IE418: Integer Programming

Jeff Linderoth

Department of Industrial and Systems Engineering
Lehigh University

2nd February 2005

Boring Stuff

- Extra Linux Class: 8AM–11AM, Wednesday February 9. Room ???
- Accounts and Passwords
  - `http://coral.ie.lehigh.edu` has user information
  - Please use `ipxx` account when solving your IP problems in COR@L!
  - Use `yppasswd` to change your password
- Homework: Due 2/9. (One week warning)
  - Make a copy of your answers before you hand them in
  - Even better, use LaTeXto write up your answers!
  - A special present for you!

Please don't call on me!

- Name some keys to solving integer programs
  - Relaxations
  - Formulation
  - Formulation so the Relaxation is "good"
- Small 'M's good.
  - Big M's baaaaaaaaaaaaaaaaaaaad.......
- How does branch-and-bound work?

LP-based Branch and Bound Algorithm

1. To start, derive an lower bound L using a heuristic method (if possible).
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 $u(S)$
   - If the LP is infeasible $\Rightarrow$ node can be pruned.
   - Otherwise, if $u(S) \leq L \Rightarrow$ node can be pruned.
   - Otherwise, if $u(S) > L$ and the solution is feasible for the MILP $\Rightarrow$ set $L \leftarrow u(S)$.
   - Otherwise, branch. Add the new subproblems to the list.
4. If the candidate list in nonempty, go to Step 2. Otherwise, the algorithm is completed.

Let's Do An Example

maximize

$$z = 5x_1 + 4x_2 + x_3 + 7x_4$$

subject to

$$\begin{aligned} x_1 + x_2 &\leq 5 \\ x_3 + x_4 &\leq 3 \\ x_1 - x_3 + x_4 &\leq 16 \\ 10x_1 + 6x_2 &\leq 45 \\ x_1, x_2 &\geq 0 \\ x_1, x_2, x_3, x_4 &\in \mathbb{Z} \end{aligned}$$

Solving the Example with B&B

- Your picture(s) here...

The Goal of Branching

- We want to divide the current problem into two or more subproblems that are easier than the original.
- We would like to choose the branching that minimizes the sum of the solution times of all the created subproblems.
  - This is the solution of the *entire subtree* rooted at the node.
- How do we know how long it will take to solve each subproblem?
  - **Answer**: We don't.
  - **Idea**: Try to predict the difficulty of a subproblem.

A Good Branching

- Imagine that when I branch, the value of the linear programming relaxation changes *a lot*!
  - I can prune the node, or should be able to prune it quickly
- So, for a given potential branching, I would like to know the upper bound that would result from processing each subproblem.
  - The branching that changes these bounds "the most" is the best branching.

Be Creative!

What are some ideas you have for deciding on a branching variable?

Predicting the Difficulty of a Subproblem

- How can I (quickly?) estimate the upper bounds that would result?
  - Partially solve the LP relaxation in each of the subproblems by performing a given number of dual simplex pivots.
  - Since we are using dual simplex, this gives us a valid bound. **Why?**
- This technique is usually called **strong branching**.
- A cheaper alternative is to use **pseudo-costs**.

Strong Branching Details

- In the case of strong branching, it may be too expensive to evaluate all possible candidates for branching.
- How do we choose the candidates to evaluate?
  - We choose them based on an estimate of their effectiveness that is very cheap to evaluate.
  - One method is to choose inequalities whose left hand side is furthest from being an integer
  - For 0-1 variables, this means those whose values are closest to 0.5.
  - We might also account for the size of the objective function coefficient.

Strong Branching Details

- The number of candidates to evaluate must be determined empirically.
    - Effective branching is more important near the top of the tree.
    - We might want to evaluate more candidates near the top of the tree.
    - More candidates almost always results in smaller trees, but the expense eventually causes an increase in running time.
- How many dual simplex pivots should we do?

Using Pseudo Costs

- The **pseudo-cost** of a variable is an estimate of the per-unit change in the objective function from forcing the value of the variable to be rounded up or down. Like a gradient!
- For each variable x_j, we maintain an up and a down pseudo-cost, denoted P_j^+ and P_j^-.
- Let f_j be the current (fractional) value of variable x_j.
- An estimate of the change in objective function in each of the subproblems resulting from branching on x_j is given by

$$
\begin{aligned}
D_j^+ &= P_j^+(1 - f_j), \\
D_j^- &= P_j^- f_j.
\end{aligned}
$$

- The question is how to get the pseudo-costs.

Obtaining and Updating Pseudo Costs

- Typically, the pseudo-costs are obtained from empirical data.
  - We observe the actual change that occurs after branching on each one of the variables and use that as the pseudo-cost.
- We can either choose to update the pseudo-cost as the calculation progresses or just use the first pseudo-cost found.
  - Several authors have noted that the pseudo-costs tend to remain fairly constant.
- The only remaining question is how to initialize. Possibilities:
  - Use the objective function coefficient.
  - Use the average of all known pseudo-costs.
  - Explicity initialize the pseudocosts using strong branching

What Does "The Most" Mean

- If we are doing typical variable branching, we create two children and have estimates of the amount the bound will change for each child
- How do we combine the two nunbers together to form one measure of goodness for a potential branch?
- Suggest to branch on the variable

$$j^* = \arg\max\{\alpha_1 \min\{D_j^+, D_j^-\} + \alpha_2 \max\{D_j^+, D_j^-\}.$$

- $\alpha_2 = 0 \Rightarrow$ we want to maximize the minimum degradation on the branch
- $(\alpha_1, \alpha_2) = (2, 1)$ seems pretty good

Putting it All Together

- Here are the choices we've discussed in branching:
  - Should we use strong branching or pseudo-costs?
  - Pseudo-costs
    - How should we initialize?
    - How should we update?
  - Strong branching
    - How do we choose the list of branching candidates?
    - How many pivots to do on each?
  - Once we have the bound estimates, how do we choose the final branching?
- Ultimately, we must use empirical evidence and intuition to answer these questions.

Priorities

How Much Do You Know?

You are smarter than integer programming!

- If you have problem specific knowledge, use it to determine which variable to branch ong
- Branch on the important variables first
  - First decide which warehouses to open, then decide the vehicle routing
  - Branch on earlier (time-based) decisions first.
- There are mechanisms for giving the variables a priority order, so that if two variables are fractional, the one with the high priority is branched on first
- Or, first branch on *all* these variables before you branch on the next class, etc.

GUB/SOS1 Branching

- $x_j \in \{0, 1\} \ \forall j$

$$\sum_{j=1}^{10000} x_j = 1$$

Which branching do you think would be better?

1. $x_1 = 1$ & $x_1 = 0 (\Rightarrow \sum_{j=2}^{10000} = 1)$, or
2. $\sum_{j=1}^{500} x_j = 1$ & $\sum_{j=501}^{10000} x_j = 1$

- The answer is It depends
- But the answer is almost assuredly (2).
- It is probably even better to look at the (infeasible) LP relaxation and "put 1/2 on each side" (Just don't break it in the middle)

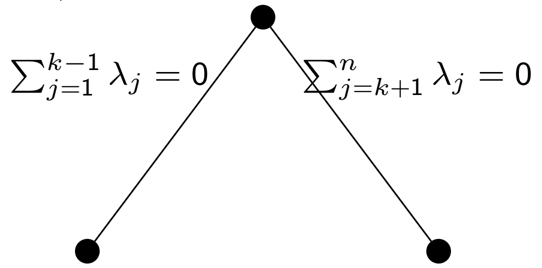
SOS2 Branching

- $\{\lambda_1, \lambda_2, \ldots \lambda_{100}\}$ is an SOS2
- Suppose:
  - $\lambda_1 = 0.2$
  - $\lambda_6 = 0.1$
  - $\lambda_8 = 0.3$
  - $\lambda_{10} = 0.1$
  - $\lambda_{17} = 0.05$
  - $\lambda_{99} = 0.25$

The \$64 Question

How would you branch?

- If $\lambda_k > 0$, then feasible solutions have $\lambda_1 = \cdots = \lambda_{k-1} = 0$, or $\lambda_{k+1} = \cdots = \lambda_{100} = 0$



- Plus the (infeasible) point is excluded on both branches.

Choices in Branch and Bound **Node Selection**

- Another important parameter to consider in branch and bound is the strategy for selecting the next subproblem to be processed.
- In choosing a search strategy, we might consider two different goals:
  - Minimizing overall solution time.
  - Finding a good feasible solution quickly.

The Best First Approach

- One way to minimize overall solution time is to try to minimize the size of the search tree.
  - We can achieve this choose 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 solution to the IP.
- Every critical node will be processed no matter what the search order.
- Best first is guaranteed to examine only critical nodes, thereby minimizing the size of the search tree.

Drawbacks of Best First

- Doesn't necessarily find feasible solutions quickly
  - Feasible solutions are "more likely" to be found deep in the tree
- Node setup costs
  - The linear program being solved may change quite a bit more one iteration to the next
- Memory usage.
  - It can require a lot of memory to store the candidate list

The Depth First Approach

- The depth first approach is to always choose the deepest node to process next.
  - Just dive until you prune, then back up and go the other way
- 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
    - LPs change very little from one iteration to the next
  - Feasible solutions are usually found quickly
- 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 $E_i = z_U^i + \sum_j \min\left(P_j^- f_j, P_j^+(1 - f_j)\right)$

Next Time:

- You should read (as review, and for more information) N&W II.4.1, II.4.2
- Introduction to IP software
- Who knows what an MPS file is?