IE418: Integer Programming

Jeff Linderoth

Department of Industrial and Systems Engineering
Lehigh University

31st January 2005

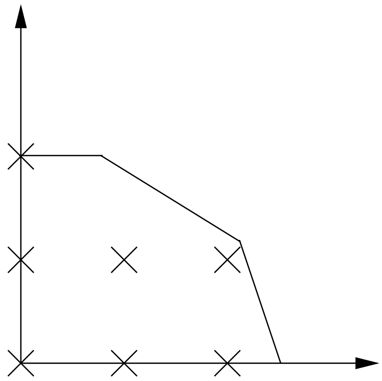
Review

- Name an application for modeling a set covering problem?
  - What *is* a set-covering problem?
- What is TSP?
  - How to model "connected"?
- What is an SOS2
  - What are they used for?
- Recite the "slide of tricks" from memory.

- Any questions on the homeworks?

A Pure Integer Program

$$z(S) = \max\{c^T x : x \in S\}, \qquad S = \{x \in \mathcal{Z}^n_+ : Ax \le b\}$$

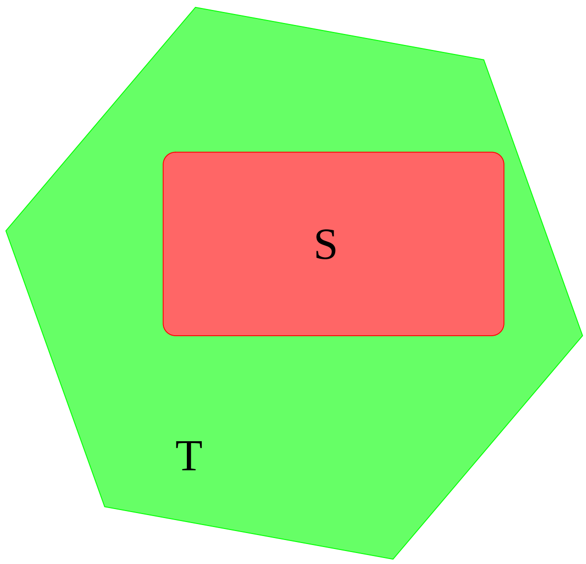


$$\begin{aligned} S &= \{(x_1, x_2) \in \mathcal{Z}^2_+ : 6x_1 + x_2 \le 15, \\ &\quad 5x_1 + 8x_2 \le 20, x_2 \le 2\} \\ &= \{(0,0), (0,1), (0,2), (1,0), \\ &\quad (1,1), (1,2), (2,0)\} \end{aligned}$$

- Note: $z(S) = z(\text{conv}(S))$
  - What *is* conv(S)??

Review

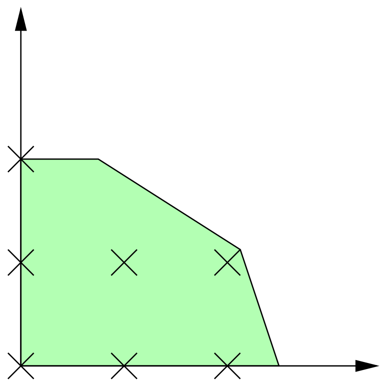
- $z_S \stackrel{\text{def}}{=} \min f(x) : x \in S$
- $z_T \stackrel{\text{def}}{=} \min f(x) : x \in T$



- What can we say about z_S and z_T?
- If $x^*_T = \arg\min f(x) : x \in T$
- And $x^*_T \in S$, then
- $x^*_T = \arg\min f(x) : x \in S$

How to Solve Integer Programs?

- Relaxations!
  - $\hat{S} \supseteq S \Rightarrow z(S) \leq z(\hat{S})$
  - $\hat{x}$ optimal with $\hat{S}$, $\hat{x} \in S \Rightarrow \hat{x}$ optimal with S.
  - People commonly use the linear programming relaxation:



$$z(LP(S)) = \max\{c^T x : x \in LP(S)\},$$
$$LP(S) = \{x \in \mathbb{R}^n_+ : Ax \leq b\}$$

- If $LP(S) = \text{conv}(S)$, we are done.

- We need only know conv(S) in the direction of c.
- The "closer" $LP(S)$ is to conv(S) the better.

GREAT Formulations

- There are a number of integer programs for which $LP(S) = \text{conv}(S)$.
  - The Assignment Problem
  - Spanning Tree Problem
  - Matching Problem

$$S = \{x \in \{0,1\}^{|N| \times |N|} \mid \sum_{i \in N} x_{ij} = 1 \; \forall j \in N, \sum_{j \in N} x_{ij} = 1 \; \forall i \in N\}$$

$$LP(S) = \{x \in \mathbb{R}_+^{|N| \times |N|} \mid \sum_{i \in N} x_{ij} = 1 \; \forall j \in N, \sum_{j \in N} x_{ij} = 1 \; \forall i \in N\}$$

- conv$(S) = LP(S)$
  - We can solve the (IP) Assignment problem by solving its LP relaxation.
- Why is this not surprising?

Solving IPs—The 3 Most Important Things

1. Formulation
2. Formulation
3. Formulation

- PPP (Production Planning Problem).
- Suppose we wish to add the constraint that we wish to make at most two products.
  - (At most two of the five x_j can be positive).

Short Modeling Review

- $z_j = \begin{cases} 1 & \text{Make product } j \\ 0 & \text{Otherwise} \end{cases}$
  - $x_j > 0 \Rightarrow z_j = 1$
  - $x_j \geq \epsilon \Rightarrow z_j = 1$
  - $\sum_{j \in N} a_j x_j \geq b \Rightarrow \delta = 1 \Leftrightarrow \sum_{j \in N} a_j x_j - (M + -\epsilon + \epsilon)\delta \leq b - \epsilon$
  - $x_j \leq M z_j$
- Add constraints
  - $x_j \leq M_j z_j \quad \forall j = 1, 2, \ldots 5.$
    - Note: There is no need for all of the M's to be the same.
  - $\sum_{j=1}^{5} z_j \leq 2.$

What about the M's?

- $M_i = 10^4 \quad \forall i = 1, 2, \ldots 5$?
- Can we make M_i smaller?

$$\begin{array}{rcll} 12x_1 + 20x_2 + 0x_3 + 25x_4 + 15x_5 & \leq & 288 & \text{(Grinding)} \\ 10x_1 + 8x_2 + 16x_3 + 0x_4 + 0x_5 & \leq & 192 & \text{(Drilling)} \\ 20x_1 + 20x_2 + 20x_3 + 20x_4 + 20x_5 & \leq & 384 & \text{Final Assembly} \\ x_i & \geq & 0 & \forall i = 1, 2, \ldots 5 \end{array}$$

A Key Point.

Small M's good, Big M's baaaaaaaaaaaaaaaaaaaaaaad!

Small M's Good. Big M's Baaaaaaaaaaaaaaaaad!

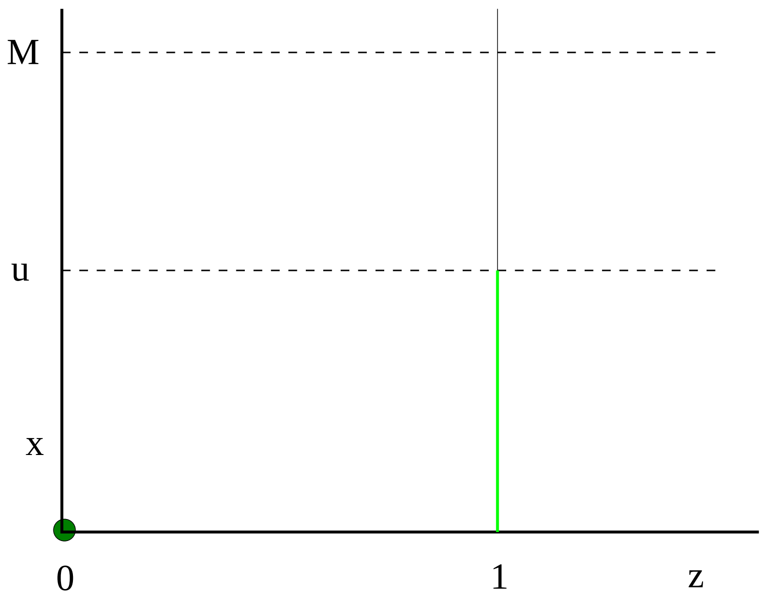
- Let's look at the geometry.

$$P = \{x \in \mathbb{R}_+, z \in \{0, 1\} : x \leq Mz, x \leq u\}$$

$$LP(P) = \{x \in \mathbb{R}_+, z \in [0, 1] : x \leq Mz, x \leq u\}$$

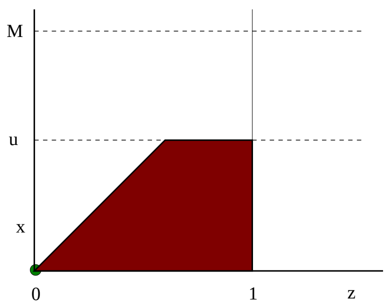
$$\text{conv}(P) = \{x \in \mathbb{R}_+, z \in \{0, 1\} : x \leq uz\}$$

P

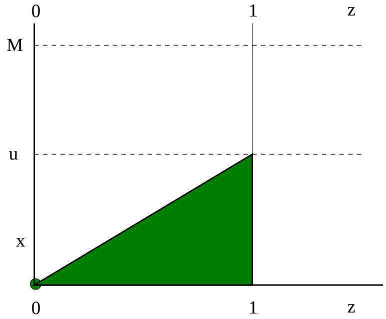


$$P = \{x \in \mathbb{R}_+, z \in \{0, 1\} : x \leq Mz, x \leq u\}$$

LP Versus Conv



$$LP(P) = \{x \in \mathbb{R}_+, z \in [0, 1] : x \leq Mz, x \leq u\}$$



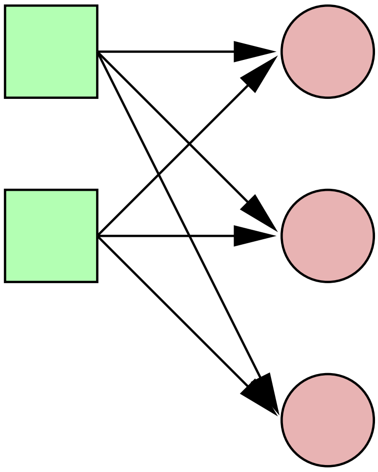
$$\text{conv(P)} = \{x \in \mathbb{R}_+, z \in [0, 1] : x \leq uz\}$$

A Key Point.

If $M = u$, $LP(P) = \text{conv}(P)$!

UFL: Uncapacitated Facility Location

- Facilities: I
- Customers: J



$$\min \sum_{j \in J} f_j x_j + \sum_{i \in I} \sum_{j \in J} f_{ij} y_{ij}$$

$$\sum_{j \in N} y_{ij} = 1 \quad \forall i \in I$$

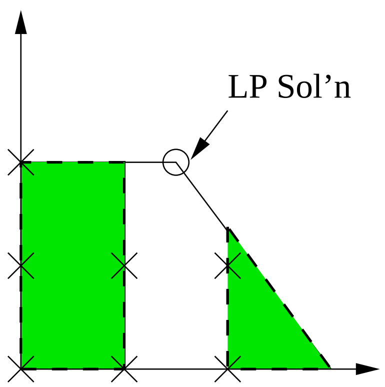
$$\sum_{i \in I} y_{ij} \leq |I| x_j \quad \forall j \in J \qquad (4)$$

OR $y_{ij} \leq x_j \quad \forall i \in I, \ j \in J \qquad (5)$

- Which formulation is to be preferred?
- $I = J = 40$. Costs random.
  - Formulation 1. 53,121 seconds, optimal solution.
  - Formulation 2. 2 seconds, optimal solution.

Feeling Lucky?

- What if we don't get an integer solution to the relaxation?
- Branch and Bound!



- Lots of ways to divide search space. People usually...
  - Partition the search space into two pieces
  - Change bounds on the variables to do this. The LP relaxations remain easy to solve.

Branch-and-Bound

- Branch-and-bound is a divide-and-conquer approach.
- Suppose S is the feasible region for some MILP: $z_{IP} \stackrel{\text{def}}{=} \max_{x \in S} c^T x$
- Consider a partition of S into subsets $S_1, \dots S_k$. Then

$$\max_{x \in S} c^T x = \max_{\{1 \le i \le k\}} \{\max_{x \in S_i} c^T x\}$$

- In other words, we can optimize over each subset separately.
- Dividing the original problem into subproblems is called branching

The Importance of Bounding

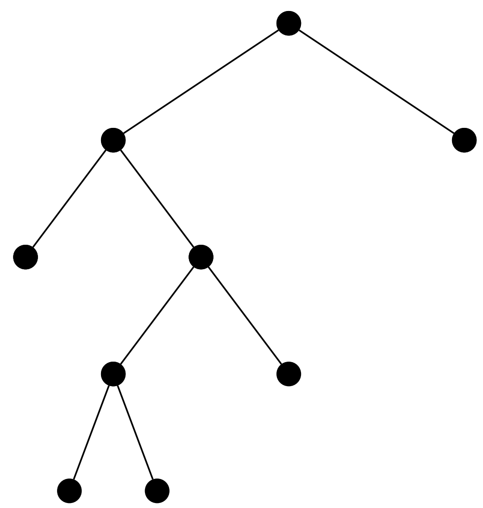
- Any feasible solution to the problem provides an lower bound L on the optimal solution value. $(\hat{x} \in S \Rightarrow z_{IP} \ge c^T \hat{x})$.
  - We can use approximate methods to obtain an lower bound.
- After branching, we obtain an upper bound $u(S_i)$ on the optimal solution value for each of the subproblems. (Why?)
  - Overall Bound: $U^t = \max_i u(S_i)$
- If $u(S_i) \le L$, then we don't need to consider subproblem i.
- We get the upper bound by solving the LP relaxation, but there are other ways too.

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 in unbounded ⇒ the MILP is unbounded. $(z_{IP} = \infty)$
  2. The LP is infeasible ⇒ MILP is infeasible. $(S = \emptyset)$
  3. We obtain a feasible solution for the MILP ⇒ it is an optimal solution to MILP. $(L = z_{IP} = U)$
  4. We obtain an optimal solution to the LP that is not feasible for the MILP ⇒ Upper Bound. $(U = z_{LP})$.
- In the first three cases, we are finished.
- In the final case, we must **branch** and recursively solve the resulting subproblems.

Terminology

- If we picture the subproblems graphically, they form a **search tree**.
- 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 processed is called a **candidate**.
- The set of candidates is the **candidate list**.



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.

The "Global" upper bound

$U^t = \max_{S \text{ is in candidate list at step } t} u(\text{parent}(S))$

Choices in Branch and Bound. **Bounding**

- Lower Bound
  - This is often called a primal heuristic.
    - Rounding, Diving, etc.
  - Often heuristics are problem dependent.
    - How do you communicate your heuristic to the IP solver?
  - Can use *metaheuristics*—Simulated Annealing, Tabu Search, Genetic Algorithms, etc...
- Upper Bound
  - Tighter is better!
  - You read about one way to tighten the relaxation—Preprocessing.
  - We will spend a good amount of time speaking of ways to "tighten" the LP relaxation.
  - Others include Lagrangian relaxation, duality-based, ...

You read N&W, I.1, right!?

Choices in Branch-and-Bound: **Branching**

- If our "relaxed" solution $\hat{x} \notin S$, we must decide how to partition the search space into smaller subproblems
- Our strategy for doing this is called a **Branching Rule**
  - Branching wisely is *very* important
  - It is most important at the top of the branch and bound tree
- $\hat{x} \notin S \Rightarrow \exists j \in N$ such that $f_j \stackrel{\text{def}}{=} \hat{x}_j - \lfloor \hat{x}_j \rfloor > 0$
- So create two problems with additional constraints
  1. $x_j \leq \lfloor \hat{x}_j \rfloor$ on one branch
  2. $x_j \geq \lceil \hat{x}_j \rceil$ on other branch

More Branching Info

- In the case of 0-1 IP, this dichotomy reduces to
  1. $x_j = 0$ on one branch
  2. $x_j = 1$ on other branch
- In general IP, branching on a variable involves imposing new bound constraints in each one of the subproblems.
- This is easily handled implicitly in most cases. Why?
- This is (by far) the most common method of branching.

Let's Do An Example

maximize

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

subject to

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

A Software Interlude...

- In this class, you will do computing.
- It will be easiest if you do your computing in COR@L Lab
  - Computation Optimization Research @ Lehigh
  - Room 362 Mohler
  - `http://coral.ie.lehigh.edu`
- For those of you who are Linux Neophytes, I want to schedule a training session.
- I will be passing around a signup sheet...
  1. Name
  2. Do you want to take training class?
  3. What three hour periods in the next 7-10 days can you *not* do a traning class?

More Software Stuff

- COR@L has lots of cool IP software like, CPLEX, XPRESS-MP, COIN-OR, MINTO, symphony, and AMPL
- More software "coming soon".
- For the time being, I'll assume you know AMPL, since I need to use *something* to demonstrate branch and bound

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.

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.

Other important branching features

- Priority Order
  - You often want to order the variables, so that important variables are branched on first.
  - First decide which warehouses to open, then decide the vehicle routing
  - Branch on earlier (time-based) decisions first.
- GUB or SOS Branching

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

- Software for solving IPs
- A few more examples of solving
- Start working on the homework!
  - I'll probably give you more homework to do

Read Please!

- Earthshattering, Groundbreaking, Seminal Papers to read.
- (They are on the course web page).
  - J. T. Linderoth and M. W. P. Savelsbergh, "A Computational Study of Branch and Bound Search Strategies for Mixed Integer Programming," INFORMS Journal on Computing, 11 (1999) pp. 173-187.
  - A. Atamtürk and M. W. P. Savelsbergh, "Integer Programming Software Systems", *Annals of Operations Research*, forthcoming.
  - J. T. Linderoth and T. K. Ralphs, "Noncommercial Software for Mixed-Integer Linear Programming", Technical Report 04T-023, Department of Industrial and Systems Engineering, Lehigh University, December, 2004.
- If you don't think I'll ask questions about these papers on the mid-term, Just Try Me! :-)