

Computational Integer Programming

JEFF LINDEROTH

ISE Department
COR@L Lab
Lehigh University
jtl13@lehigh.edu



Enterprise-Wide Optimization (EWO) Tele-Seminar
Allentown, PA
January 17, 2007



Jeff's Theory of Teaching

"Children need encouragement. So when a kid gets an answer right, tell him it was a lucky guess. That way, the child develops a good, lucky feeling."

-Jack Handey



The Problem of the Day

(Linear) Mixed-Integer Programming Problem: (MIP)

$$\max\{c^T x + h^T y \mid Ax + Gy \leq b, x \in \mathbb{Z}_+^n, y \in \mathbb{R}_+^p\}$$

Applications

Too numerous to mention



The Knapsack Problem



- A burglar has a knapsack of size b . He breaks into a store that carries a set of items N . Each item has profit c_j and size a_j .
- What items should the burglar select in order to optimize his heist?

$$x_j = \begin{cases} 1 & \text{Item } j \text{ goes in the knapsack} \\ 0 & \text{Otherwise} \end{cases}$$

$$z_{\text{HEIST}} = \max\left\{ \sum_{j \in N} c_j x_j : \sum_{j \in N} a_j x_j \leq b, x_j \in \{0, 1\} \forall j \in N \right\}.$$

- **Integer Knapsack Problem:**

$$z_{\text{HEIST}} = \max\left\{ \sum_{j \in N} c_j x_j : \sum_{j \in N} a_j x_j \leq b, x_j \in \mathbb{Z}_+ \forall j \in N \right\}.$$



Fall into the...



- Given m machines and n jobs, find a least cost assignment of jobs to machines not exceeding the machine capacities
- Each job j requires a_{ij} units of machine i 's capacity b_i

$$\begin{aligned} \min z &\equiv \sum_{i=1}^m \sum_{j=1}^n c_{ij} x_{ij} \\ \text{s.t. } \sum_{j=1}^n a_{ij} x_{ij} &\leq b_i \quad \forall i \quad (\text{Machine Capacity}) \\ \sum_{i=1}^m x_{ij} &= 1 \quad \forall j \quad (\text{Assign all jobs}) \\ x_{ij} &\in \{0, 1\} \quad \forall i, j \end{aligned}$$



Selecting from a Set

- We can use constraints of the form $\sum_{j \in T} x_j \geq 1$ to represent that **at least one** item should be chosen from a set T .
 - Similarly, we can also model that **at most one** or **exactly one** item should be chosen.
- **Example:** Set covering problem
- If A in a 0-1 matrix, then a set covering problem is any problem of the form

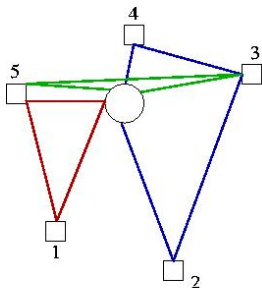
$$\begin{aligned} \min \quad & c^T x \\ \text{s.t.} \quad & Ax \geq e^1 \\ & x_j \in \{0, 1\} \quad \forall j \end{aligned}$$

- **Set Packing:** $Ax \leq e$
- **Set Partitioning:** $Ax = e$



¹It is common to denote the vector of 1's as e

Vehicle Routing



	x_1	x_2	x_3	...	
Customer 1 :	1	0	0	\vdots	= 1
Customer 2 :	0	1	0	\vdots	= 1
Customer 3 :	0	1	1	\vdots	= 1
Customer 4 :	0	1	0	\vdots	= 1
Customer 5 :	1	0	1	\vdots	= 1

- This is a **very** flexible modeling trick
- You can list **all feasible routes**, allowing you to handle “weird” constraints like time windows, strange precedence rules, nonlinear cost functions, etc.



The Farmer's Daughter

- This is *The Most Famous Problem in Combinatorial Optimization!*
- A traveling salesman must visit all his cities at minimum cost.
- Given directed (complete) graph with node set N . ($G = (N, N \times N)$)
- Given costs c_{ij} of traveling from city i to city j
- Find a minimum cost **Hamiltonian Cycle** in G
- **Variables:** $x_{ij} = 1$ if and only if salesman goes from city i to city j



TSP (cont.)

$$\min \sum_{i \in N} \sum_{j \in N} c_{ij} x_{ij}$$

$$\sum_{i \in N} x_{ij} = 1 \quad \forall j \in N \quad \text{Enter Each City}$$

$$\sum_{j \in N} x_{ij} = 1 \quad \forall i \in N \quad \text{Leave Each City}$$

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

Subtour elimination constraint:

(“No Beaming”)

$$\sum_{i \in S} \sum_{j \notin S} x_{ij} \geq 1 \quad \forall S \subseteq N, 2 \leq |S| \leq |N| - 2$$

Alternatively:

(“No Beaming”)

$$\sum_{i \in S} \sum_{j \in S} x_{ij} \leq |S| - 1 \quad \forall S \subseteq N, 2 \leq |S| \leq |N| - 2$$



TSP Trivia Time!

What is This Number?

101851798816724304313422284420468908052573419683296
8125318070224677190649881668353091698688.

- Is this...
 - a) The number of gifts that Jacob Linderoth's grandparents bought him for Christmas?
 - b) The number of subatomic particles in the universe?
 - c) The number of subtour elimination constraints when $|N| = 299$.
 - d) All of the above?
 - e) None of the above?



Answer Time

- The answer is (e). (a)–(c) are all too small (as far as I know) :-). (It is (c), for $|N| = 300$).
- “Exponential” is **really** big.
- Yet people have solved TSP’s with $|N| > 16,000!$
- You will learn how to solve these problems too!
- The “trick” is to only add the subset of constraints that are necessary to prove optimality.
 - This is a trick known as **branch-and-cut**, where the inequalities are added only as needed



Modeling a Restricted Set of Values

- We may want variable x to only take on values in the set $\{a_1, \dots, a_m\}$.
- We introduce m binary variables $y_j, j = 1, \dots, m$ and the constraints

$$x = \sum_{j=1}^m a_j y_j, \quad \sum_{j=1}^m y_j = 1, \quad y_j \in \{0, 1\} \quad \forall j = 1, 2, \dots, m$$

- The set of variables $\{y_1, y_2, \dots, y_m\}$ is called a **special ordered set (SOS)** of variables.
- The a_1, a_2, \dots, a_m defines the order. (The **reference row**).



Example—Building a warehouse

- Suppose we are modeling a facility location problem in which we must decide on the size of a warehouse to build.
- The choices of sizes and their associated cost are shown below:

Size	Cost
10	100
20	180
40	320
60	450
80	600

Warehouse sizes and costs



Warehouse Modeling

- Using binary decision variables x_1, x_2, \dots, x_5 , we can model the cost of building the warehouse as

$$\text{COST} \equiv 100x_1 + 180x_2 + 320x_3 + 450x_4 + 600x_5.$$

- The warehouse will have size

$$\text{SIZE} \equiv 10x_1 + 20x_2 + 40x_3 + 60x_4 + 80x_5,$$

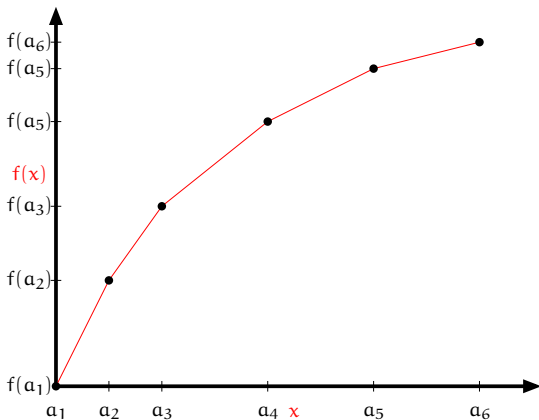
- and we have the SOS constraint

$$x_1 + x_2 + x_3 + x_4 + x_5 = 1.$$



Piecewise Linear Cost Functions

- We can use binary variables to model arbitrary piecewise linear functions.
- The function is specified by ordered pairs $(a_i, f(a_i))$



SOS2

- This is typically modeled using **special ordered sets of type 2**

SOS2

A set of variables of which at most two can be positive. If two are positive, they must be adjacent in the set.

$$\min \sum_{i=1}^k \lambda_i f(\mathbf{a}_i)$$

$$\text{s.t. } \sum_{i=1}^k \lambda_i = 1$$

$$\lambda_i \geq 0$$

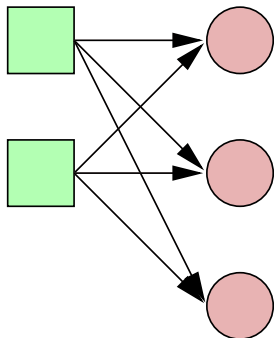
$$\{\lambda_1, \lambda_2, \dots, \lambda_k\} \quad \text{SOS2}$$

- The adjacency conditions of SOS2 are enforced by the solution algorithm
- (All) commercial solvers allow you to specify SOS2



The Impact of Formulation: UFL

- Facilities: I
- Customers: J



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

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

$$\sum_{i \in I} y_{ij} \leq |I| x_j \quad \forall j \in J \quad (1)$$

$$\text{OR } y_{ij} \leq x_j \quad \forall i \in I, j \in J \quad (2)$$

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



Preprocessing

Bound Tightening

- Examine coefficient matrix and bounds on variables and “deduce” if constraints are redundant or bounds on variables can be tightened.
- For example (if x binary, y continuous),

$$3x_1 + x_2 + y \leq 10 \Rightarrow y \leq 6.$$

- Similar techniques to those used in linear programming
- Brearley et al. [1975] is a good reference for this



More Preprocessing

Coefficient Reduction

If there is a binary knapsack row

$$\sum_{j=1}^n a_{ij}x_j \leq b_i$$

and by looking at variable bounds, one can establish that

$$\sum_{j=1, j \neq k}^n a_{ij}x_j \leq b_i - \delta, \forall \text{ feasible } x,$$

then replace constraint with

$$\sum_{j=1, j \neq k}^n a_{ij}x_j + (a_k - \delta)x_k \leq b_i - \delta$$



More Preprocessing

Probing

- Tentatively fix a variable to 0 or 1, and then do “preprocessing” again.
- (Can be) expensive operation
- Can learn **logical** implications between variables.
- Used for inequalities and heuristics

Reduced Cost Fixing

- Use duality information from LP solution to show that some (non-basic) variables must remain fixed at their current (integer) values in every optimal solution
- See Savelsbergh [1994] for a good reference on preprocessing



The Bag of Tricks



- There are **lots** of things you can model with binary variables:
 - Fixed-charge
 - Either-or
 - If-then
 - Limiting cardinality of positive variables
 - Economies of scale
- But sometimes it's hard to **derive** the models



The Slide of Tricks. Indicator Variables...

- $\delta = 1 \Rightarrow \sum_{j \in N} a_j x_j \leq b$
 - $\sum_{j \in N} a_j x_j + M\delta \leq M + b$
- $\sum_{j \in N} a_j x_j \leq b \Rightarrow \delta = 1$
 - $\sum_{j \in N} a_j x_j - (m - \epsilon)\delta \geq b + \epsilon$
- $\delta = 1 \Rightarrow \sum_{j \in N} a_j x_j \geq b$
 - $\sum_{j \in N} a_j x_j + m\delta \geq m + b$
- $\sum_{j \in N} a_j x_j \geq b \Rightarrow \delta = 1$
 - $\sum_{j \in N} a_j x_j - (M + \epsilon)\delta \leq b - \epsilon$

Definitions

- δ : Indicator variable ($\delta \in \{0, 1\}$).
- M : Upper bound on $\sum_{j \in N} a_j x_j - b$
- m : Lower bound on $\sum_{j \in N} a_j x_j - b$
- If $a_j \in \mathbb{Z}$, $x_j \in \mathbb{Z}$, then we can take $\epsilon = 1$, else let $\epsilon = 0$



A More Realistic Example

- PPP—Production Planning Problem. (A simple linear program).
- An engineering plant can produce five types of products: p_1, p_2, \dots, p_5 by using two production processes: grinding and drilling. Each product requires the following number of hours of each process, and contributes the following amount (in hundreds of dollars) to the net total profit.

	p_1	p_2	p_3	p_4	p_5
Grinding	12	20	0	25	15
Drilling	10	8	16	0	0
Profit	55	60	35	40	20



PPP – More Info

- Each unit of each product take 20 manhours for final assembly.
- The factory has three grinding machines and two drilling machines.
- The factory works a six day week with two shifts of 8 hours/day. Eight workers are employed in assembly, each working one shift per day.



PPP

maximize

$$55x_1 + 60x_2 + 35x_3 + 40x_4 + 20x_5 \quad (\text{Profit/week})$$

subject to

$$12x_1 + 20x_2 + 0x_3 + 25x_4 + 15x_5 \leq 288 \quad (\text{Grinding})$$

$$10x_1 + 8x_2 + 16x_3 + 0x_4 + 0x_5 \leq 192 \quad (\text{Drilling})$$

$$20x_1 + 20x_2 + 20x_3 + 20x_4 + 20x_5 \leq 384 \quad \text{Final Assembly}$$

$$x_i \geq 0 \quad \forall i = 1, 2, \dots, 5$$



Another PPP Modeling Example

- Let's model the following situation.
 - If we manufacture P1 or P2 (or both), then at least one of P3, P4, P5 must also be manufactured.
- We first need indicator variables z_j that indicate when each of the $x_j > 0$.
 - How do we model $x_j > 0 \Rightarrow z_j = 1$?
 - **Hint:** This is equivalent to $z_j = 0 \Rightarrow x_j = 0$



Modeling the Logic

Answer: $x_j \leq Mz_j$

- Given that we have included the constraints $x_j \leq Mz_j$, we'd like to model the following implication:
 - $z_1 + z_2 \geq 1 \Rightarrow z_3 + z_4 + z_5 \geq 1$
- Can you just “see” the answer?
- I can't. So let's try the “formulaic” approach.
- **Important Trick:** Think of it in two steps
 - $z_1 + z_2 \geq 1 \Rightarrow \delta = 1$
 - $\delta = 1 \Rightarrow z_3 + z_4 + z_5 \geq 1$.



Look up the Tricks

- First we model the following:
 - $z_1 + z_2 \geq 1 \Rightarrow \delta = 1$
- The formula from the bag o' tricks
- $\sum_{j \in N} a_j x_j \geq b \Rightarrow \delta = 1 \Leftrightarrow \sum_{j \in N} a_j x_j - (M + \epsilon)\delta \leq b - \epsilon$
- M : Upper Bound on $\sum_{j \in N} a_j z_j - b$
 - $M = 1$ in this case. ($z_1 \leq 1, z_2 \leq 1, b = 1$).
- $\epsilon = 1$ in this case
- Just plug in the formula $\sum_{j \in N} a_j x_j - (M + \epsilon)\delta \leq b - \epsilon$
 - $z_1 + z_2 - 2\delta \leq 0$



Second Part

- Want to model the following:
 - $\delta = 1 \Rightarrow z_3 + z_4 + z_5 \geq 1$.
- The formula from the bag o' tricks
- $\delta = 1 \Rightarrow \sum_{j \in N} a_j x_j \geq b \Leftrightarrow \sum_{j \in N} a_j x_j + m\delta \geq m + b$
- m : lower bound on $\sum_{j \in N} a_j x_j - b$.
 - $m = -1$. ($z_1 \geq 0, z_2 \geq 0, b = 1$).
- Plug in the formula:
 - $z_3 + z_4 + z_5 - \delta \geq 0$
- It works! (Check for $\delta = 0, \delta = 1$).



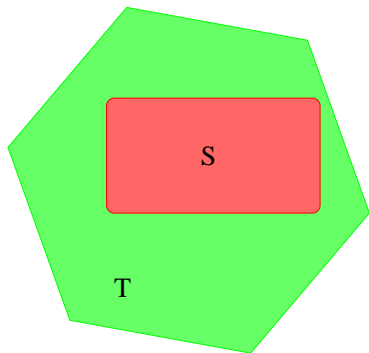
Cool Things You Can Now Do

- Either constraint 1 or constraint 2 must hold
 - Create indicators δ_1, δ_2 , then $\delta_1 + \delta_2 \geq 1$
- At least one constraint of all the constraints in M should hold
 - $\sum_{i \in M} \delta_i \geq 1$
- At least k of the constraints in M must hold
 - $\sum_{i \in M} \delta_i \geq k$
- If x , then y
 - $\delta_y \geq \delta_x$



Relaxations

- $z(S) \stackrel{\text{def}}{=} \min_{x \in S} f(x)$
- $z(T) \stackrel{\text{def}}{=} \min_{x \in T} f(x)$

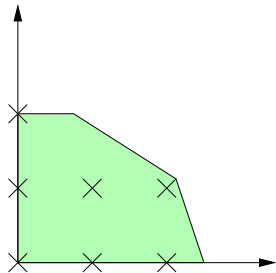


- Independent of f, S, T : $z(T) \leq z(S)$
- If $x_T^* = \arg \min_{x \in T} f(x)$
- And $x_T^* \in S$, then
- $x_T^* = \arg \min_{x \in S} f(x)$



A Pure Integer Program

$$z(S) = \min\{c^T x : x \in S\}, \quad S = \{x \in \mathbb{Z}_+^n : Ax \leq b\}$$



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



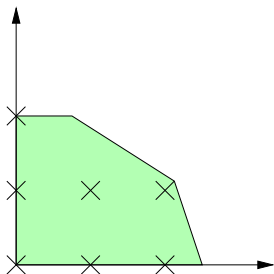
How to Solve Integer Programs?

- **Relaxations!**

- $T \supseteq S \Rightarrow z(T) \leq z(S)$
- People commonly use the linear programming relaxation:

$$z(\text{LP}(S)) = \min\{c^T x : x \in \text{LP}(S)\}$$

$$\text{LP}(S) = \{x \in \mathbb{R}_+^n : Ax \leq b\}$$



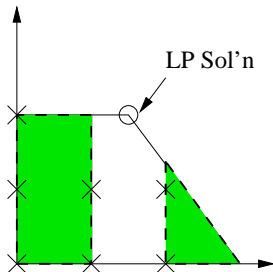
- If $\text{LP}(S) = \text{conv}(S)$, we are done.
- Minimum of **any** linear function over **any** convex set occurs on the boundary

- We need only know $\text{conv}(S)$ in the direction of c .
- The “closer” $\text{LP}(S)$ is to $\text{conv}(S)$ the better.



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.

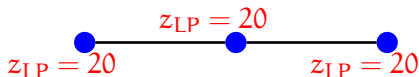


Choices in Branch-and-Bound: **Branching**

- 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



Some Branching Facts

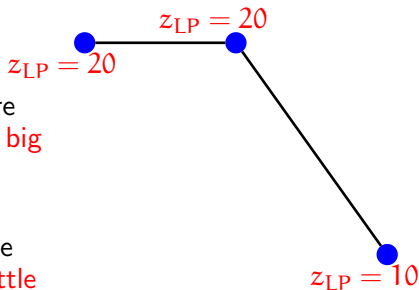


- 1 An Example Branch
- 2 A **bad** branch.
 - The amount of work for this subtree has doubled
- 3 Reducing upper bound **vs.** increasing lower bound:
 - These are somewhat **conflicting goals**



Proof By Picture

- 1 **Improving Upper Bound:** Make sure that your branching decision has a **big** impact on **both** children
 - Now our upper bound is 7
- 2 **Improving Lower Bound:** Make sure that your branching decision has **little** impact on **at least one** child
 - You still have “the same” amount of work to do on the left branch



A Natural Branching Idea

- To make bound go down on both branches, choose to branch on the “most fractional” variable

$$j \in \arg \min_I \{ |f(\hat{x}_j) - 0.5| \}.$$

$f(z)$: Fractional part of z



Nature Is Bad!

*Most fractional branching is no better than choosing a **random** fractional variable to branch on!*

Alex Martin, MIP'06



A Better Branching Idea: Pseudocosts

- Keep track of the impact of branching on x_j :

$$z_j^- \stackrel{\text{def}}{=} \max_{x \in R(S) \cap x_j \leq \lfloor \hat{x}_j \rfloor} \{c^T x + h^T y\} \quad z_j^+ \stackrel{\text{def}}{=} \max_{x \in R(S) \cap x_j \geq \lceil \hat{x}_j \rceil} \{c^T x + h^T y\}$$

$$P_j^- = \frac{z_{\text{LP}} - z_j^-}{f(\hat{x}_j)} \quad P_j^+ = \frac{z_{\text{LP}} - z_j^+}{1 - f(\hat{x}_j)}$$

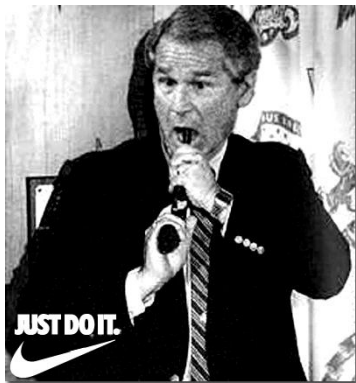
- When you choose to branch on x_j (with value x'_j) again, compute estimated LP decreases as

$$D_j^- = P_j^- f(x'_j) \quad D_j^+ = P_j^+ (1 - f(x'_j))$$

Problem!?

What do you use *initially*!

Just Do It

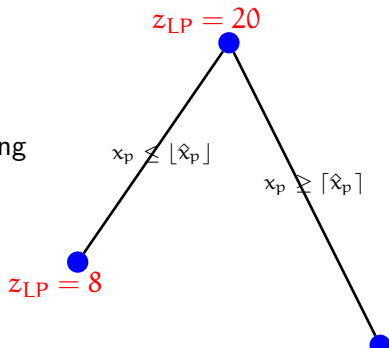


- Initialize pseudocosts by *explicitly* computing them for all yet-to-be-branched-on variables
- With a little imagination, this is a branching method in and of itself: **Strong Branching**.



(Full) Strong Branching

- 1 At **each** node n at which a branching decision must be made:
- 2 For **each** $j \in \mathcal{F}_n$: Compute z_j^- , z_j^+
- 3 Branch on $\max_{j \in \mathcal{F}_n} f(z_j^-, z_j^+)$



How To Combine?

- Try the weighting function $\mathcal{W}(z_{LP} - z_i^-, z_{LP} - z_i^+)$ for

$$\mathcal{W}(a, b) \stackrel{\text{def}}{=} \{\alpha_1 \min(a, b) + \alpha_2 \max(a, b)\},$$

- $\alpha_1 = 3.7214541$, $\alpha_2 = 1$ seems to work OK. :-)



Speeding up Strong Branching

Obvious Ideas

- 1 Limit number of pivots β
- 2 Limit Candidate Set $|C|$

Good Ideas!

- 1 Q-phase selection
 - $C_1 \supseteq C_2 \supseteq C_3 \supseteq \dots \supseteq C_Q$
 - $\beta_1 \leq \beta_2 \leq \beta_3 \leq \dots \leq \beta_Q$
- 2 Limit number of times that you perform strong branching on any variable, then “switch” to pseudocosts.
 - Reliability branching (Achterberg, Koch, Martin)



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 on
- 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.



Branching Rules in Commercial Packages

CPLEX (CPX_PARAM_VARSEL)

- Most fractional
- Min Fractional: (**Very** bad idea if you want to prove optimality)
- Pseudocosts
- Strong Branching: CPX_PARAM_STRONGCANDLIM, CPX_PARAM_STRONGITLIM
- Pseudo-Reduced Costs

XPRESS

- Pseudocosts
- Strong Branching: SBEST, SBITERLIMIT, SBESTIMATE
- VARSELECTION: Controls how to combine up and down degradation estimates



Choices in Branch and Bound **Node Selection**

- We've talked about one choice in branch and bound: **Which variable.**
- Another important choice in branch and bound is the strategy for selecting the next subproblem to be processed.
 - That said, in general, the branching variable selection method has a larger impact on solution time than the node selection method
- **Node selection** is often called **search strategy**
- 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 Proof. **Gasp!**
 - 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.

QUITE ENOUGH DONE



Drawbacks of Best First

- 1 Doesn't necessarily find feasible solutions quickly
 - Feasible solutions are “more likely” to be found deep in the tree
- 2 Node setup costs are high
 - The linear program being solved may change quite a bit from one node evaluation to the next
- 3 Memory usage is high
 - It can require a lot of memory to store the candidate list, since the tree can grow “broad”



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.



Hybrid Strategies

- Go depth-first until you find a feasible solution, then do best-first search

A Key Insight

If you *knew* the optimal solution value, the best thing to do would be to go depth first

- Go depth-first for a while, then make a best-first move.
- What is “for a while”?
 - Estimate z_E as the optimal solution value
 - Go depth-first until $z_{LP} \leq z_E$
 - Then jump to a better node
- This is what the commercial packages do!



Estimate-based Strategies

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



Node Selection in Commercial Packages

CPLEX

- `CPX_PARAM_NODESEL`: Best bound, two different best-estimates, and (pure) depth-first.
- `CPX_PARAM_BTTOL`: Controls likelihood of stopping dive

XPRESS

- `NODESELECT`: Best, Pure Best, Deepest, Pure Best for k nodes, then Best, Pure depth
- `BACKTRACK`: Sets whether to jump/backtrack to “best bound” or “best estimate” node.



Cutting Planes

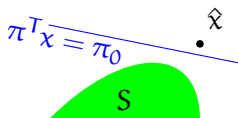
- Sometimes we can get a better formulation by **dynamically** improving it.

- An inequality $\pi^T x \leq \pi_0$ is a **valid inequality** for S if $\pi^T x \leq \pi_0 \forall x \in S$

- Alternatively: $\max_{x \in S} \{\pi^T x\} \leq \pi_0$

- **Thm:** (Hahn-Banach). Let $S \subset \mathbb{R}^n$ be a closed, convex set, and let $\hat{x} \notin S$. Then there exists $\pi \in \mathbb{R}^n$ such that

$$\pi^T \hat{x} > \max_{x \in S} \{\pi^T x\}$$



Two Classes of Valid Inequalities

Structure-Specific


- (Lifted) Knapsack Covers
- (Lifted) GUB Covers
- Flow Covers
- Flow Path
- Clique Inequalities
- Implication Inequalities

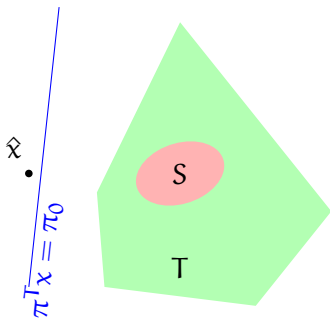
Structure-Independent

- Gomory Cuts
- Lift and Project Cuts
- Mixed Integer Rounding Cuts
- Split Cuts



Valid Inequalities From Relaxations

- **Idea:** Inequalities valid for a relaxation are valid for original 
- Generating valid inequalities for a relaxation is often easier.



- **Separation Problem** over T :
Given \hat{x}, T find (π, π_0) such that
 $\pi^T \hat{x} > \pi_0, \pi^T x \leq \pi_0 \forall x \in T$



Simple Relaxations

- **Idea:** Consider **one row** relaxations 💡
- If $P = \{x \in \{0, 1\}^n \mid Ax \leq b\}$, then for any row i , $P_i = \{x \in \{0, 1\}^n \mid a_i^T x \leq b_i\}$ is a relaxation of P .
- If the intersection of the relaxations is a good approximation to the true problem, then the inequalities will be quite useful.
- Crowder et al. [1983] is the seminal paper that shows this to be true for IP.



Knapsack Covers

$$K = \{x \in \{0, 1\}^n \mid a^T x \leq b\}$$

- A set $C \subseteq N$ is a **cover** if $\sum_{j \in C} a_j > b$
- A cover C is a **minimal cover** if $C \setminus j$ is not a cover $\forall j \in C$
- If $C \subseteq N$ is a cover, then the *cover inequality*

$$\sum_{j \in C} x_j \leq |C| - 1$$

is a valid inequality for S

- Sometimes (minimal) cover inequalities are **facets** of $\text{conv}(K)$



Example

$$K = \{x \in \{0, 1\}^7 \mid 11x_1 + 6x_2 + 6x_3 + 5x_4 + 5x_5 + 4x_6 + x_7 \leq 19\}$$

$$LP(K) = \{x \in [0, 1]^7 \mid 11x_1 + 6x_2 + 6x_3 + 5x_4 + 5x_5 + 4x_6 + x_7 \leq 19\}$$

- $(1, 1, 1/3, 0, 0, 0, 0) \in LP(K)$
 - **CHOPPED OFF BY** $x_1 + x_2 + x_3 \leq 2$
- $(0, 0, 1, 1, 1, 3/4, 0) \in LP(K)$
 - **CHOPPED OFF BY** $x_3 + x_4 + x_5 + x_6 \leq 3$



Other Substructures

- **Single node flow:** [Padberg et al., 1985]

$$S = \left\{ x \in \mathbb{R}_+^{|\mathcal{N}|}, y \in \{0, 1\}^{|\mathcal{N}|} \mid \sum_{j \in \mathcal{N}} x_j \leq b, x_j \leq u_j y_j \quad \forall j \in \mathcal{N} \right\}$$

If you have this structure, you may want to employ **flow covers** and **flow-path** inequalities

- **Set Packing:** [Borndörfer and Weismantel, 2000]

$$S = \left\{ y \in \{0, 1\}^{|\mathcal{N}|} \mid Ay \leq e \right\}$$

$A \in \{0, 1\}^{|\mathcal{M}| \times |\mathcal{N}|}$, $e = (1, 1, \dots, 1)^T$. If you have this structure, you may wish to employ **clique** inequalities or (maybe) lifted-odd-hole inequalities



The Chvátal-Gomory Procedure

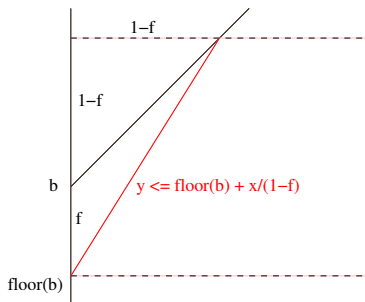
- A **general** procedure for generating valid inequalities for integer programs
- Let the columns of $A \in \mathbb{R}^{m \times n}$ be denoted by $\{a_1, a_2, \dots, a_n\}$
- $S = \{y \in \mathbb{Z}_+^n \mid Ay \leq b\}$.
 - 1 Choose nonnegative multipliers $u \in \mathbb{R}_+^m$
 - 2 $u^T Ay \leq u^T b$ is a valid inequality ($\sum_{j \in N} u^T a_j y_j \leq u^T b$).
 - 3 $\sum_{j \in N} \lfloor u^T a_j \rfloor y_j \leq u^T b$ (Since $y \geq 0$).
 - 4 $\sum_{j \in N} \lfloor u^T a_j \rfloor y_j \leq \lfloor u^T b \rfloor$ is valid for S since $\lfloor u^T a_j \rfloor y_j$ is an integer
- **Simply Amazing:** This simple procedure **suffices** to generate every valid inequality for an integer program



Mixed Integer Rounding—MIR

Almost **everything** comes from considering the following very simple set, and observation.

- $X = \{(x, y) \in \mathbb{R} \times \mathbb{Z} \mid y \leq b + x\}$
- $f = b - \lfloor b \rfloor$: **fractional**
- $LP(X)$
- $\text{conv}(X)$
- $y \leq \lfloor b \rfloor + \frac{1}{1-f}x$ is a valid inequality for X



Extension of MIR

$$X_2 = \left\{ (x^+, x^-, y) \in \mathbb{R}_+^2 \times \mathbb{Z}^{|\mathcal{N}|} \mid \sum_{j \in \mathcal{N}} a_j y_j + x^+ \leq b + x^- \right\}$$

- The inequality

$$\sum_{j \in \mathcal{N}} \left(\lfloor a_j \rfloor + \frac{(f_j - f)^+}{1 - f} \right) y_j \leq \lfloor b \rfloor + \frac{x^-}{1 - f}$$

is valid for X_2

- $f_j \stackrel{\text{def}}{=} a_j - \lfloor a_j \rfloor$, $(t)^+ \stackrel{\text{def}}{=} \max(t, 0)$
- X_2 is a one-row relaxation of a general *mixed* integer program
 - Continuous variables aggregated into two: x^+, x^-



Gomory Mixed Integer Cut is a MIR Inequality

- Consider the set

$$X = \left\{ (x^+, x^-, y_0, y) \in \mathbb{R}_+^2 \times \mathbb{Z} \times \mathbb{Z}_+^{|\mathcal{N}|} \mid y_0 + \sum_{j \in \mathcal{N}} a_j y_j + x^+ - x^- = b \right\}$$

which is essentially the row of an LP tableau

- Relax the equality to an inequality and apply MIR
- Gomory Mixed Integer Cut:**

$$\sum_{j \in \mathcal{N}_1} f_j y_j + x^+ + \frac{f}{1-f} x^- + \sum_{j \in \mathcal{N}_2} \left(f_j - \frac{f_j - f}{1-f} \right) y_j \geq f$$



Implied Bound (Implication) Cuts

- Logical implications discovered during preprocessing are sometimes between pairs of binary variables
 - $x_i = 1 \Rightarrow x_j = 0 \Leftrightarrow x_i + x_j \leq 1$
- Sometimes implication is found between binary variable x and continuous variable y
 - $x = 0 \Rightarrow y \leq \alpha \Rightarrow y \leq \alpha + (U - \alpha)x$
- Lots of other sorts of inequalities can be derived, as discussed by Savelsbergh [1994].



Cuts in Commercial Software Systems

CPLEX

- Cover Cuts (Knapsack Covers): CPX_PARAM_COVERS
- Cover Cuts with GUB: CPX_PARAM_GUBCOVERS
- Lift and project (type) CPX_PARAM_DISJCUTS
- Flow Covers: CPX_PARAM_FLOWCOVERS
- Flow Paths: CPX_PARAM_FLOWPATHS
- Cliques: CPX_PARAM_CLIQUES
- Gomory Cuts: CPX_PARAM_FRACCUTS
- Implication Cuts: CPX_PARAM_IMPLBD
- MIR Cuts: CPX_PARAM_MIRCUTS



Cuts in Commercial Software Packages

XPRESS

- “Cover” Cuts: COVERCUTS (TREECOVERCUTS)
- Gomory Cuts: GOMCUTS (TREEGOMCUTS) LNPBest, LNPIterLimit
- “Cover” cuts – include knapsack covers, gub covers, and flow covers
- “Gomory cuts” – are really lift and project cuts



Too Many Parameters?

- If you think there are too many parameters to control these algorithms, you are right!
- CPLEX and XPRESS agree with you.
- In recent versions they have added “meta”-parameters, the effect of which is to set many base parameters

Examples

- CPLEX: CPX_PARAM_MIPEMPHASIS
- XPRESS: CUTSTRATEGY, HEURSTRATEGY



Heuristics

- Sadly I didn't have time to include much discussion, but all quality MIP solvers also include (a variety) of heuristic procedures that work to find feasible solutions.
- This has been an area of recent emphasis.
- Commercial vendors don't like people to know exactly what they are doing

Heuristics

- Diving (CPLEX and XPRESS)
- Feasibility Pump (XPRESS)
- RINS (CPLEX)
- Local Branching (Neither?)
- Solution Polishing (CPLEX v 10)



Callbacks

- Sadly, I also didn't have time to cover the callback mechanisms through which users can
 - Customize the branching selection
 - Generate valid inequalities
 - Do heuristics to find feasible solutions
- But both XPRESS and CPLEX have these capabilities
- (Perhaps) more open solver frameworks make algorithm customization easier



ABACUS

- ABACUS [Jünger and Thienel, 2001] is a pure solver framework written in C++.
- ABACUS was for some time a commercial code, but has recently been released open source [Thienel, 2004] under the GNU Library General Public License (LGPL).



BCP

- BCP is a pure solver framework developed by Ladányi. It is a close relative of SYMPHONY, described below.
- Released Open Source under the Common Public license
<http://www.coin-or.org>
- Allows for the addition of both columns (branch-and-price) and rows (branch-and-cut)
- Can be instrumented to run in parallel



BonsaiG

- Black box MILP solver available at Hafer [2004], written by Lou Hafer at Simon Fraser
- Features:
 - Dynamic (agreesive) preprocessing
 - its own built-in LP Solver
- Not actively maintained anymore



CBC

- CBC: Coin Branch and Cut
- Written by John Forrest (author of most of OSL)
- Available at COIN site Forrest [2004]
- Actively developed with many recent improvements



GLPK

- GLPK is the GNU Linear Programming Kit, a set of subroutines comprising a callable library and black box solver for solving linear programming and MILP instances Makhorin [2004].
- Released under LGPL
- Developer: Andrew Makhorin
- GLPK also comes equipped with GNU MathProg (GMPL), an algebraic modeling language similar to AMPL.
- Clean code – following GNU coding standards



lp_solve()

- Lp_solve is a black box solver and callable library for linear and mixed-integer programming currently developed and maintained by Kjell Eikland and Peter Notebaert
- Distributed as open source under the GNU Library General Public License (LGPL).
- Has its own (relatively active) YAHOO Group: Berkelaar [2004].
- **Target Audience:** Similar to that of GLPK—users who want a lightweight, self-contained solver with a callable library API implemented in a number of popular programming languages, including C, VB and Java, as well as an AMPL interface.



MINTO

- MINTO (Mixed INTeger Optimizer) is a black box solver and solver framework for MILP.
- Chief architects of MINTO: George Nemhauser and Martin Savelsbergh, and a majority of the software development was done by Savelsbergh.
- MINTO was developed at the Georgia Institute of Technology and is available under terms of an agreement created by the Georgia Tech Research Institute.
- The current maintainer of MINTO is Jeff Linderoth of Lehigh University.
- MINTO is available only in library form for a number of platforms Nemhauser and Savelsbergh [2004].
- Advantages: Sophisticated preprocessing and cutting planes
- Can be called from AMPL
- Available on NEOS <http://www-neos.mcs.anl.gov> (as are a few others)



Symphony

- SYMPHONY is a black box solver, callable library, and solver framework for MILPs that evolved from the COMPSys framework of Ralphs and Ladányi [1996], Ralphs [1995].
- The source code for packaged releases, with full documentation and examples, is available for download [2004] and is licensed under the Common Public License (CPL).
- Advantages:
 - Actively Maintained
 - Can solve for “efficient frontier” of “bi-criteria” MIPs



Separated at Birth?

MINTO



≠

SYMPHONY



Noncommercial MIP Software

	Version Number	LP Solver	File Format	Callable API	Framework API	User's Manual
ABACUS	2.3	C/S/X	no	none	C++	yes
BCP	11/1/04	OSI	no	none	C++	yes
bonsaiG	2.8	DYLP	MPS	none	none	yes
CBC	0.70	OSI	MPS	C++/C	C++	no
GLPK	4.2	GLPK	MPS/GMPL	OSI/C	none	yes
lp_solve	5.1	lp_solve	MPS/LP/GMPL	C/VB/Java	none	yes
MINTO	3.1	OSI	MPS/AMPL	none	C	yes
SYMPHONY	5.0	OSI	MPS/GMPL	OSI/C	C	yes

Table: List of solvers and main features



MIP Feature Key

- e: pseudo-cost branching
- f: branching on the variables with the largest fractional part
- h: branching on hyperplanes
- g: GUB branching
- i: branching on first or last fractional variable (by index)
- p: penalty method
- s: strong branching
- x: SOS(2) branching and branching on semi-continuous variables

For the column denoting search strategies, the codes stand for the following:

- b: best-first
- d: depth-first
- e: best-estimate
- p: best-projection
- r: breadth-first
- h(x,z): a hybrid method switching from strategy 'x' to strategy 'z'
- 2(x,z): a two-phase method switching from strategy 'x' to strategy 'z'



Noncommercial MIP Code Features

	Preproc	Built-in Cut Generation	Column Generation	Primal Heuristic	Branching Rules	Search Strategy
ABACUS	no	no	yes	no	f,h,s	b,r,d,2(d,b)
BCP	no	no	yes	no	f,h,s	h(d,b)
bonsaiG	no	no	no	no	p	h(d,b)
CBC	yes	yes	no	yes	e,f,g,h,s,x	2(d,p)
GLPK	no	no	no	no	i,p	b,d,p
Ip_solve	no	no	no	no	e,f,i,x	d,r,e,2(d,r)
MINTO	yes	yes	yes	yes	e,f,g,p,s	b,d,e,h(d,e)
SYMPHONY	no	yes	yes	no	e,f,h,p,s	b,r,d,h(d,b)

Table: Algorithmic features of solvers



Inequality Classes in Noncommercial MIP Solvers

Name	Knapsack	GUB	Flow	Clique	Implication	Gomory	MIR
CBC	yes	no	yes	yes	yes	yes	yes
MINTO	yes	yes	yes	yes	yes	no	no
SYMPHONY	yes	no	yes	yes	yes	yes	yes

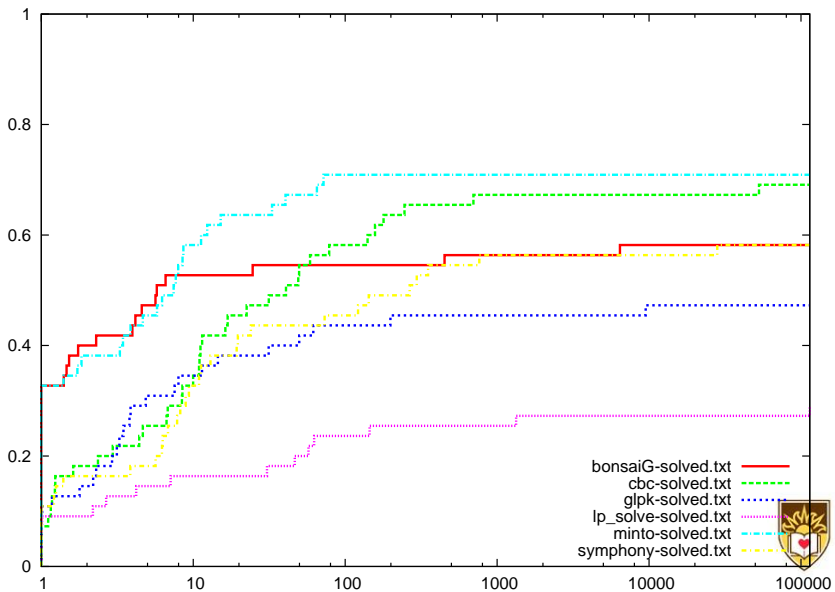
Table: Classes of valid inequalities generated by black box solvers

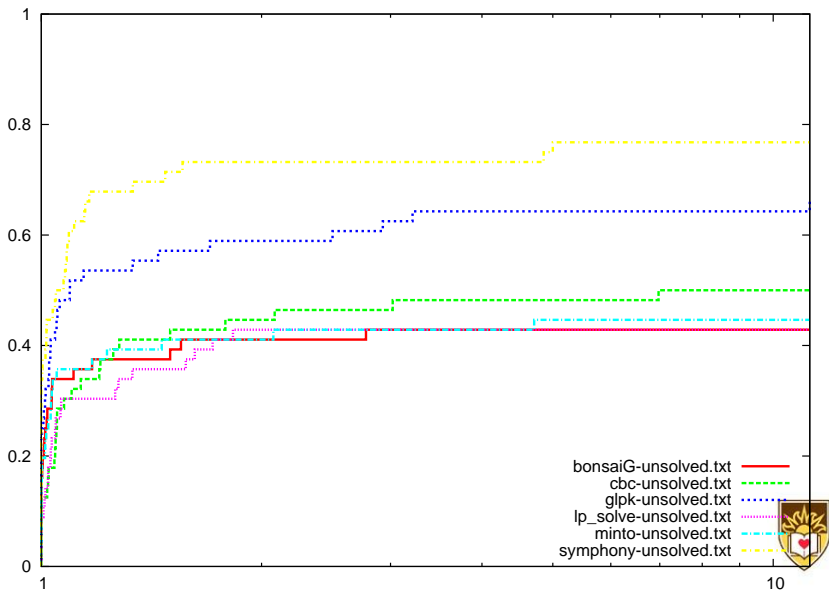


Performance Profiles

- A relative measure of the effectiveness of a solver s when compared to a group of solvers \mathcal{S} on a set of problem instances \mathcal{P} .
 - γ_{ps} : *quality measure* of solver $s \in \mathcal{S}$ when solving problem $p \in \mathcal{P}$
 - $r_{ps} = \gamma_{ps} / (\min_{s \in \mathcal{S}} \gamma_{ps})$
 - $\rho_s(\tau) = |\{p \in \mathcal{P} \mid r_{ps} \leq \tau\}| / |\mathcal{P}|$.
- $\rho_s(\tau)$: fraction of instances for which the performance of solver s was within a factor of τ of the best.
- A performance profile for solver s is the graph of $\rho_s(\tau)$.
- In general, the “higher” the graph of a solver, the better the relative performance.







Conclusions

- Bad Idea to try and do an entire course in an hour
- Integer Programming is the (commercially) “Most Important” optimization problem
- **Amazing** increases in efficiency of codes. Literally, instances can be solved **billions** of times faster than they could ten years ago
- Commercial codes are (by a wide margin) the state-of-the-art
- Noncommercial codes are reasonable for many instances



- M. Berkelaar. Ip_solve 5.1, 2004. Available from http://groups.yahoo.com/group/lp_solve/.
- R. Borndörfer and R. Weismantel. Set packing relaxations of some integer programs. *Mathematical Programming*, 88:425 – 450, 2000.
- A. Brearley, G. Mitra, and H. Williams. Analysis of mathematical programming problems prior to applying the simplex method. *Mathematical Programming*, 8:54–83, 1975.
- H. Crowder, E. L. Johnson, and M. W. Padberg. Solving large scale zero-one linear programming problems. *Operations Research*, 31:803–834, 1983.
- J. Forrest. CBC, 2004. Available from <http://www.coin-or.org/>.
- L. Hafer. bonsaiG 2.8, 2004. Available from <http://www.cs.sfu.ca/~lou/BonsaiG/dwnldreq.html>.
- M. Jünger and S. Thienel. The ABACUS system for branch and cut and price algorithms in integer programming and combinatorial optimization. *Software Practice and Experience*, 30: 1325–1352, 2001.
- L. Ladányi. *Parallel Branch and Cut and Its Application to the Traveling Salesman Problem*. PhD thesis, Cornell University, May 1996.
- A. Makhorin. GLPK 4.2, 2004. Available from <http://www.gnu.org/software/glpk/glpk.html>.
- G. Nemhauser and M. Savelsbergh. MINTO 3.1, 2004. Available from <http://coral.ie.lehigh.edu/minto/>.
- M. Padberg, T. J. Van Roy, and L. Wolsey. Valid linear inequalities for fixed charge problems. *Operations Research*, 33:842–861, 1985.
- T. Ralphs. SYMPHONY 5.0, 2004. Available from <http://www.branchandcut.org/SYMPHONY/>
- T. Ralphs. *Parallel Branch and Cut for Vehicle Routing*. PhD thesis, Cornell University, May 1995.



- M. W. P. Savelsbergh. Preprocessing and probing techniques for mixed integer programming problems. *ORSA Journal on Computing*, 6:445–454, 1994.
- S. Thienel. ABACUS 2.3, 2004. Available from <http://www.informatik.uni-koeln.de/abacus/>.

