Integer Programming

Lecture 11

Putting it All Together: Search Strategies

- In the last lecture, we discussed how to *branch*, i.e., divide the feasible region of a subproblem into two pieces.
- After branching, we still have to face the question of what node to process next.
- The strategy for deciding what node to work on next is called the *search strategy*.
- In other words, we are determining the priority function that will be used in the priority queue we use to keep track of the candidate nodes.
- In choosing a search strategy, we might consider our goal:
  - Minimize the time required to find a provably optimal solution.
  - Find the best possible solution in a limited amount of time.
- In practice, we may want some of each.

Basic Strategies: Best First

- A reasonable approach to minimizing overall solution time is to try to minimize the size of the search tree.
- In theory, we can do this by choosing the subproblem with the *best bound* (highest upper bound, if we are maximizing).
- A candidate node is said to be *critical* if its bound exceeds the value of an optimal solution to the IP.
- Every critical node will be processed no matter what the search order.
- Under mild conditions, best first is guaranteed to examine only critical nodes, thereby minimizing the size of the search tree (why?).
- However, it has some drawbacks:
  - Doesn't find feasible solutions quickly (why?).
  - Node setup costs.
  - Memory usage.
  - Fewer variables fixed by reduced cost (more about this later).

What Bound Do We Use?

- We have so far left out one detail: exactly what bound we assign initially to a new candidate subproblem?
- One option is to use the final bound of the parent node, but this does not allow us to distinguish between two children with the same parent.
- A better option is to simply use the same estimate of the bound we computed during branching.
  - If we used strong branching, then use the estimate computed during the pre-solve.
  - If we are using pseudo-cost branching, use that estimate.
- Below, we will also see some alternatives that use estimates of the optimal solution value of the subproblem itself (not the relaxation).

Basic Strategies: Depth First

- The depth first approach is to always choose the "deepest" node to process next.
- This avoids *most* of the problems with best first:
  - The number of candidate nodes is minimized (saving memory).
  - The node set-up costs are minimized.
  - Feasible solutions are found more quickly (why?).
- Unfortunately, if the initial lower bound is not very good, then we may end up processing lots of non-critical nodes.
- We want to avoid this extra expense if possible.

Estimate-based Strategies: Finding Feasible Solutions

- Let's focus on a strategy for finding feasible solutions quickly.
- One approach is to try to estimate the value of the optimal solution

$$z_i = \max_{x \in \mathcal{S}_i} c^\top \mathsf{x}$$

  to each subproblem itself (not the relaxation).
- For any subproblem S_i, let
  - $s_i = \sum_j \min(f_j, 1 - f_j)$ be the sum of the integer infeasibilities,
  - $U(i)$ be the upper bound, and
  - L the global lower bound.
- Also, let S_0 be the root subproblem.
- The *best projection* criterion is

$$E_i = U(i) + \left(\frac{L - U(0)}{s_0}\right) s_i$$

- The *best estimate* criterion uses the pseudo-costs to obtain

$$E_i = U(i) + \sum_j \min\left(P_j^- f_j, P_j^+ (1 - f_j)\right)$$

Interpretation of Best Projection

- Best projection is based on the implicit assumption that there is a linear relationship between s_i and the gap $U(i) - z_i$.
- In order to solve the subproblem, we need to reduce the sum of the integer infeasibilities to zero by, e.g., further branching.
- Reducing the infeasibility reduces the upper bound.
- We try to figure out what the bound will be when the infeasibility is zero and this is our estimate.
- It is not always the case that our assumption about the linear relationship holds, but it seems to hold empirically in some cases.