Integer Programming
ISE 418

Lecture 23

Dr. Ted Ralphs

Reading for This Lecture

- Achterburg "Constraint Integer Programming" (2007)
- M.W.P. Savelsbergh "Preprocessing and Probing for Mixed Integer Programming Problems."
- A. Atamturk, G. Nemhauser, and M.W.P. Savelsbergh, "Conflict Graphs in Solving Integer Programming Problems."
- T. Achterberg, R.E. Bixby, Z. Gu, E Rothberg, And D. Weninger, "Presolving Reductions in Mixed Integer Programming."

Preprocessing and Probing

- Often, it is possible to simplify a model using logical arguments.
- Most commercial IP solvers have a built-in preprocessor.
- Effective preprocessing can pay large dividends.
- Let the upper and lower bounds on x_j be u_j and l_j.
- The most basic type of preprocessing is calculating *implied bounds*.
- Let (π, π_0) be a valid inequality.
- If $\pi_1 > 0$, then

$$x_1 \leq (\pi_0 - \sum_{j:\pi_j>0} \pi_j l_j - \sum_{j:\pi_j<0} \pi_j u_j)/\pi_1$$

- If $\pi_1 < 0$, then

$$x_1 \geq (\pi_0 - \sum_{j:\pi_j>0} \pi_j l_j - \sum_{j:\pi_j<0} \pi_j u_j)/\pi_1$$

Basic Preprocessing

- Again, let (π, π_0) be any valid inequality for $\mathcal{S}$.
- The constraint $\pi x \leq \pi_0$ is redundant if

$$\sum_{j:\pi_j>0} \pi_j u_j + \sum_{j:\pi_j<0} \pi_j l_j \leq \pi_0.$$

- $\mathcal{S}$ is empty (IP is infeasible) if

$$\sum_{j:\pi_j>0} \pi_j l_j + \sum_{j:\pi_j<0} \pi_j u_j > \pi_0.$$

- For any IP of the form $\max\{cx | Ax \leq b, l \leq x \leq u\}, x \in \mathbb{Z}^n$,
  - If $a_{ij} \geq 0 \forall i \in [1..m]$ and $c_j < 0$, then $x_j = l_j$ in any optimal solution.
  - If $a_{ij} \leq 0 \forall i \in [1..m]$ and $c_j > 0$, then $x_j = u_j$ in any optimal solution.

Probing for Integer Programs

- It is also possible in many cases to fix variables or generate new valid inequalities based on logical implications.
- Consider (π, π_0), a valid inequality for 0-1 integer program.
- If $\pi_k > 0$ and $\pi_k + \sum_{j:\pi_j<0} \pi_j > \pi_0$, then we can fix x_k to zero.
- Similarly, if $\pi_k < 0$ and $\sum_{j:\pi_j<0, j\neq k} \pi_j > \pi_0$, then we can fix x_k to one.
- Example: Generating logical inequalities

Generation of the Conflict Graph

- As describe earlier, a *conflict* is a pair of variables and associated values that are mutually incompatible.
- For example, we may derive that binary variables x_1 and x_2 cannot both take value 1 simultaneously.
- These conflicts can be generated in a number of ways:
  - during preprocessing;
  - during cut generation; or
  - when the LP relaxation is infeasible;
- Known conflicts can be stored as a *conflict graph* in which the nodes correspond to variable-value pairs and the edges correspond to conflicts.
- The graph can be used to guide branching decisions, fix variable values, etc.

Improving Coefficients

- Suppose again that (π, π_0) is a valid inequality for a 0-1 integer program.
- Suppose that $\pi_k > 0$ and $\sum_{j:\pi_j>0, j\neq k} \pi_j < \pi_0$.
- If $\pi_k > \pi_0 - \sum_{j:\pi_j>0, j\neq k} \pi_j$, then we can set
  - $\pi_k \leftarrow \pi_k - (\pi_0 - \sum_{j:\pi_j>0, j\neq k} \pi_j)$, and
  - $\pi_0 \leftarrow \sum_{j:\pi_j>0, j\neq k} \pi_j$.
- Similarly, suppose that $\pi_k < 0$ and $\pi_k + \sum_{j:\pi_j>0, j\neq k} \pi_j < \pi_0$.
- Then we can again set $\pi_k \leftarrow \pi_0 - \sum_{j:\pi_j>0, j\neq k} \pi_j)$

Preprocessing Based on Problem Structure

- Example: Preprocessing Methods in Set Partitioning
  - Duplicate columns
  - Dominated rows
  - Column is a sum of other columns
  - Extended row clique
  - Singleton row
  - Rows differ by two entries

Preprocessing and Probing in Branch and Bound

- In practice, these rules are applied iteratively until none applies.
- Applying one of the rules may cause a new rule to apply.
- Rules that explicitly use the global lower bound can be reapplied whenever a new incumbent is found.
- Furthermore, all rules can be reapplied after branching.
- These techniques can make a very big difference.

Root Node Processing

- Typically, more effort is put into processing the root node than other nodes in the tree.
- Work done in the root node will impact the processing of every subsequent node.
- Dual bounding
  - Cut generation in the root node can be thought of as an additional pre-processing step to strengthen the formulation before enumeration.
  - Cut generation in the root node can also be used to predict effectiveness of such techniques elsewhere in the tree.
- Primal bounding
  - Primal bounds found in the root node can have a big impact on the search.
  - They help to improvement variable bounds by reduced cost and can also lead to more effective/efficient search strategies.
  - As with cut generation, we use performance in the root node as an indicator of efficacy throughout the tree.

Node Pre/Post-Processing: Bound Improvement by Reduced Cost

- Consider an integer program $\max_{x \in \mathbb{Z}^n}\{cx \mid Ax \leq b, 0 \leq x \leq u\}$.
- Suppose the linear programming relaxation has been solved to optimality and row zero of the tableau looks like

$$z = \bar{a}_{00} + \sum_{j \in NB_1} \bar{a}_{0j} x_j + \sum_{j \in NB_2} \bar{a}_{0j}(x_j - u_j)$$

  where NB_1 are the nonbasic variables at 0 and NB_2 are the nonbasic variables at their upper bounds u_j.
- In addition, suppose that a lower bound $\underline{z}$ on the optimal solution value for IP is known.
- Then in any optimal solution

$$\begin{aligned} x_j &\leq \left\lfloor \frac{\bar{a}_{00} - \underline{z}}{-\bar{a}_{0j}} \right\rfloor \text{ for } j \in NB_1, \text{ and} \\ x_j &\geq u_j - \left\lceil \frac{\bar{a}_{00} - \underline{z}}{\bar{a}_{0j}} \right\rceil \text{ for } j \in NB_2. \end{aligned}$$

Node Pre/Post-Processing: Other Techniques

- Bound improvement in the root node
  - Whenever a new lower bound is found by a heuristic or otherwise, we can apply bound improvement in the root node.
  - To do so, we save the reduced costs of the variables in the root node.
  - We can do this for multiple bases obtained during the processing of the root node.
  - The bound improvements found in this way can be immediately applied to all candidate and active nodes.
- Techniques similar to those applied in the root node can also be applied during the processing of individual nodes.
- New implications may be available once branching constraints are applied.

Node Pre/Post-Processing: Conflict Analysis

- Whenever a node is found to be infeasible, we derive a *conflict*.
- The branching constraints imposed to arrive at that node cannot all be imposed simultaneously.
- These conflicts can be used to derive cuts and may also contribute to enhancement of the conflict graph.