Search in Mixed-Integer Linear Programming

JOHN W. CHINNECK
SYSTEMS AND COMPUTER ENGINEERING
CARLETON UNIVERSITY
OTTAWA, CANADA

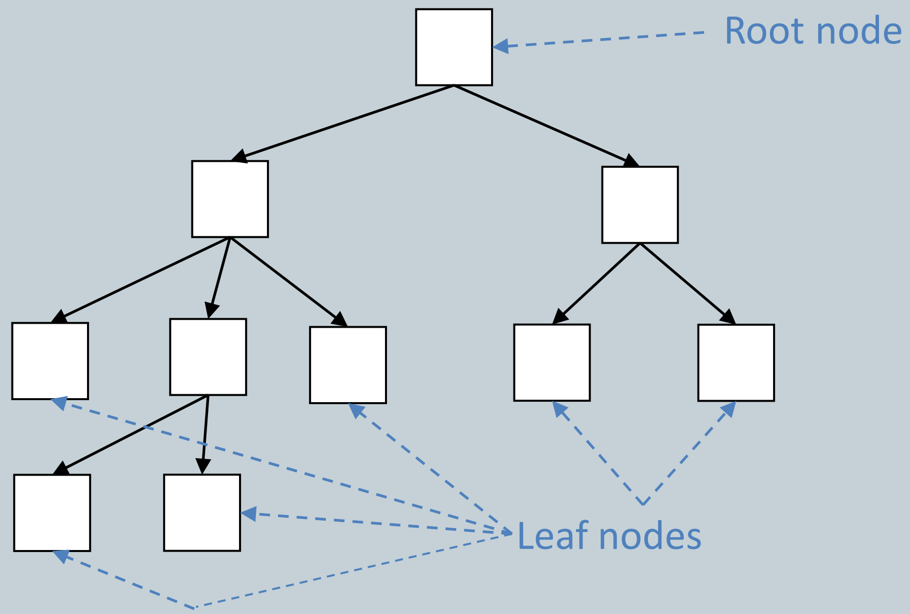
I. Fundamentals

BRANCH AND BOUND BASICS
WHAT IS MIXED-INTEGER PROGRAMMING (MIP)?
BRANCH AND BOUND FOR MIP
OBSERVATIONS

Branch and Bound Basics

- *Application*: searching a discrete space
- *Node*: represents subsets of possible solutions.
- *Branch*: generate child nodes from current node.

AH Land, A Doig (1960). *An Automatic Method of Solving Discrete Programming Problems*, Econometrica 28.



The Bounding Function

- *Bound*: ***optimistic*** bound on the best possible solution at descendent of current node.
  - As accurate as possible, but…
  - *Under*estimate for minimization; *Over*estimate for maximization
- *Incumbent*: best complete ***feasible*** solution yet found. Updated as solution proceeds.
- *Prune*: remove node under certain conditions
  - **Descendent cannot be optimum**: bounding function value worse than incumbent *objective function value*.
  - **Node and descendents cannot be feasible**: decisions thus far prevent one or more constraints from ever being satisfied.
- *Stop*: when incumbent *objective function value* is better than (or equal to) the *best bound* on any node.
  - Optimistic bounding function guarantees optimality.
  - "Better than *or equal to*" finds alternative optima

What is Mixed-Integer Programming?

- Linear objective function (Z) and constraints
- Variables: continuous / integer / binary
  - At least one integer or binary variable
  - Hereafter: "integer" includes "binary"
- MILP (or MIP) includes:
  - Mixed problems (at least one continuous variable)
  - Pure integer problems
  - Pure binary problems
- Integer variables make it a discrete search problem
- *Goal:* best solution that also satisfies all integrality conditions

Branch and Bound for MIP

- *Bounding function* at a node:
  - linear program (LP) solution ignoring integer restrictions.
  - Called the *LP-relaxation*.
- *Integer-feasible solution*:
  - All integer variables have integer values.
- *Leaf nodes* are either:
  - **Integer-feasible** (no descendent will be better)
  - **Infeasible** (and no descendent will be feasible)
- *Intermediate node*:
  - solution satisfies all linear constraints and bounds (original or added), but not all integrality constraints.

Designing a MIP B&B Algorithm

3 major search rule design decisions:

- Branching variable selection
- Branching direction selection
- Node selection: which node to explore next?

Numerous other heuristics:

- Local search
- Root node heuristics
- Etc.

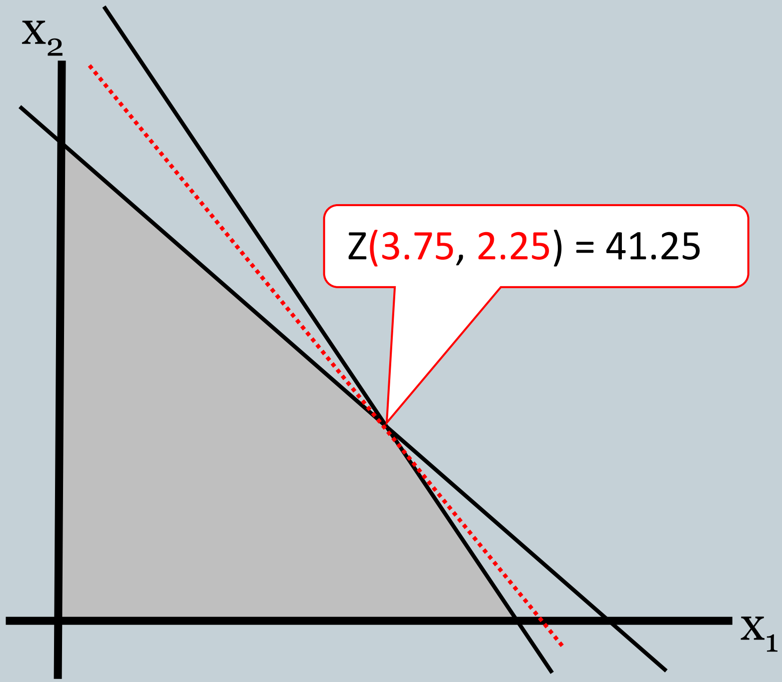
MIP: B&B framework (guarantees optimum), plus numerous heuristics

Branching Variable and Direction Selection

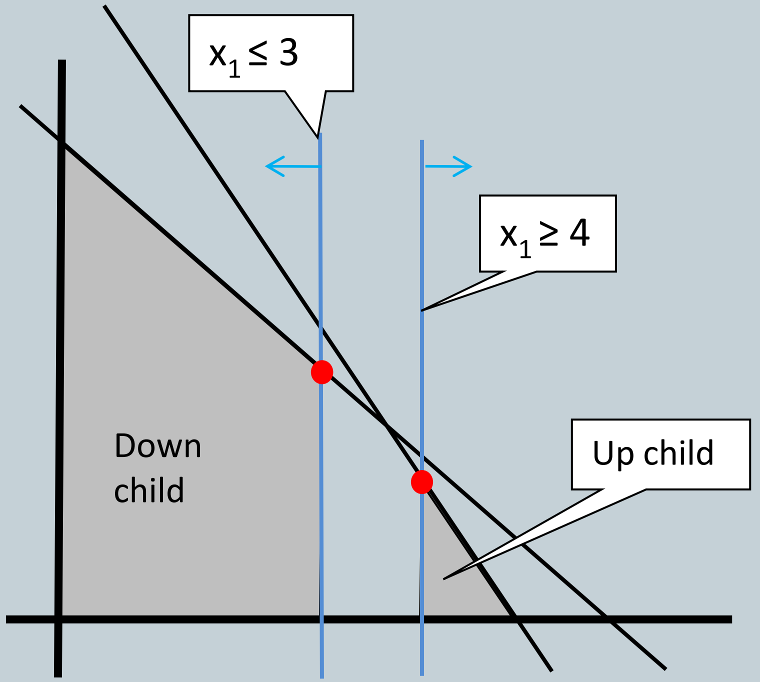
- *Candidate variable*: integer variable having non-integer value in current LP relaxation solution.
- E.g. x_3 =5.7 in LP solution. Branching on x_3 creates two child nodes:
  - **Down** branch: parent LP + revised bound $x_3 \leq 5$
  - **Up** branch: parent LP + revised bound $x_3 \geq 6$
- Search design issues:
  - How to choose the branching variable?
  - How to choose the branching direction (up or down)?
    - The other child node may be visited later…

Branching on a Variable

- LP at parent node



- Branch on x_1:
  - two child node LPs
  - x_1 integer in child solns



Node Selection

- Search issue: which node to explore next?

- Depth-first:
  - Choose next node from among last nodes created
  - Common choice for MIP
  - Big advantage:
    - Next LP identical to last one solved, except for one bound
    - Next solution very quick due to advanced LP start

- Many other options (more later…)

Simple Example

Maximize $Z = 8x_1 + 5x_2$

s.t. $x_1 + x_2 \leq 6$

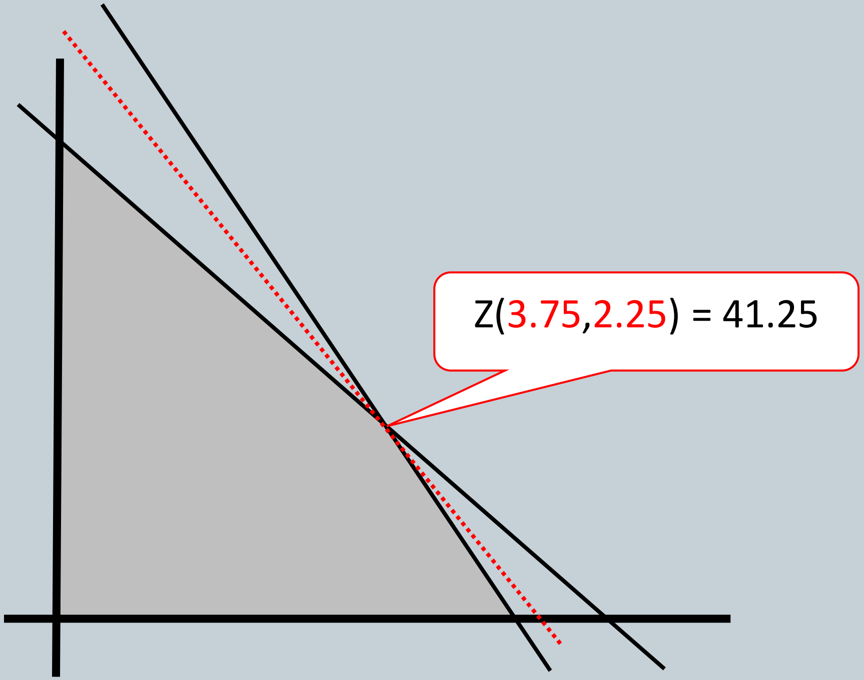
$9x_1 + 5x_2 \leq 45$

x_1, x_2 are integer and nonnegative

- Search rules:
  - *Node selection*: depth first. Simple backtrack at leaf.
  - *Branching variable selection*: natural order.
  - *Branching direction*: down.

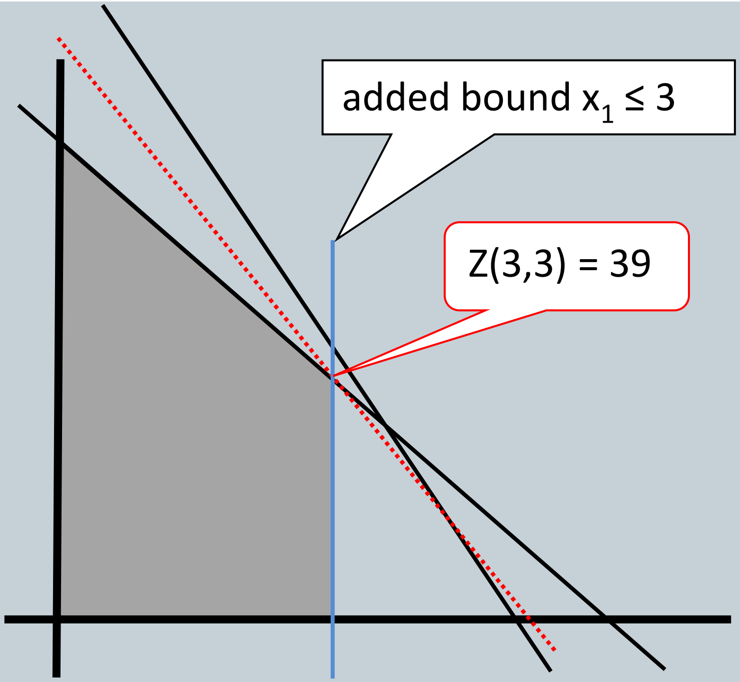
1. Root Node

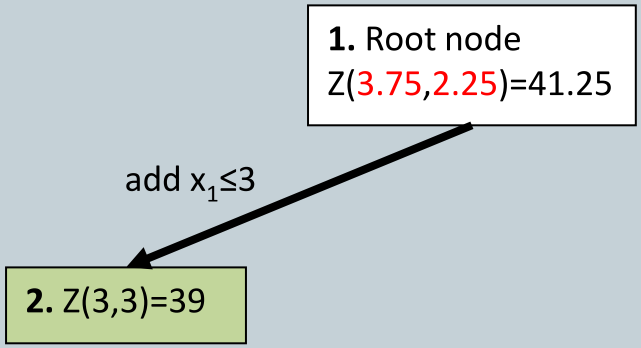
- *Root node LP* solution: Z(3.75, 2.25)=41.25



- Both variables are candidates: choose x_1, branch down.

2. Add Bound: $x_1 \leq 3$

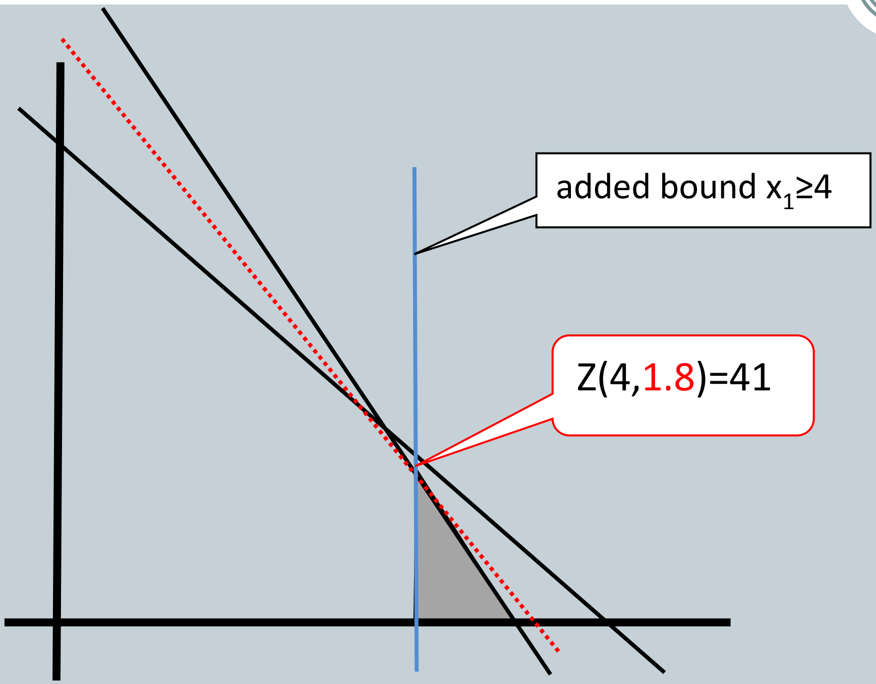




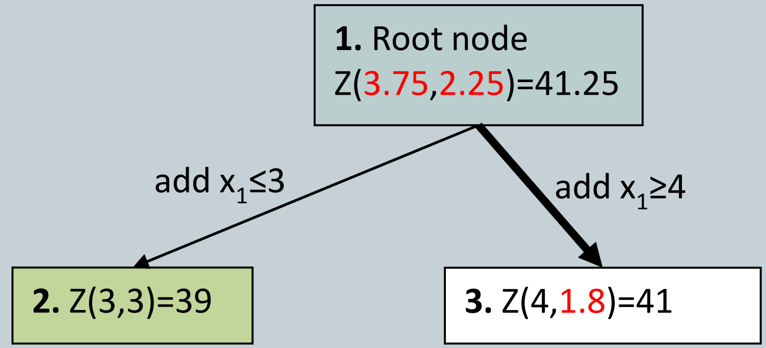
- Integer-feasible!
- First incumbent: Z(3,3)=39

- Still active nodes with bound > incumbent so continue.
- *Next:* backtrack and branch on x_1, up.

3. Add bound: $x_1 \geq 4$

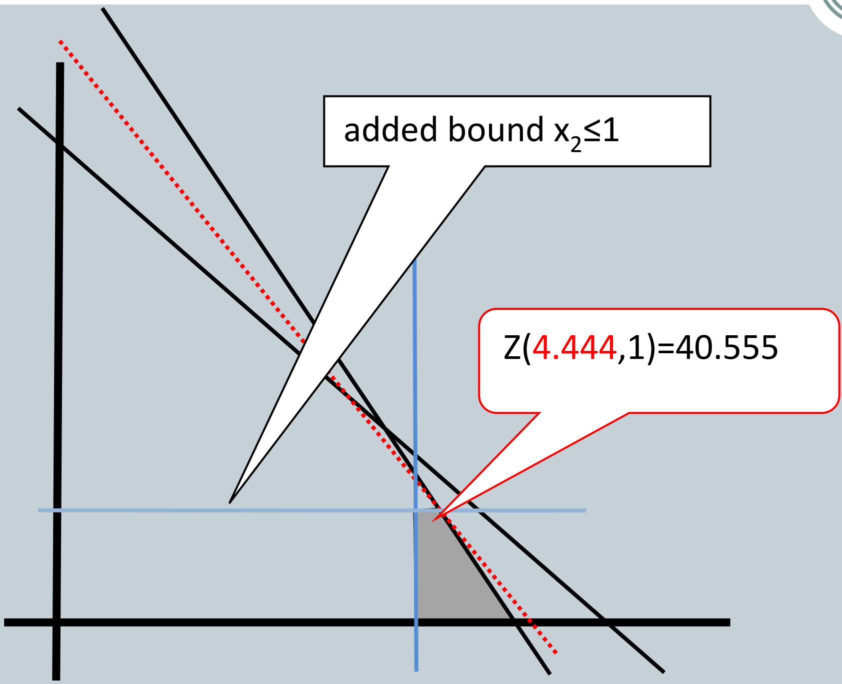


- Not integer-feasible, bound > incumbent.

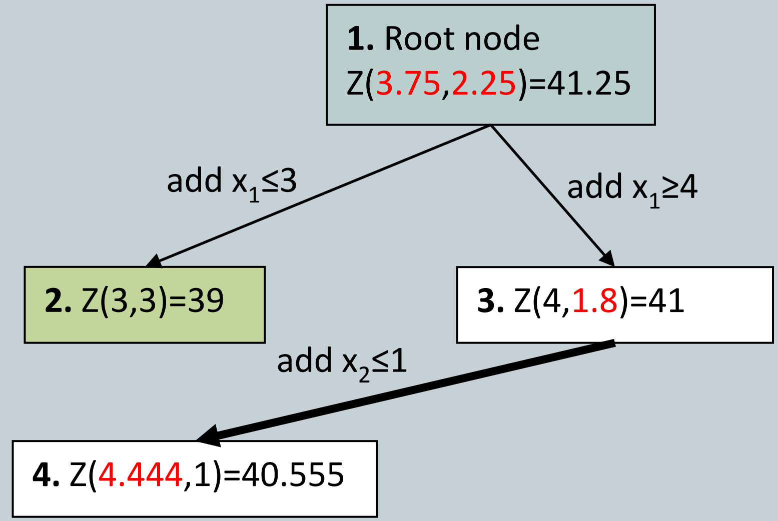


- Still active nodes with bound > incumbent so continue.
- *Next:* continue depth-first, branch on x_2, down.

4. Branch Down on x_2

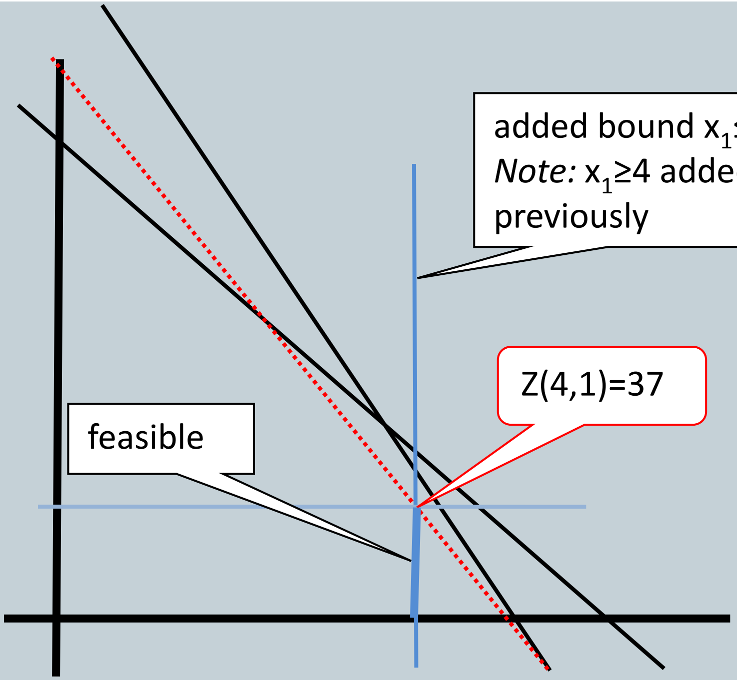


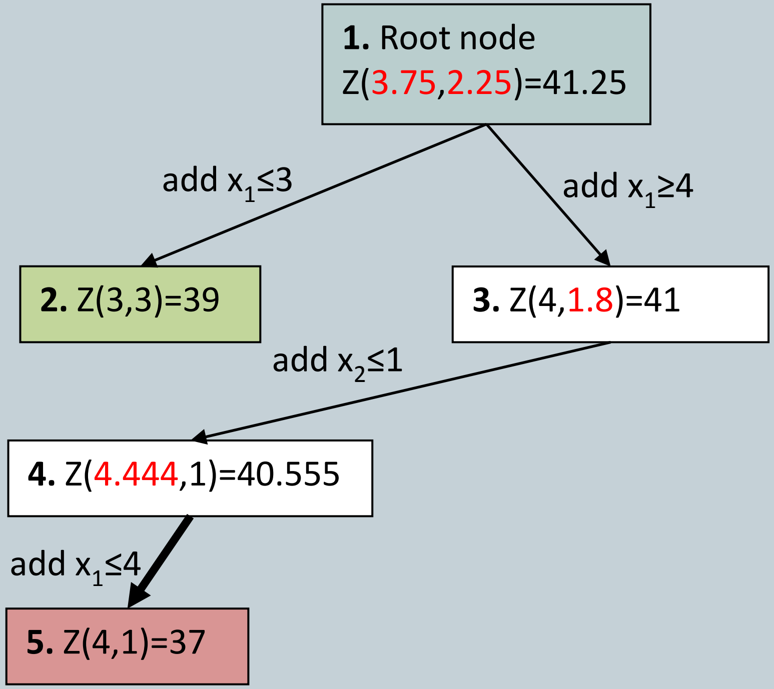
- Not integer-feasible, bound > incumbent.



- Still active nodes with bound > incumbent so continue.
- *Next:* continue depth-first, branch on x_1,down

5. Branch Down on x_1

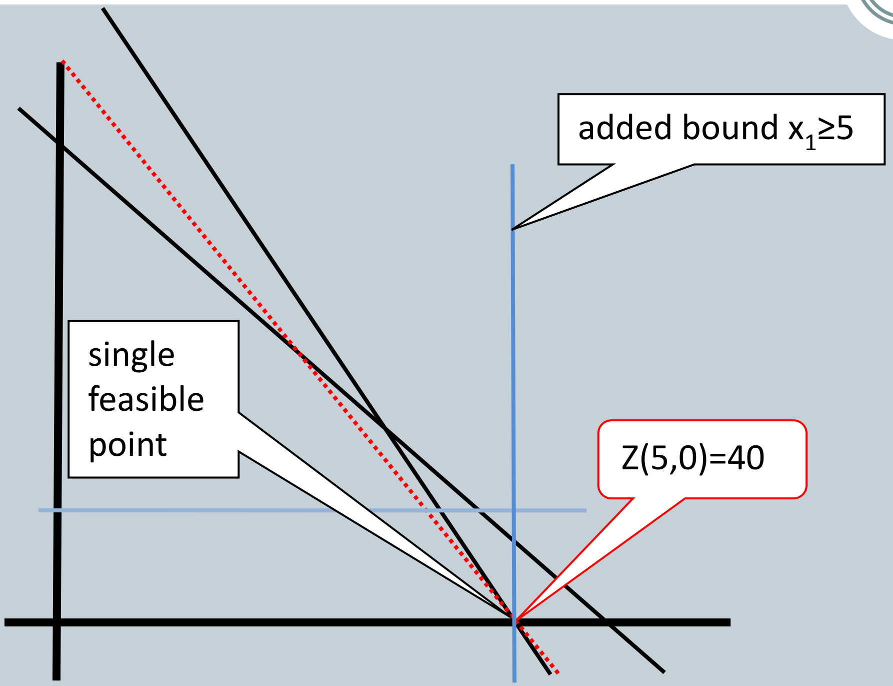


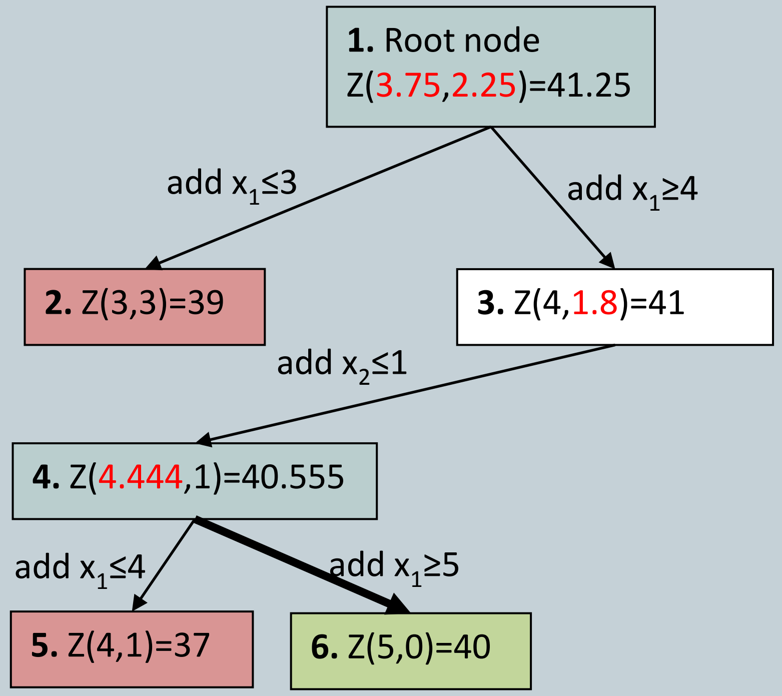


- Integer-feasible, but worse than incumbent: prune.

- Still active nodes with bound > incumbent so continue.
- *Next:* backtrack, branch on x_1, up.

6. Backtrack, add $x_1 \geq 5$

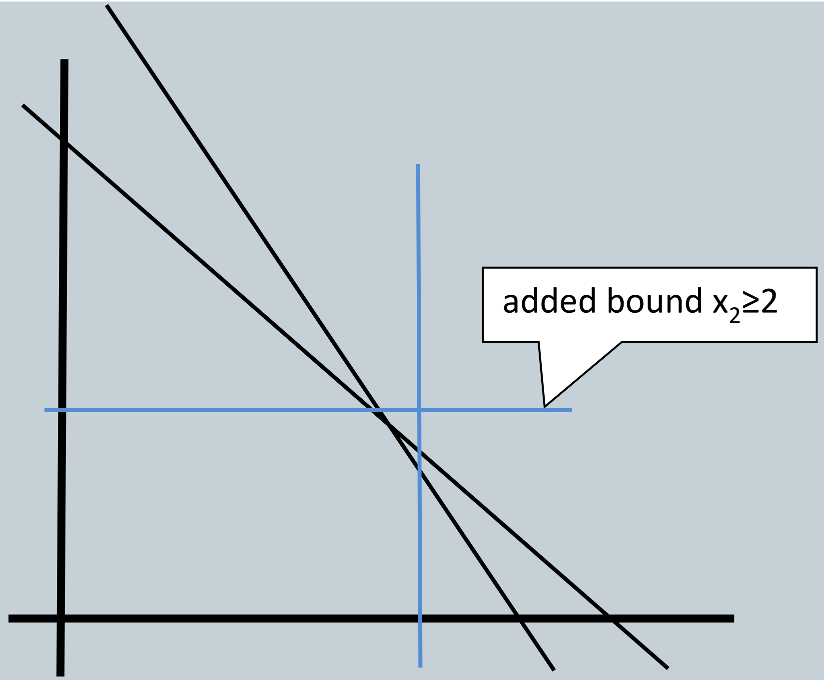




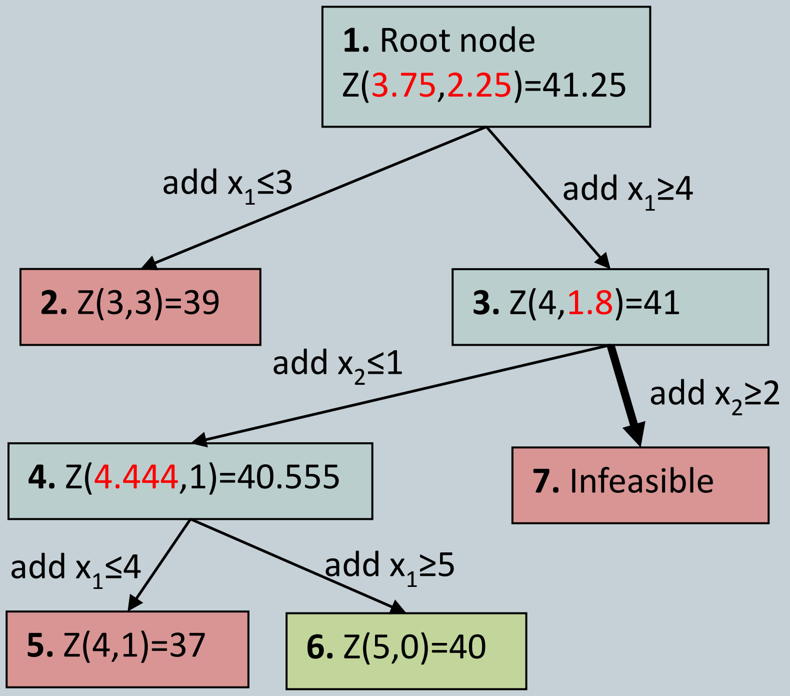
- Feasible, replaces incumbent.

- Still active nodes with bound > incumbent so continue.
- *Next:* backtrack, branch on x_2, up.

7. Backtrack, Add $x_2 \geq 2$



- Infeasible!



- Nowhere to backtrack: exit.
- Solution: Z(5,0)=40.

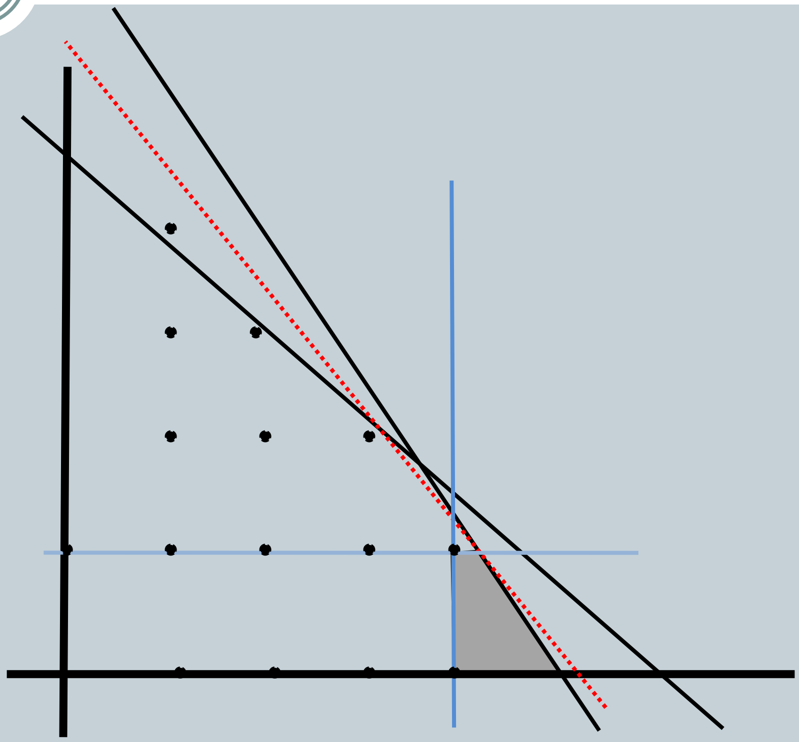
General B&B Algorithm for MIP

N: list of unexplored nodes, initially empty. No incumbent at start.

1. Solve root node LP relaxation. Add it to *N*.
2. Choose node from *N* for exploration.
3. Solve LP relaxation for current node.
   - If LP solution infeasible: go to Step 7.
   - If LP solution is integer-feasible:
     - Worse than incumbent, then go to Step 7.
     - Better than incumbent, replace it, go to Step 7.
4. Choose candidate variable in current node for exploration.
5. Create two child nodes using branching variable, add to *N*.
6. Go to Step 2.
7. If *N* is empty then:
   1. If no incumbent, exit with infeasible outcome.
   2. Else exit with incumbent as optimum solution.
8. Go to Step 2.

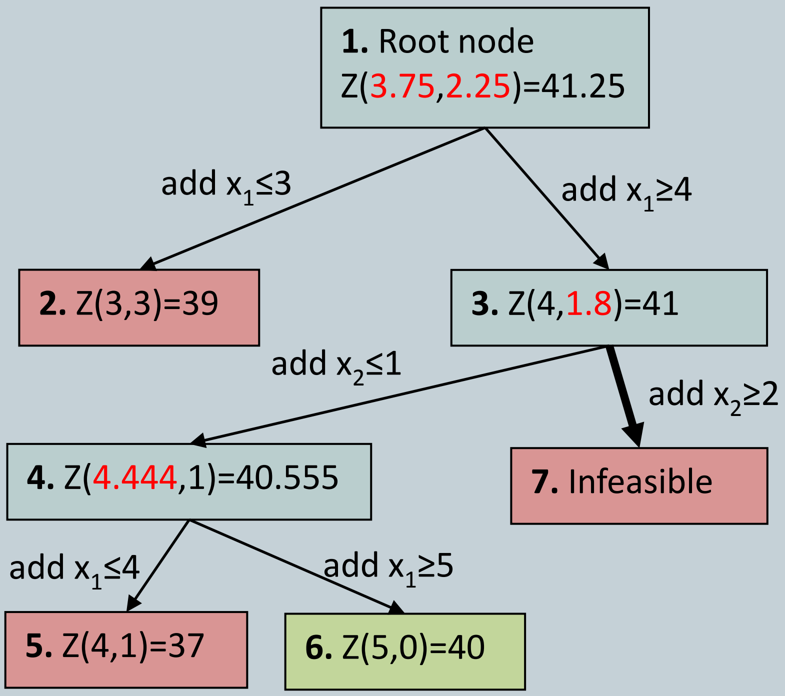
Observations: Squaring-off

- Adding bounds “squares off” the feasible region
- Objective function eventually “catches” on a squared-off cornerpoint
- Number of candidates generally decreases deeper in tree



Depth and Bounding Function Value

- Bounding function values get worse (or stay the same) as you descend
- Each new level removes part of parent feasible region:
  - LP relaxation solution can only get worse (or stay the same)
- Solution stalls when bounds do not change much between levels

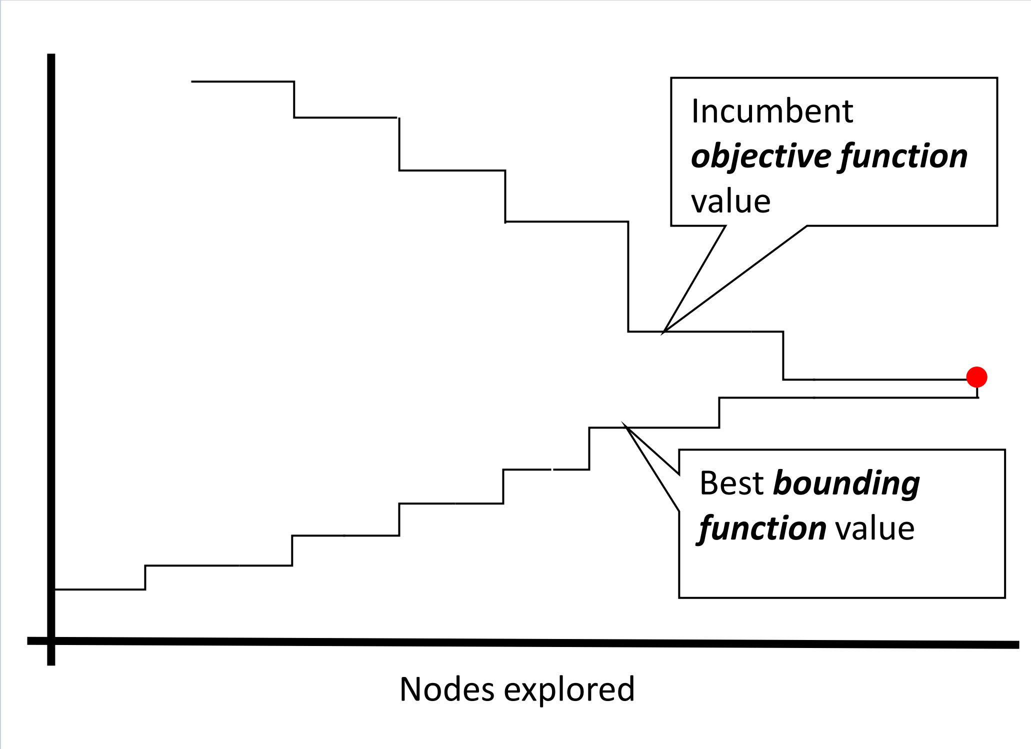


Observations

- ***Any*** set of tree rules (node selection, branching variable selection and direction) will solve the MIP correctly.
  - Different sets of rules generate different trees
  - Some trees are much more efficient!
- Simplex method preferred for LP solutions because of ease of advanced start in child nodes.
- Some MIPs do not terminate (rare).
- Good early incumbent helps prune the search tree.
  - Nodes with worse values of bounding function are removed.

Converging Bounds

- MIP solved when the upper and lower bounds converge:
  - incumbent objective function value
  - best bounding function value
- To speed the process:
  - Better incumbents early
  - Tighter bounding function values



Converging bounds when minimizing

Measuring Solution Speed

- Total solution time: the gold standard.
- Total simplex iterations
  - Approximates total time
  - Ignores non-LP time (e.g. choosing node, variable, etc.)
  - Useful if running on heterogeneous machines
- Total number of nodes
  - May not correlate with time at all
  - E.g. Depth-first search may have many more nodes but take much less time due to simplex advanced starts.
- Example: *pk1*
  - *Depth-first*: **4058 s**; 9,778,734 iterations; 1,965,503 nodes
  - *Best-projection*: 12,623 s; **4,329,434 iterations**; **820,924 nodes**

State of the Art

NODE SELECTION
BRANCHING VARIABLE SELECTION
BRANCHING DIRECTION SELECTION
OTHER CONCEPTS

Node Selection

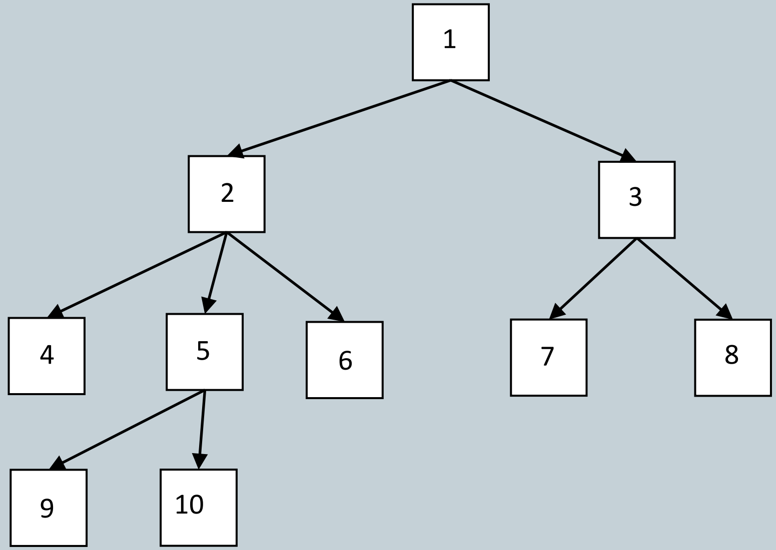
- General goal (unrealistic): *always choose a node that is an ancestor of an optimum node.*
  - i.e. Avoid superfluous search
- How much difference does it make?
  - *Mas76*: depth-first 1,307 s, best-projection 20,610 s
- Philosophies:
  - *Pattern-based*: breadth-first, depth-first
  - *Forecasting*: best-first, best-estimate, best-projection

Depth-First Node Selection

- Choose next node from among last nodes created
  - Also need rule for branching direction: branch up or down?
- Backtrack at leaf node:
  - Choose last created active node.
- Speed advantage for MIP:
  - Next LP identical to last one solved, except for one bound
  - Next solution very quick due to advanced LP start
- Often finds first incumbent early

Breadth-First Node Selection

- Add nodes to bottom of a list as they are created
- Choose next node from top of list



Best-First Node Selection

- Forecasting method, but with limited lookahead
  - Just a bound on how well you *might* do
- Choose unexplored node with best bounding function value anywhere in tree
  - Unexplored nodes initially given bounding function value from parent node.
- Disadvantage for MIP:
  - Frequent re-starts of simplex solution without having the factorized basis from the parent node.

Best-Projection Node Selection

- Forecasting, with lookahead:
  - *Project* objective function value at a feasible descendent of current node.
- Assume constant rate of worsening of Z per unit integer infeasibility at the root node solution.
  - For minimization, $Z_{incumbent} > Z_{root}$:

$$estimate_i = Z_i + \left(\frac{Z_{incumbent} - Z_{root}}{Inf_{root}} \right) \times Inf_i$$

- For min: choose node that gives smallest estimate.

An Aside: Pseudo-Costs

Estimate effect on Z due to change in value of variable

- Minimizing: $Z_{child} \geq Z_{parent}$, so $\Delta Z = Z_{child} - Z_{parent}$
- f_j = fractional part of variable, e.g. 0.7 if x =9.7
- Calculate separately for up and down branches on every integer variable,

$$P_j^{down} = \Delta Z_j^{down} / f_j$$

$$P_j^{up} = \Delta Z_j^{up} / (1 - f_j)$$

- Many different estimating and updating schemes

Best-Estimate Node Selection

- Forecasting, with lookahead based on pseudo-costs
  - *Estimate* Z at a feasible descendent of current node using pseudo-costs for each candidate variable

- For minimization:

$$estimate_i = Z_i + \sum_j \min\left\{P_j^{down} f_j, P_j^{up}(1 - f_j)\right\}$$

Other Node Selection Variants

- *Most feasible node selection*: choose node having smallest sum of fractional values over all candidate variables

- Combinations:
  - Depth-first to first incumbent, then best-first
  - Interleave best-estimate with occasional best-first
  - Etc.

Triggering Backtrack or Jumpback

- Assuming depth-first node selection:
  backtrack at a leaf (LP-infeasible or integer-feasible)
- *Any other reasons to backtrack or jumpback?*
  - *Jumpback: select a node other than the backtrack node*
- Trigger using *aspiration value*:
  - User-selected limit on objective function value:
    - Bounding function value must be at least this good to explore node
  - Backtrack or jumpback if node bound is worse than the aspiration value

Branching Variable Selection

- How much difference does it make?
  - *Momentum1*: time to first incumbent
  - Cplex 9.0 default: time out at 28,800 s. Method B: 75 s.
- Most common idea:
  - Choose variable that *worsens Z the most* in child node
  - Gives a tighter bound on descendent nodes
- Some methods choose variable ***and*** direction

Simple Variable Selection

- Choose variable that is *closest to feasibility*
- Choose variable that is *farthest from feasibility* (closest to $f_j = 0.5$)

Pseudo-Cost Variable Selection

- Choose variable whose pseudo-cost *worsens Z the most* in one of the child nodes:

  $$\mathrm{Max}_j\{P_j^{up}\times(1-f_j),\ P_j^{down}\times f_j\}$$

Alternatively choose variable that has:

- Maximum sum of degradations:

  $$\mathrm{Max}_j\{P_j^{up}\times(1-f_j) + P_j^{down}\times f_j\}$$

- Maximum minimum degradation:

  $$\mathrm{Max}_j\{\min(P_j^{up}\times(1-f_j),\ P_j^{down}\times f_j)\}$$

- Maximum product of degradations:

  $$\mathrm{Max}_j\{P_j^{up}\times(1-f_j) \times P_j^{down}\times f_j\}$$

Strong Branching and Variants

- Full strong branching:
  - Solve LP for up and down direction for ***every*** candidate varb.
  - Choose variable and direction that degrade Z the most
  - Computationally very expensive!

- Approximations to full strong branching:
  - Limit the number of simplex iterations in each LP
  - Limit which candidate variables are tested (e.g. based on pseudo-costs)

Driebeek and Tomlin

1. Approximate strong branching:
   - Just one dual simplex pivot for each LP
     [can actually just be estimated, not performed]
2. Choose *variable* that has largest degradation in either direction
3. Choose *direction* that gives smallest degradation

- Default branching method in GLPK

Many Variants:

- Hybrid strong/pseudo-cost branching
  - Strong branching high in tree
  - Pseudo-cost branching below a certain level

- Reliability branching:
  - Pseudo-cost branching, *except*…
  - Strong branching on varbs with uninitialized pseudo-costs and unreliable pseudo-costs

- Etc.

Branching Direction Selection

Common rules:

- Branch *up* always
  - Generally best in practice.
- Branch *down* always
- Branch to *closest* bound
- Branch to *farthest* bound
- Direction that forces branching variable away from its value at the root node
- Solver-proprietary rules

Other Concepts

- Branch and Cut
- Branch and Price
- Preprocessing and Probing
- Neighbourhood search:
  - Limited B&B search in the “neighbourhood” of promising node
- Special Ordered Sets:
  - Enforce specified order of variable selection under certain conditions
- Specialized feasibility-seeking algorithms:
  - OCTANE for binary problems
  - Pivot-and-complement, pivot-and-shift
  - The feasibility pump prior to B&B
- No-good learning
- General disjunctions:
  - Linear disjunctions that are not axis-parallel
- Parallel processing
- Etc.!

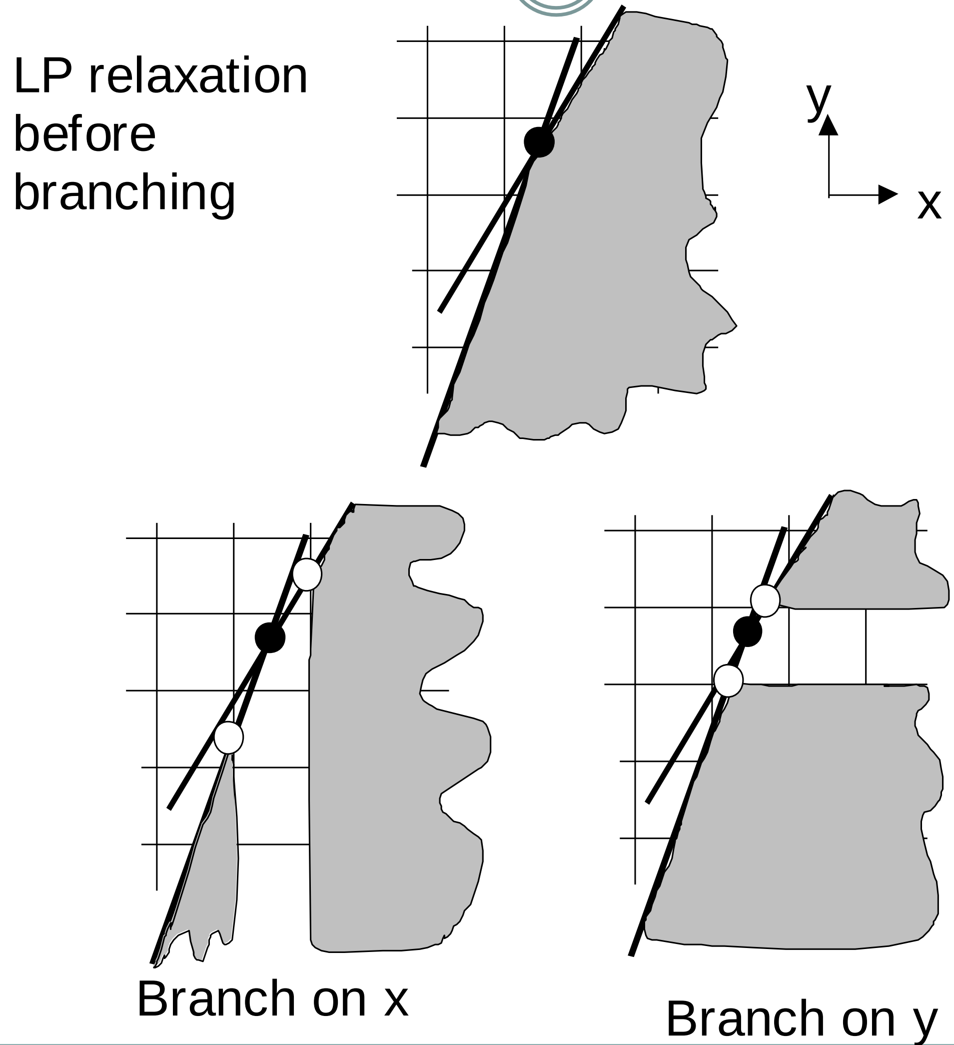
New Directions

ACTIVE-CONSTRAINT VARIABLE SELECTION
NEW NODE SELECTION METHODS
BRANCHING TO FORCE CHANGE
GENERAL DISJUNCTIONS

Active-Constraint Variable Selection

- Concept:
  - LP-relaxation optimum is fixed by *active constraints*
  - For different child optima, must impact the active constraints
  - Choose candidate variable that has *most impact on active constraints* in current LP-relaxation solution
- *Constraint-oriented* approach vs. usual objective-function-oriented approaches
- Focus on reaching first incumbent quickly

Illustration



Estimating Impact on Active Constraints

1. Calculate "weight" W_{ik} of each candidate i in each active constraint k
   - *0 if the candidate does not appear in constraint*
2. For each candidate, total the weights over all of the active constraints.
3. Choose candidate having largest total weight.

- *Dynamic* variable ordering: changes at each node
- Many variants: A through P

Overview of Weighting Methods

- Is candidate variable in active constraint or not?
- Relative importance of active constraint:
  - Smaller weight if more candidate or integer variables: changes in other variables can compensate for changes in selected variable.
  - Normalize by absolute sum of coefficients.
- Relative importance of candidate variable within active constraint:
  - Greater weight if coefficient size is larger: candidate variable has more impact.
- Sum weights over all active constraints? Look at biggest impact on single constraint?
- Etc.

Some of the Better Weighting Schemes

- **A:** W_{ik}=1.
- **B:** W_{ik} = 1/ [Σ(|coeff of *all* variables|].
- **L:** W_{ik} = 1/(no. *integer* variables)
- **O:** W_{ik} = |coeff$_i$|/(no. of *integer* variables)
- **P:** W_{ik} = |coeff$_i$|/(no. of *candidate* variables)

H methods: choose largest *individual* value of W_{ik}

- **H$_M$:** W_{ik} = 1/[no. *candidate* variables]
- **H$_O$:** W_{ik} = |coeff$_i$|/(no. of *integer* variables)

- *Variants*: voting, multiply by dual costs, etc.

Test Models

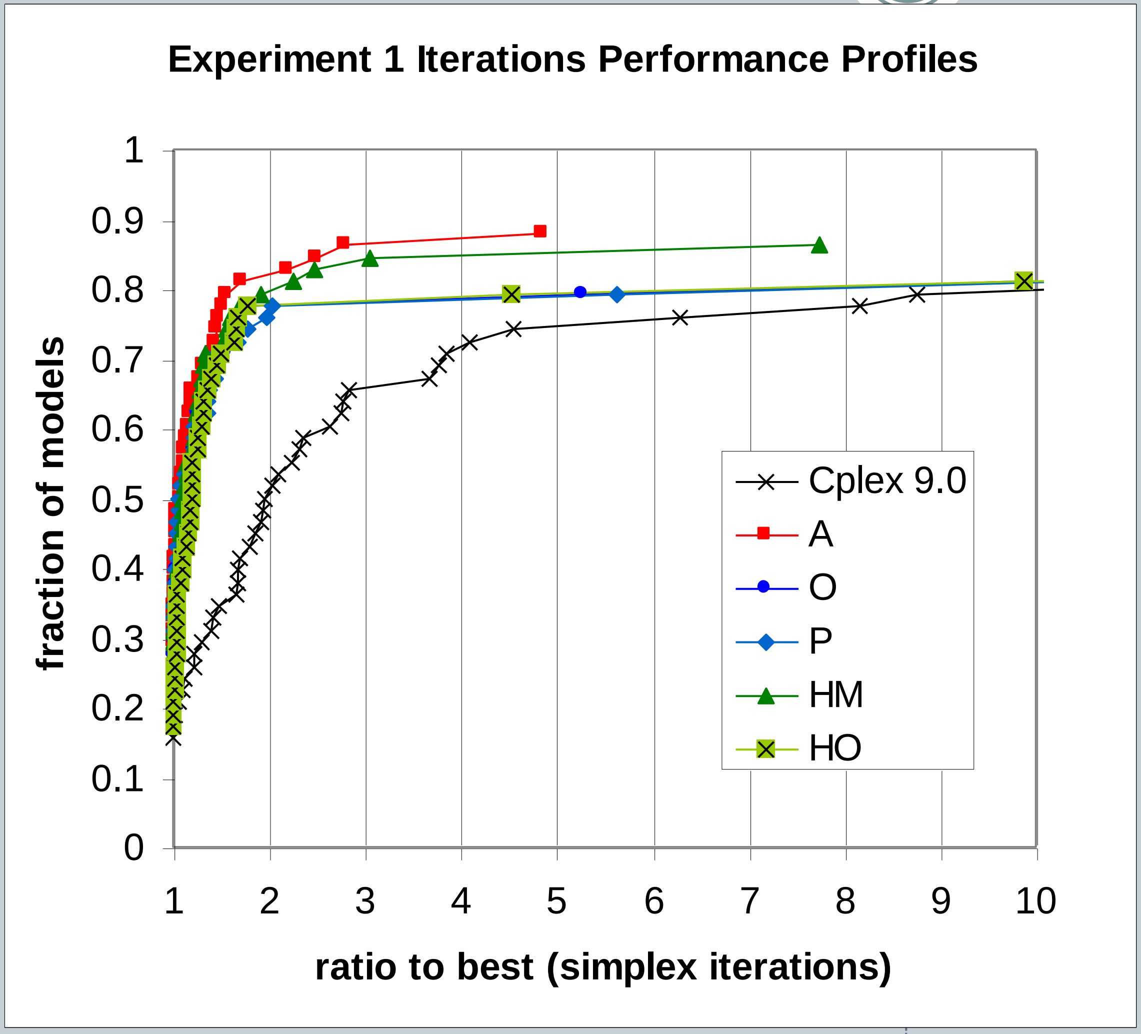
MIPLIB 2003 set

- 60 models (58 used: 2 time out on all methods)
- Range of difficulties
- Rows: 6–159,488
- Variables: 62–204,880
  - Integer variables: 1–3,303
  - Binary variables: 18–204,880
  - Continuous variables: 1–13,321
- Nonzeroes: 312–1,024,059

Experiments

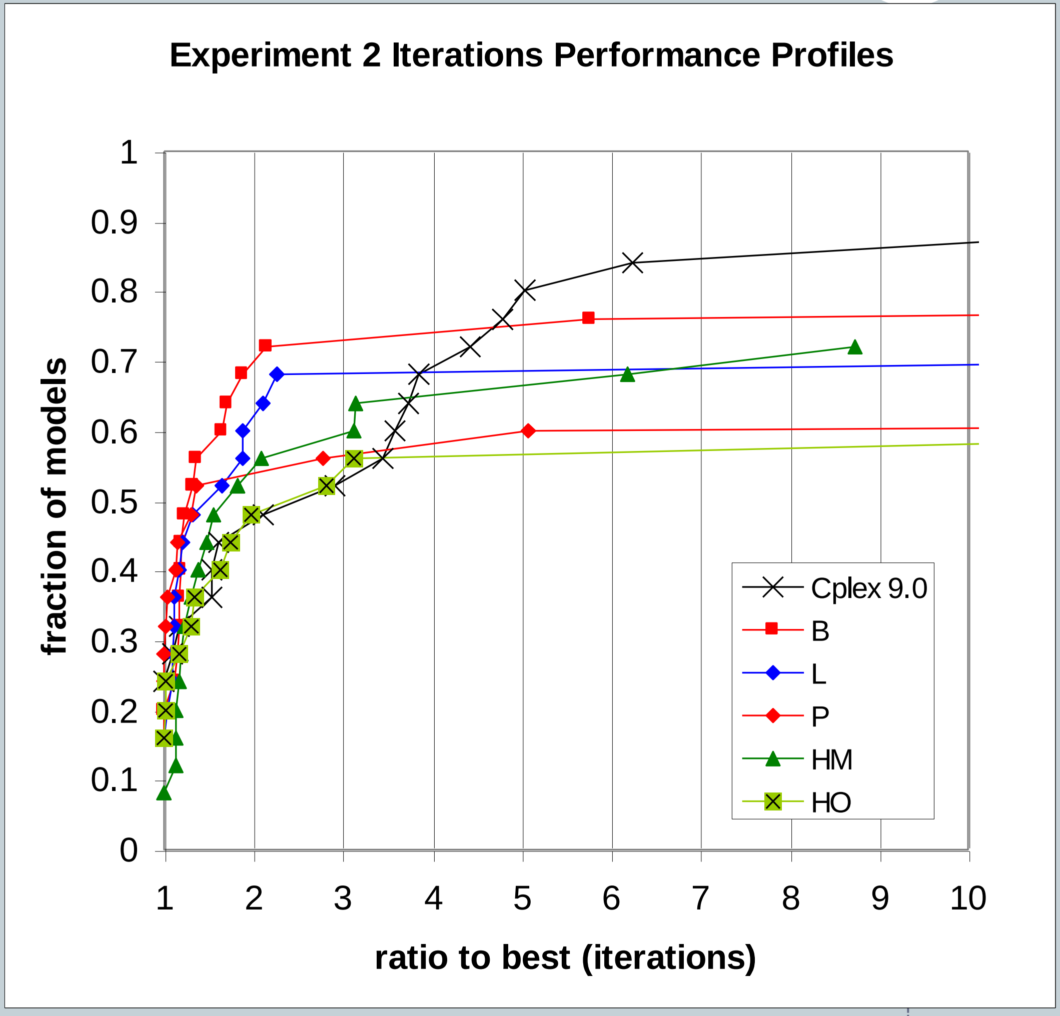
- Cplex 9.0 (baseline): all default settings, except:
  - MIP emphasis: find feasible solution
  - *Experiment 1 (basic B&B)*: all heuristics **off**
  - *Experiment 2*: all heuristics turned **on**
- Active Constraint solver:
  - Built on top of Cplex
    - Callbacks set branching variable
    - Data structures not optimized: inefficient search
  - Node selection:
    - *Experiment 1*: Straight depth-first, branch up
    - *Experiment 2*: Cplex default
- *Goal:* find first integer-feasible solution quickly.

Experiment 1: Heuristics Off



- Method A faster than Cplex: 43/58 tests (74.1%)

Experiment 2: Heuristics On



- 25 models used:
  - 32 solved at root node
  - 3 failed on all methods
- Method B faster than Cplex: 14/25 models (56%)
- Cplex heuristics on:
  - Half of models solve *faster* than before
  - Half of models solve *slower* than before

First Incumbent Better than Cplex

- Optimality Gap measures “distance” of solution from (unknown) optimum solution

- For minimization:
  - Z_{low}: lowest bound on an active node
  - Optimality gap: $[|Z_{low} - Z_{incumbent}|]/[\varepsilon+|Z_{low}|]$

- Experimental Results:
  - **Exp. 1:** active constraint methods have smaller gaps than Cplex (53% for A, 78% for method P)
  - **Exp. 2:** active constraint methods have smaller gaps than Cplex (75% for B, 50% for H_M)

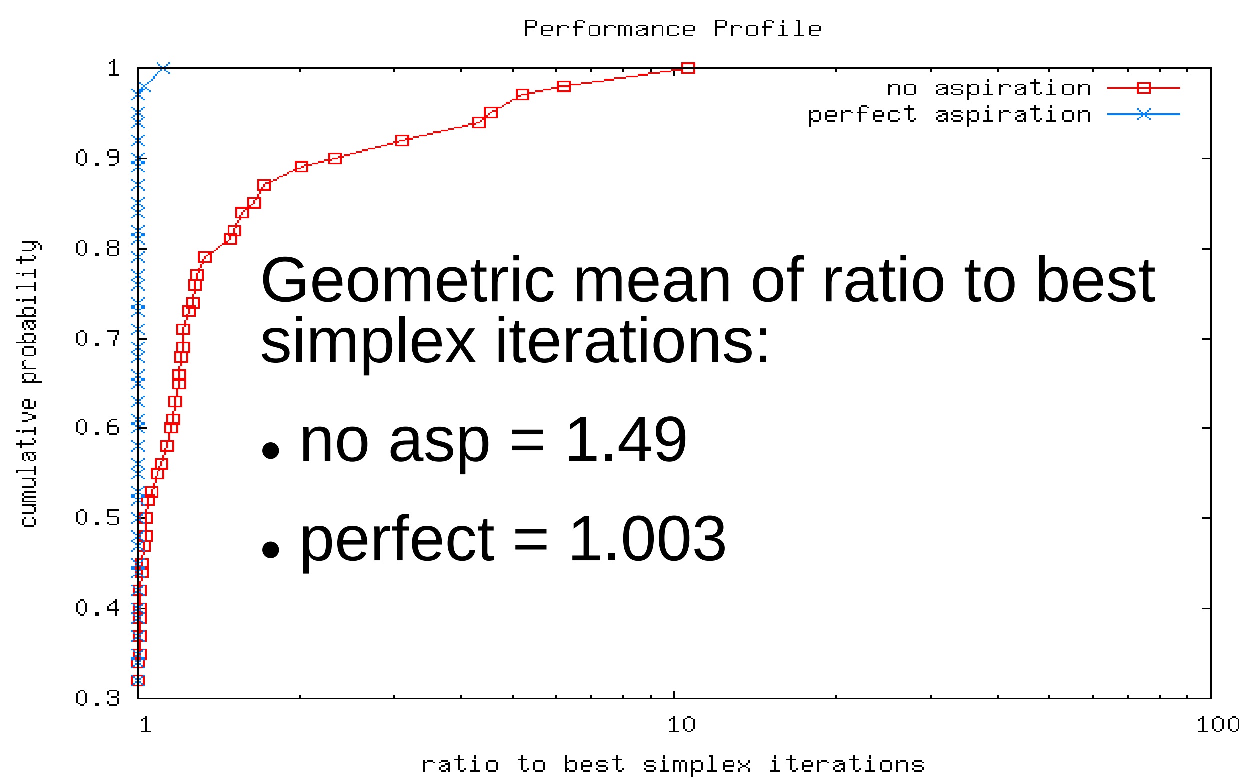
New Node Selection Methods

- Triggering Backtrack
  - Feasibility Depth Extrapolation
  - Modified Best Projection Aspiration
- Choosing Node When Backtracking
  - Modified Best Projection
  - Distribution-based Backtracking
  - Active Node Search Threshold: Changing Methods
- *Goal:* optimum solution as quickly as possible

Triggering Backtrack or Jumpback

- Try to trigger when all node descendents:
  - ***Unlikely*** to be optimal, or
  - ***Unlikely*** to be feasible.

- Potential improvement:
  - Suppose perfect aspiration



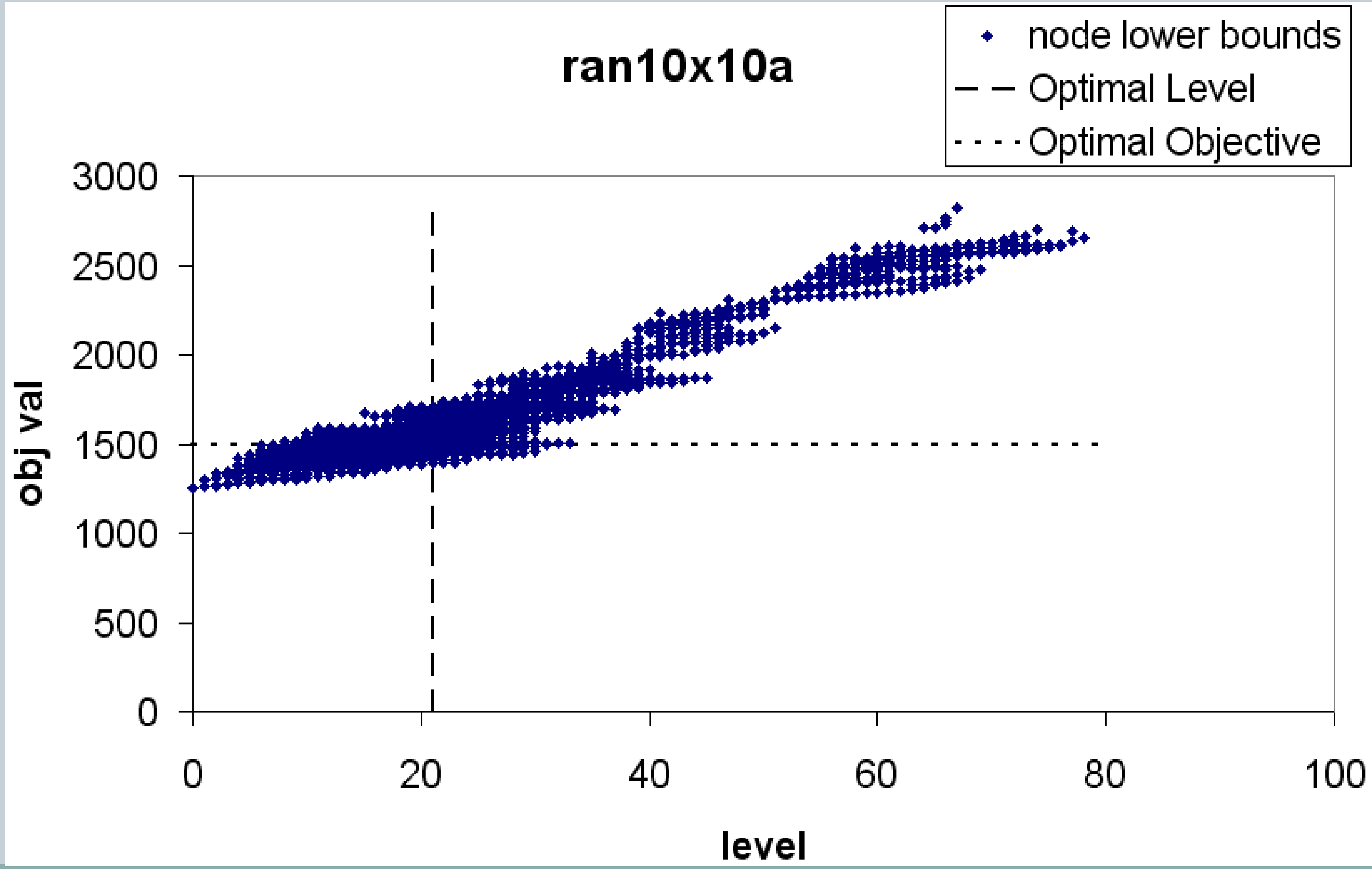
Set Aspiration by Estimating Z*

- Z*: optimum objective function value
- Z^a: aspiration value based on estimate of Z*

- Adapting node selection methods to estimate Z*:
  - Best-Estimate (uses pseudo-costs)
  - Best-Projection (uses ratio of degradation in Z between root and incumbent to reduction in integer infeasibility).
- Very little improvement: need something better!

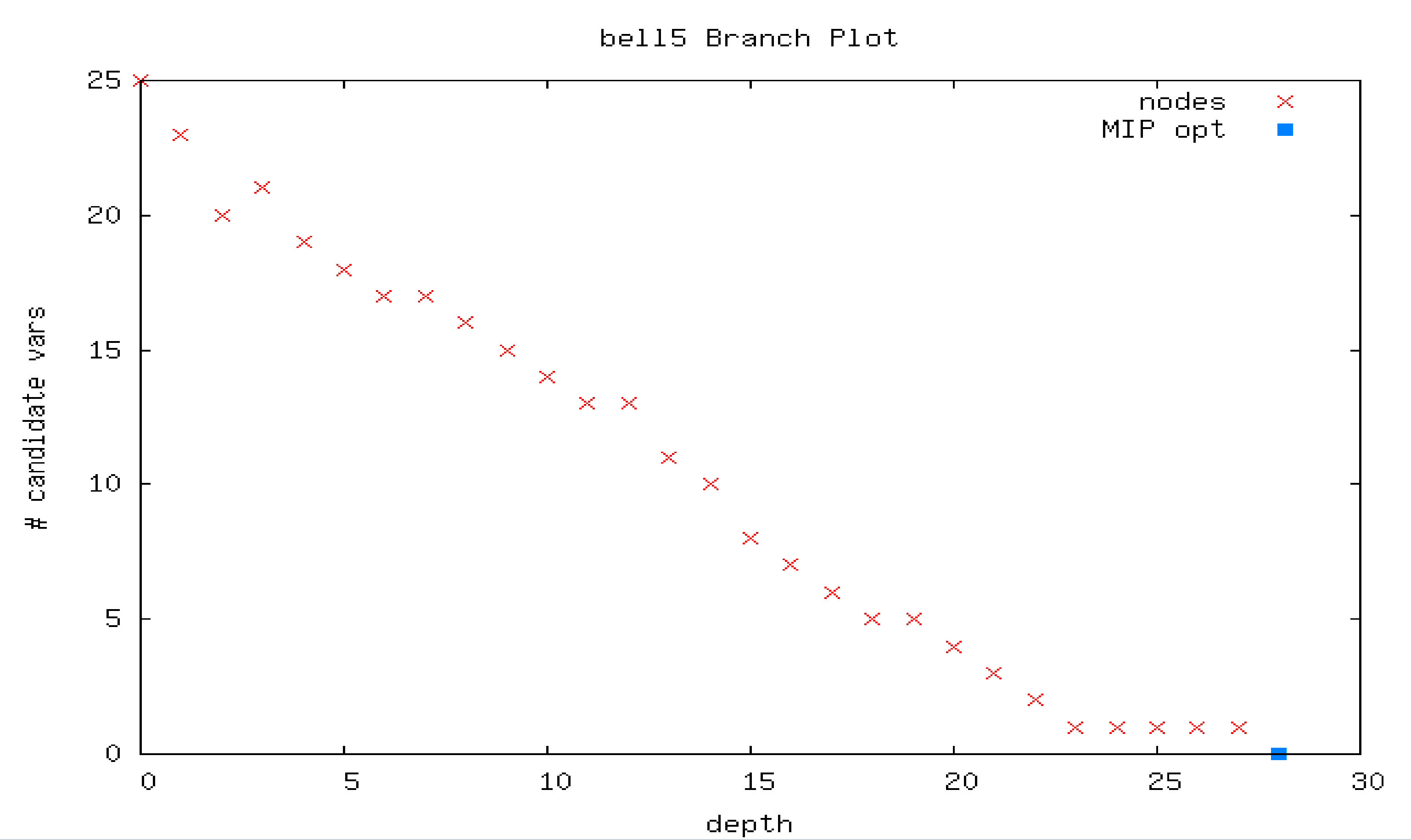
New: Projection Based on Depth of Z*

- *Observation:* nodes at depth of Z* have similar value

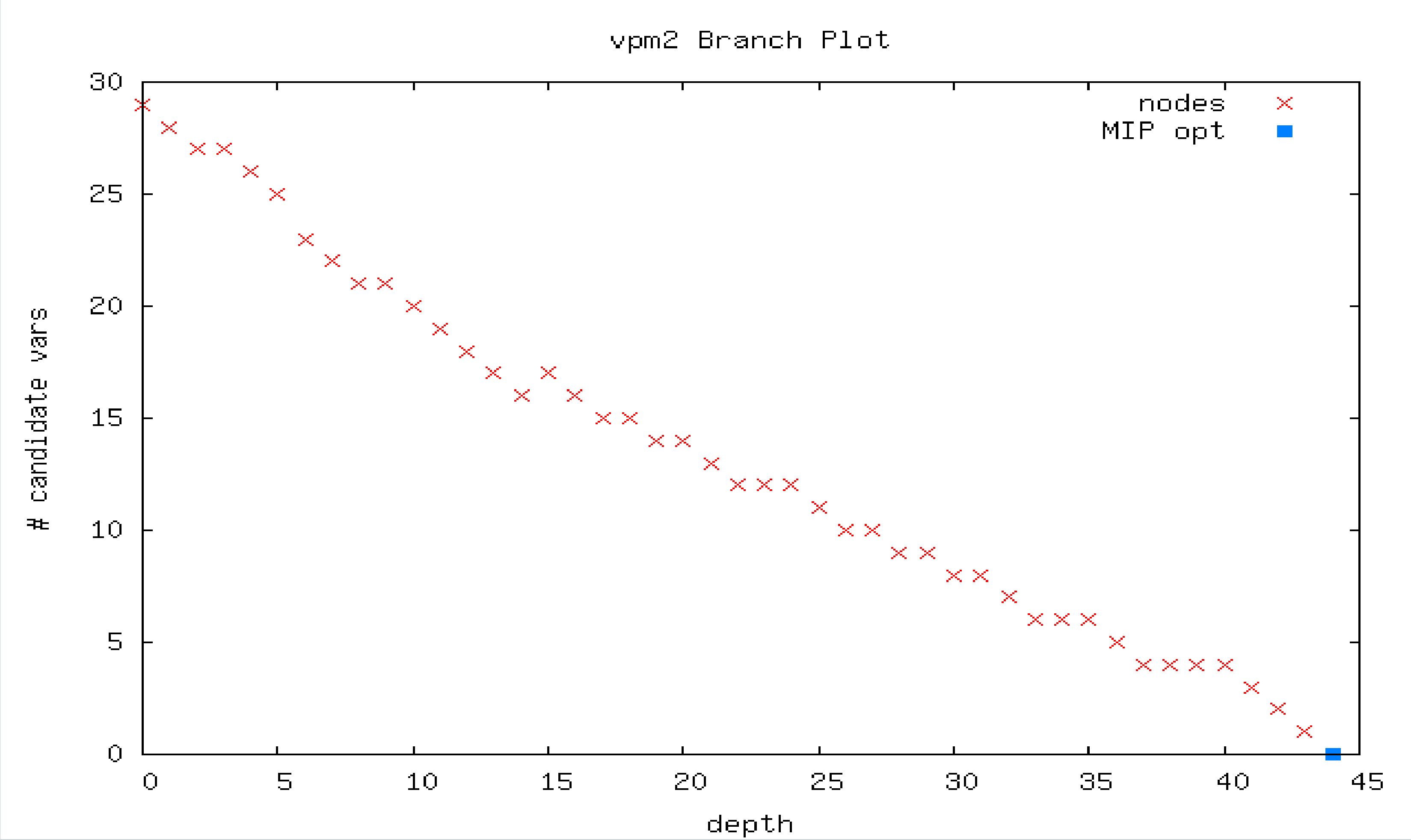


Frequent Pattern

- Zero candidates = feasible solution

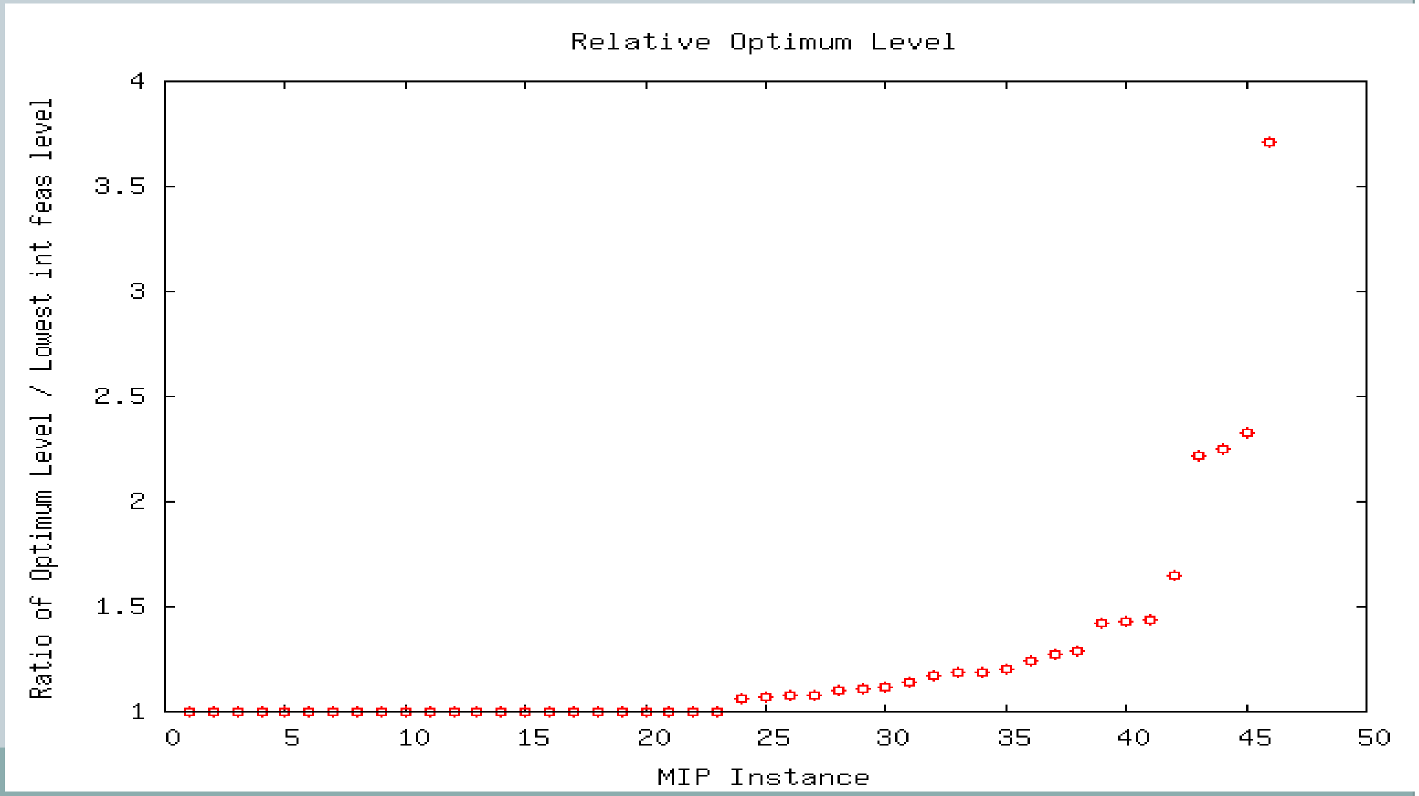


Linear Projection to Estimate Z* Depth



Reconciling Multiple Active Nodes

- Each active node has its own projection of Z* depth
  - Which one should we use?
  - Frequent pattern of optima: use the shallowest projection!



Method: Linear Extrapolation to Estimate Z*

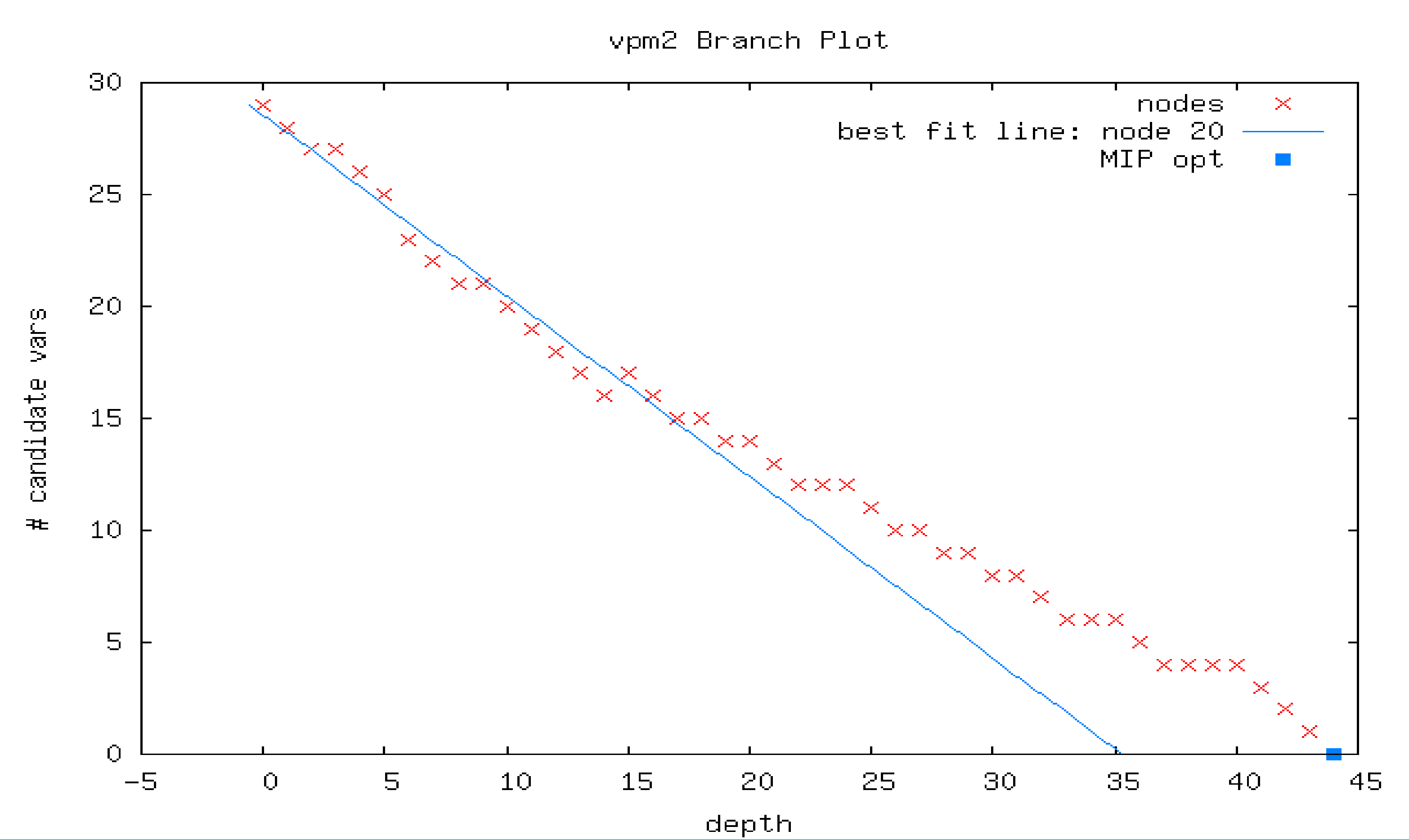
- For every active node with depth ≥ 20
  - Fit least-squares line to number of candidates vs. depth using all ancestor nodes
  - Project depth of closest feasible solution (zero candidates)
- *k* = smallest extrapolated depth over all nodes
- Z^a = max of Z^i over all nodes at depth (conservative)

Notes

- Z^a is the aspiration value that is set (minimization)
- 20 chosen empirically: enough data to extrapolate

Example

- Usually close, not perfect…



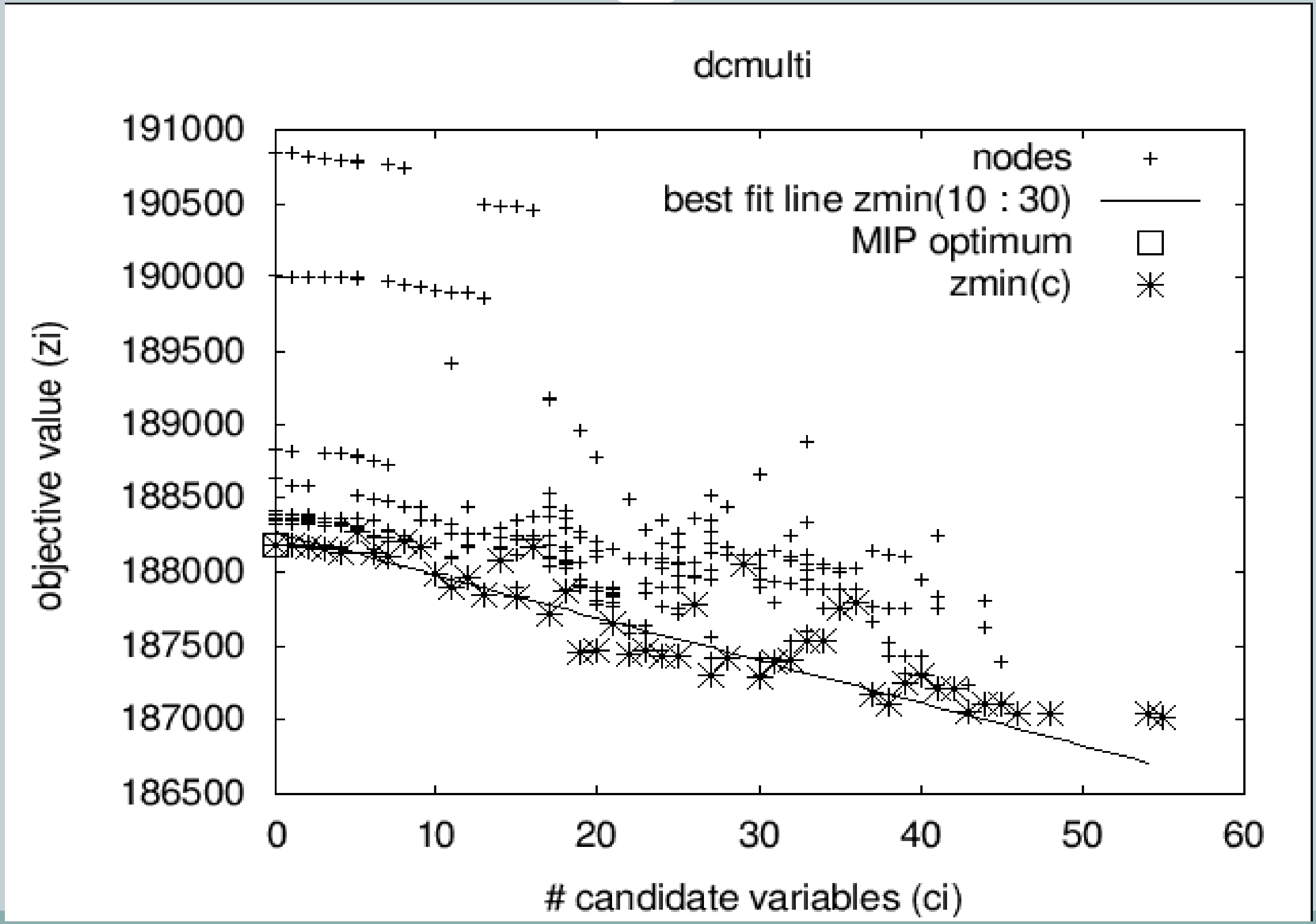
New: Modified Best Projection Aspiration

Best projection for node selection (minimization):

- $Z^a = Z^i + (Z^{inc} - Z^o)s^i/s^o$
  - s^i: sum of integer infeasibilities at node i
  - s^o: sum of integer infeasibilities at root node
- But we don't always have an incumbent!

- $Z^{min}(c)$: min Z at given number of candidates (c)
  - $Z^{min}(0)$ is optimum objective function value
- There is a pattern…

Projecting $Z^{min}(c)$



Modified Best Projection Aspiration

- $Z^a = Z^i + C^i[Z^{min}(C^{min})-Z^o]/(C^o-C^{min})$
  - C^i: number of candidate variables at node i
  - C^{min}: minimum number of candidate variables at any node
  - Two-point projection: root node through min candidates node

Notes:

- Eliminates need for an incumbent
- Closeness to feasibility measure:
  - number of candidate variables instead of sum of integer infeasibilities
- Also useful for node selection

New: Distribution-based Jumpback

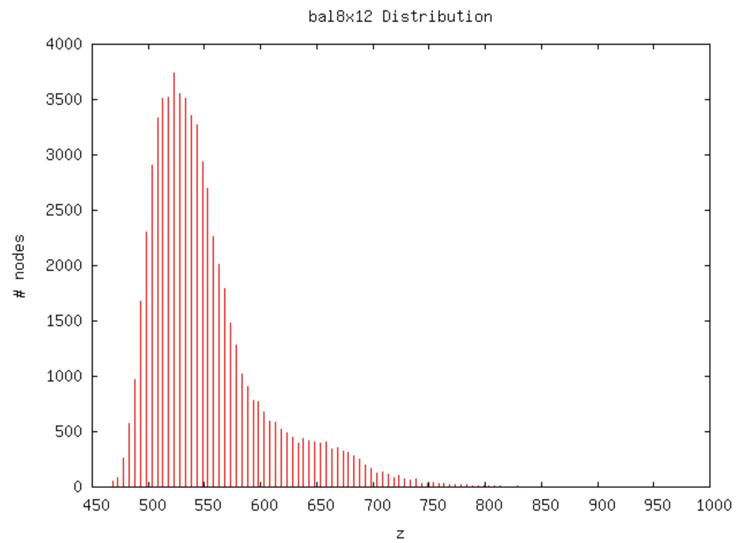
Balance pursuit of ***both*** feasibility and optimality

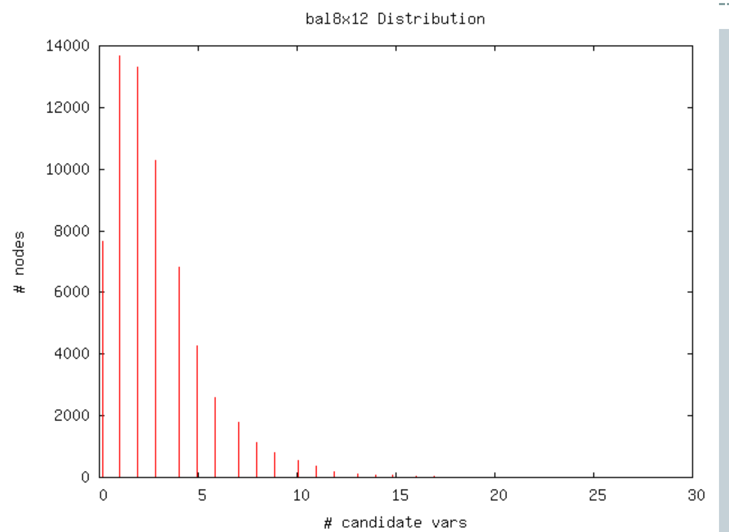
- Minimizing: smaller Z^i and C^i both desirable
- Z^i tends to be *large* where C^i is *small*, and vice versa

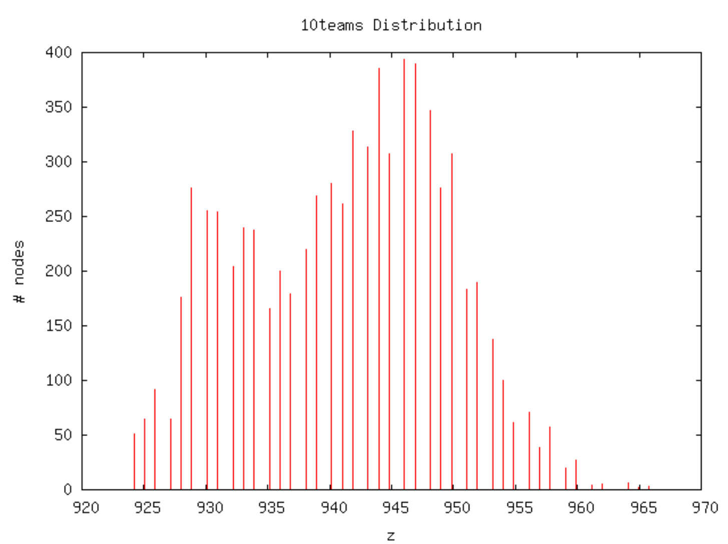
Ranges quite different: how to balance?

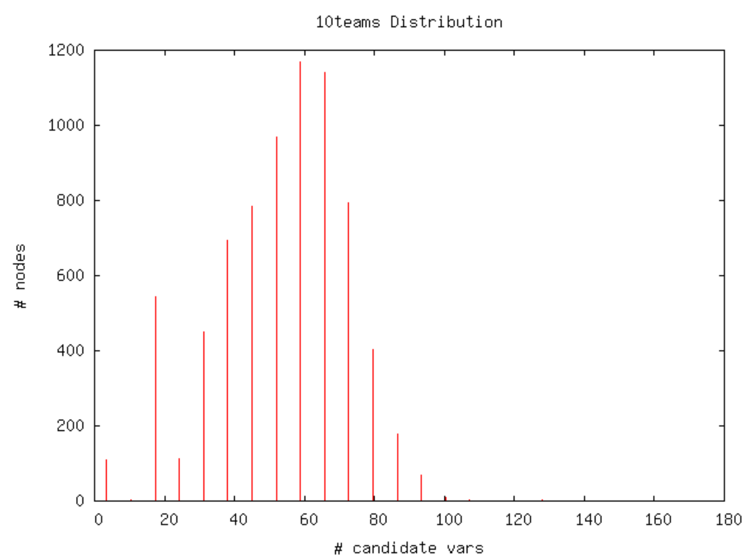
- *Assume* independent normal probability distribns
- Normalize ranges of Z^i and C^i
- $P(Z \leq Z^i, C \leq C^i) = F_Z(Z^i) \times F_C(C^i)$
- Choose node n where $n = arg\ min_i\ P(Z \leq Z^i, C \leq C^i)$
  i.e. node n has lowest prob. of being "beaten"

Distributions: Gaussian-like





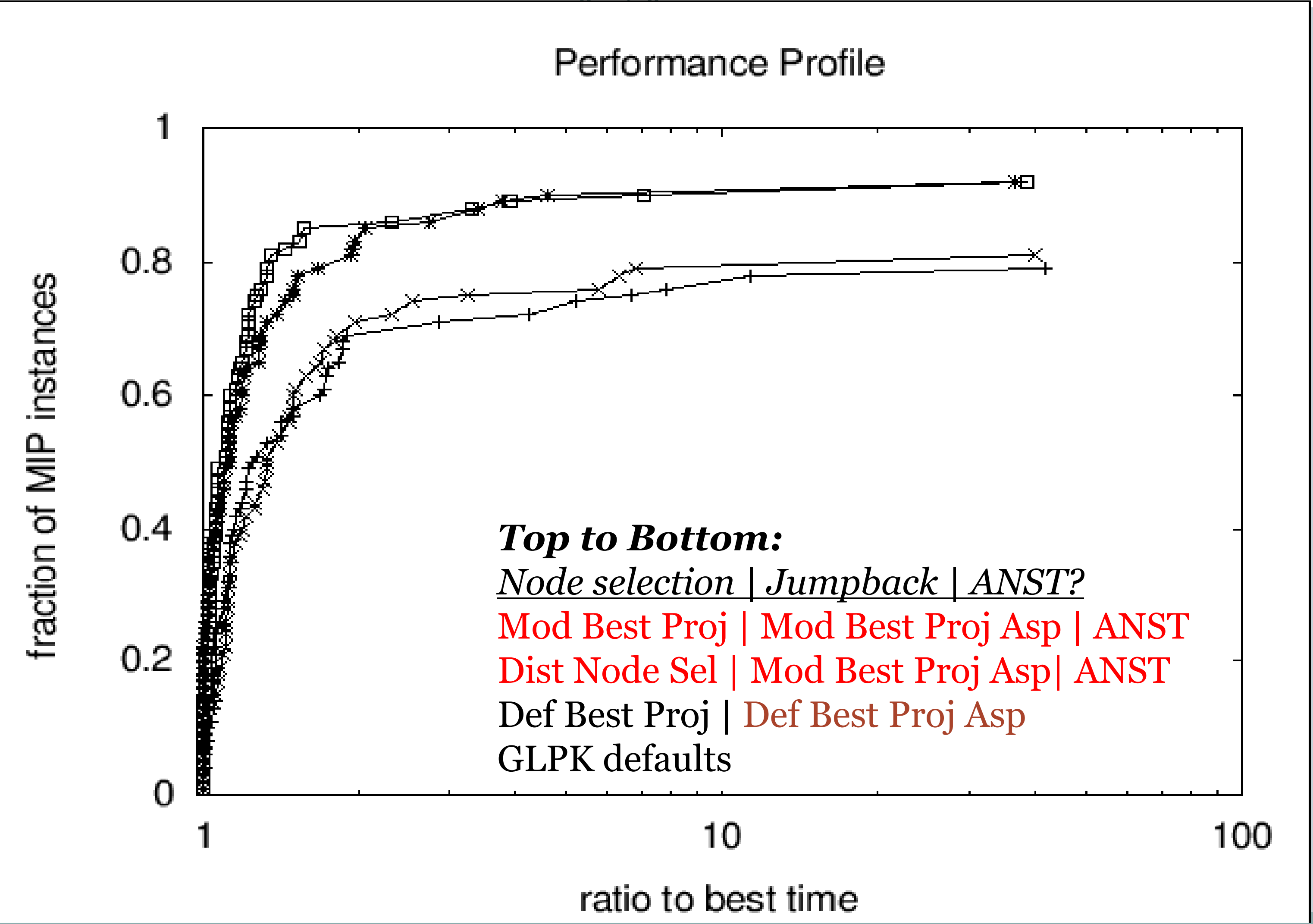




New: Active Node Search Threshold

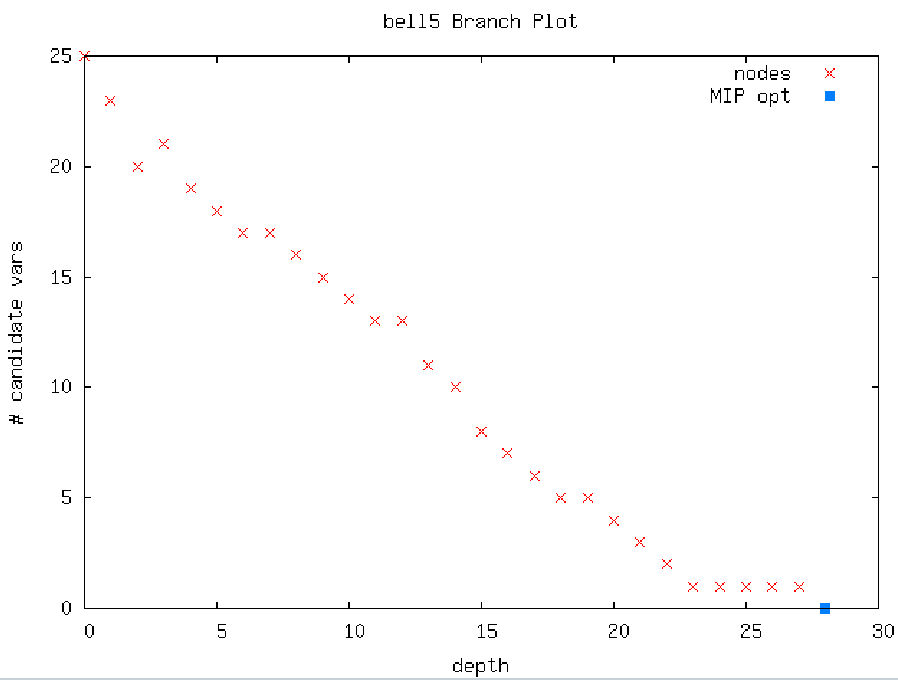
- Advanced node selection can be time-consuming
- ANST: switch to simple depth-first backtracking under certain conditions
  - R_t = (cum. time for node selection)/(cum. time for all else)
  - If $R_t > 0.1$, then switch to simple depth-first node selection

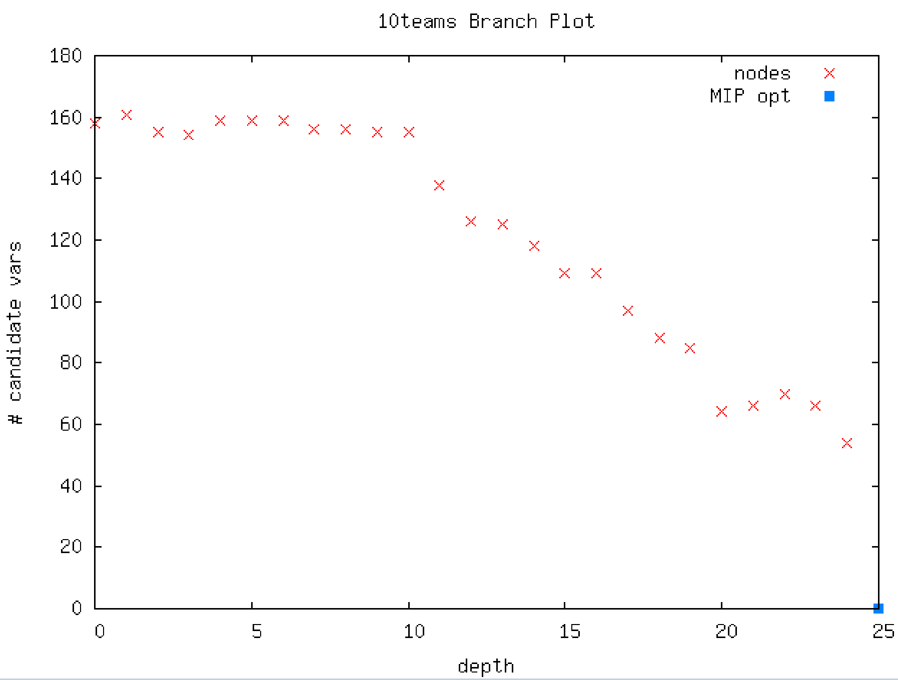
Testing Numerous Combinations



Branching to Force Change

- *Question:* What is the best branching strategy to reach first incumbent quickly?
  - Force more candidates to integrality at each branch.





Basic Question

- You can either:
  a) Branch to have ***largest*** probability of satisfying constraints in a MIP, or
  b) Branch to have ***smallest*** probability of satisfying constraints in a MIP.

- *Which policy leads to the first feasible solution more quickly?*

Clue: Active Constraints Variable Selection

- Choose candidate variable having greatest impact on the *active constraints* in current LP relaxation
  - All other methods look at impact on *objective fcn*
- Reaches integer-feasibility very quickly

- ***Method A***:
  - choose candidate variable appearing in largest number of active constraints, branch **up**

Clue: “Multiple Choice” Constraints

$x_1 + x_2 + x_3 + \ldots x_n \{\leq,=\} 1$, where x_i are binary

- *Branch down*: other x_i can take real values
- *Branch up*: all x_i forced to integer values

E.g.: $x_1 + x_2 + x_3 + x_4 = 1$ at (0.25, 0.25, 0.25, 0.25)

Branching on x_1 :

- *Branch down*: (0, 0.333, 0.333, 0.333) *or others*
- *Branch up*: (1, 0, 0, 0) is *only* solution

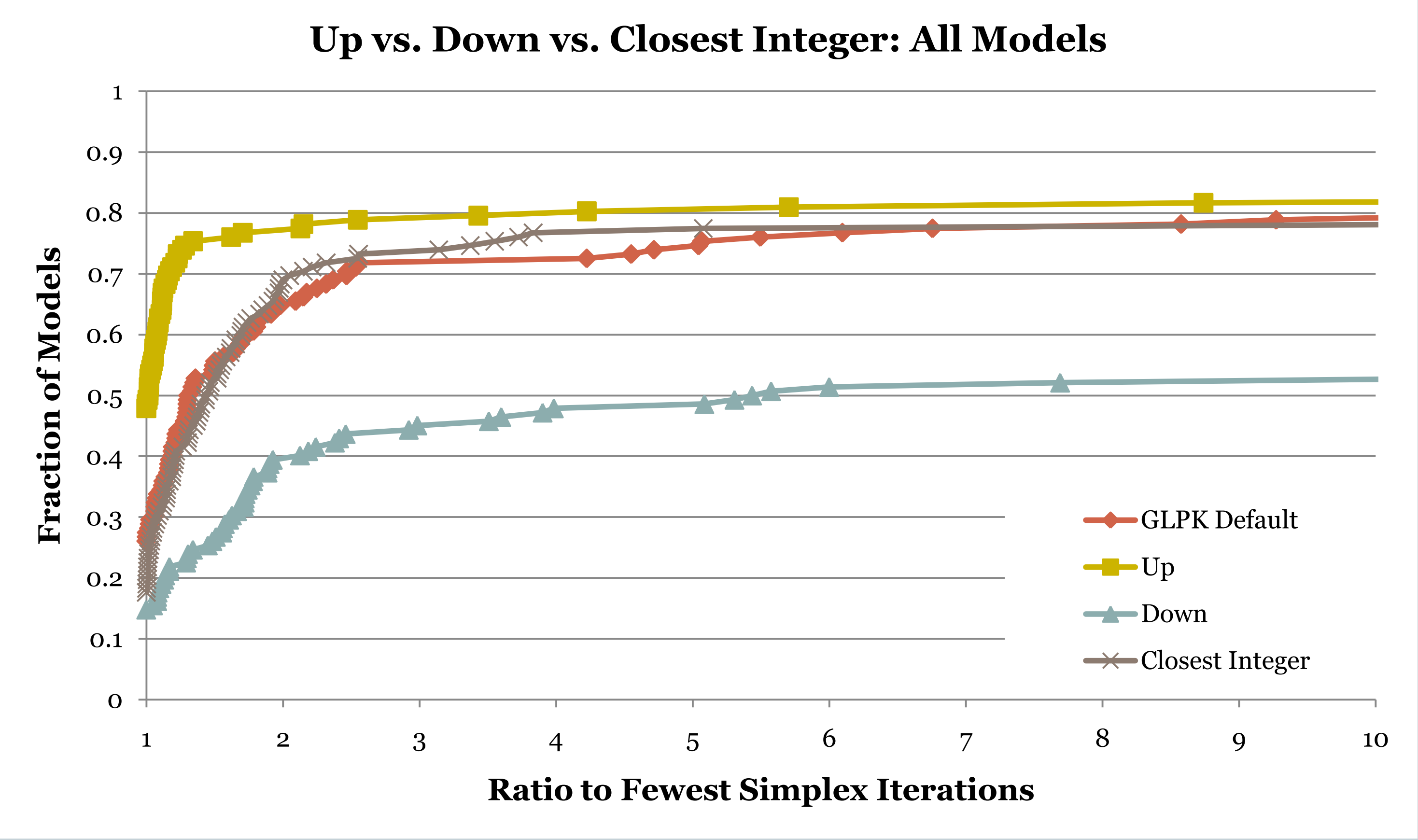
New Principle

- ***Branch to Force Change***
  - E.g. branch **up** on multiple choice constraints
  - E.g. active constraint branching variable selecn

- In general:
  - *Branch to cause change that will **propagate** to as many candidate variables as possible.*
    - Hope that *many* will take integer values.

Branching Direction

- UP is best. *Why?*



New: Probability-Based Branching

Counting integer solutions (Pesant and Quimper 2008)

- $l \leq \mathbf{cx} \leq u$: l, $\mathbf{c}$, u are integer values, $\boldsymbol{x}$ integer
- Example: $x_1 + 5x_2 \leq 10$ where $x_1, x_2 \geq 0$

Value of x_2	***Range for x_1***	***Soln count***	***Soln density***
x_2=0	[0,10]	11	11/18 = 0.61
x_2=1	[0,5]	6	6/18 = 0.33
x_2=2	[0]	<u>1</u>	1/18 = 0.06
Total solutions		18	

- Choose x_2 =0 for max prob of satisfying constraint
- *Is this the best thing to do?*

Generalization

Assume:

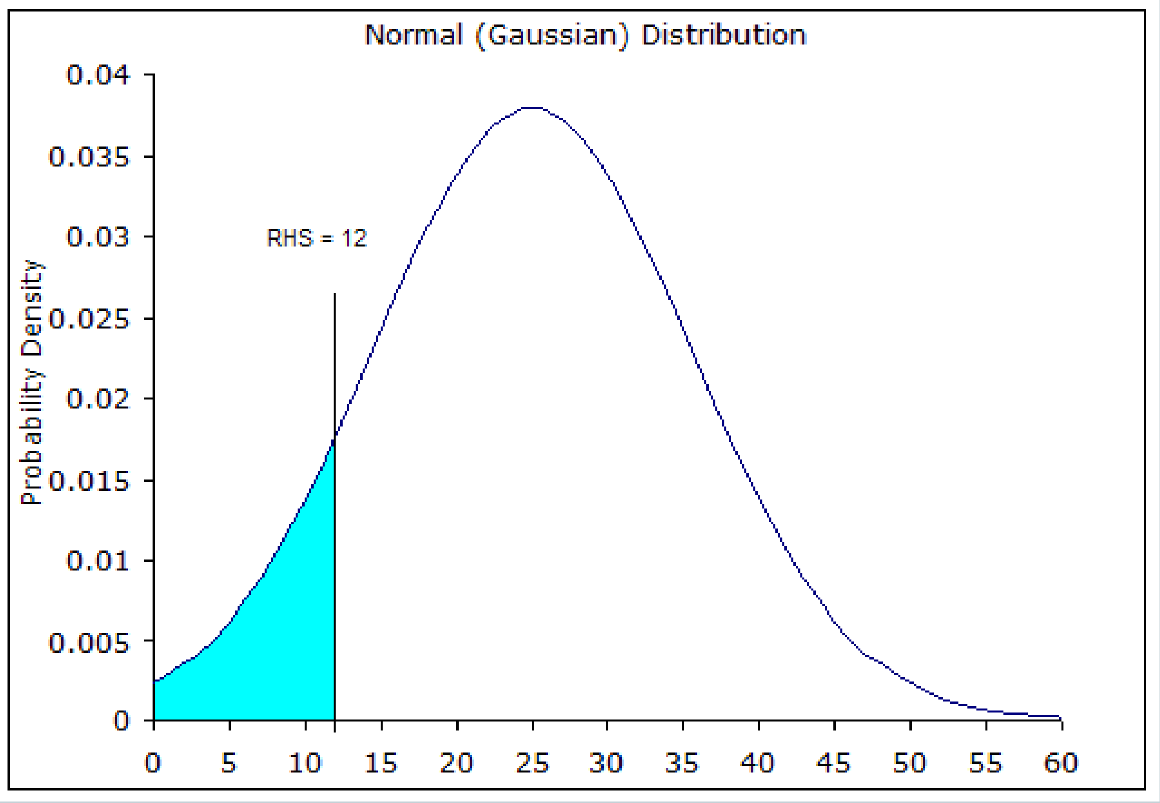
- All variables bounded, real-valued
- Uniform distribution within range

Result:

- linear combination of variables yields approx. *normal distribution* for function value
- Example: $g(\boldsymbol{x}) = 3x_1 + 2x_2 + 5x_3$, $0 \le \boldsymbol{x} \le 5$
  has mean 25, variance 110.83
- *Plot*…. look at $g(\boldsymbol{x}) \le 12$

$$g(\boldsymbol{x}) = 3x_1 + 2x_2 + 5x_3 \leq 12,\ 0 \leq \boldsymbol{x} \leq 5$$

- Probability density plot
  - Cumulative prob of satisfying function in blue

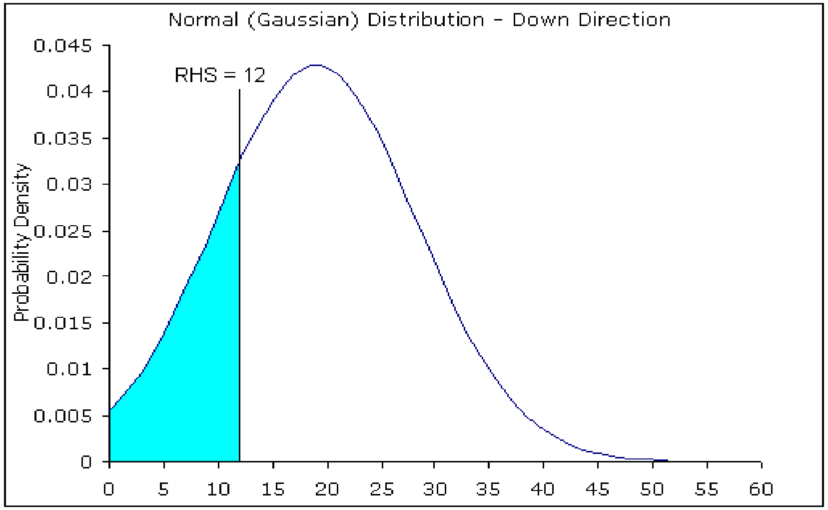


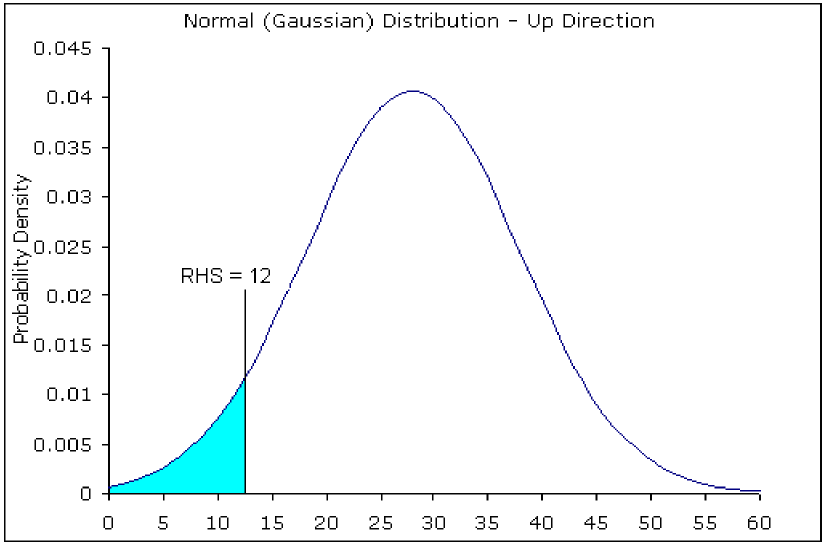
Use for Branching

- ***Different*** distributions for DOWN and UP branches due to changed variable ranges
- ***Different*** cumulative probabilities of satisfying constraint in each direction

Example:

- Branch on x_1=1.5
- *Down*: x_1 range [0,1], p=0.23
- *Up*: x_1 range [2,5], p=0.05





Handling Equality Constraints

- Look at ***centeredness*** of RHS value in UP and DOWN prob. curves
- For each direction:
  - Calculate cum. prob. of ≤ RHS
  - Calculate cum. prob. of ≥ RHS
  - Calculate ratio:
    (*smaller* cum. prob.)/(*larger* cum. prob.)
  - Least centered = zero; most centered = 1
- For "highest prob." methods, choose ***most centred*** direction, i.e. ratio closest to 1
- For "lowest prob." methods, choose ***least centred*** direction, i.e. ratio closest to zero

New Branching Direction Methods

Given branching variable, choose direction:

- Try UP and DOWN for each active constraint branching variable is in. Choose direction:
  - LCP: lowest cum. prob. in **<u>any</u>** active constraint
  - HCP: highest cum. prob. in **<u>any</u>** active constraint
  - LCPV: direction **<u>most often</u>** having lowest cum. prob.
  - HCPV: direction **<u>most often</u>** having highest cum. prob.

Choose Both Variable ***and*** Direction

- VDS-LCP: choose ***varb and direction*** having lowest cum. prob. among all candidate varbs and all active constraints containing them
- VDS-HCP: choose ***varb and direction*** having highest cum. prob. among all candidate varbs and all active constraints containing them

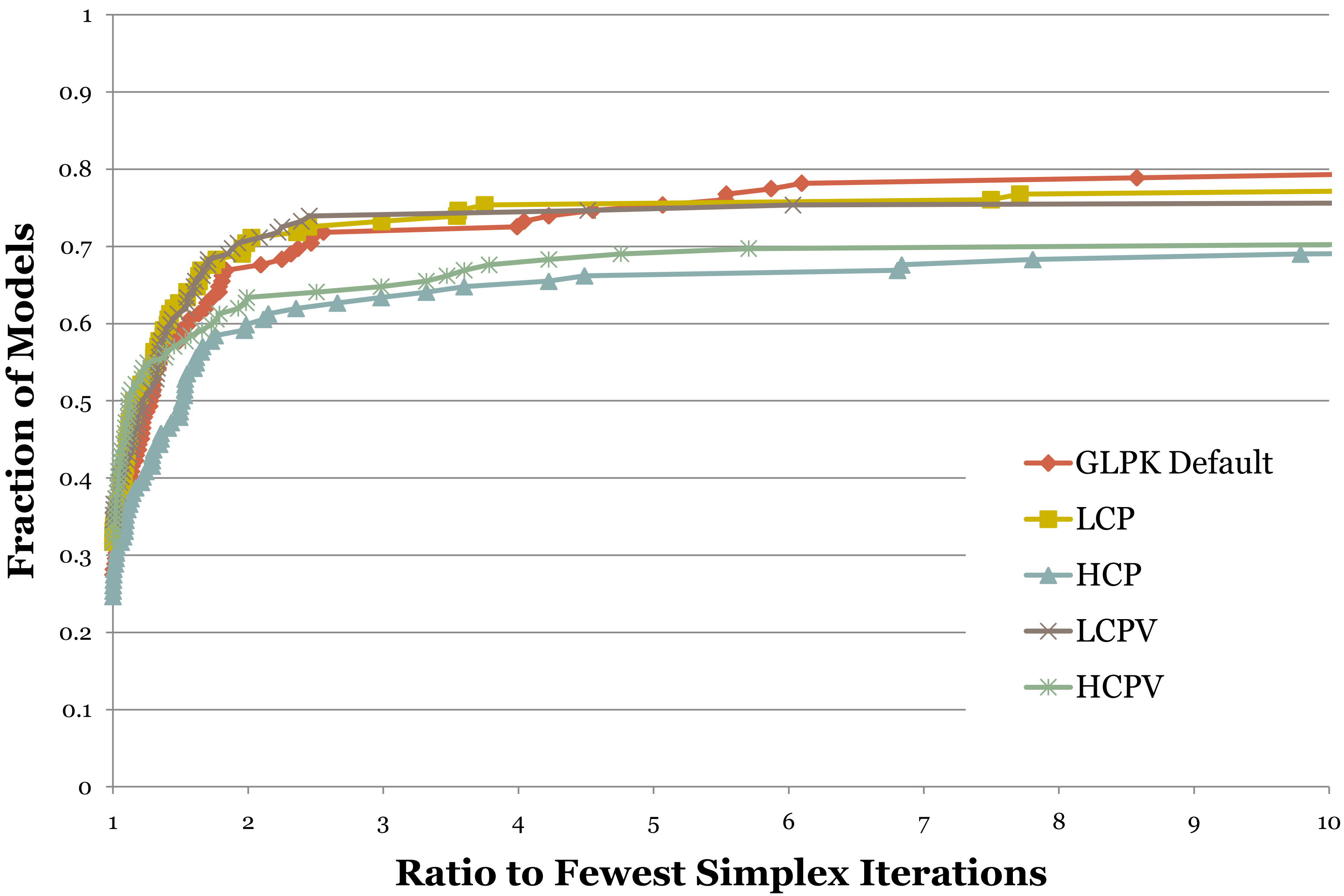
New: Violation-Based Methods

- Fix all variable values ***except*** branching variable. What is effect of branching UP vs. DOWN?
  - *Inequality:* is active constraint violated or still satisfied?
  - *Equality:* construct cum. prob. curves for up/down
    - "violated": less centred direction
    - "satisfied": more centred direction
- MVV: Most Violated Votes method
  - Choose direction that violates largest number of active constraints containing branching varb.
- MSV: Most Satisfied Votes method

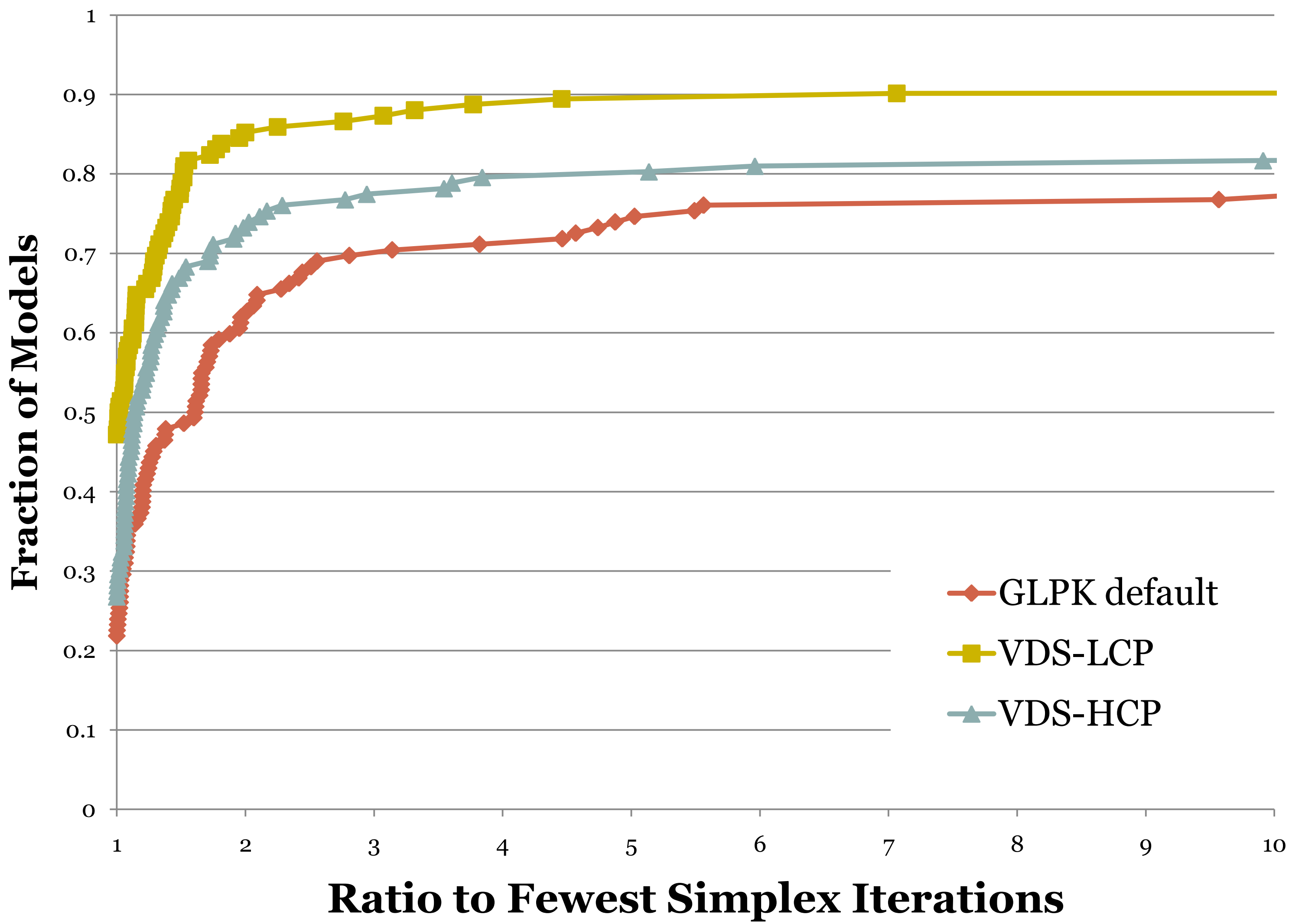
Experiments

- Compare methods in pairs:
  - Branching to *high* vs. *low* prob. of satisfying active constraints
- GLPK default included in all comparisons
- Branching variable selection: GLPK default
  - Except for variable-and-direction methods

LCP vs. HCP; LCPV vs. HCPV: All Models

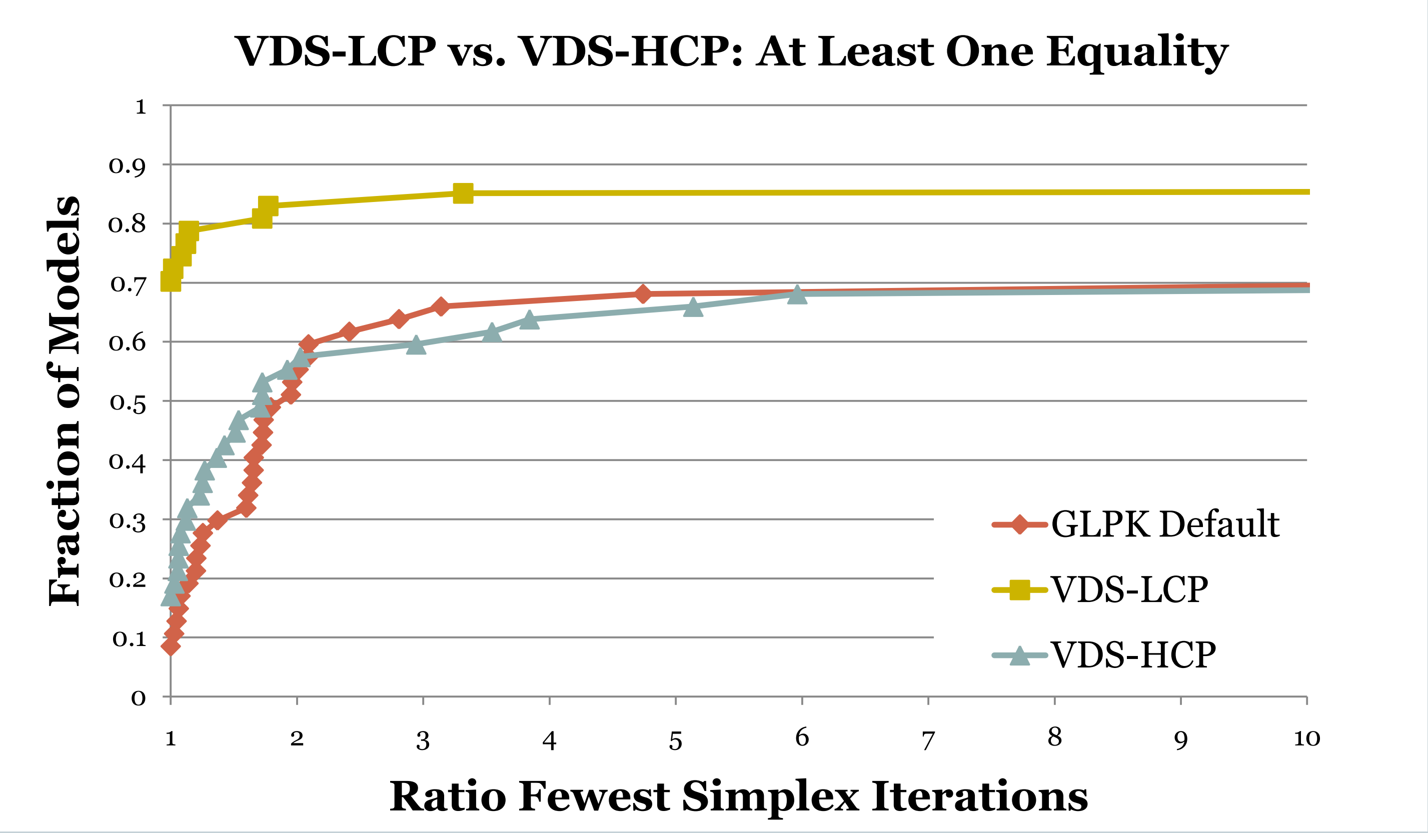


VDS-LCP vs. VDS-HCP: All Models

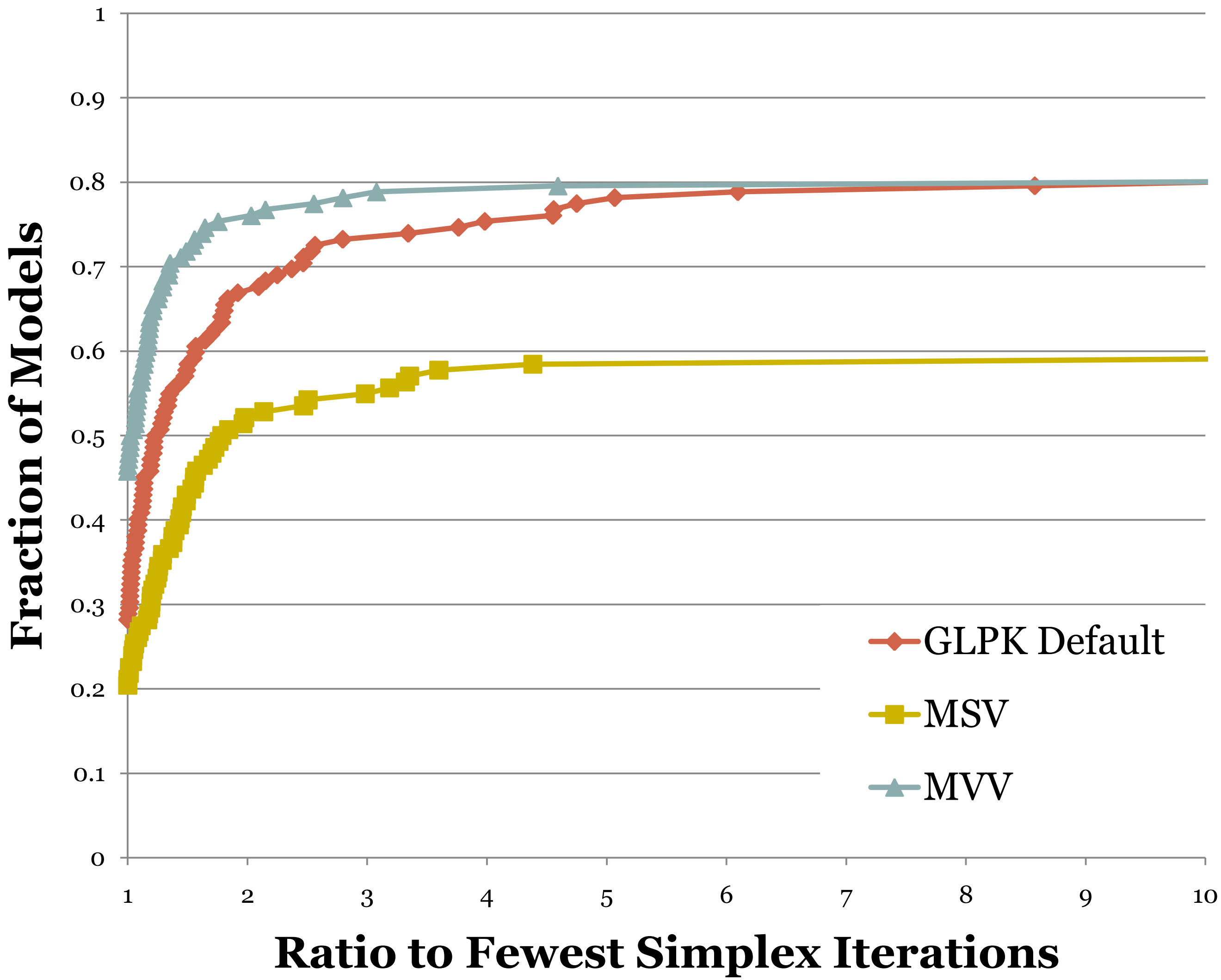


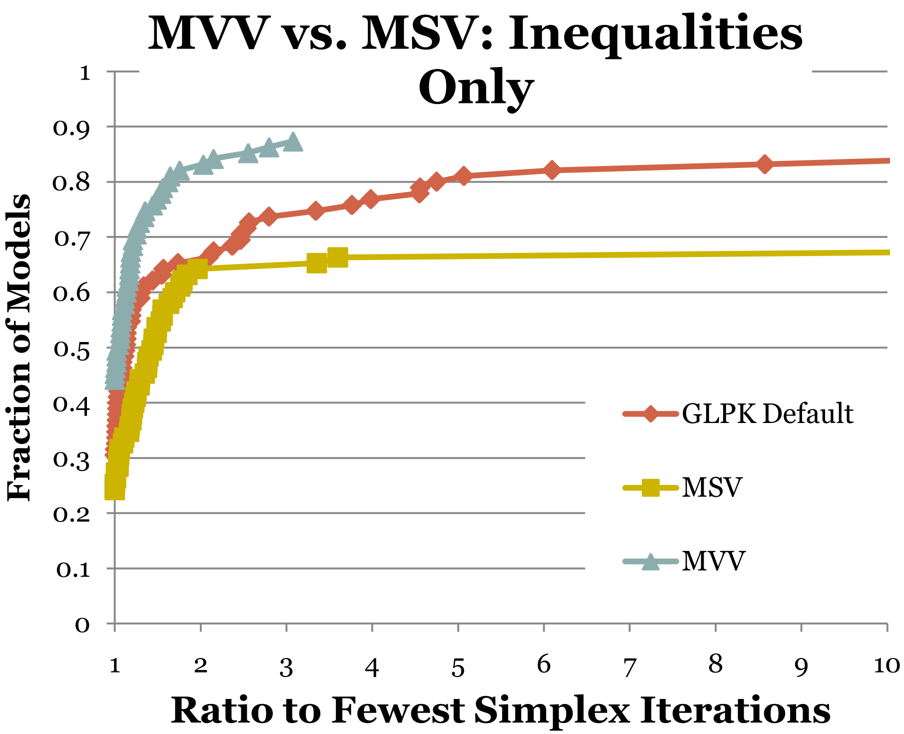
VDS Methods With Equality Constraints

- VDS-LCP even more dominant
- The centering strategy is effective

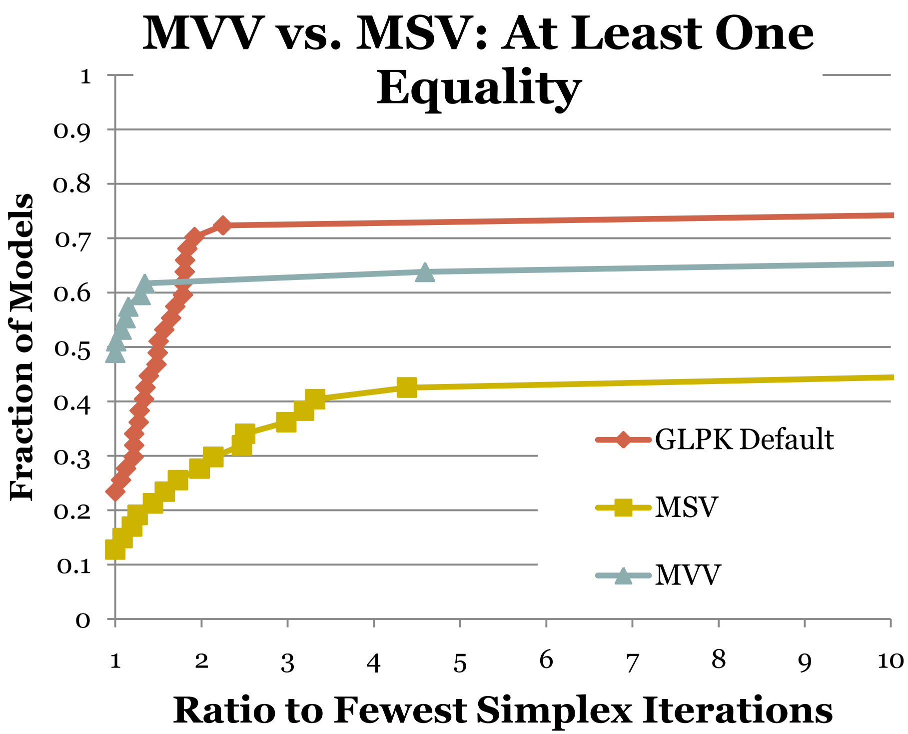


MVV vs. MSV: All Models



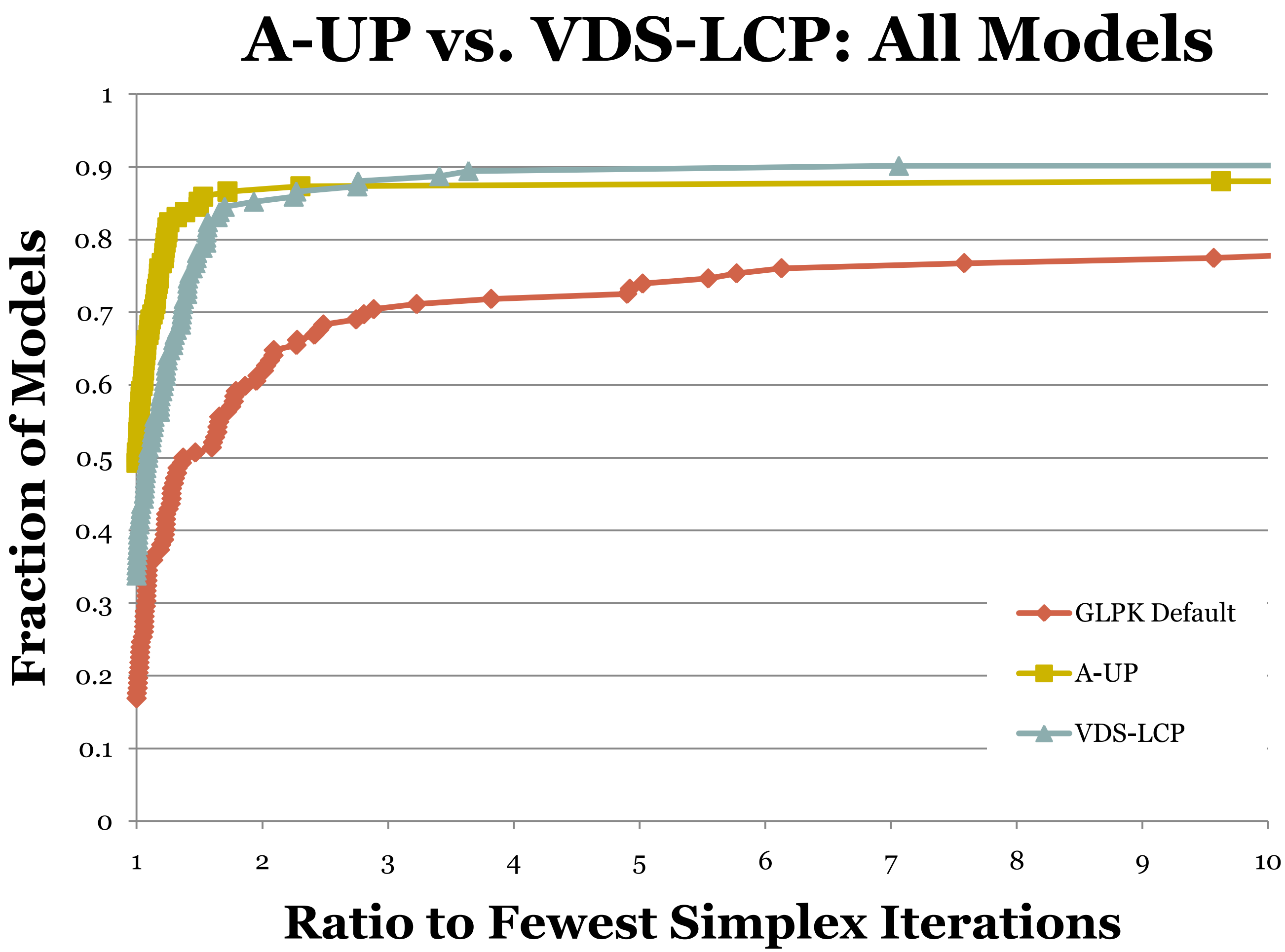


MVV vs. MSV: Inequalities Only
Fraction of Models
1
0.9
0.8
0.7
0.6
0.5
0.4
0.3
0.2
0.1
0
1 2 3 4 5 6 7 8 9 10
Ratio to Fewest Simplex Iterations
GLPK Default
MSV
MVV



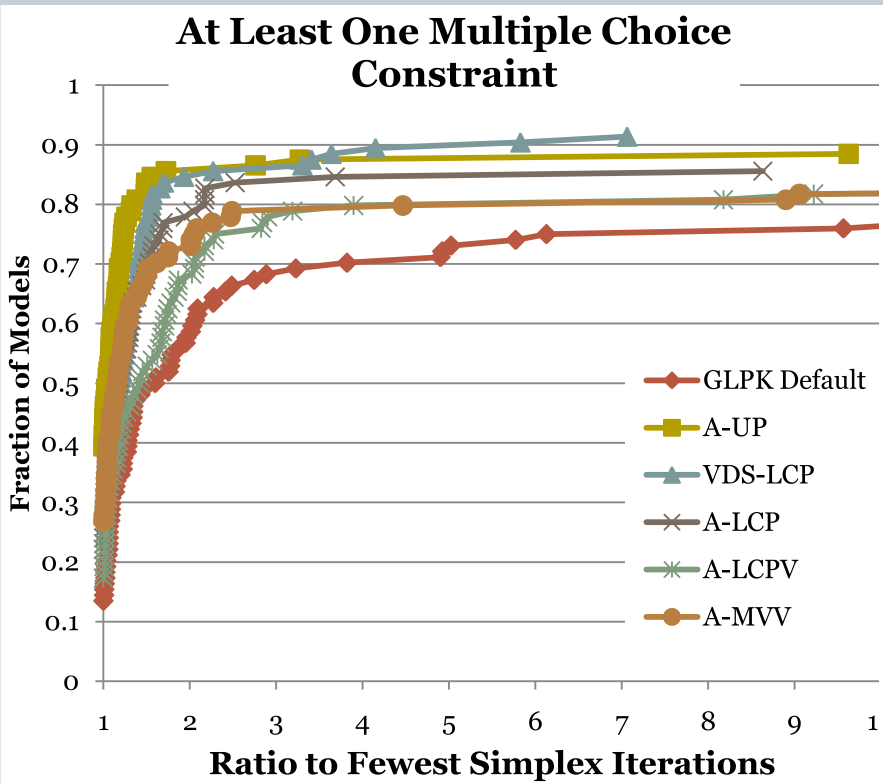
MVV vs. MSV: At Least One Equality
Fraction of Models
1
0.9
0.8
0.7
0.6
0.5
0.4
0.3
0.2
0.1
0
1 2 3 4 5 6 7 8 9 10
Ratio to Fewest Simplex Iterations
GLPK Default
MSV
MVV

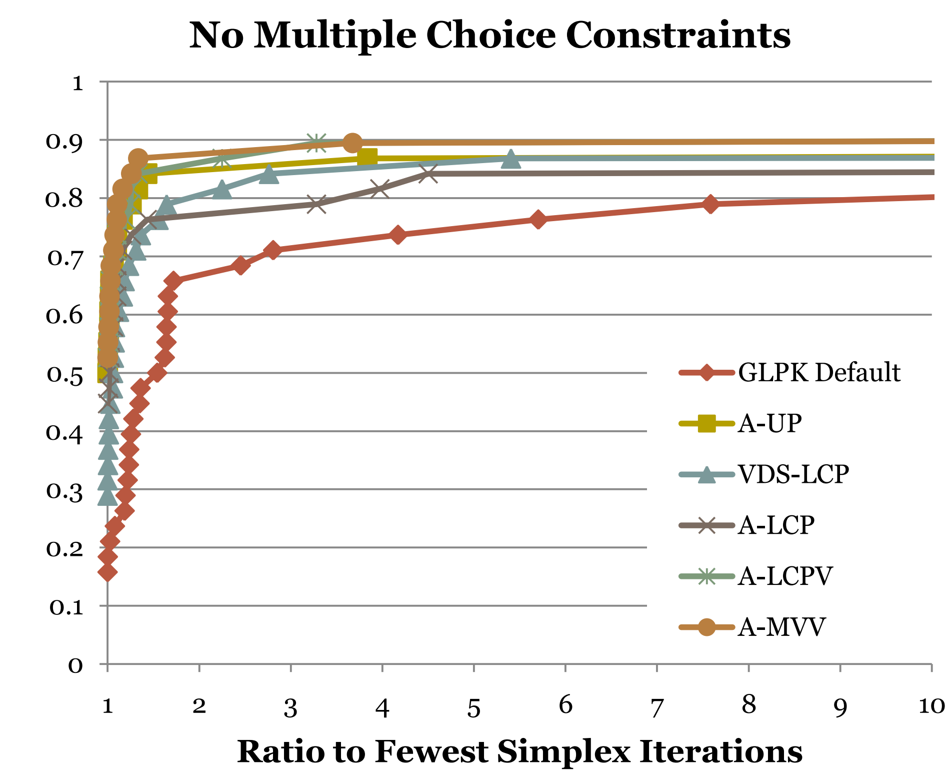
Best Methods: A-UP vs. VDS-LCP



Branching Up Revisited

- UP is good because many models have multiple choice constraints!
  - 104 of 142 (73%) models have at least one





Conclusions

- **Branching to force change** in the candidate variables is fastest to first feasible solution
  - **LCP** better than **HCP**
  - **LCPV** better than **HCPV**
  - **VDS-LCP** better than **VDS-HCP**
  - **MVV** better than **MSV**
  - *Surprise!* Branching in low probability direction is best
- Constraint types have an impact:
  - Equality constraints; multiple choice constraints

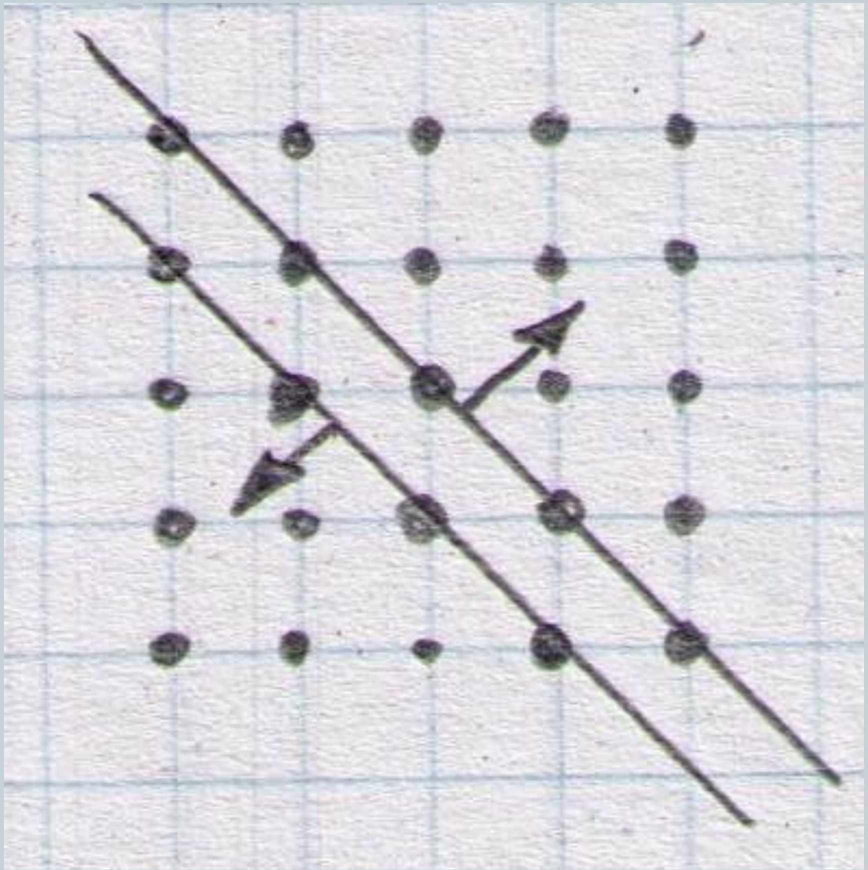
Observations

- MIP:
  - Linear constraints always satisfied, integrality more difficult
  - ***Branch to force integrality as much as possible***

- Constraint programming:
  - Integrality always satisfied, constraints more difficult
  - ***Branch to satisfy constraints as much as possible***

General Disjunctions

- Faster integer-feasibility in MIPs by using general disjunctions
- Observation:
  "45 degree" general disjunctions leave no integer solutions in their "interior"

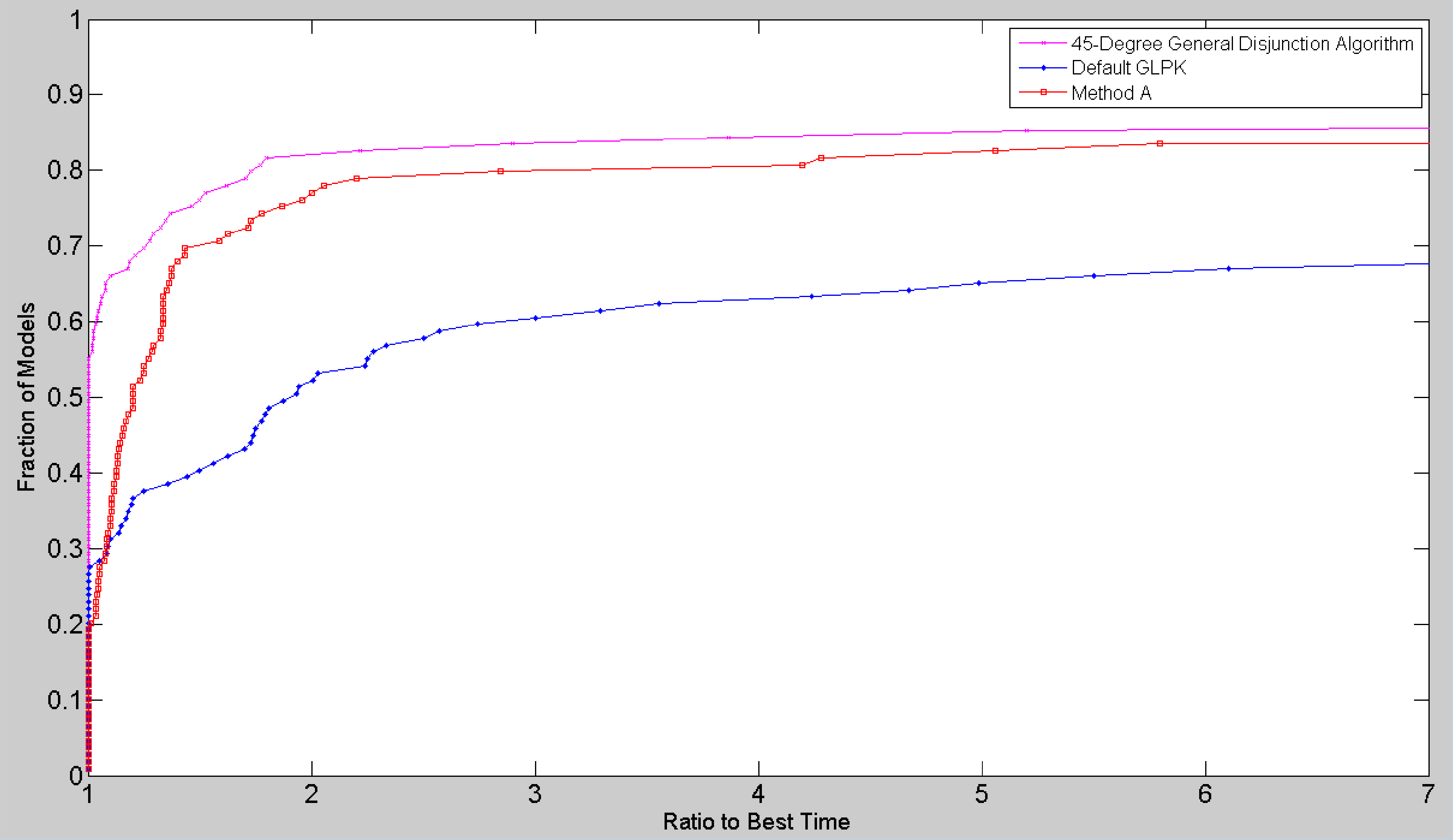


Method

- Find 45 degree general disjunction that is:
  - as ***parallel*** as possible or
  - as ***perpendicular*** as possible

  to active constraint having the most candidate varbs:
  - Active ***inequality*** chosen: as ***parallel*** as possible
  - Active ***equality*** chosen: as ***perpendicular*** as possible
- All coefficients must be +1, -1 or 0 for 45 degree
  - Many possibilities
  - Simple rules try to match/reverse signs in chosen constraint
- Branch in the direction that forces change
- 45 degree general disjunction only when "stuck"

Results



45-Degree General Disjunction Algorithm
Default GLPK
Method A
Fraction of Models
0
0.1
0.2
0.3
0.4
0.5
0.6
0.7
0.8
0.9
1
1
2
3
4
5
6
7
Ratio to Best Time

References

- J. Pryor and J.W. Chinneck (2011), “Faster Integer-Feasibility in Mixed-Integer Linear Programs by Branching to Force Change", *Computers and Operations Research*, vol. 38, no. 8, pp. 1143-1152.

- D.T. Wojtaszek and J.W. Chinneck (2010), “Faster MIP Solutions via New Node Selection Rules”, *Computers and Operations Research*, vol. 37, no. 9, pp. 1544-1556.

- J. Patel and J.W. Chinneck (2007), "Active-Constraint Variable Ordering for Faster Feasibility of Mixed Integer Linear Programs", *Mathematical Programming* Series A, vol. 110, pp. 445-474.

Cross-Fertilization

ACTIVE CONSTRAINTS BRANCHING VARIABLE SELECTION
BRANCHING TO FORCE CHANGE
NOGOOD BRANCHING
STRONG BRANCHING

Active Constraints Branching Variable Selection

Constraint Programming

- *Backdoor variable*:
  - Assigning a value to a backdoor variable simplifies the problem as much as possible
- Is the active constraints branching variable selection method choosing a "backdoor" variable?

Branching to Force Change

Constraint Programming:

- ***Fail-first:*** select varb having fewest remaining legal values
- ***Degree heuristic:*** select variable appearing in most constraints on other varbs whose values are not yet set

Satisfiability:

- ***MAXO:*** select literal that appears most often
- ***MOMS:*** select literal appearing most often in clauses of minimum size
- ***MAMS:*** combine MAXO and MOMS
- ***Jeroslaw-Wang:*** weights small clauses more heavily

Nogood Branching

Constraint Programming

- Intelligent backtracking
  - Backtrack on members of the *conflict set*
- Conflict-directed backtracking
  - The backtrack set is minimal [same as an Irreducible Infeasible Subset of an infeasibility]
- Constraint Learning / Nogood Learning
  - Add constraints based on the minimal infeasible set

Strong Branching

Satisfiability

- UP (unit propagation):
  - Make test assignment for each unassigned literal; count the number of unit propagations triggered
- SUP (selective unit propagation):
  - Reduce testing of literals by first running MAXO, MOMS, MAMS, Jeroslaw-Wang to identify at most 4 candidates