Lecture 25 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. So far it is the best-performing MILP solver in the Mittelmann benchmark.
- Gurobi can now solve nonconvex MIQCQP. Starting with Gurobi 11, simple MINLP capability is added.

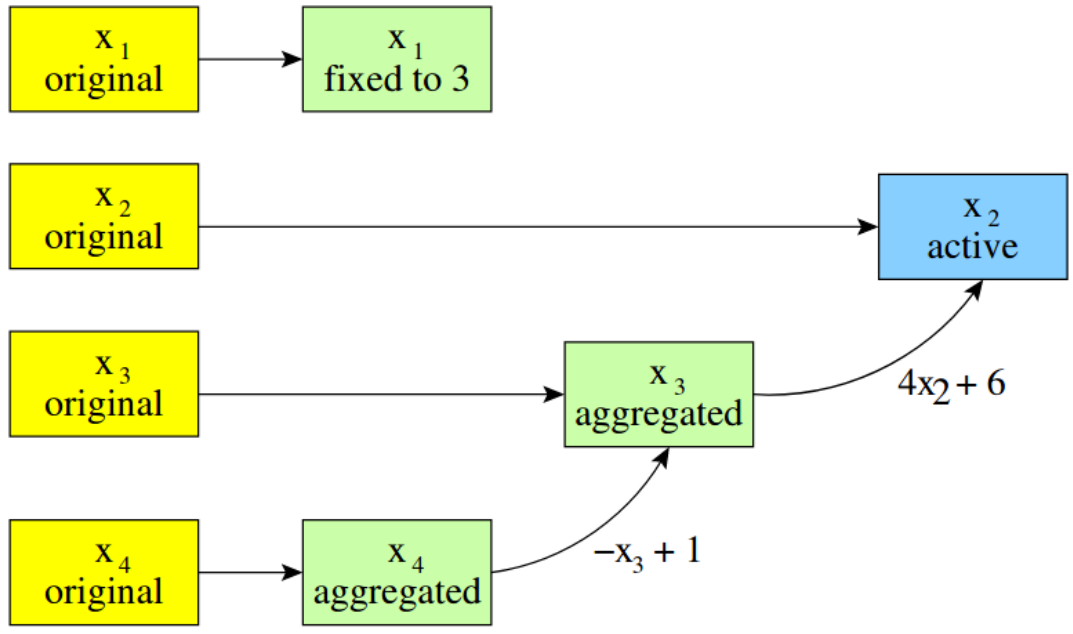
What's inside the box?

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

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

Preprocessing

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



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://www.gurobi.com/documentation/current/refman/cuts.html`

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 $\hat{x}$.
- 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

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

  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:

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

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):

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

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 $x^0 \in P$, 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:

$$\min\{\Delta(x, \hat{x}) \mid x \in P\}$$

- 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).
- 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 $c_1^T x$, $c_2^T x$, ..., $c_n^T x$, 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

`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
```

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