to devote to preprocessing by changing the parameters. `https://www.gurobi.com/documentation/current/refman/presolve.html` to change a parameter in pyomo.

Cutting planes

Tighten the LP relaxation

- Boolean Quadric Polytope (BQP) cuts
- Clique cuts
- Cover cuts
- Disjunctive cuts
- Flow cover cuts
- Flow path cuts
- Gomory fractional cuts
- Zero-half cuts
- Generalized upper bound (GUB) cover cuts
- Implied bound cuts: global and local
- Lift-and-project cuts
- Mixed integer rounding (MIR) cuts
- Multi-commodity flow (MCF) cuts
- Reformulation Linearization Technique (RLT) cuts

There are parameters in CPLEX/Gurobi to control how "aggressive" you want to generate the cuts.
`https://docs.gurobi.com/projects/optimizer/en/current/reference/parameters.html#cuts`

Heuristics

Heuristics are to find good feasible solutions (upper bounds) quickly.

- Neighborhood search
- Local branching
- Relaxation induced neighborhood search (RINS)
- Solution polishing
- Feasibility pump

Parameter to control how much time to spend on heuristics
`https://www.gurobi.com/documentation/current/refman/heuristics.html`

Neighborhood search

Let x^* be the solution to the LP relaxation. Explore neighborhoods defined by $\lfloor x_i^* \rfloor \leq x_i \leq \lceil x_i^* \rceil$, $i \in I$, i.e., add these bound constraints to the MILP and solve the subMIP in the neighborhood of the LP solution. This only applies when x is has general integer variables. For binary variables, one can just do a simple rounding heuristic.

- The search within the subMIP is truncated by limiting the number of nodes explored.

Local Branching

- Assume we have a feasible solution $\hat{x}$, the so-called reference solution, and let $S := \{j \in I \mid \hat{x}_j = 1\}$ denote the binary support of $\hat{x}$. are the binary variables being 1.

  $I \setminus S$ 0

- For a given positive integer parameter k, we define the k-OPT neighborhood $\mathcal{N}(\hat{x}, k)$ of $\hat{x}$ as the set of the feasible solutions satisfying

$$\Delta(x, \hat{x}) := \sum_{j \in S} (1 - x_j) + \sum_{j \in I \setminus S} x_j \leq k, \tag{1}$$

$x = \hat{x}$

  known as the *local branching constraint*.

- This constraint requires at most k variables have values different from $\hat{x}$.
- Constraint (1) can also be used to branch within a branch and bound:

$$\begin{aligned} &\Delta(x, \hat{x}) \leq k \quad \text{(left branch)} \\ \text{or} \quad &\Delta(x, \hat{x}) \geq k + 1 \quad \text{(right branch)} \end{aligned}$$

Relaxation Induced Neighborhood Search

- A similar concept of neighborhood takes into account simultaneously both
  - the *incumbent solution* $\bar{x}$, and
  - the *solution of the continuous relaxation* $\tilde{x}$,

  at a given node of the branch-and-bound tree.
- $\bar{x}$ and $\tilde{x}$ are compared and all the binary variables that assume the same value are **hard-fixed** in an associated MILP.
- This associated MILP is then solved by using the MILP solver as a black-box.
- In case the incumbent solution is improved, $\bar{x}$ is updated in the rest of the tree.
- This method turns out to give very competitive results on general MILPs.

RINS Formulation

RINS: Let $\tilde{x}$ be a known feasible solution and let $\hat{x}$ be an LP-solution at some node in the search tree. We create a new MILP (Danna et al., 2005):

$$
\begin{aligned}
&\text{minimize } z = cx \\
&\text{s.t.} \\
&\qquad\qquad Ax \le b, \\
&\qquad\qquad x_i = \tilde{x}_i \quad \forall i \quad \text{ s.t. } \tilde{x}_i = \hat{x}_i \\
&\qquad\qquad x \in \mathbb{Z}^r \times \mathbb{R}^{n-r}.
\end{aligned}
$$

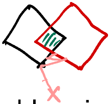
Solution Polishing

- Idea: Solution polishing can yield better solutions in situations where good solutions are otherwise hard to find. More time-intensive than other heuristics, solution polishing is actually a variety of branch and cut that works after an initial solution is available. In fact, it requires a solution to be available for polishing, either a solution produced by branch and cut, or a MIP start supplied by a user.
- Rothberg, 2007 developed an evolutionary algorithm for polishing MIP solutions. It can be seen as a combination of genetic algorithm (mutating, combining solutions from different MIPs). It requires solving sub MIPs.

Feasibility Pump: The Basic Scheme

- We start from any $x^0 \in P$ where P denotes the LP relaxation, and round to obtain $\hat{x}^0$.
- We look for a point $\hat{x}^1 \in P$ which is as close as possible to $\hat{x}^0$ by solving the problem:

$$\min\{\underbrace{\Delta(x,\hat{x})}_{\text{distance metric}} \mid x \in P\}$$



- If we choose the measure $\Delta(x,\hat{x})$ properly, this problem is easily solvable.
- If $\hat{x}^1$ satisfies the integrality constraints, we are done.
- Otherwise, we obtain $\hat{x}^1$ by rounding $\tilde{x}^1$, and repeat.
- From a geometric point of view, this is like an "alternating projection" algorithm (project to the polyhedron and project to the integer set). converge when the two sets are convex
- These satisfy feasibility in a complementary but partial way:
  1. $\tilde{x}^i$, satisfies the linear constraints,
  2. $\hat{x}^i$, the integrality requirements.

Branching Priority

- Affects the order of branching.
- For example, in scheduling problem, first branch on the assignment of jobs to machines, then branch on the sequence of the jobs.

`https://www.gurobi.com/documentation/current/refman/branchpriority.html`

Solution Pool

Goal: find multiple solutions in an MILP.

https://www.gurobi.com/documentation/current/refman/poolsearchmode.html

Multi-objective

If you have multiple objectives $c_1^T x$, $c_2^T x$, $\ldots$, $c_n^T x$, you can solve the problem

$$\min_x \sum_{i=1}^{n} \lambda_i c_i^T x$$

where $\lambda_i \geq 0$, $\forall i$, $\sum_{i=1}^{n} \lambda_i = 1$.
By changing the values of λ, we obtain the "Pareto front" of the multi-objective optimization problem where we cannot improve any objective without sacrificing others. `https://www.gurobi.com/documentation/current/refman/multiple_objectives.html`

LP algorithms

- Primal simplex
- Dual simplex
- Barrier with crossover
- Barrier without crossover

LP method `https://www.gurobi.com/documentation/current/refman/method.html`
Crossover `https://www.gurobi.com/documentation/current/refman/crossover.html`

- Takeaway: if you are solving an LP and doesn't care whether you get an extreme point or not, use "barrier without crossover" is fastest.
- If you are solving an MILP, the root relaxation has to return an extreme point. The reason is to use dual simplex to solve the node relaxations after branching starts. The dual feasible solution remain feasible after branching, which provides a good warmstart.

Parallel Computing

- By default, MIP solvers use all the threads on your computer/server. Branch and bound can be easily parallelized
- However, empirical results show it is best to use 6-8 threads. Having more threads may be counter-productive

`https://www.gurobi.com/documentation/current/refman/threads.html` Change the number of threads

Solver APIs

- The MIP solvers are usually written in C. But it has APIs in C, C++, Java, Python, Matlab, R. All these APIs are solver-dependent.

How does Pyomo work?

- Pyomo is designed to be a solver-independent algebraic modeling language.
- Pyomo can "talk" to different solvers. However, there are different ways to send your model data to the solvers.
  - LP file. write you model as a (.lp) file on your hard-drive. Then, the solver reads the LP file. This approach does not need to use the solver's python API.
  - Pyomo persistent solvers: communicate with the solver through the their python API.

Gurobi persistent solver:

`https://github.com/Pyomo/pyomo/blob/main/pyomo/solvers/plugins/solvers/gurobi_persistent.py`

Lazy Constraints/Callback

- Lazy constraints are useful when the full set of constraints is too large to explicitly include in the initial formulation. When a MIP solver reaches a new solution, for example with a heuristic or by solving a problem at a node in the branch-and-bound tree, it will give the user the chance to provide constraints that would make the current solution infeasible.
- An example is the subtour elimination constraints.
- Lazy callbacks give more flexibility of what to implement when an integer feasible solution is found. This used in the implementation of single-tree Benders decomposition, single tree OA (QG).
- example `https://stackoverflow.com/questions/58200552/pyomo-and-gurobi-does-pyomo-support-solver-callbacks-t`

User cut callback

- User cuts provide a way for the user to tighten the LP relaxation using problem-specific knowledge that the solver cannot or is unable to infer from the model. Just like with lazy constraints, when a MIP solver reaches a new node in the branch-and-bound tree, it will give the user the chance to provide cuts to make the current relaxed (fractional) solution infeasible in the hopes of obtaining an integer solution.

More on callbacks `https://www.gurobi.com/documentation/current/refman/cb_codes.html`

- Key take-away: Lazy constraints are necessary to get the optimal (feasible) solution. User cuts are not necessary (it only helps to solve the problem faster).

Solver log file

- **Read this carefully every time you solve a nontrival optimization model!**
- Check matrix, rhs range. root node relaxation. root node bound after adding cuts.
- Easy problem can be solved fast, without doing much branch and bound.
- Hard problem takes exponentially many branch and bound nodes. Large memory consumption. Check your memory consumption from time to time or set an upper limit.
- Diagnose why it is slow: difficult to find good feasible solutions or closing the gap.

```
CPU model: AMD EPYC 7643 48-Core Processor, instruction set [SSE2|AVX|AVX2]
Thread count: 48 physical cores, 96 logical processors, using up to 8 threads

Optimize a model with 92155 rows, 50135 columns and 262513 nonzeros
Model fingerprint: 0x81f9d80d
Variable types: 29975 continuous, 20160 integer (20160 binary)
Coefficient statistics:
  Matrix range     [1e-02, 5e+02]
  Objective range  [9e+02, 2e+03]
  Bounds range     [0e+00, 0e+00]
  RHS range        [1e+00, 1e+08]
Found heuristic solution: objective -814886.9986
Presolve removed 83494 rows and 41814 columns
Presolve time: 0.35s
Presolved: 8661 rows, 8321 columns, 41716 nonzeros
Variable types: 6327 continuous, 1994 integer (1994 binary)
Root relaxation presolved: 8659 rows, 8319 columns, 41712 nonzeros

Root barrier log...

Ordering time: 0.01s

Barrier statistics:
 AA' NZ     : 8.906e+04
 Factor NZ  : 3.210e+05 (roughly 10 MB of memory)
 Factor Ops : 1.805e+07 (less than 1 second per iteration)
 Threads    : 8
```

```
CPU model: AMD EPYC 7643 48-Core Processor, instruction set [SSE2|AVX|AVX2]
Thread count: 48 physical cores, 96 logical processors, using up to 8 threads

Optimize a model with 92155 rows, 50135 columns and 262513 nonzeros
Model fingerprint: 0x81f9d80d          min c^T x
Variable types: 29975 continuous, 20160 integer (20160 binary)
Coefficient statistics: 1e-8   5e-5   Ax ≤ b
  Matrix range (A) [1e-02, 5e+02]
  Objective range (c) [9e+02, 2e+03]
  Bounds range     [0e+00, 0e+00]
  RHS range (b)    [1e+00, 1e+08] → suspicious big-M
Found heuristic solution: objective -814886.9986
Presolve removed 83494 rows and 41814 columns
Presolve time: 0.35s
Presolved: 8661 rows, 8321 columns, 41716 nonzeros
Variable types: 6327 continuous, 1994 integer (1994 binary)
Root relaxation presolved: 8659 rows, 8319 columns, 41712 nonzeros

Root barrier log...

Ordering time: 0.01s

Barrier statistics:
 AA' NZ     : 8.906e+04
 Factor NZ  : 3.210e+05 (roughly 10 MB of memory)
 Factor Ops : 1.805e+07 (less than 1 second per iteration)
 Threads    : 8

                  Objective                Residual
Iter       Primal          Dual         Primal    Dual     Compl     Time
   0  -8.75349353e+10  3.16759838e+11  3.51e+04 3.61e+03  2.85e+07     0s
   1  -5.65473859e+10  4.49136693e+09  4.93e+03 9.37e-11  4.19e+06     0s
   2  -4.66624680e+09  2.12223081e+09  4.31e+02 9.91e-11  4.16e+05     0s
   3  -5.93650592e+08  5.33710820e+08  5.43e+01 1.23e-10  6.09e+04     0s
   4  -1.41554632e+08  1.48462275e+08  1.28e+01 9.00e-11  1.46e+04     0s
   5  -6.17438000e+07  5.76147676e+07  5.54e+00 5.02e-11  5.82e+03     0s
   6  -2.17774077e+07  1.42426566e+07  1.94e+00 3.62e-11  1.71e+03     0s
   7  -8.40739562e+06  4.59748907e+06  7.39e-01 1.43e-11  6.06e+02     0s
   8  -6.47436121e+06  3.03782035e+06  5.70e-01 1.36e-11  4.41e+02     0s
   9  -4.47422322e+06  2.35107115e+06  3.96e-01 9.09e-12  3.16e+02     0s
  10  -3.05997438e+06  1.42584289e+06  2.75e-01 7.15e-12  2.07e+02     0s
  11  -2.41341865e+06  1.28100790e+06  2.20e-01 6.84e-12  1.70e+02     0s
  12  -2.20664641e+06  1.18989676e+06  2.02e-01 8.35e-12  1.57e+02     0s
  13  -1.98828288e+06  1.17144512e+06  1.84e-01 8.32e-12  1.46e+02     0s
  14  -1.54978590e+06  8.44878454e+05  1.47e-01 5.74e-12  1.10e+02     0s
  15  -1.07826885e+06  6.08789586e+05  1.05e-01 5.13e-12  7.78e+01     0s
  16  -5.82641701e+05  3.53376691e+05  6.21e-02 4.97e-12  4.31e+01     0s
  17  -3.62968591e+05  1.97842701e+05  4.05e-02 2.79e-12  2.59e+01     0s
  18  -1.95442249e+05  1.13637501e+05  2.38e-02 3.14e-12  1.43e+01     1s
  19  -7.15981068e+04  6.24467786e+04  1.07e-02 1.81e-12  6.19e+00     1s
  20   6.33863689e+03  4.23819434e+04  1.85e-03 2.93e-12  1.66e+00     1s
  21   1.05186928e+04  2.81456352e+04  1.21e-03 2.18e-12  8.12e-01     1s
  22   1.51678553e+04  2.35368659e+04  4.31e-04 1.96e-12  3.84e-01     1s
  23   1.66147354e+04  1.99499477e+04  1.89e-04 1.17e-12  1.53e-01     1s
  24   1.69590686e+04  1.90483655e+04  1.32e-04 1.18e-12  9.61e-02     1s
  25   1.72897664e+04  1.81986844e+04  7.65e-05 1.05e-12  4.20e-02     1s
  26   1.75170934e+04  1.79693432e+04  3.80e-05 1.33e-12  2.09e-02     1s
  27   1.76892413e+04  1.77892494e+04  8.51e-06 1.76e-12  4.62e-03     1s
  28   1.77322731e+04  1.77461969e+04  1.30e-06 1.82e-12  6.44e-04     1s
  29   1.77401344e+04  1.77408199e+04  2.73e-09 2.73e-12  3.12e-05     1s
  30   1.77401610e+04  1.77401729e+04  1.38e-10 2.73e-12  5.44e-07     1s
  31   1.77401626e+04  1.77401627e+04  1.14e-10 2.73e-12  2.03e-09     1s

Barrier solved model in 31 iterations and 0.58 seconds (0.58 work units)
```

```
Optimal objective 1.77401626e+04


Root relaxation: objective 1.774016e+04, 6363 iterations, 0.32 seconds (0.30
work units)

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

     0     0 17740.1626    0  581 -814887.00 17740.1626   102%     -    2s
H    0     0                    -96621.07624 17740.1626   118%     -    2s
     0     0 17707.4841    0  624 -96621.076 17707.4841   118%     -    3s
     0     0 17704.5241    0  630 -96621.076 17704.5241   118%     -    3s
     0     0 17660.0868    0  588 -96621.076 17660.0868   118%     -    3s
     0     0 17660.0868    0  586 -96621.076 17660.0868   118%     -    3s
H    0     0                    -29247.14276 17660.0868   160%     -    6s
     0     0 17660.0868    0  123 -29247.143 17660.0868   160%     -    6s
H    0     0                    -14563.64994 17660.0868   221%     -    6s
```

heuristic finds a solution

```
     0     0 17660.0868    0  123 -14563.650 17660.0868   221%     -    6s
     0     0 17660.0868    0  126 -14563.650 17660.0868   221%     -    6s
     0     0 17660.0868    0  125 -14563.650 17660.0868   221%     -    6s
     0     0 17660.0868    0  231 -14563.650 17660.0868   221%     -    7s
     0     0 17660.0868    0  196 -14563.650 17660.0868   221%     -    7s
     0     0 17660.0868    0  186 -14563.650 17660.0868   221%     -    7s
     0     0 17660.0868    0  194 -14563.650 17660.0868   221%     -    7s
     0     0 17660.0868    0  187 -14563.650 17660.0868   221%     -    7s
     0     0 17660.0868    0  188 -14563.650 17660.0868   221%     -    7s
     0     0 17660.0868    0  151 -14563.650 17660.0868   221%     -    7s
     0     0 17660.0868    0  164 -14563.650 17660.0868   221%     -    7s
     0     0 17660.0868    0  164 -14563.650 17660.0868   221%     -    7s
     0     0 17660.0868    0  169 -14563.650 17660.0868   221%     -    8s
     0     0 17660.0868    0   86 -14563.650 17660.0868   221%     -    8s
H    0     0                    -7148.918660 17660.0868   347%     -    9s
     0     2 17660.0868    0   69 -7148.9187 17660.0868   347%     -    9s
     3     8 17660.0868    2  143 -7148.9187 17660.0868   347%  1127   10s
H   32    40                    -1440.475097 17510.6047  1316%   791   10s
H   32    40                    1318.5313397 17510.6047  1228%   791   10s
H   58    62                    3330.7868465 17510.6047   426%   523   11s
H   59    62                    4276.5071333 17510.6047   309%   515   11s
H  106   107                    4276.5071340 17510.6047   309%   430   12s
H  188   195                    4276.5073972 17510.6047   309%   294   13s
H  230   230                    4789.4576283 17510.6047   266%   276   14s
H  231   230                    5845.1096079 17510.6047   200%   275   14s
   265   259 10627.4584   26  105 5845.10961 17510.6047   200%   273   15s
H  725   448                    9351.5655274 17510.6047  87.2%   286   22s
H  726   448                    11089.017488 17510.6047  57.9%   286   22s
H  761   433                    11089.018283 16804.7463  51.5%   282   22s
  1231   626 11731.0990   59   72 11089.0183 16596.9487  49.7%   264   25s
H 1507   738                    11345.897832 16596.9487  46.3%   262   29s
  1508   737 11596.0990   45   86 11345.8978 16596.9487  46.3%   262   30s
  1516   742 11774.9584   71  116 11345.8978 16596.9487  46.3%   260   36s
  1717   776 13460.5704   28   88 11345.8978 16596.9487  46.3%   264   40s
H 1913   746                    11345.898980 16596.9487  46.3%   248   42s
  2238   844 12517.9584   32   68 11345.8990 16596.9487  46.3%   233   45s
H 2469   839                    11761.920659 16596.9487  41.1%   225   46s
  3079  1054 12020.3437  109  106 11761.9207 16596.9487  41.1%   205   54s
H 3082  1019                    11761.921451 16596.9487  41.1%   205   54s
  3092  1024 infeasible  110       11761.9215 16596.9487  41.1%   206   55s
H 3094   810                    11774.958412 16596.9487  41.0%   206   55s
  4236  1639 11868.5460  247  101 11774.9584 16596.9487  41.0%   190   60s
H 4304  1639                    11774.959215 16596.9487  41.0%   189   60s
  5530  2609 12071.6251   35  104 11774.9592 16058.5000  36.4%   179   65s
  6796  3647 11909.9584   47  100 11774.9592 15531.8030  28.7%   173   70s
H 6903   715                    12071.625079 15531.8030  28.7%   174   70s
```

```
H 8102   587                      12071.627454 14885.4915  23.3%   163   73s
  8501   692 12198.6527   36   99 12071.6275 14233.3151  17.9%   161   75s
  9402  1180 12821.8972   83   93 12071.6275 13562.7084  12.4%   164   80s
 10476  1809 12821.8972   58   91 12071.6275 12824.4084  6.24%   168   86s
 11568  2273 infeasible  234      12071.6275 12824.4084  6.24%   172   91s
 12694  2779 12574.1774   50  122 12071.6275 12824.4084  6.24%   179   97s
 14132  3863 12595.1316  146   79 12071.6275 12824.4084  6.24%   178  102s
 14684  4467 12087.5381  225  124 12071.6275 12824.4084  6.24%   176  105s
 16345  5169 12821.8972   90   71 12071.6275 12824.4084  6.24%   175  110s
 17768  6323 12263.9173   90   94 12071.6275 12824.4084  6.24%   175  115s
 19614  7623     cutoff   58      12071.6275 12824.4084  6.24%   173  122s
 20605  8210 12294.5174  221  113 12071.6275 12824.4084  6.24%   172  125s
 21573  8219 12821.8972   55   89 12071.6275 12824.4084  6.24%   173  130s
 22590  9644 12222.1346  146   97 12071.6275 12824.4084  6.24%   172  137s
 23664 10334 12821.8972   89   76 12071.6275 12824.4084  6.24%   172  141s
 24605 11053 12821.8972  122  102 12071.6275 12824.4084  6.24%   172  145s
 26937 12674 12558.4932   97   95 12071.6275 12824.4084  6.24%   173  153s
 28404 13642 12481.7713  108  104 12071.6275 12824.4084  6.24%   171  157s
 29823 14798 12807.5866   77  102 12071.6275 12824.4084  6.24%   172  207s
 31514 15733 12821.8972  134   61 12071.6275 12824.4084  6.24%   171  211s
*32476 15441             268    12074.383586 12824.4084  6.21%   169  211s
 32750 16295 12081.3655  351   78 12074.3836 12824.4084  6.21%   169  215s
H34111 15060                    12101.373966 12824.4084  5.97%   169  218s
H34454  1258                    12821.897187 12824.4084  0.02%   169  218s
 35062   843     cutoff   99      12821.8972 12824.4084  0.02%   169  222s
H35069   843                    12821.898429 12824.4084  0.02%   169  222s
H35217   843                    12821.899229 12824.4084  0.02%   169  222s
 35714   768 infeasible   79      12821.8992 12824.4084  0.02%   170  226s
 36432   654 12824.4084   92  109 12821.8992 12824.4084  0.02%   173  230s
 37040   477 infeasible   81      12821.8992 12824.4084  0.02%   175  235s
 38605   305 12824.4084   96   96 12821.8992 12824.4084  0.02%   180  244s
 39219   229     cutoff   90      12821.8992 12824.4084  0.02%   183  249s
 39947   234 infeasible   49      12821.8992 12824.4084  0.02%   185  253s
 40719   249 12824.4084   69   99 12821.8992 12824.4084  0.02%   187  258s
 41505   310 infeasible   59      12821.8992 12824.4084  0.02%   188  262s
 42399   490 infeasible   96      12821.8992 12824.4084  0.02%   190  267s
 43465   584 12824.4084   69   95 12821.8992 12824.4084  0.02%   192  273s
H43852   584                    12821.900927 12824.4084  0.02%   192  273s
 44046   678 infeasible   77      12821.9009 12824.4084  0.02%   193  279s
 45186   811 12824.4084   79   96 12821.9009 12824.4084  0.02%   195  284s
 46149  1010     cutoff   86      12821.9009 12824.4084  0.02%   197  291s
 47374  1210 12824.4084   70   75 12821.9009 12824.4084  0.02%   200  298s
 48794  1435 12824.4084   68   73 12821.9009 12824.4084  0.02%   202  305s
 50126  1471 12824.4084   83  107 12821.9009 12824.4084  0.02%   204  312s
 51105  1476     cutoff   71      12821.9009 12824.4084  0.02%   206  320s
 52398  1421 12824.4084   77  109 12821.9009 12824.4084  0.02%   208  327s
 53623  1407 12824.4084   66   97 12821.9009 12824.4084  0.02%   211  335s
 54938  1214 12824.4084   90   68 12821.9009 12824.4084  0.02%   213  344s
 56249  1057     cutoff   93      12821.9009 12824.4084  0.02%   215  353s
 57566   910 12824.4084   81  120 12821.9009 12824.4084  0.02%   218  361s
 58936   689 infeasible   76      12821.9009 12824.4084  0.02%   219  371s
 60407   532     cutoff   89      12821.9009 12824.4084  0.02%   221  379s
 61744   403 12824.4084  108  123 12821.9009 12824.4084  0.02%   223  387s
 62935   320     cutoff   93      12821.9009 12824.4084  0.02%   224  394s
 63996   276 12824.4084  103  109 12821.9009 12824.4084  0.02%   224  401s
 64860   200     cutoff   97      12821.9009 12824.4084  0.02%   225  407s
 65647   190     cutoff   65      12821.9009 12824.4084  0.02%   226  412s
 66448   156 12824.4084   72   85 12821.9009 12824.4084  0.02%   227  417s
 67059   169 12824.4084   84   97 12821.9009 12824.4084  0.02%   227  423s
 67792    67 12824.4084   88   92 12821.9009 12824.4084  0.02%   228  428s
 68370    16     cutoff  106      12821.9009 12824.4084  0.02%   228  432s

Cutting planes:
  Gomory: 4
```

```
  Projected implied bound: 3
  MIR: 85
  Flow cover: 8
  RLT: 1

Explored 68724 nodes (15775565 simplex iterations) in 434.82 seconds (799.87
work units)
Thread count was 8 (of 96 available processors)

Solution count 10: 12821.9 12821.9 12821.9 ... 11775

Optimal solution found (tolerance 1.00e-04)
Best objective 1.282190092696e+04, best bound 1.282190092696e+04, gap 0.0000%
```

Typical causes of numerical issues

- Scale: too large of a range of numerical coefficients. Setting parameter `NumericFocus` can help but should be avoided. Ideally, this model should be reformulated.

```
Matrix range     [9e-06, 6e+01]
Objective range  [1e+00, 1e+00]
Bounds range     [1e-06, 1e+03]
RHS range        [0e+00, 0e+00]
```

- Rounding of numerical coefficients: Ex: Don't write $1/3$ as 0.333, if possible multiply constraint with 3
- wrong answer with big-M
  - y ≤ 1000000 x
    x binary
    y ≥ 0
  - With default value of `IntFeasTol` (1e−5):
    - x = 0.0000099999, y = 9.9999 is integer feasible!
    - y can be positive without forcing x to 1

Check log file for warnings

- Warning during optimization

```
Warning: 1 variable dropped from basis
Warning: switch to quad precision
Warning: Markowitz tolerance tightened to 0.0625
```

- Warnings after optimization is finished

```
Warning: max integrality violation (5.0000e-05) exceeds
         tolerance
Warning: max SOS violation (9.9353e-05) exceeds tolerance
Warning: max constraint violation (2.5435e-05) exceeds
         tolerance
Warning: max bound violation (2.5435e-05) exceeds tolerance
         (possibly due to large matrix coefficient range)
Numerical trouble encountered
```

References

- CPLEX user manual https://www.ibm.com/docs/en/icos/22.1.1?topic=optimizers-users-manual-cplex
- Gurobi Documentation: https://www.gurobi.com/documentation/
- Gurobi Youtube channel https://www.youtube.com/@GurobiVideos/featured
- Paul Rubin, User Cuts versus Lazy Constraints https://orinanobworld.blogspot.com/2012/08/user-cuts-versus-lazy-constraints.html
- Sonja Mars, Modeling: Best Practices & Techniques. https://assets.gurobi.com/pdfs/user-events/2016-frankfurt/Best-Practices.pdf