Computational Integer Programming
Universidad de los Andes

Lecture 7

Dr. Ted Ralphs

Reading for This Lecture

- Linderoth and Savelsbergh (1991)
- Savelsbergh (1994)
- Martin (2001)

Putting it All Together: Search Strategies

- In Lecture 4, 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 choosing a search strategy, we might consider our goal, as in branching:
  - 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).

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 to each subproblem and pick the best.
- For any subproblem S_i, let
  - $s^i = \sum_j \min(f_j, 1 - f_j)$ be the sum of the integer infeasibilities,
  - z_U^i be the upper bound, and
  - z_L the global lower bound.
- Also, let S_0 be the root subproblem.
- The *best projection* criterion is

$$E_i = z_U^i + \left(\frac{z_L - z_U^0}{s^0} \right) s^i$$

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

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

Advanced Strategies: Proving Optimality

- For many combinatorial problems, we can find a "good" solution heuristically.
- In such cases, we are more concerned with minimizing the time to prove optimality.
- To retain the advantages of both best first and depth first search, we can use a combined strategy.
  - Proceed depth-first until the bound in the current node falls below the best bound by more than a given percentage.
  - Proceed depth-first until the difference between the current bound and the best bound is more than a given percentage of the "global gap."
- Note that if we actually knew the value of the optimal solution, then we could simply do pure depth-first search.
- Hence, another strategy is to estimate the optimal solution value and proceed depth-first until the bound falls below the estimate.

Hybrid Strategy

- In cases where we do not have a good feasible solution going, we might also try a *hybrid strategy*.
  - First try to find good feasible solutions.
  - Then switch to proving optimality.

Measuring Progress

- An important question is how we know if we're making progress?
- How much longer will it be until completion?
- The traditional measures of progress are
  - Optimality gap
  - Number of candidate nodes
- These measures are not ideal in many respects.
- Current research is being conducted into what measures are more appropriate.