Computational Integer Programming
Universidad de los Andes

Lecture 8

Dr. Ted Ralphs

Reading for This Lecture

- Wolsey Section 9.6
- Nemhauser and Wolsey Section II.6
- Martin "Computational Issues for Branch-and-Cut Algorithms" (2001)
- Linderoth and Ralphs "Noncommercial Software for Mixed-Integer Linear Programming"

Branch and Cut

- *Branch and cut* is an LP-based branch-and-bound scheme in which the linear programming relaxations are augmented by valid inequalities.
- The valid inequalities are generated dynamically using separation procedures.
- We iteratively try to improve the current bound by adding valid inequalities.
- In practice, branch and cut is the method typically used for solving difficult mixed-integer linear programs.
- Computational component of branch and cut
  - Preprocessing
  - Cut generation
  - Managing the LP relaxation
  - Search strategy
  - Branching strategy
  - Primal heuristics

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 S.
- The constraint $\pi x \leq \pi_0$ is redundant if

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

- 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 $\max\{cx | Ax \leq b, l \leq x \leq b\}, x \in \mathbf{Z}^n$,
  - 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.

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 $\pi_k \leftarrow \pi_k - (\pi_0 - \pi_j - \sum_{j:\pi_j>0, j\neq k} \pi_j)$

Bound Improvement by Reduced Cost

- Consider an integer program $\max\{cx \mid Ax \leq b, 0 \leq x \leq u\}$
- 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

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

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.
- Bound improvement by reduced cost can be reapplied whenever a new bound is computed.
- Furthermore, all rules can be reapplied after branching.
- These techniques can make a very big difference.

Preprocessing Based on Problem Structure

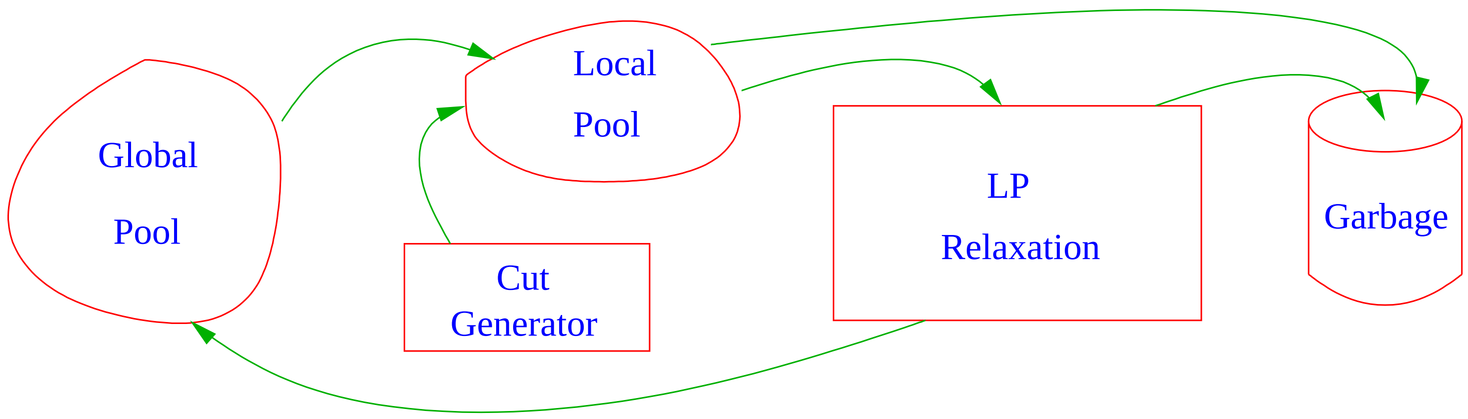
- Example: Preprocessing Methods in Set Partitioning
  - Duplicate columns
  - Dominated rows
  - Column is a sum of other columns
  - Extended row clique
  - Singleton row
  - Rows differ by two entries

Managing the LP Relaxations

- In practice, the number of inequalities generated can be HUGE.
- We must be careful to keep the size of the LP relaxations small or we will sacrifice efficiency.
- This is done in two ways:
  - Limiting the number of cuts that are added each iteration.
  - Systematically deleting cuts that have become *ineffective*.
- How do we decide which cuts to add?
- And what do we do with the rest?
- What is an ineffective cut?
  - One whose dual value is (near) zero.
  - One whose slack variable is basic.
  - One whose slack variable is positive.

Managing the LP Relaxations

- Below is a graphical representation of how the LP relaxation is managed in practice.
- Newly generated cuts enter a buffer (called the *local cut pool*).
- Only a small number of the most violated cuts from the buffer are added each iteration.
- Cuts that prove effective locally are eventually sent to the global pool for future use.



Cut Generation and Management

- A significant question in branch and cut is what classes of valid inequalities to generate and when?
- It is generally not a good idea to try all cut generation procedures on every fractional solution arising.
- For generic mixed-integer programs, cut generation is most important in the root node.
- Using cut generation only in the root node yields a procedure called *cut and branch*.
- Depending on the structure of the instance, different classes of valid inequalities may be effective.
- Sometimes, this can be predicted ahead of time (knapsack inequalities).
- In other cases, we have to use past history as a predictor of effectiveness.
- Generally, each procedure is only applied at a dynamically determined frequency.

Cut Sharing

- Note that cuts generated by the C-G procedure are not globally valid, i.e., at other search tree nodes (why not?).
- Structural cuts generated using problem-specific separation algorithms are globally valid by definition.
- Sometimes these cuts are difficult to generate and some amount of luck may be involved in finding "important" ones.
- The advantage of generating globally valid inequalities is that they can be used later in other search tree nodes.
- This sharing of polyhedral information can help create a much better approximation of the convex hull of solutions.
- This is done through the use of one or more *cut pools*.

Cut Pools

- Cut pools can be used to store cuts that have proven effective for later use.
- The cut pool can be considered as an auxiliary mechanism for performing separation.
- The solver can check the cut pool periodically to see if it contains any inequalities that can separate the current LP solution.
- In this way, the cut pool can be thought of as a global database of polyhedral information that is queried to obtain localized descriptions.
- As already noted, the number of generated cuts can be HUGE, so the cut pool must be carefully maintained (more on that later).

Managing the Cut Pool

- Cut pools can easily grow quite large.
- We need to limit their size for two reasons
  - Memory
  - Efficiency
- We limit the size of the cut pool by
  - Eliminating duplicates.
  - Only allowing in cuts that have already proved effective.
  - Purging cuts that are underutilized or irrelevant.
- How do we judge which cuts are irrelevant?

Searching the Cut Pool

- Searching the cut pool means locating cuts in the list that are violated by a given LP solution.
- Computing the violation of each cut with respect to a given solution can take time.
- We still may not want to search through the entire pool.
- Which cuts do we check?
  - Those that have proven the most effective.
  - Those that were generated "nearby" in the search tree.
- Another idea for increasing the efficiency of the cut pool is to have multiple pools servicing different parts of the tree.

Overview of the Bounding Process