Computational Methods in Optimization
IE496

Lecture 27

Dr. Ted Ralphs

Branch and Bound

- *Branch and bound* is the most commonly-used algorithm for solving MILPs.
- It is a divide and conquer approach.
- Suppose F is the feasible region for some MILP and we wish to solve $\min_{x \in F} c^\top x$.
- Consider a partition of F into subsets $F_1, \ldots F_k$. Then

$$\min_{x \in F} c^\top x = \min_{\{1 \leq i \leq k\}} \{\min_{x \in F_i} c^\top x\}$$

- In other words, we can optimize over each subset separately.
- Idea: If we can't solve the original problem directly, we might be able to solve the smaller *subproblems* recursively.
- Dividing the original problem into subproblems is called *branching*.
- Taken to the extreme, this scheme is equivalent to complete enumeration.

Branch and Bound

- Next, we discuss the role of *bounding*.
- For the rest of the lecture, assume all variables have finite upper and lower bounds.
- Any feasible solution to the problem provides an upper bound $u(F)$ on the optimal solution value.
- We can use approximate methods to obtain an upper bound.
- Idea: After branching, try to obtain a *lower bound* $b(F_i)$ on the optimal solution value for each of the subproblems.
- If $b(F_i) \geq u(F)$, then we don't need to consider subproblem i.
- One easy way to obtain a lower bound is by solving the *LP relaxation* obtained by dropping the integrality constraints.

LP-based Branch and Bound

- In LP-based branch and bound, we first solve the LP relaxation of the original problem. The result is one of the following:

  1. The LP is infeasible $\Rightarrow$ MILP is infeasible.
  2. We obtain a feasible solution for the MILP $\Rightarrow$ optimal solution.
  3. We obtain an optimal solution to the LP that is not feasible for the MILP $\Rightarrow$ lower bound.

- In the first two cases, we are finished.
- In the third case, we must branch and recursively solve the resulting subproblems.

Branching in LP-based Branch and Bound

- The most common way to branch is as follows:
  - Select a variable i whose value $\hat{x}_i$ is fractional in the LP solution.
  - Create two subproblems.
    - In one subproblem, impose the constraint $x_i \leq \lfloor \hat{x}_i \rfloor$.
    - In the other subproblem, impose the constraint $x_i \geq \lceil \hat{x}_i \rceil$.
- Such a method of branching is called a *branching rule*.
- Why is this a valid branching rule?
- What does it mean in a 0-1 integer program?

Continuing the Algorithm After Branching

- After branching, we solve each of the subproblems recursively.
- Now we have an additional factor to consider.
- If the optimal solution value to the LP relaxation is greater than the current upper bound, we need not consider the subproblem further.
- This is the key to the efficiency of the algorithm.
- Terminology
  - If we picture the subproblems graphically, they form a *search tree*.
  - Each subproblem is linked to its *parent* and eventually to its *children*.
  - Eliminating a problem from further consideration is called *pruning*.
  - The act of bounding and then branching is called *processing*.
  - A subproblem that has not yet been considered is called a *candidate* for processing.
  - The set of candidates for processing is called the *candidate list*.

LP-based Branch and Bound Algorithm

1. To start, derive an upper bound U using a heuristic method.
2. Put the original problem on the candidate list.
3. Select a problem S from the candidate list and solve the LP relaxation to obtain the bound $b(S)$.
   - If the LP is infeasible $\Rightarrow$ node can be pruned.
   - Otherwise, if $b(S) \geq U$ $\Rightarrow$ node can be pruned.
   - Otherwise, if $b(S) < U$ and the solution is feasible for the MILP $\Rightarrow$ set $U \leftarrow b(S)$.
   - Otherwise, branch and add the new subproblem to the candidate list.
4. If the candidate list in nonempty, go to Step 2. Otherwise, the algorithm is completed.

Choices in Branch and Bound

- Selecting the next candidate to process.
  - "Best-first" always chooses the candidate with the lowest lower bound.
  - This rule minimizes the size of the tree (why?).
  - There may be practical reasons to deviate from this rule.
- Choosing a branching rule.
  - Branching wisely is extremely important.
  - A "poor" branching can slow the algorithm significantly.
  - We will cover methods of branching in detail in IE418.
- There are also alternative methods of lower bounding, although LP relaxation is the most common.

The Importance of Formulation

- The most vital aspect of branch and bound is obtaining "good" lower bounds.
- In this respect, not all formulations are created equal.
- Choosing the right one is critical.
- A typical MILP can have many alternative formulations.
- Each formulation corresponds to a different polyhedron enclosing the integer points that are feasible for the problem.
- The more closely the polyhedron approximates the convex hull of the integer solutions, the better the bound will be.

Example: Facility Location Problem

- We are given n potential facility locations and m customers.
- There is a fixed cost c_j of opening facility j.
- There is a cost d_{ij} associated with serving customer i from facility j.
- We have two sets of binary variables.
  - y_j is 1 if facility j is opened, 0 otherwise.
  - x_{ij} is 1 if customer i is served by facility j, 0 otherwise.
- Here is one formulation:

$$\min \sum_{j=1}^{n} c_j y_j + \sum_{i=1}^{m} \sum_{j=1}^{n} d_{ij} x_{ij}$$

$$\text{s.t.} \quad \sum_{j=1}^{n} x_{ij} = 1 \qquad \forall i$$

$$\sum_{i=1}^{m} x_{ij} \leq m y_j \quad \forall j$$

$$x_{ij}, y_j \in \{0,1\} \forall i, j$$

Example: Facility Location Problem

- Here is another formulation for the same problem:

$$
\begin{aligned}
\min \quad & \sum_{j=1}^{n} c_j y_j + \sum_{i=1}^{m} \sum_{j=1}^{n} d_{ij} x_{ij} \\
\text{s.t.} \quad & \sum_{j=1}^{n} x_{ij} = 1 && \forall i \\
& x_{ij} \leq y_j && \forall i, j \\
& x_{ij}, y_j \in \{0, 1\} && \forall i, j
\end{aligned}
$$

- Notice that the set of integer solutions contained in each of the polyhedra is the same (why?).
- However, the second polyhedra strictly includes the first one.
- Therefore, the second polyhedra will yield better lower bounds and be better for branch and bound.
- Notice that the second formulation includes more constraints, but will likely solve more quickly.

Formulation Strength and Ideal Formulations

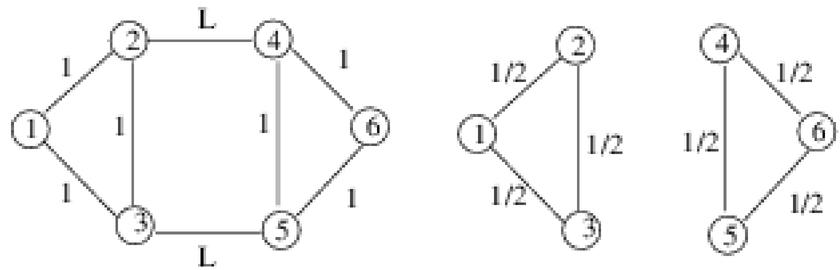
- Consider two formulations A and B for the same ILP.
- Denote the corresponding feasible regions for their LP relaxations as P_A and P_B.
- Formulation A is said to be *at least as strong as* formulation B if $P_A \subseteq P_B$.
- If the inclusion is strict, then A is *stronger than* B.
- If F is the set of all feasible integer solutions for the ILP, then we must have $conv(F) \subseteq P_A$ (why?).
- A is *ideal* if $conv(F) = P_A$

Strengthening Formulations

- Often, a given formulation can be strengthened with additional inequalities satisfied by all feasible integer solutions.
- Example: The Perfect Matching Problem
  - We are given a set of n people that need to paired in teams of two.
  - Let c_{ij} represent the "cost" of the team formed by person i and person j.
  - We wish to minimize cost over all teams.
  - We can represent this problem on an undirected graph $G = (N, E)$.
  - The nodes represent the people and the edges represent pairings.
  - We have $x_e = 1$ if the endpoints of e are matched, $x_e = 0$ otherwise.

$$
\begin{aligned}
min \quad & \sum_{e=\{i,j\}\in E} c_e x_e \\
s.t. \quad & \sum_{\{j | \{i,j\}\in E\}} x_{ij} = 1, && \forall i \in N \\
& x_e \in \{0,1\}, && \forall e = \{i,j\} \in E.
\end{aligned}
$$

Valid Inequalities for Matching



- Consider the graph on the left above.
- The optimal perfect matching has value $L + 2$.
- The optimal solution to the LP relaxation has value 3.
- This formulation can be extremely weak.
- Add the valid inequality $x_{24} + x_{35} \geq 1$.
- Every perfect matching satisfies this inequality.

The Odd Set Inequalities

- We can generalize the inequality from the last slide.
- Consider the cut S corresponding to any odd set of nodes.
- The *cutset* corresponding to S is

$$\delta(S) = \{\{i,j\} \in E | i \in s, j \notin S\}.$$

- An *odd cutset* is any $\delta(S)$ for which $|S|$ is odd.
- Note that every perfect matching contains at least one edge from every odd cutset.
- Hence, each odd cutset of induces a possible valid inequality.

$$\sum_{e \in \delta(S)} x_e \geq 1, S \subset N, |S| odd.$$

Using the New Formulation

- If we add all of the odd set inequalities, the new formulation is ideal.
- However, the number of inequalities is exponential in size.
- Only a small number of these inequalities will be active at the optimal solution.
- Recall the concept of a *constraint generation algorithm*.
- We can generate these inequalities on the fly.
- This can be done efficiently.

Constraint Generation Algorithm for Matching

1. Solve the initial LP relaxation.
2. If the solution is feasible, STOP.
3. Otherwise, look for a violated odd set inequality.
4. Add the inequality and reoptimize from the current basis.
5. Go to Step 2.

Branch and Cut Algorithms

- If we combine constraint generation with branch and bound, we get *branch and cut*.
- The relaxation at each node is strengthened using valid inequalities.
- This increases the lower bound and improves efficiency.
- Branch and cut is the current state of the art for solving ILPs.

The Traveling Salesman Problem

- We are given a set N of *customers*, along with a cost c_{ij} associated with traveling between customers i and j.
- We want to order the customers so that the cost of visiting all customers in the specified order and then returning to the starting point is minimized.
- We consider an undirected graph $G = (N, E)$ where each edge $\{i, j\}$ has associated cost c_{ij}.
- Our problem is to find a minimum cost *Hamiltonian tour* in this graph.
- Integer programming formulation:

$$
\begin{aligned}
& min \sum_{e \in E} c_e x_e && (1)\\
s.t. \quad & \sum_{\{j \mid \{i,j\} \in E\}} x_e = 2 \;\forall i \in N, && (2)\\
& \sum_{\substack{\{i,j\} \in E \\ i \in S, j \notin S}} x_e \geq 2 \;\forall S \subseteq N, |S| > 2, && (3)\\
& x_e \in \{0, 1\} \;\forall e \in E. && (4)
\end{aligned}
$$

Solving the Traveling Salesman Problem

- Constraints (**??**) are called the *subtour elimination constraints*.
- Once again, we see that the number of these constraints is exponential.
- In this case, however, the formulation is not ideal—we must use branch and cut.
- We can solve the LP relaxation by using constraint generation.
  - Solve the LP without constraints (**??**) to obtain $\hat{x}$.
  - Construct a network by associating the capacity $\hat{x}_e$ with each edge e.
  - If the minimum cut in this network has capacity < 2, this corresponds to a violated subtour elimination constraint. Add the constraint to the relaxation and resolve.
  - If the minimum cut in this network has capacity ≥ 2, then all constraints (**??**) are satisfied and the relaxation is solved.
- We can now embed this subroutine inside a branch and bound algorithm to solve the TSP.

A Branch and Cut Algorithm for the TSP

- At each node in the search tree, solve the relaxation (**??**)-(**??**) along with the constraints imposed by branching.
- This LP can be solved using the previously discussed constraint generation algorithm.
- If the optimal solution to the relaxation is not integral, then branch on some fractional variable and continue.
- This branch and cut algorithm will solve reasonably sized instances of the TSP.

Gomory Inequalities

- The *Gomory procedure* is a generic procedure for generating valid inequalities for mixed-integer linear programs.
- It assume no special problem structure.
- Consider a pure integer program with feasible region $\mathcal{P}$ represented in standard form.
- For a given $u \in \mathbb{R}^m$, we have that $uAx = ub$ for all $x \in \mathcal{P} \cap \mathbb{Z}^n$.
- Because $x \geq 0$ for all $x \in \mathcal{P} \cap \mathbb{Z}^n$, it follows that

$$\lfloor uA \rfloor x \leq ub \, \forall x \in \mathcal{P} \cap \mathbb{Z}^n.$$

- Since $\lfloor uA \rfloor \in \mathbb{Z}^n$, it finally follows that

$$\lfloor uA \rfloor x \leq \lfloor ub \rfloor \, \forall x \in \mathcal{P} \cap \mathbb{Z}^n.$$

- This last inequality is called a *Gomory inequality*.

Generating Gomory Inequalities

- Gomory inequalities are easy to generate in LP-based branch and bound.
- If the solution to the current LP relaxation is not feasible, then we must have $(B^{-1}b)_i \notin \mathbb{Z}$ for some i between 1 and m.
- Taking u to be the i^{th} row of B^{-1}, we see that

$$x_l + \sum_{j \in NB} \lfloor ua_j \rfloor x_j \leq \lfloor ub \rfloor, \; \forall x \in \mathcal{P} \cap \mathbb{Z}^n,$$

  where

  - l is the index of the i^{th} basic variable,
  - NB is the set of indices of the nonbasic variables, and
  - a_j is the j^{th} column of A.

- Eliminating x_l from the above inequality using the equation $uAx = ub$ for all $x \in \mathcal{P} \cap \mathbb{Z}^n$, we obtain

$$\sum_{j \in NB} (ua_j - \lfloor ua_j \rfloor) x_j \geq ub - \lfloor ub \rfloor,$$