Outline

1. Introduction and Orientation (Lodi)

Part I: Achieving Integer-Feasibility Quickly

2. Classic Feasibility-Seeking Algorithms (Chinneck)
3. Active Constraint Variable Selection (Chinneck)
4. Branching to Force Change (Chinneck)
5. The Feasibility Pump (Lodi)

Part II: Reaching Optimality Quickly

6. New Node Selection Rules (Chinneck)
7. Local Branching and RINS (Lodi)

Part III: Analyzing Infeasible MIPs

8. Isolating Infeasible Subsystems (Chinneck)
9. Repairing MIP Infeasibility via Local Branching (Lodi)
10. Conclusions (Chinneck)

MIP Feasibility-Seeking: Classic Algorithms

John W. Chinneck

Systems and Computer Engineering, Carleton University, Ottawa, Canada

Find First Feasible Solution Quickly

- *Why?*
  - Integer-feasibility may be the only goal.
  - Shortens time to optimality:
    - First incumbent prunes subsequent tree. Early incumbent important.
    - If backtracking algorithm is good, then closest integer-feasible descendent usually has best objective function value.
  - Helps ensure solution in case of time-out.
  - Helpful in infeasibility analysis.

“Classic” feasibility-focused heuristics

For pure binary problems:

- Pivot-and-complement
- OCTANE

For general MIPs:

- Pivot-and-shift

BIP: Pivot-and-Complement

- Inequality-constrained Binary Integer Program (BIP)
- Feasibility-seeking first phase
- Main insight:
  - BIP has LP equivalent in which all binary varbs are nonbasic at upper or lower bound
  - One basic variable per constraint
  - Hence all slack variables must be basic
- BIP: max $\boldsymbol{cx}$ s.t. $\boldsymbol{Ax} \leq \boldsymbol{b}$, x_j binary
- LP: max $\boldsymbol{cx}$ s.t. $\boldsymbol{Ax}+\boldsymbol{y}=\boldsymbol{b}$, $\boldsymbol{0} \leq \boldsymbol{x} \leq \boldsymbol{1}$, $\boldsymbol{y} \geq \boldsymbol{0}$, y_i basic
- Balas and Martin 1980

Pivot-and-Complement

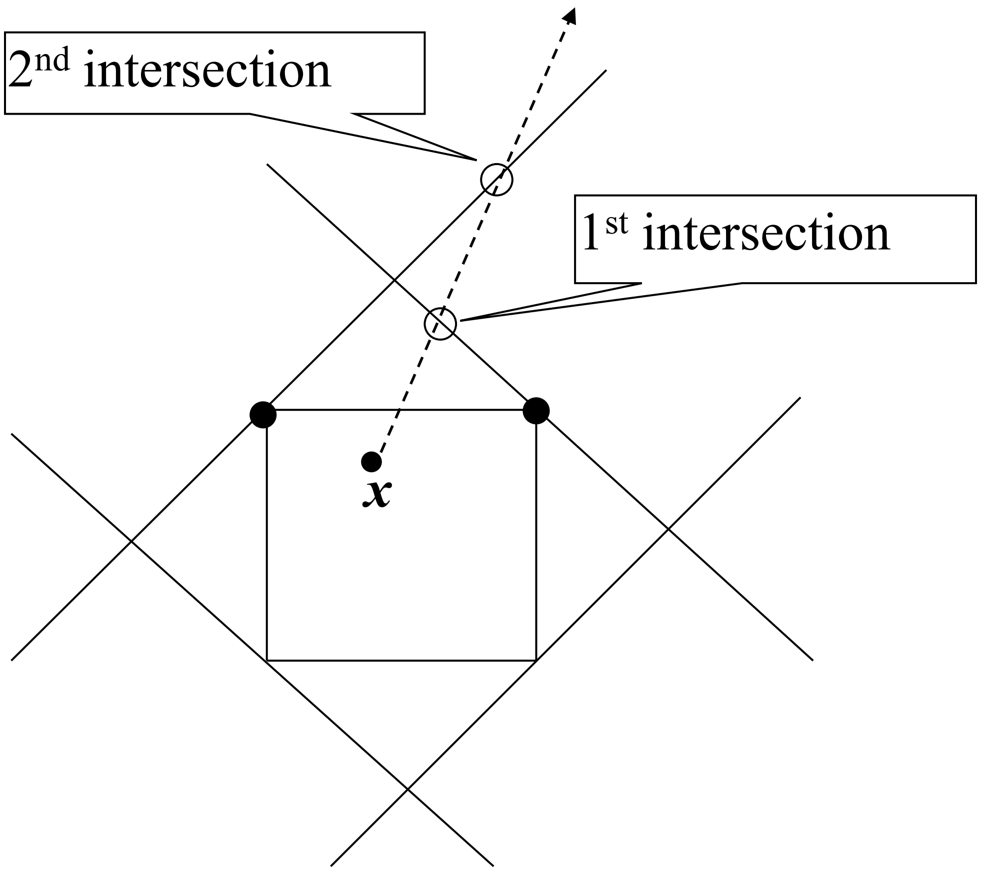
Operations to force slacks to be basic:

- **Type 1 pivot:** maintain LP feasibility, exchange nonbasic slack and basic binary varb
- **Type 2 pivot:** maintain LP feasibility, exchange slack for slack or binary for binary but reduce sum of integer infeasibility
- **Type 3 pivot:** sacrifice LP feasibility, exchange nonbasic slack for basic binary
- **Complement:** flip the values of 1 or 2 binary varbs to reduce an infeasibility measure
- **Rounding and truncating solutions.**

BIP: OCTANE

- *OCTA*hedral *N*eighbourhood *E*valuation
- Main insight:
  - N-dimension octagon around binary n-cube associates octagon facets with binary solutions
  - Given current soln (e.g. LP-relaxation) and improvement direction:
    - Improving rays cross extended facets of octagon
    - Crossed facet has associated binary solution
  - A kind of neighbourhood search
- Balas, Ceria, Dawande, Margot, Pataki 2001

OCTANE



- Find first k octagon facet intersections
- Check associated binary solutions

OCTANE Details

- Unit cube actually centred at origin, so offset by ½
- OCTANE not run at every node of branch-and-cut tree
  - Every node in first 5 levels of tree
  - Every 8^{th} node thereafter

MIP: Pivot-and-Shift

- Extension of pivot-and-complement
- Initial feasibility-seeking stage:
  - Rounding
  - Pivot-and-shift operations
  - Small neighbourhood searches

- Balas and Martin 1986; Balas, Schmieta and Wallace 2004

Types of Pivots

Operations:

- **Type 1 pivot:** maintain LP feasibility, exchange basic int varb and nonbasic continuous varb
- **Type 2 pivot:** maintain LP feasibility and improve obj fcn, exchange continuous varb with cont, or int varb with int
- **Type 3 pivot:** maintain LP feasibility while reducing int infeasibility, exchange cont varb with cont, or int varb with int

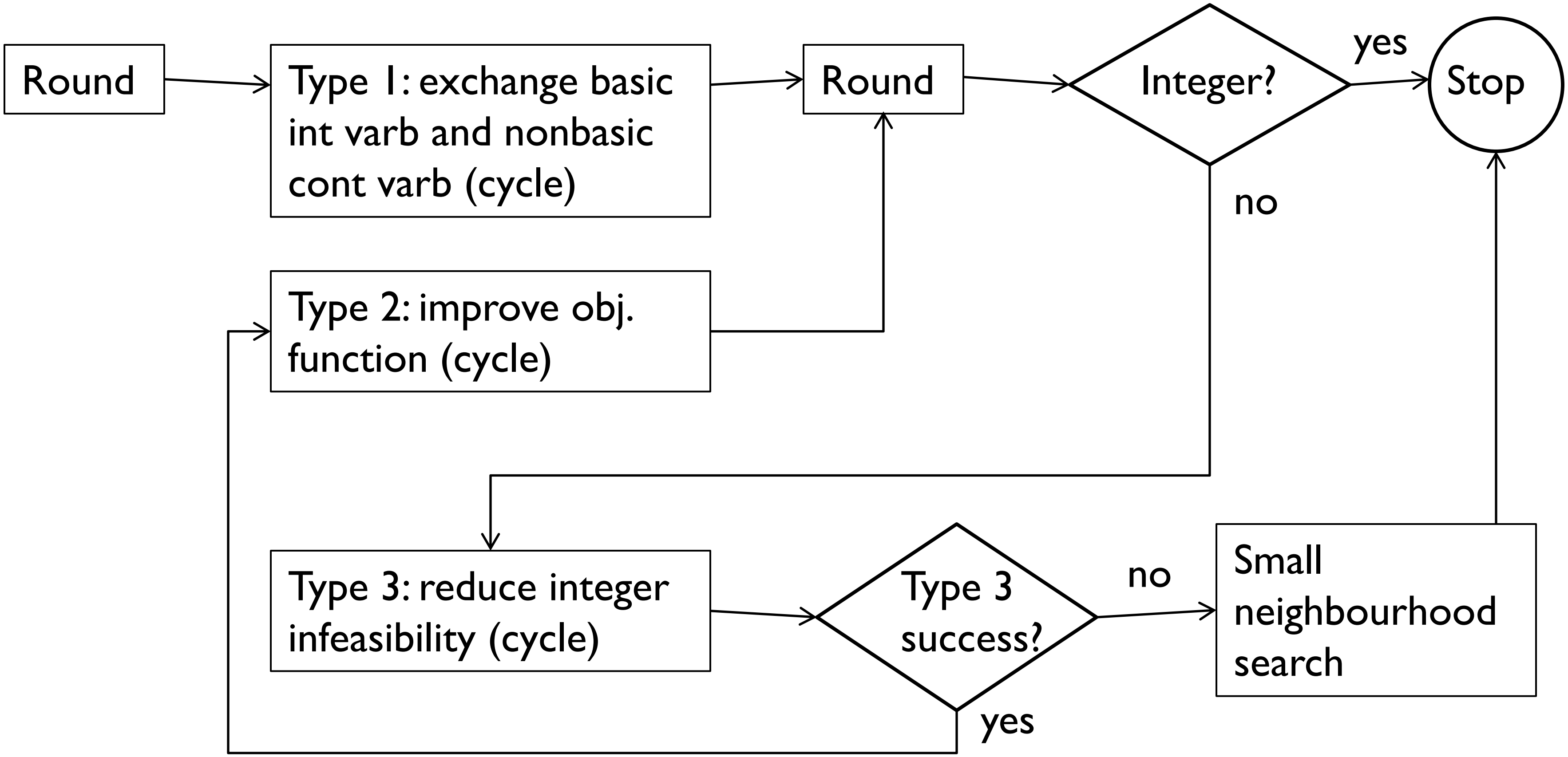
Feasibility maintained:

- Entering basic variable (col) chosen according to type of pivot
- Leaving basic variable (row) chosen by minimum ratio test

Other Operations

- **Rounding (*shifting*).**
- **Small neighbourhood search:**
  - MIP search in neighbourhood around a near-feasible soln (tot int infeas < limit, e.g. 0.1).

Pivot-and-Shift Flowchart



Pivot-and-Shift Details

- Time limit
- Abandon in favour of Xpress-MP solver if:
  - No integer-feasible soln within time limit
  - Integer soln obtained by rounding has obj fcn value 40%+ worse than bounding fcn value of unrounded soln

- Empirical tests:
  - Much faster to first feasibility than standard Xpress-MP.

Conclusions

- Significant progress 1980-mid 2000s
- Recent renewed interest:
  - Updated pivot-and-shift (2004)
  - The feasibility pump (2005)
  - Active constraints branching (2006)
  - Etc…….

Outline

1. Introduction and Orientation (Lodi)

<u>Part I: Achieving Integer-Feasibility Quickly</u>

2. Classic Feasibility-Seeking Algorithms (Chinneck)
3. Active Constraint Variable Selection (Chinneck)
4. Branching to Force Change (Chinneck)
5. The Feasibility Pump (Lodi)

<u>Part II: Reaching Optimality Quickly</u>

6. New Node Selection Rules (Chinneck)
7. Local Branching and RINS (Lodi)

<u>Part III: Analyzing Infeasible MIPs</u>

8. Isolating Infeasible Subsystems (Chinneck)
9. Repairing MIP Infeasibility via Local Branching (Lodi)
10. Conclusions (Chinneck)

Active-Constraint Variable Selection

John W. Chinneck, Jagat Patel
Systems and Computer Engineering, Carleton University, Ottawa, Canada

Branch & Bound (simplified)

After start-up…

1. If no unexplored nodes left then exit: optimal or infeasible.
2. Choose unexplored node for expansion and solve its LP relaxation.
   - Infeasible: discard the node, go to Step 1.
   - Feasible and integer-feasible: check for new incumbent, go to Step 1.
3. ***Choose branching variable*** in current node and create two new child nodes.

Main B&B Design Decisions

- How choose next node from list?
  - Depth-first?
    - Usual choice for efficiency of basis re-use.
  - Global best value of bounding function?
    - Original objective function?
    - minimum sum of integrality violations?
  - Breadth-first?
  - Etc.
- How choose branching variable?
- How choose branching direction?

Is Branching Variable Selection Important?

	B&B nodes to First Feasible Soln	
model	**Cplex 9.0**	**Active-Constraints Method**
aflow30a	23,481	22 (A, H_M, H_O, O, P)
aflow40b	100,000+ (limit)	33 (H_O, O, P)
fast0507	14,753	26 (A)
glass4	7,940	62 (A, H_M, H_O, O, P)
nsrand-ipx	3,301	18 (H_M)
timtab2	14,059	100,000+ (limit)

Traditional Branching Variable Selection

- Based on estimated impact on ***objective function***
- *Goal:* maximize degradation in the objective function value at optimal solution of child node LP relaxations.
- e.g. pseudo-costs

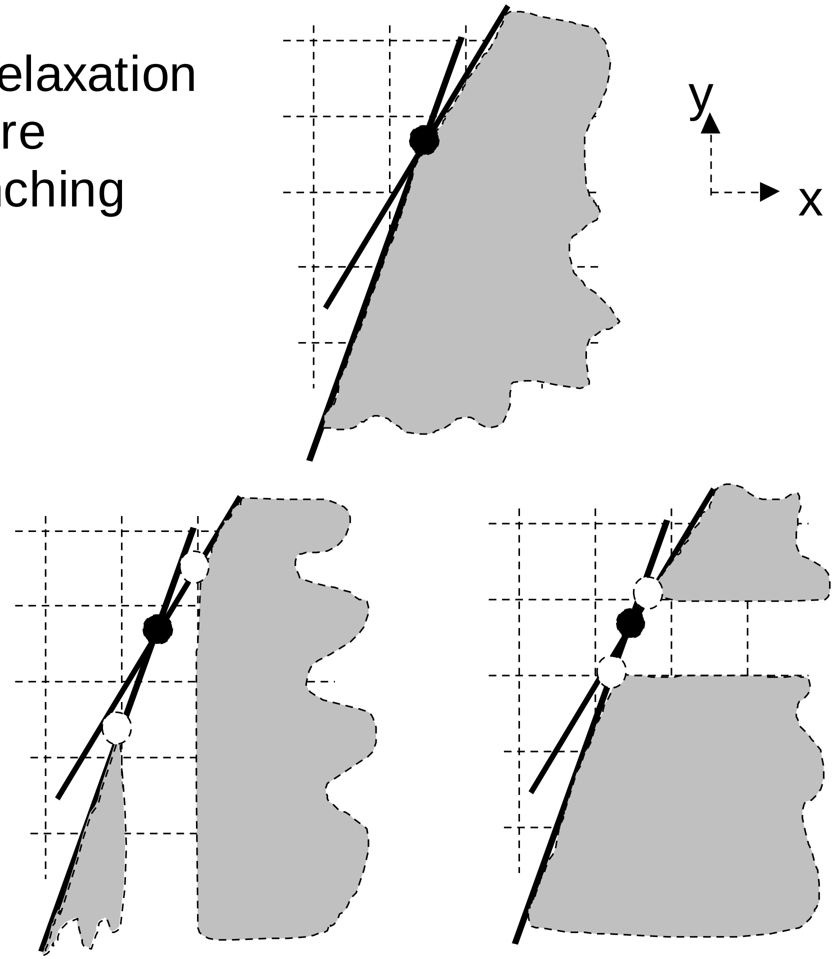
Active Constraints Approach

Goal: make child node LP-relaxation optima *far* from parent node LP-relaxation optimum.

- *Active constraints* fix the position of the LP optimum solution in parent, so…
- Choose branching variable that has *most impact on the active constraints* in parent LP relaxation optimum solution.
  - Select variable that is *most tightly constrained* first
- ***Constraint-oriented*** approach.
- *Note:* "active constraints" include tight degenerate constraints

Impact of the Branching Variable

LP relaxation before branching



Branch on x

Branch on y

Estimating Candidate Variable Impact on Active Constraints

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

Dynamic variable ordering: changes at each node.

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 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.

Methods A, B, L

Numerous variants. Subset of best:

- **A:** W_{ik}=1.
  - *Is candidate variable present in the active constraint?*
- **B:** W_{ik} = 1/ [Σ(|coeff of *all* variables|].
  - *Like A, but relative impact of a constraint normalized by absolute sum of coefficients*
- **L:** W_{ik} = 1/(no. *integer variables*)
  - *Like A, but relative impact of a constraint normalized by number of integer variables it contains*
  - *Related to MOMS rule?*

Methods M, O, P

- **M:** W_{ik} = 1/(no. *candidate variables*)
  - *Like A, but relative impact of a constraint normalized by number of candidate variables it contains*
  - *Not used directly: see H methods*
- **O:** W_{ik} = $|\text{coeff}_i|$/(no. of *integer variables*)
  - *Like L, but size of coefficient affects weight of varb in constraint*
- **P:** W_{ik} = $|\text{coeff}_i|$/(no. of *candidate variables*)
  - *Like M, but size of coefficient affects weight of varb in constraint*

Methods H_M, H_O

- **H** methods: for a given base method, choose the variable that has largest weight in any *single* active constraint
  - Do not sum across active constraints
- **H_M**: based on method M
- **H_O**: based on method O

Experimental Setup: Solvers

- 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
    - No optimization of data structures for active constraint methods: inefficient searching
  - Node selection:
    - *Experiment 1*: Straight depth-first, branch up
    - *Experiment 2*: Cplex default

Experimental Setup: Premature Termination

- Time limit: 28,800 seconds (8 hours)
  - Data structures not optimized for active constraint methods, hence penalizes them
- Node Limits:
  - 100,000 nodes
  - Limit on active-constraint methods: (Cplex nodes + 1000)
- Tree memory, node file size:
  - Never exceeded.

Experimental Setup: Metrics

- Number of B&B nodes
- Number of simplex iterations
  - No. of B&B nodes does not penalize for jumping around tree, reducing ability to use advanced starts
  - Tracks well with solution time (except as noted later)
- Feasibility Success Ratio
  - Fraction of cases where better than Cplex
- Quality Success Ratio
  - fraction of cases in which the first feasible solution has optimality gap equal to or smaller than optimality gap for first feasible solution returned by Cplex
- Performance Profiles

Experimental Setup: Test Models

- MIPLIB 2003 set
  - 60 models
  - Range of difficulties
  - Rows: 6–159488
  - Cols: 62–204880
    - Integer variables: 1–3,303
    - Binary variables: 18–204,880
    - Continuous variables: 1–13,321
  - Nonzeroes: 312–1,024,059

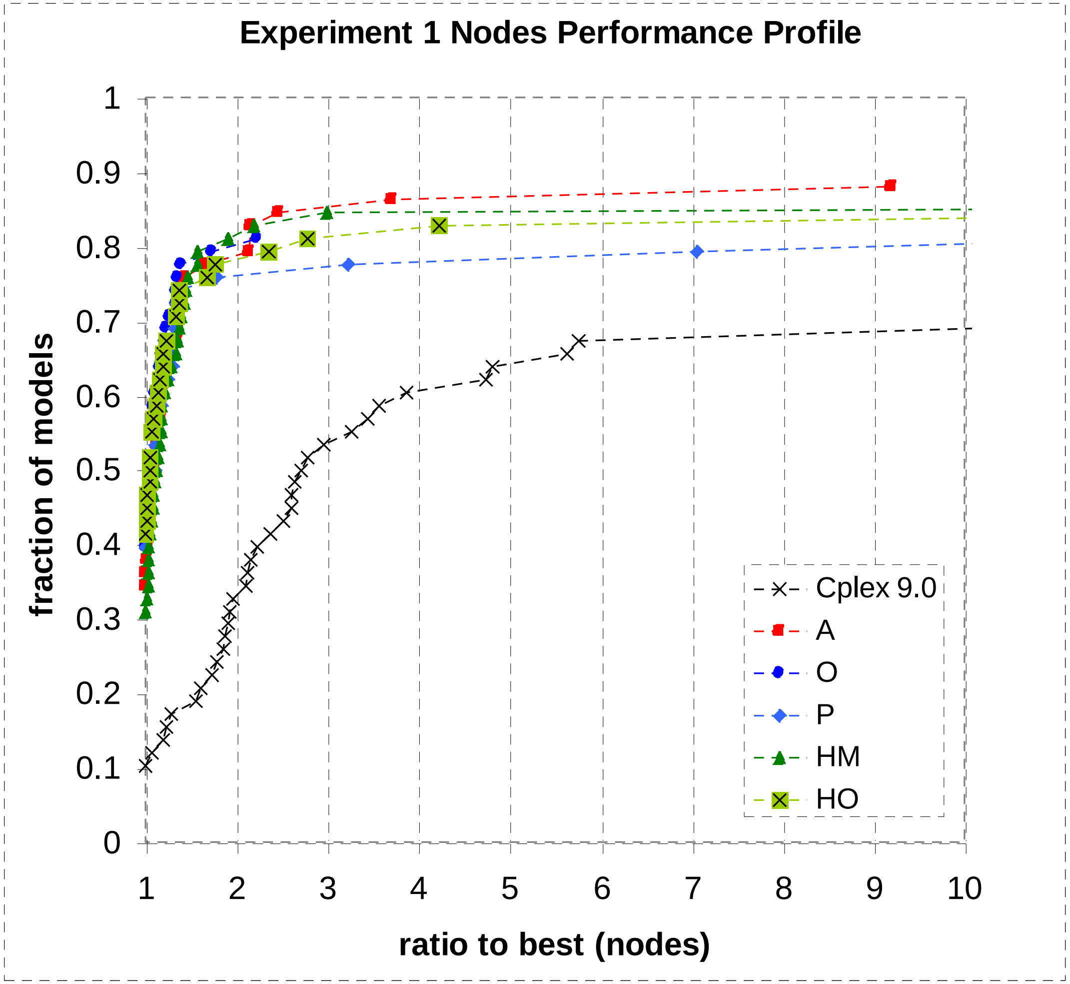
Experiment 1: Notes

- All internal heuristics *off*
- 58 models used
  - 2 models prematurely terminated by all methods, including Cplex

Experiment 1: Number of Nodes

	All 58 Models				40 Comparable Models		
method	times within 10% of best	fewer nodes than Cplex	FSR	times term. (fewer nodes at time-out)	Avg. nodes:	(avg. nodes)/ (Cplex avg. nodes)	avg. ratio to best
Cplex 9.0	7			**4**	1967.5		58.22
A	30	**47**	**0.810**	7 (2)	149.5	0.076	1.19
H_M	28	45	0.776	8(2)	130.5	0.066	1.18
H_O	35	45	0.776	9 (3)	123.3	0.063	1.47
O	**36**	43	0.741	11 (3)	**116.1**	**0.059**	**1.11**
P	32	44	0.759	10 (2)	156.2	0.079	1.37

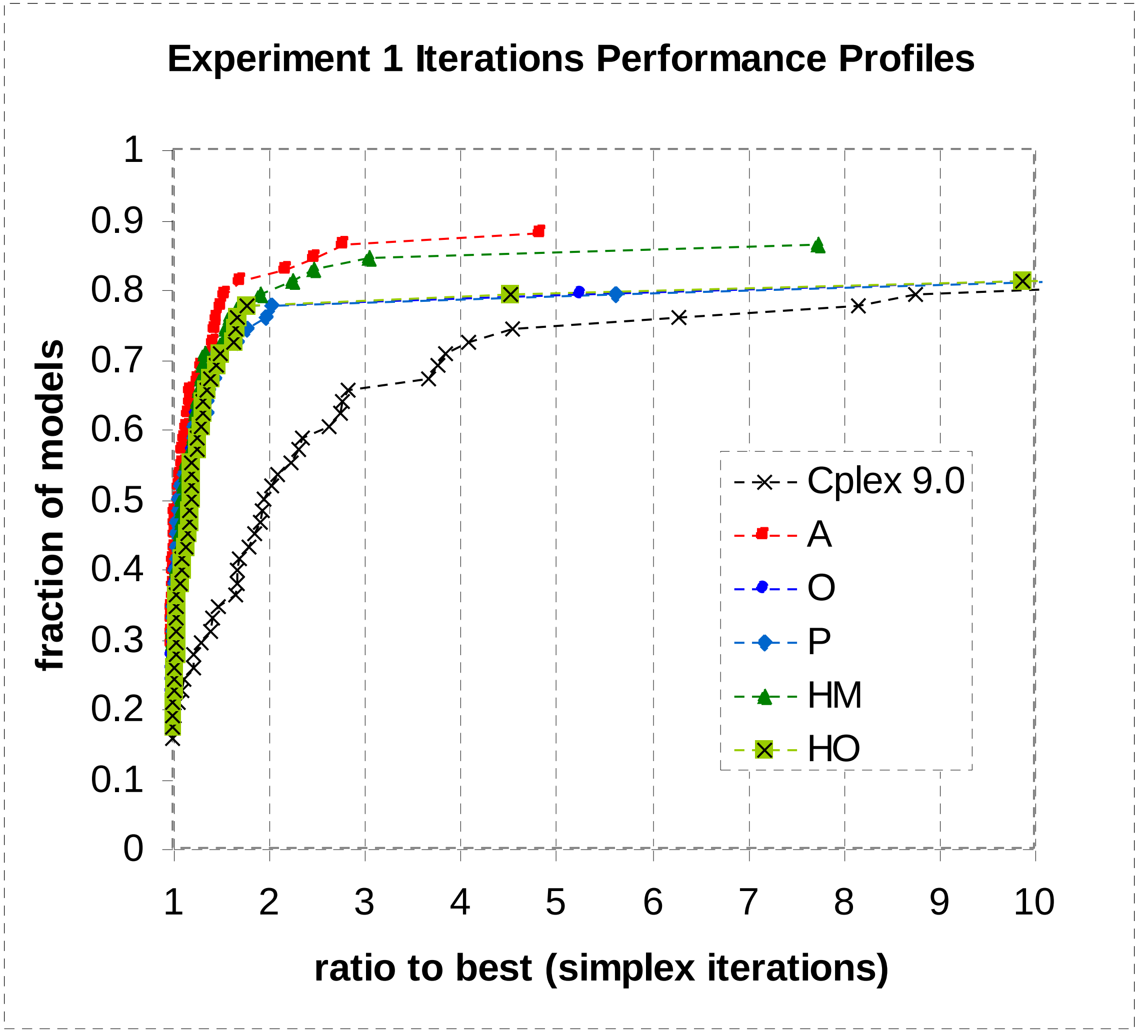
Exp 1: Nodes Peformance Profiles



Experiment 1: Simplex Iterations

	All 58 Models				40 Comparable Models		
method	times within 10% of best	fewer itns than Cplex	FSR	times term. (fewer itns at time-out)	Avg. itns:	(avg. itns)/ (Cplex avg. itns) [w/o *disctom*]	avg. ratio to best
Cplex 9.0	12			**4**	55052		14.93
A	**30**	**43**	**0.741**	7 (3)	36484	0.663 [**0.214**]	**1.17**
H_M	28	40	0.690	8(3)	**35173**	**0.639** [0.245]	1.18
H_O	23	40	0.690	9 (3)	117320	2.131 [0.237]	1.48
O	25	37	0.638	11 (4)	117401	2.133 [0.239]	1.38
P	**30**	41	0.707	10 (3)	216100	3.925 [0.232]	1.67

Exp 1: Simplex Iterations Perf. Profiles



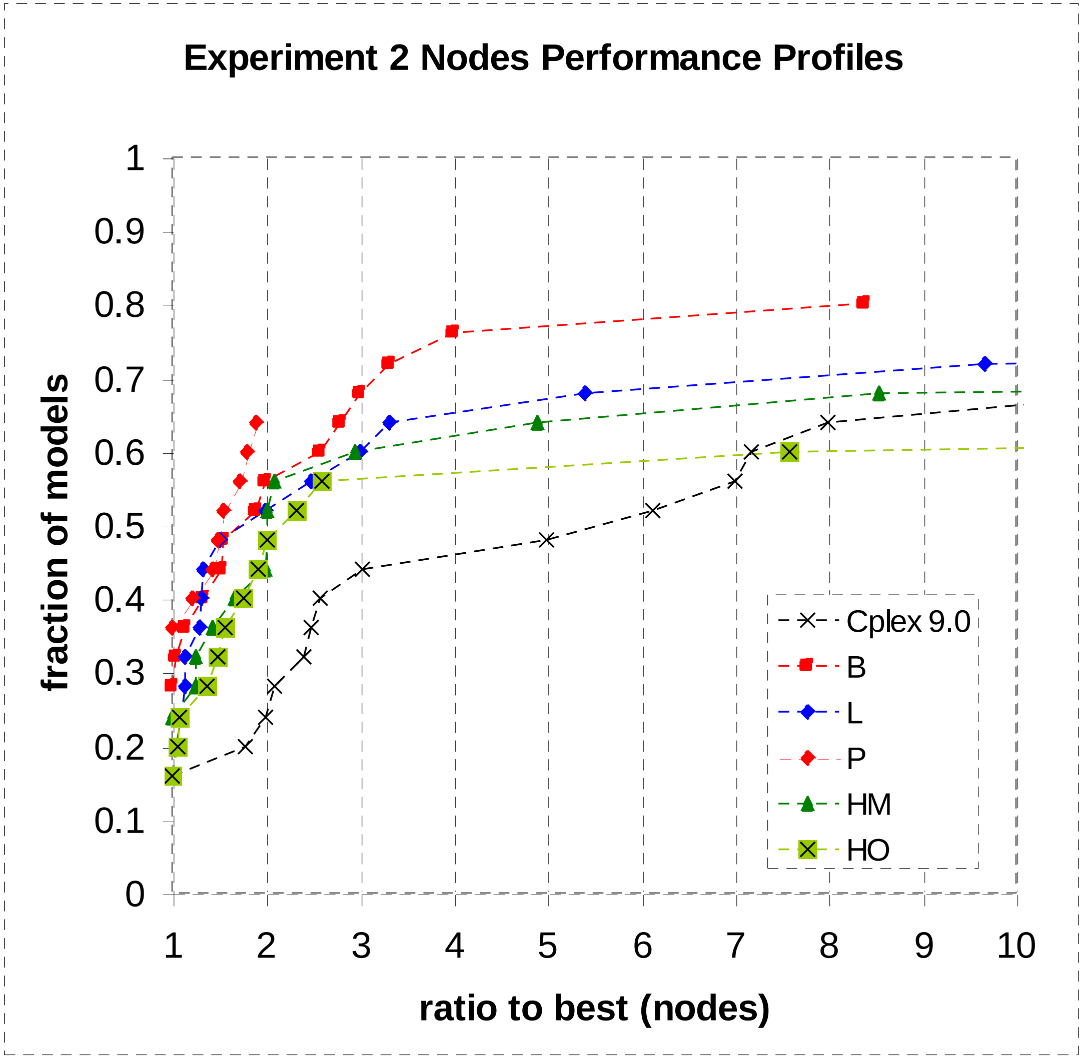
Experiment 2: Notes

- All internal heuristics *on*.
- 25 models used:
  - 3 models prematurely terminated by all methods
  - 32 models solved at root node
- Heuristics impact is mixed:
  - Many models solved at root node
  - Others: half slower with heuristics on, half faster.
    - 1 model solvable with heuristics off, but not solvable with heuristics on

Experiment 2: Number of Nodes

	All 25 Models				12 Comparable Models		
method	times within 10% of best	fewer nodes than Cplex	FSR	times term. (fewer nodes at time-out)	Avg. nodes:	(avg. nodes)/ (Cplex avg. nodes)	avg. ratio to best
Cplex 9.0	4			**1**	1214.6		23.86
B	**9**	**17**	**0.680**	5 (1)	235.0	0.193	2.02
L	7	**17**	**0.680**	6 (1)	**233.0**	**0.192**	2.01
H_M	6	16	0.640	7 (2)	262.9	0.216	2.13
H_O	6	13	0.520	8 (2)	260.9	0.215	1.96
P	**9**	15	0.600	9 (1)	293.8	0.242	**1.27**

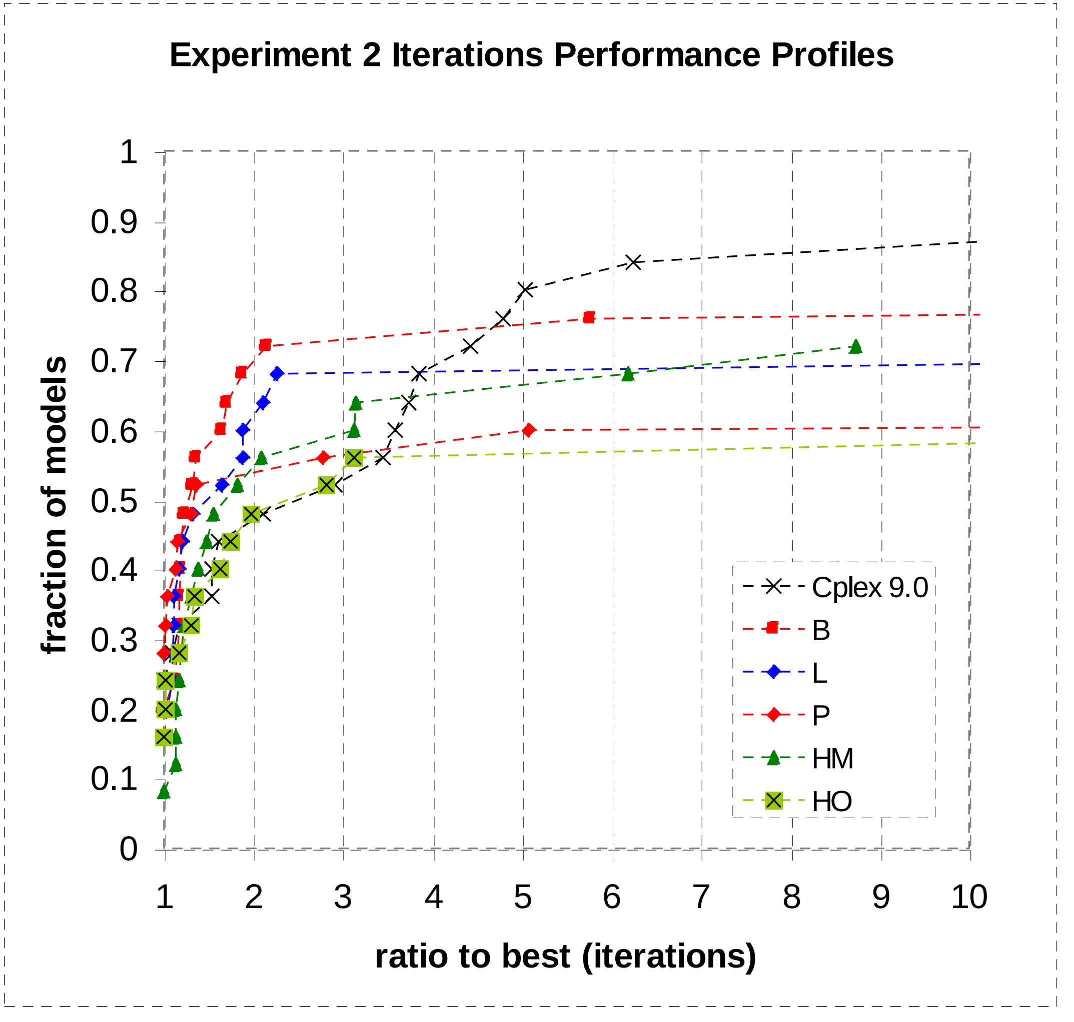
Exp 2: Nodes Performance Profiles



Experiment 2: Simplex Iterations

	All 25 Models				12 Comparable Models		
method	times within 10% of best	fewer itns than Cplex	FSR	times term. (fewer itns at time-out)	Avg. itns	(avg. itns)/ (Cplex avg. itns) [w/o *disctom*]	avg. ratio to best
Cplex 9.0	7			**1**	**32578**		6.89
B	5	14	0.56	5 (1)	400552	12.295 [0.452]	4.37
L	7	14	0.56	6 (2)	400233	12.285 [0.437]	4.38
H_M	2	14	0.56	7 (3)	108898	**3.343** [0.760]	**2.15**
H_O	6	**15**	**0.60**	8 (3)	418697	12.852 [0.785]	4.66
P	**9**	14	0.56	9 (2)	609275	18.702 [**0.367**]	5.90

Exp 2: Simplex Iterations Perf. Profiles



Quality Success Ratios

Experiment 1 over 40 comparable models	
method	QSR
A	0.53
H_M	0.55
H_O	0.58
O	0.70
P	**0.78**

Experiment 2 over 12 comparable models	
method	QSR
B	**0.75**
H_M	0.50
H_O	0.50
L	0.58
P	0.33

Experiment 1 Conclusions

- Active constraints branching variable selection is *much* better than commercial state of the art in achieving feasibility quickly:
  - Much faster in almost all cases.
  - Optimality gap at first feasible solution is usually better.
- Several methods very good
  - Simple method A the best.

Experiment 2 Conclusions

- Active constraints branching variable selection is better than commercial state of the art in achieving feasibility quickly:
  - Faster more often than not.
  - Optimality gap at first feasible solution is usually better for most methods.
- Cplex heuristics have uneven results
  - How do heuristics, models, and active constraints methods interact?
  - Active constraints methods can be used internally to heuristics.

Integration with Other Methods

- Octane and Pivot-and-Shift:
  - Comparing reported results: active constraint methods better
  - Active constraint methods integrate easily with both methods: use when selecting among variables to branch on
- Feasibility Pump
  - Use after feasibility pump finished

Reference

Jagat Patel, 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.

Ongoing Research: New Methods

- Choose candidate varb whose boundary has most oblique angle to an active constraint
- Tie-breaking:
  - Many methods give numerous ties, e.g. A
  - Pair with another method to break the ties
  - *Choose randomly?*
- Branching direction
  - How predict whether to branch up or down?
  - E.g.: branch to "inside" of an inequality

Ongoing Research: New Approach

Now: same method from start to end

- Should different methods be used depending on conditions at current node?
  - Special case:
    - Presence of active "hard" constraints
      (all binary variables, all coefficients are 1s)
    - Choose *only* from among candidate varbs in hard constraints
  - Other special cases?
  - Classifier to determine method to use at node, based on conditions at the node
- Promising so far: first leaf found very often feasible

Ongoing Research: Properties of Solution Trees

- 1-2 candidate variables very common.
  - Theory: more nodes are closer to leaves, where there are few candidate variables
- When most oblique angle is high (70°+), there are few candidate variables.
  - Theory: happens far down in the B&B tree, so most facets squared off by added bounds.

Ongoing Research: Best Choice at Node

Basic data:

- Full expansion on *all* candidate varbs, both up and down directions, at every node in smaller MIPLIB 2003 models
  - Calculate total Integer Infeasibility (II) for *all* choices
  - Use II reduction between parent and child to identify "best" choice at a node
  - *Comment: "ultra-strong" branching an effective method!*
- Data used to train classifier:
  - Which varb selection method to use at this node?

Future Research

- Extension to finding optimum solution
  - Use active constraint method to first feasibility, objective-based method thereafter?
  - Incorporate aspiration level as another constraint?

Outline

1. Introduction and Orientation (Lodi)

<u>Part I: Achieving Integer-Feasibility Quickly</u>

2. Classic Feasibility-Seeking Algorithms (Chinneck)
3. Active Constraint Variable Selection (Chinneck)
4. Branching to Force Change (Chinneck)
5. The Feasibility Pump (Lodi)

<u>Part II: Reaching Optimality Quickly</u>

6. New Node Selection Rules (Chinneck)
7. Local Branching and RINS (Lodi)

<u>Part III: Analyzing Infeasible MIPs</u>

8. Isolating Infeasible Subsystems (Chinneck)
9. Repairing MIP Infeasibility via Local Branching (Lodi)
10. Conclusions (Chinneck)

John W. Chinneck
Jennifer Pryor

Systems and Computer Engineering
Carleton University, Ottawa, Canada

Faster Integer Feasibility in MIPs by Branching to Force Change

A 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?*

Outline

1. B&B algorithms for MIPs
2. A new principle
3. Experimental setup
4. Evaluating simple branching direction heuristics
5. New probability-based branching methods
6. New violation-based methods
7. Experiments: branching to force change
8. A-UP vs. VDS-LCP
9. Branching up revisited
10. Contributions

1. B&B Algorithms for MIPs

Main ingredients:

- **Node selection** heuristic
- **Branching variable** selection heuristic
  - Choose from among *candidate variables*
- **Branching direction** selection heuristic
  - $k \le x \le k+1$, where k and $k+1$ are closest integers
  - **Branch down:** add $x \le k$ and solve new LP relaxn
  - **Branch up:** add $x \ge k+1$ and solve new LP relaxn

Node selection

- Many possible heuristics
- Depth-first is typical
  - LP advanced start based on parent LP solution
- Back-tracking
  - When current dive ends at leaf node (feasible or infeasible)
  - Many different heuristics

Branching

Assume node has been selected:

- If there are *k* candidate branching variables, and can branch up or down, then there are 2*k* branching possibilities.

Main categories of methods:

A. Choose branching variable, then choose branching direction
   - Most common method
   - Branching variable selection well researched
   - Branching direction selection little researched

B. Choose branching variable and direction simultaneously
   - Very few methods

What is the Best *Branching* Heuristic for Feasibility?

Metric:
shortest time to first integer-feasible solution

- Sometimes feasibility is the ***only*** goal
- Early incumbent shortens time to optimality (better pruning)
- If node selection method is effective, reaching an integer-feasible descendent quickly helps shorten time to optimality

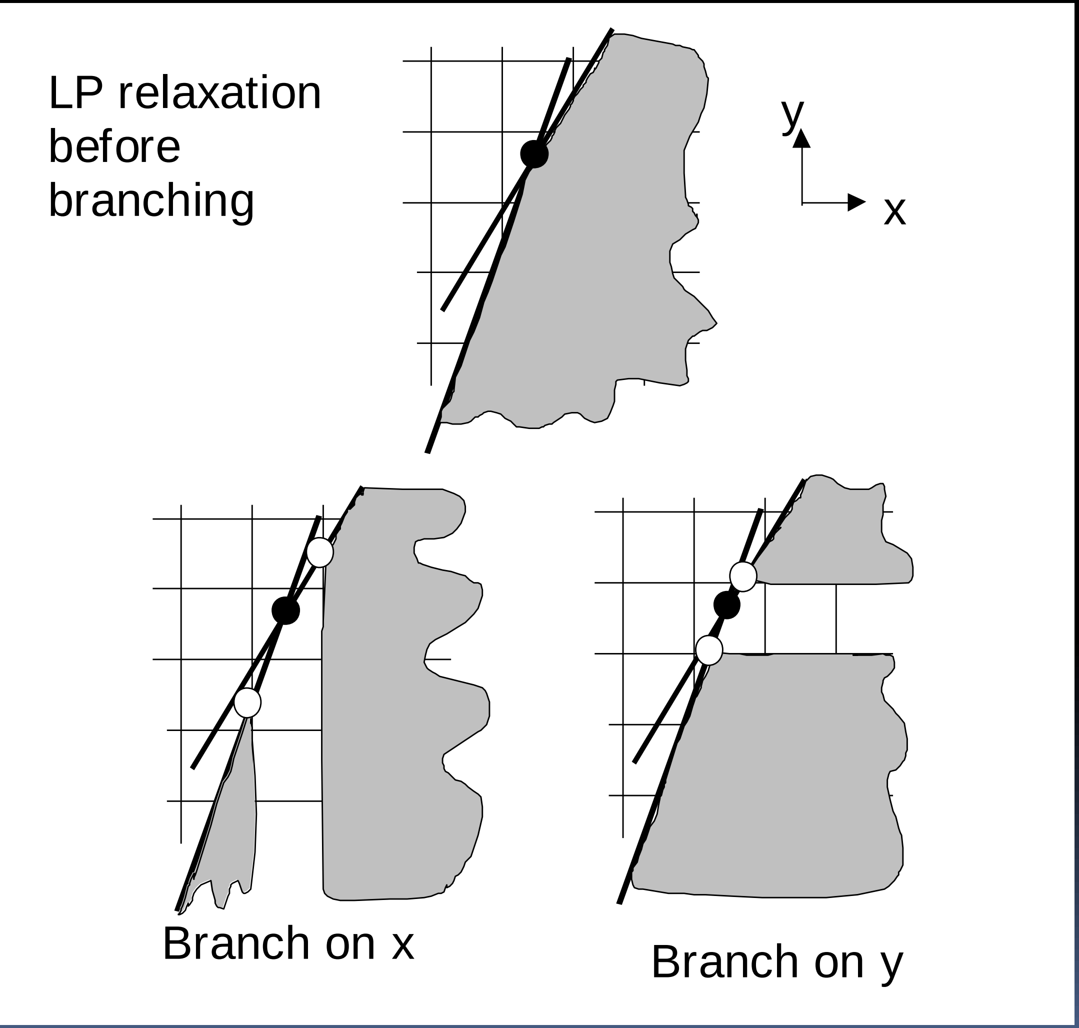
Branching Variable Selection

- Active Constraints Variable Selection (Patel and Chinneck 2007):
  - 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

Active Constraints Results

	B&B nodes to First Feasible Soln	
model	**Cplex 9.0**	**Active-Constraints Method**
aflow30a	23,481	22 (A, H_M, H_O, O, P)
aflow40b	100,000+ (limit)	33 (H_O, O, P)
fast0507	14,753	26 (A)
glass4	7,940	62 (A, H_M, H_O, O, P)
nsrand-ipx	3,301	18 (H_M)
timtab2	14,059	100,000+ (limit)

Impact of the Branching Variable



Branching Direction Selection

- Usually available in a solver:
  - UP always
  - DOWN always
  - CLOSEST INTEGER
- Sometimes available in a solver:
  - FARTHEST INTEGER
  - Specialized heuristics ("let solver choose")…
- No method dominates *in the literature*

Branching Variable *and* Direction

- Driebeek and Tomlin
  - Estimate *objective function* degradation for variable/direction combination using a dual pivot
  - *Largest* degradation chooses variable
  - *Smaller* of two degradations chooses direction
  - Default branching method in GLPK

"Multiple Choice" Constraints

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

- Branch down: 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:

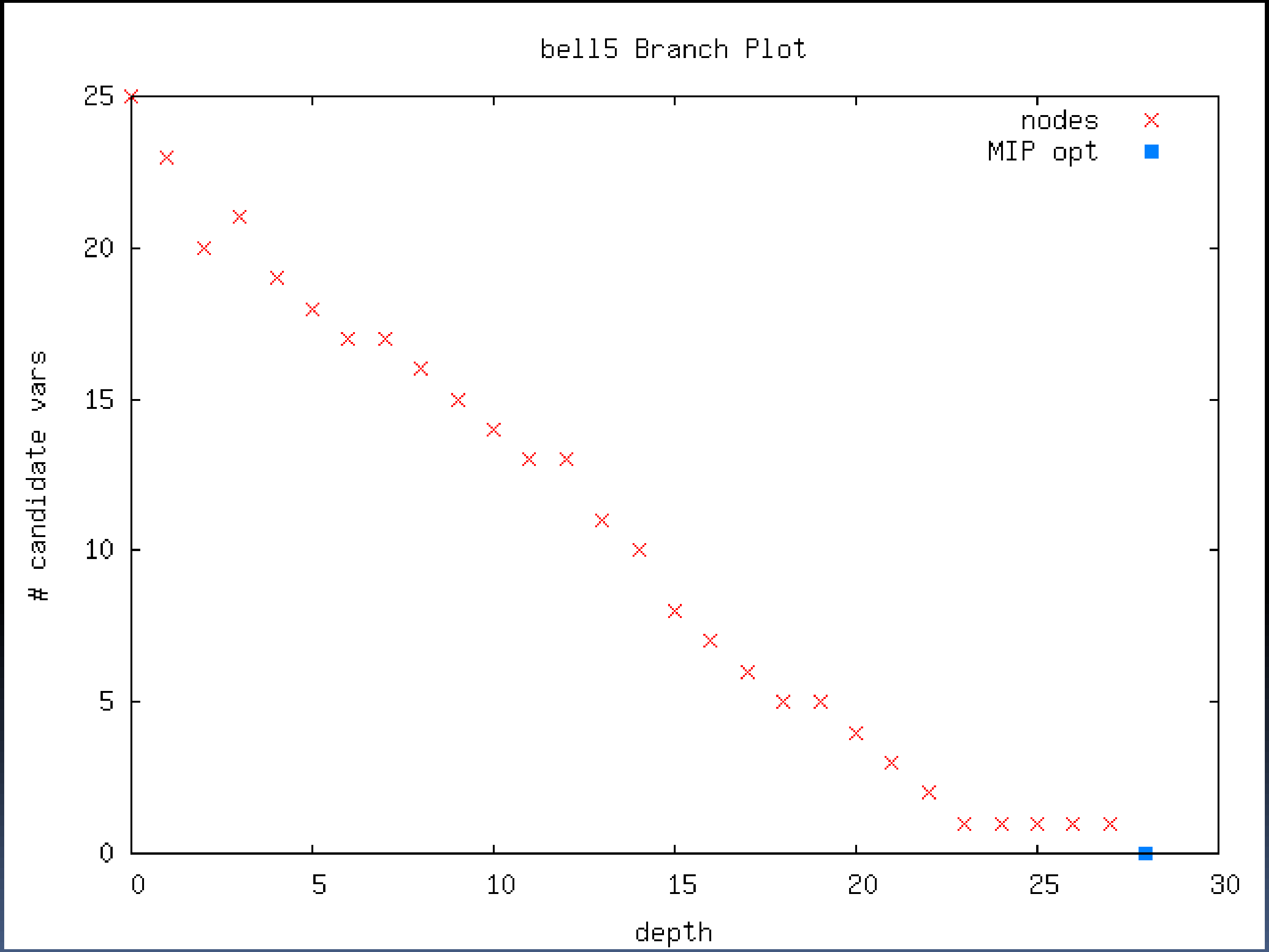
- Branch down: (0, 0.333, 0.333, 0.333) *or others*
- Branch up: (1, 0, 0, 0) is *only* solution

2. A New Principle

- Observations
  - *Often*: each branching forces roughly 1 candidate variable to integrality
  - *Desirable*: force as many candidates as possible to integrality at each branch

- *Note: integer-feasible when number of candidate variables is zero*

Frequent Pattern

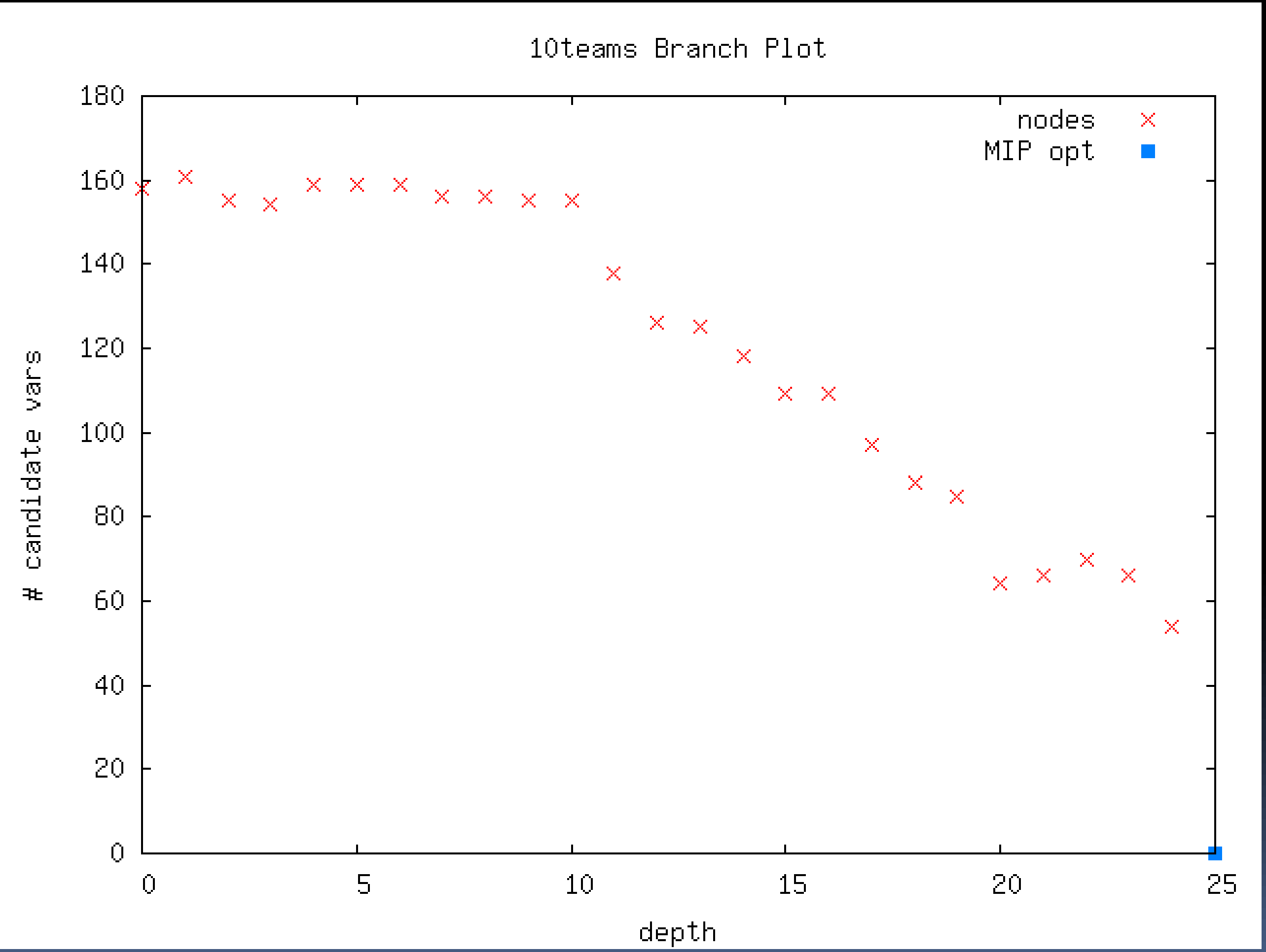


New Principle

- Branch to Force Change
  - E.g. Branch up on multiple choice constraints
  - E.g. Active constraint branching variable selection

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

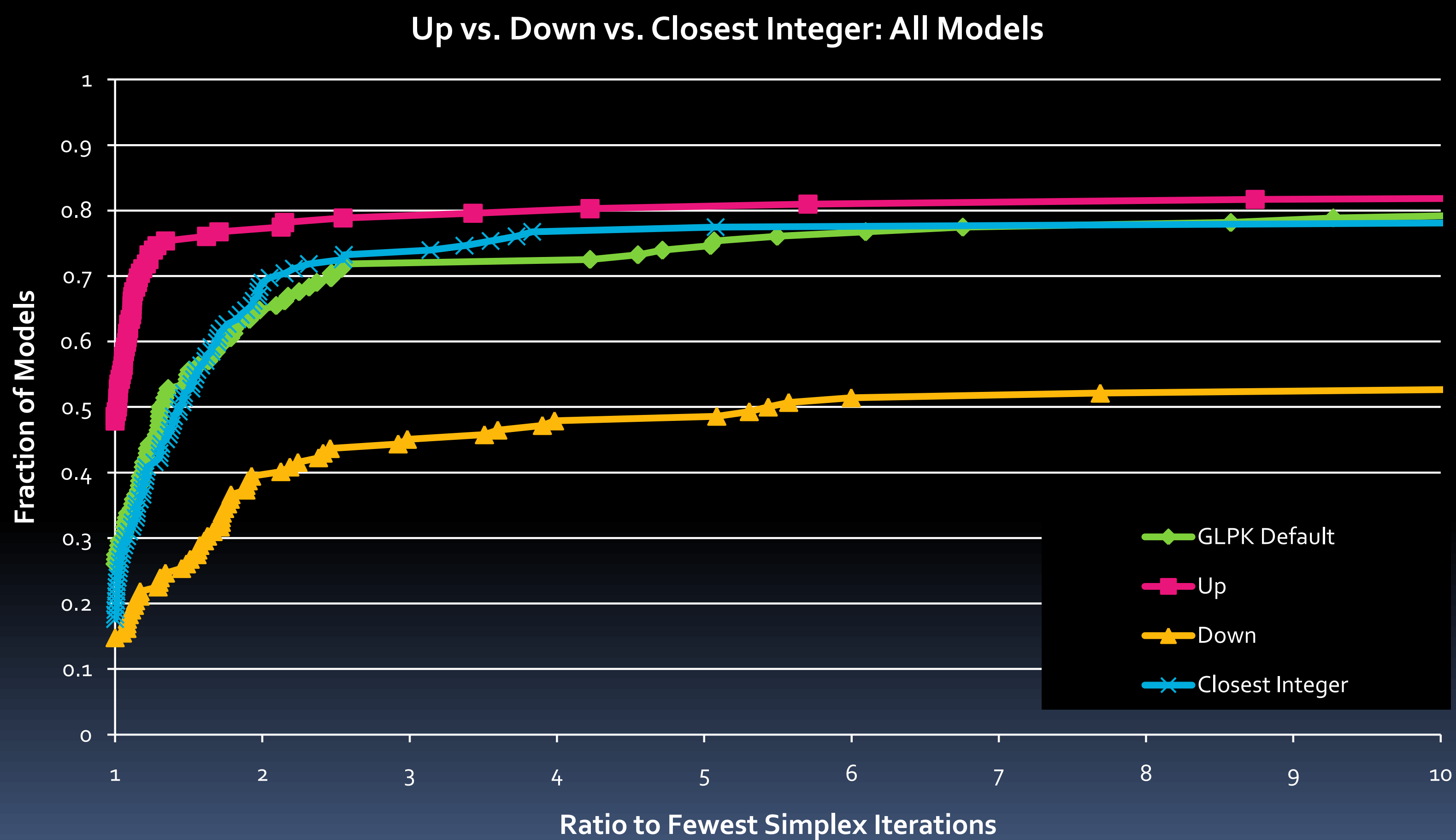
Reach Integrality Faster



3. Experimental Setup

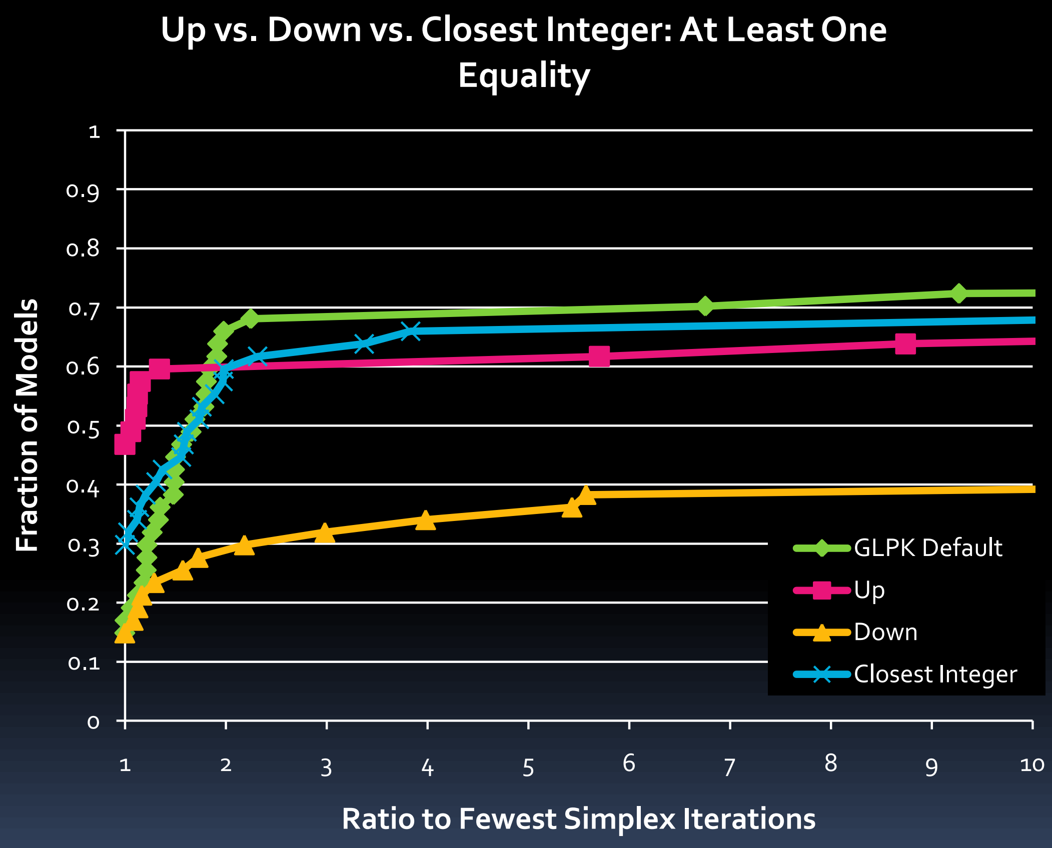
- Modifications to GLPK 4.28
- Stopping: first feasible solution, or two hours
- Node selection:
  - Driebeek and Tomlin (GLPK default), or
  - Depth first
- Test models
  - 142 total, 47 equality-containing, 95 equality-free
  - 56 from MIPLIB2003
  - 11 from MIPLIB 3.0
  - 7 from MIPLIB 2.0
  - 68 from COR@L
- Speed metric: number of simplex iterations
  - Due to variety of machines

4. Evaluating Simple Branching Direction Heuristics



Branching UP Usually Best

- Folklore: branching up is best
  - Empirically supported
  - UP is best, DOWN is worst
- Affected by equality constraints



5. Probability-based Branching

Counting solutions (Pesant and Quimper 2008)

- $l \leq \mathbf{cx} \leq u$: l, $\mathbf{c}$, u are integer values, $\mathbf{x}$ 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]	1	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

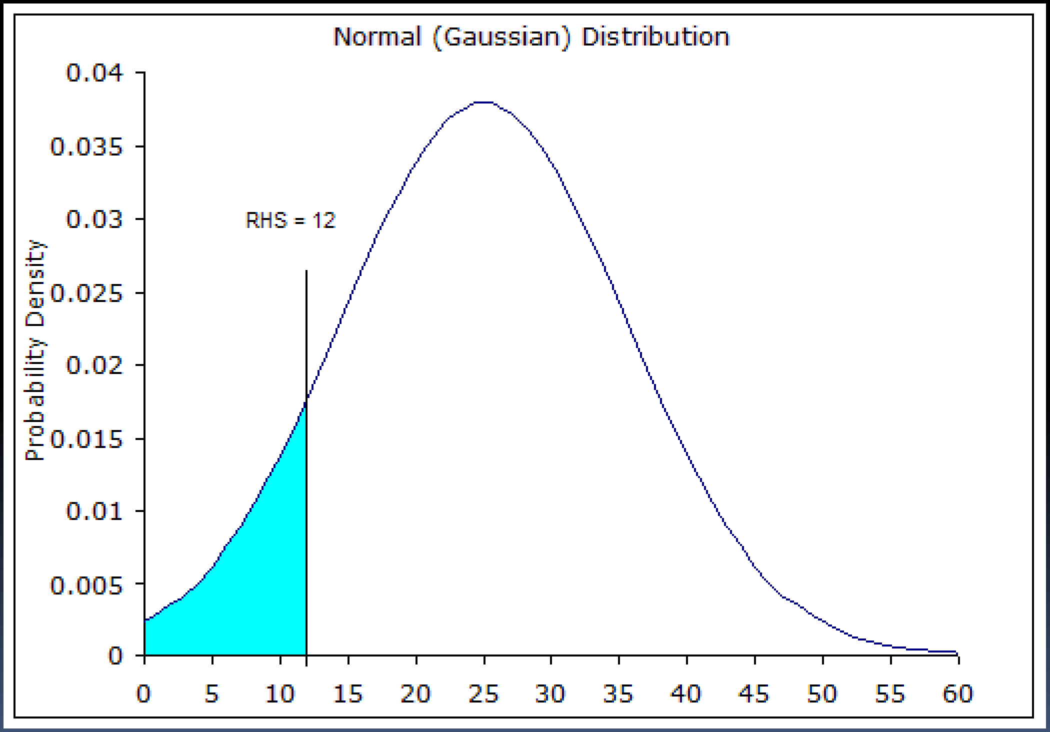
Result:

- linear combination of variables yields *normal distribution* for function value
- *Mean:* $\sum a_i(l_i+u_i)/2$, where x_i has range $[l_i, u_i]$
- *Variance:* $\sum a_i^2[(u_i-l_i+1)^2-1]/12$
- Example: $g(\boldsymbol{x}) = 3x_1 + 2x_2 + 5x_3$, $0 \leq \boldsymbol{x} \leq 5$ has mean 25, variance 110.83
- *Plot….* Look at $g(\boldsymbol{x}) \leq 12$

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

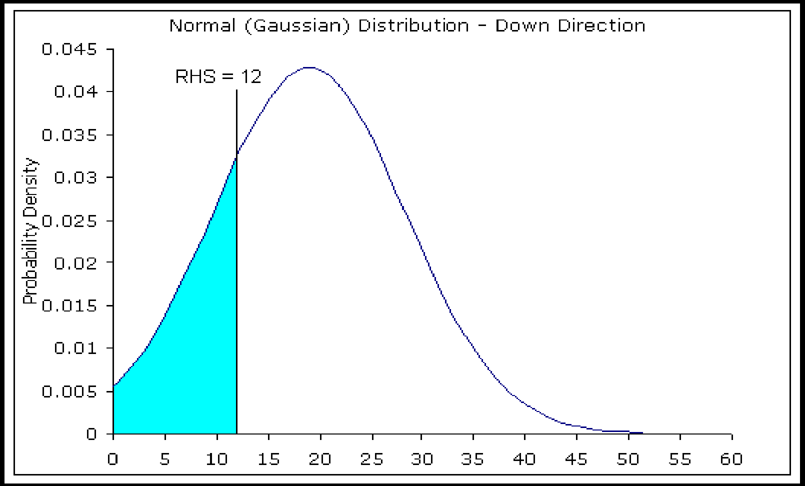
Probability density plot

- Cumulative prob of satisfying function in blue



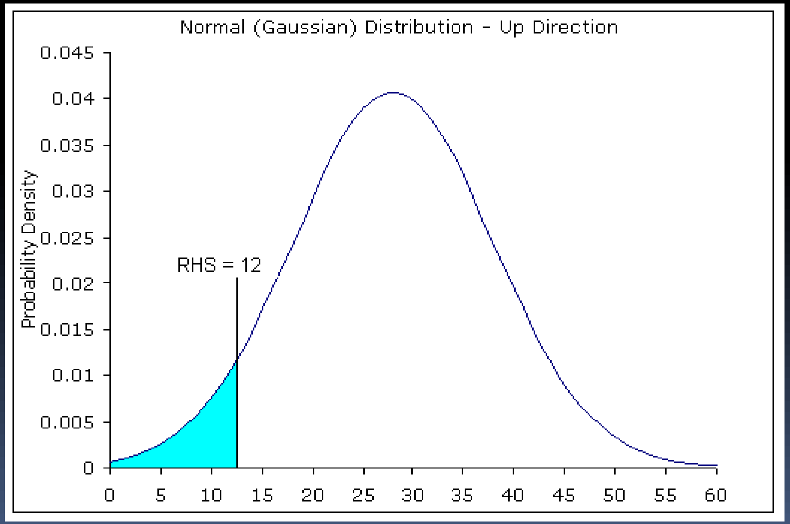
Use for Branching

- Separate distributions for DOWN and UP branches due to changed variable ranges
- Calculate cumulative probability 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



New: Handling Equality Constraints

- Look at *centeredness* of RHS value in the two prob. curves created by branching UP or DOWN
- For each of branch-UP and branch-DOWN:
  - Calculate cum. prob. of being less than RHS
  - Calculate cum. prob. of being more than 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 the branching variable:

- Choose direction based on cum. prob. in <u>*any*</u> active constraint branching variable is in:
  - LCP: Lowest Cum. Prob. in any active constraint
  - HCP: Highest Cum. Prob. in any active constraint
- Choose direction based on *votes* using cum. prob. in <u>*all*</u> active constraints branching variable is in:
  - LCPV: direction most often selected based on lowest cum. prob.
  - HCPV: direction most often selected based on highest cum. prob.

New Simultaneous Variable and Direction Methods

- 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

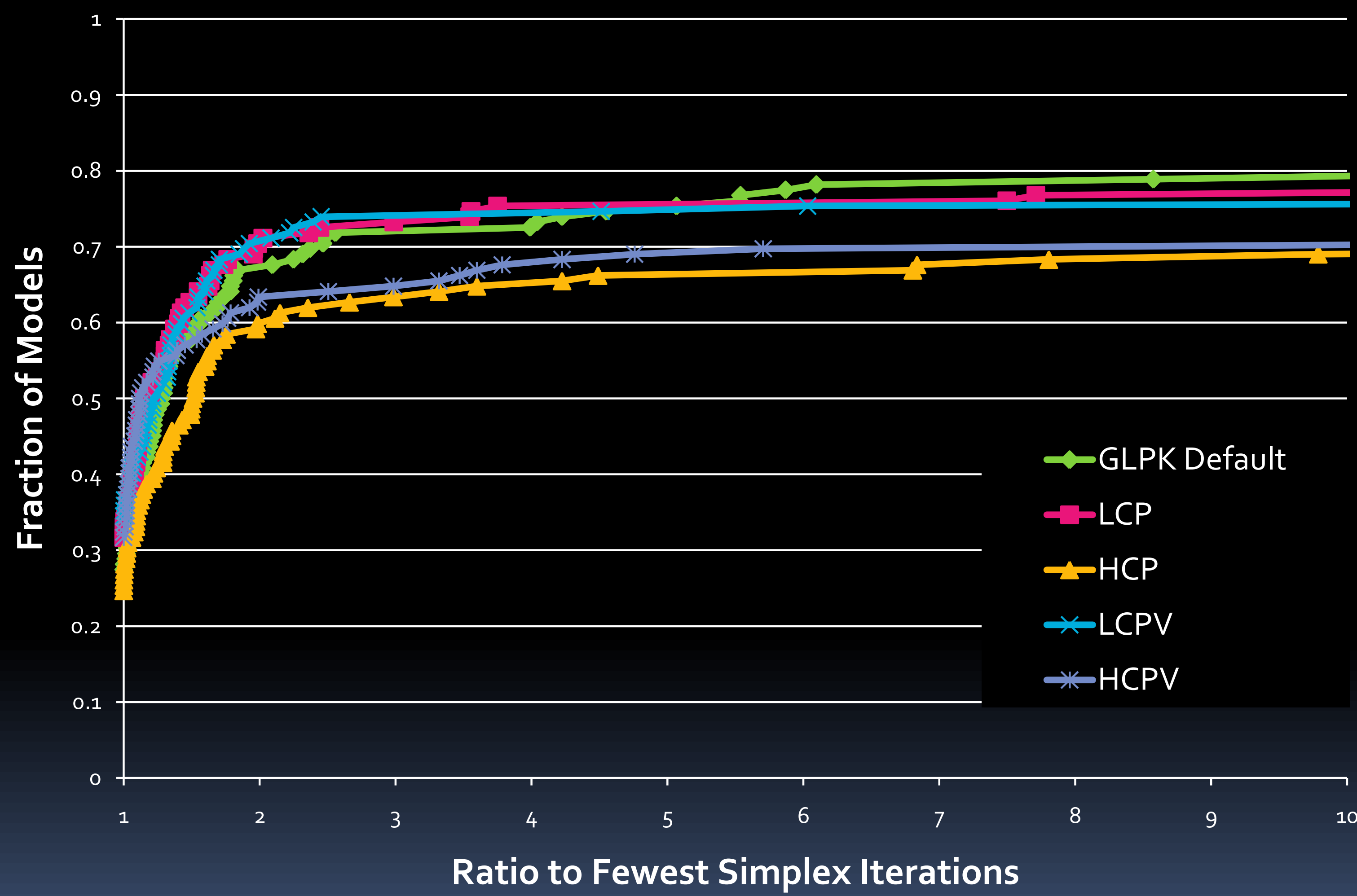
6. New Violation-Based Methods

- If all variable values except branching variable are fixed, what happens when branching direction is UP vs. DOWN?
  - *Inequality:* is act. 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

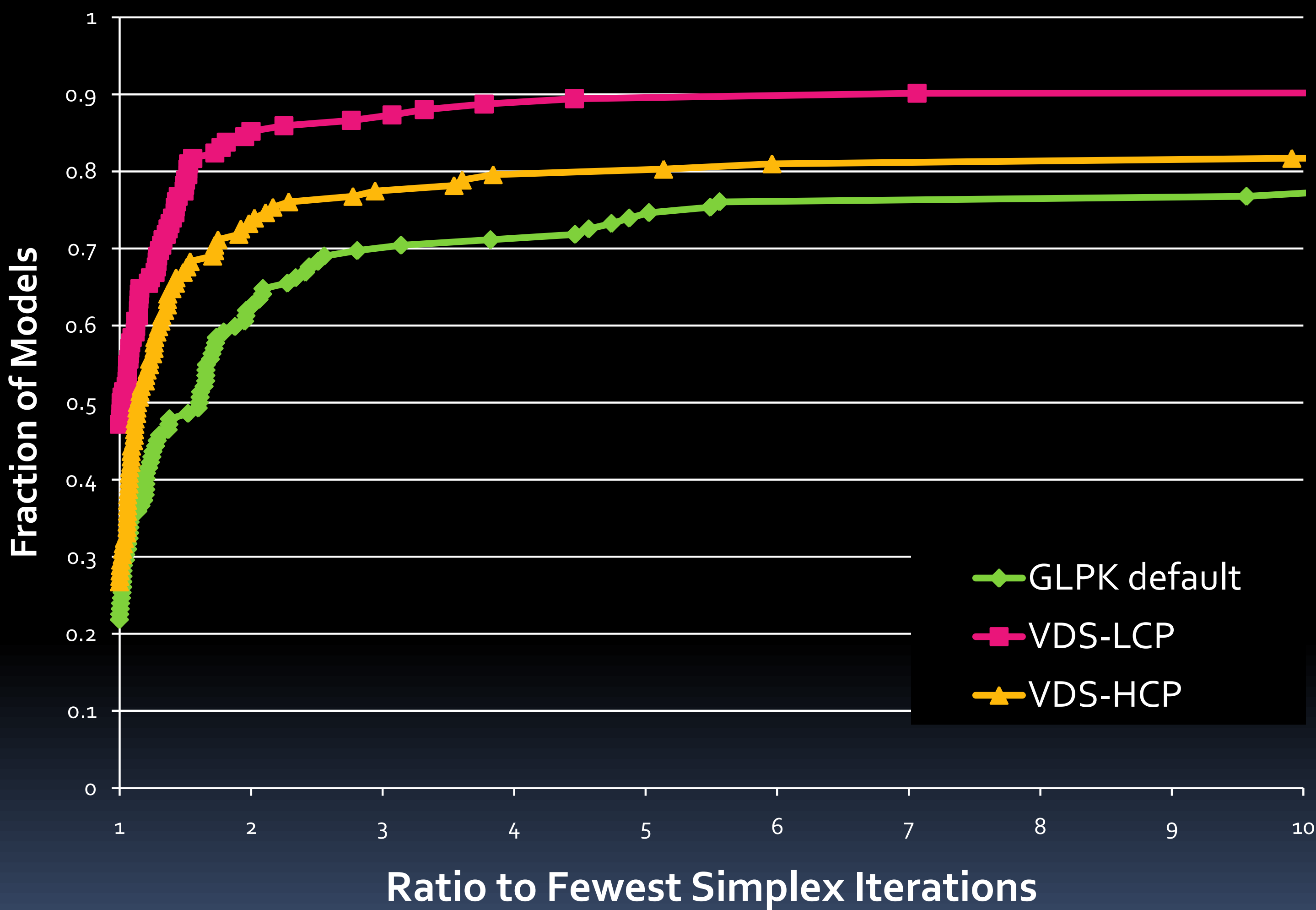
7. Experiments: Branching to Force Change

- 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/LCPV vs. HCP/HCPV: All Models

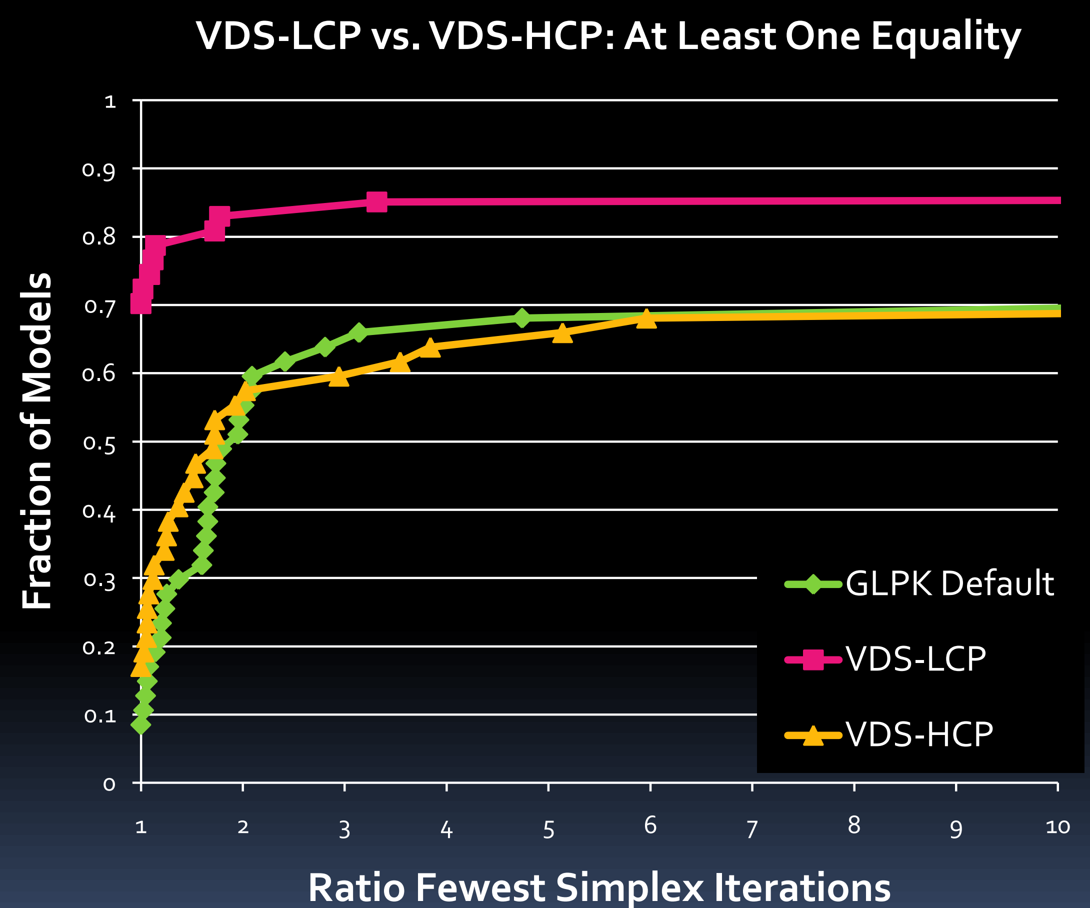


VDS-LCP vs. VDS-HCP: All Models

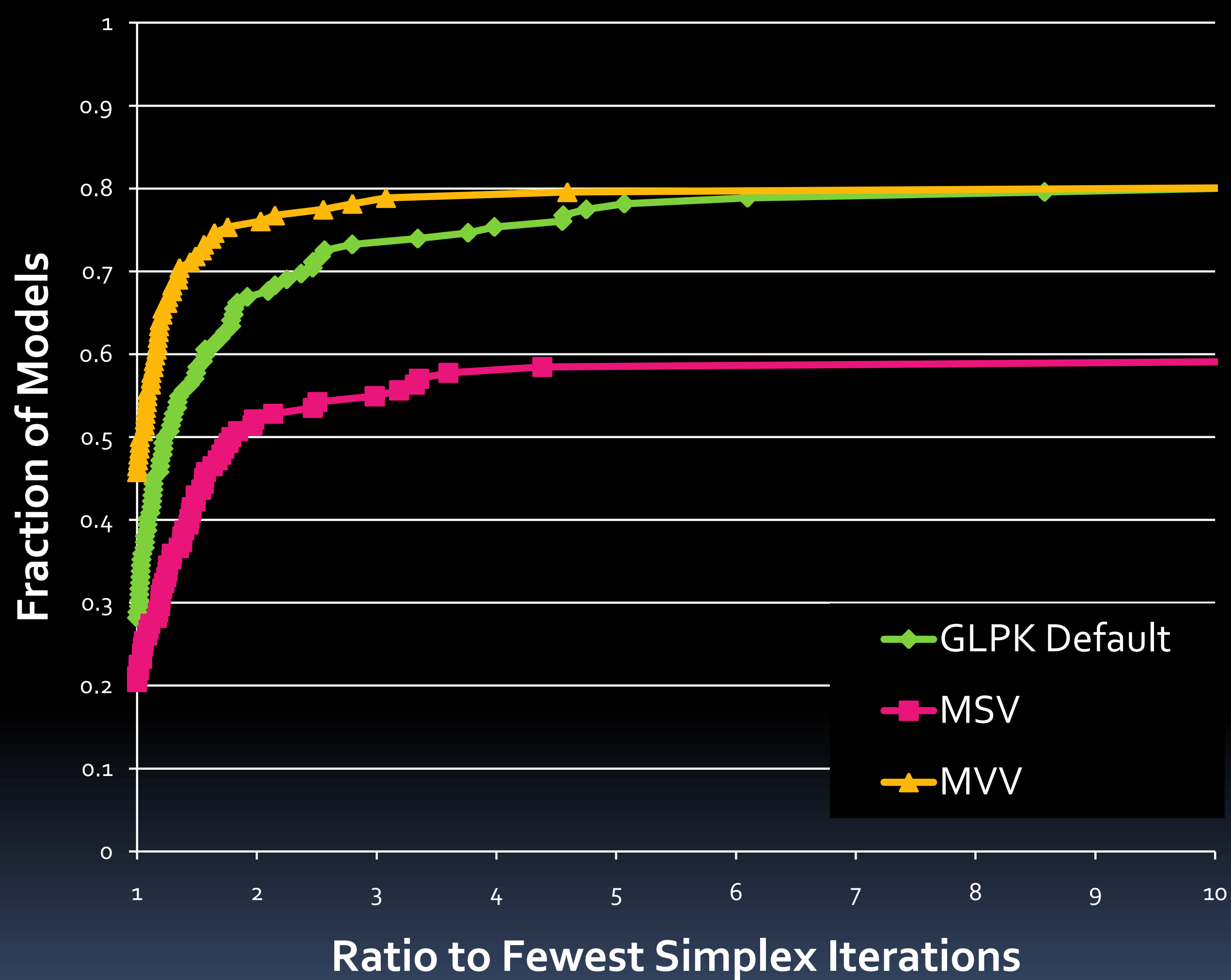


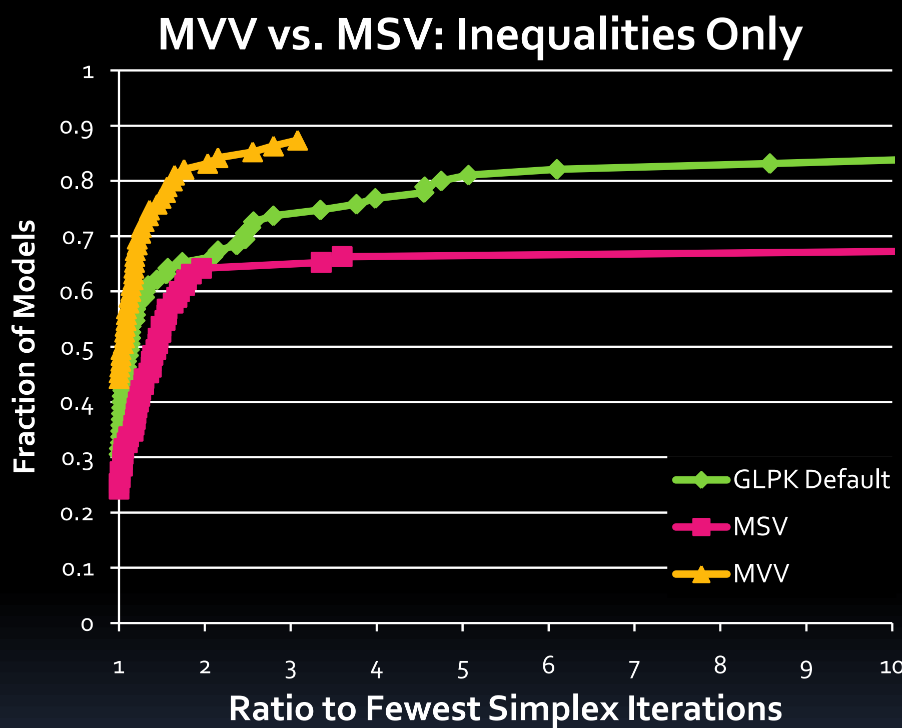
VDS Methods With Equality Constraints

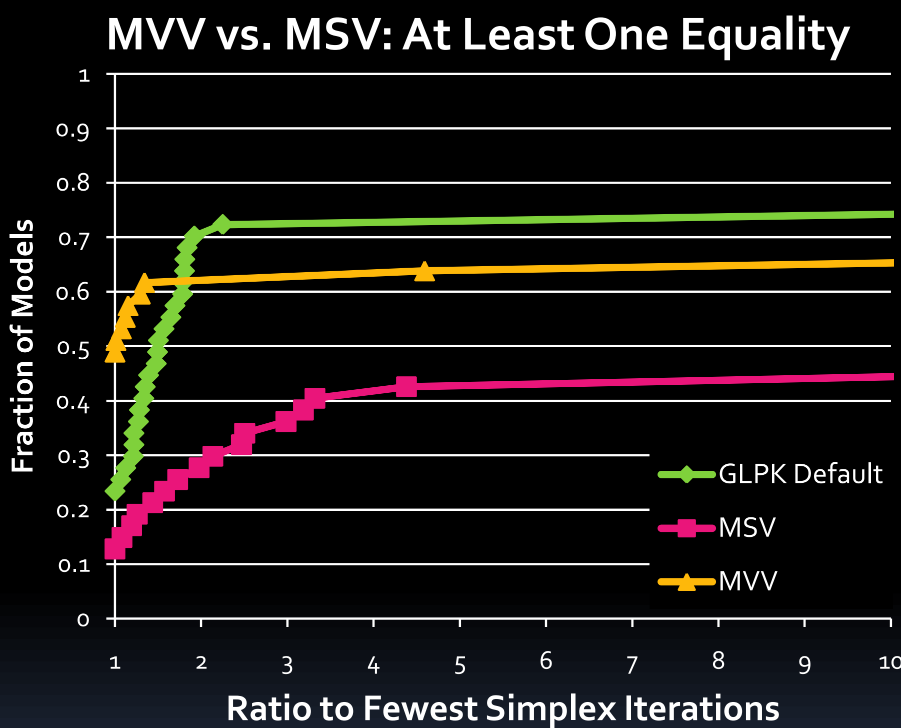
- VDS-LCP even more dominant
- The centering strategy is effective



MVV vs. MSV: All Models




MVV vs. MSV: Inequalities Only
Fraction of Models
Ratio to Fewest Simplex Iterations
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
8
9
10
GLPK Default
MSV
MVV


MVV vs. MSV: At Least One Equality
Fraction of Models
Ratio to Fewest Simplex Iterations
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
8
9
10
GLPK Default
MSV
MVV

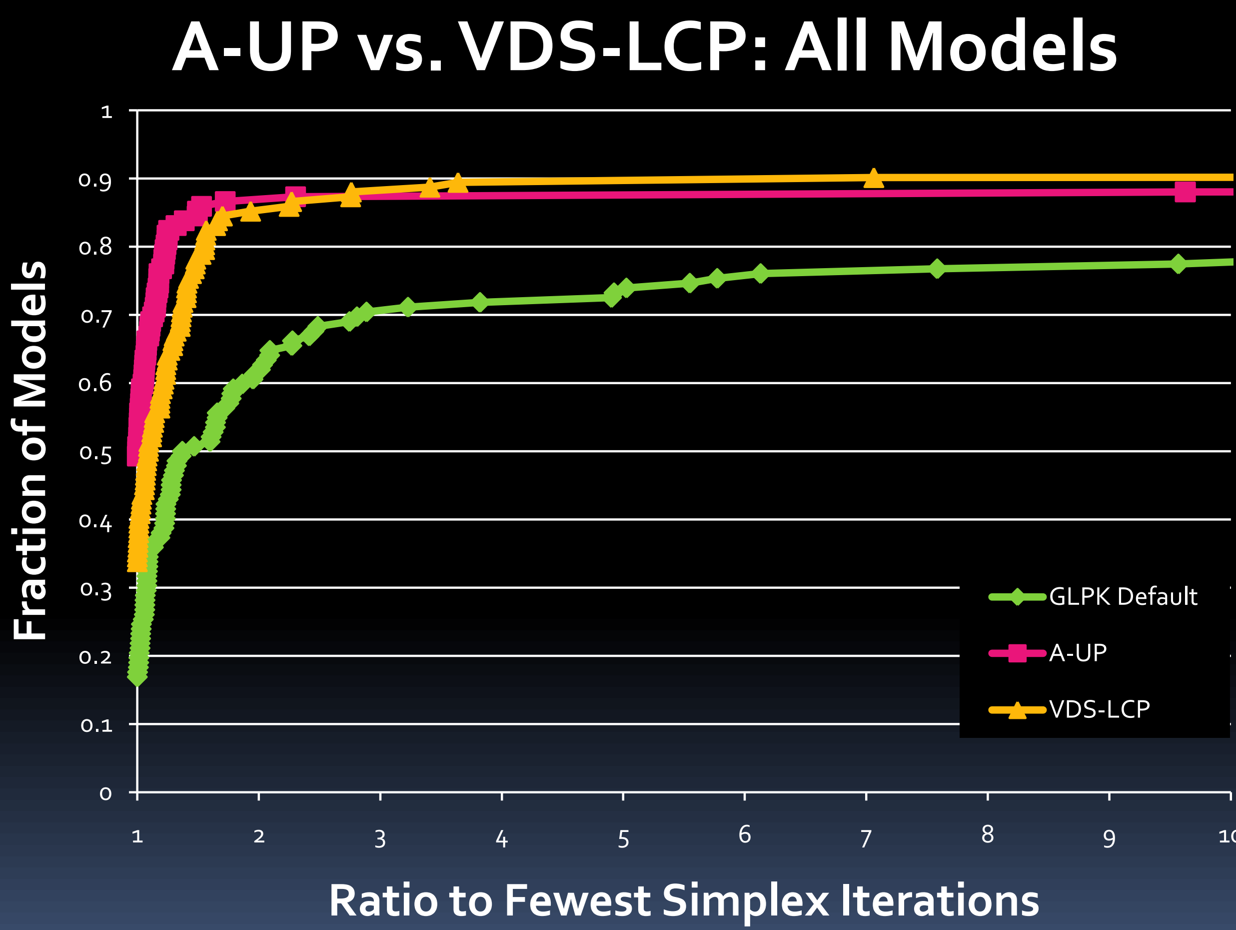
Effect of Branching Variable Heuristic

	fraction fewest simplex iterations	fraction solved
GLPK Default	0.1620	0.8239
GLPK-UP	0.2887	0.8592
A-UP	**0.3662**	**0.8944**
GLPK-LCP	0.1831	0.8310
A-LCP	**0.3028**	**0.8592**
GLPK-LCPV	0.1901	0.7958
A-LCPV	**0.2394**	**0.8521**
GLPK-MVV	0.2042	0.8310
A-MVV	**0.3028**	**0.8521**

Conclusions Thus Far

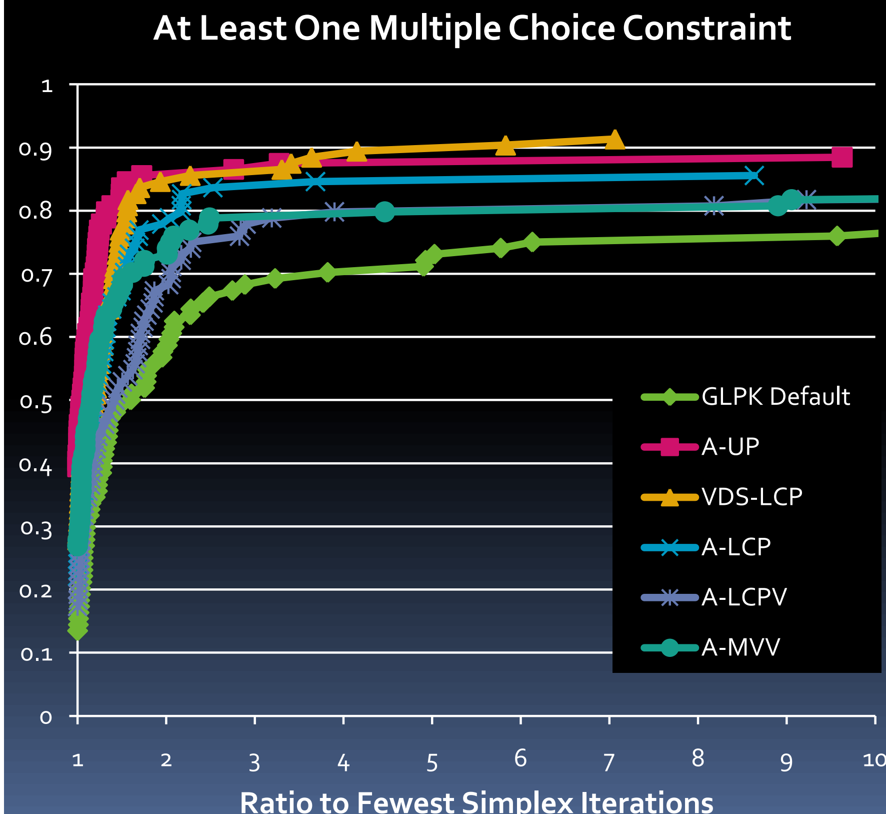
- Branching to force change in the candidate variables is fastest to first feasible solution
  - LCP better than HCP
  - LCPV better than HCPV
  - VDS-LCP better than VDS-HCP
  - MVV better than MSV
- Constraint types have an impact:
  - Equality constraints; multiple choice constraints
- One counter-example: set covering
  - Feasible solution easy: set all variables to 1

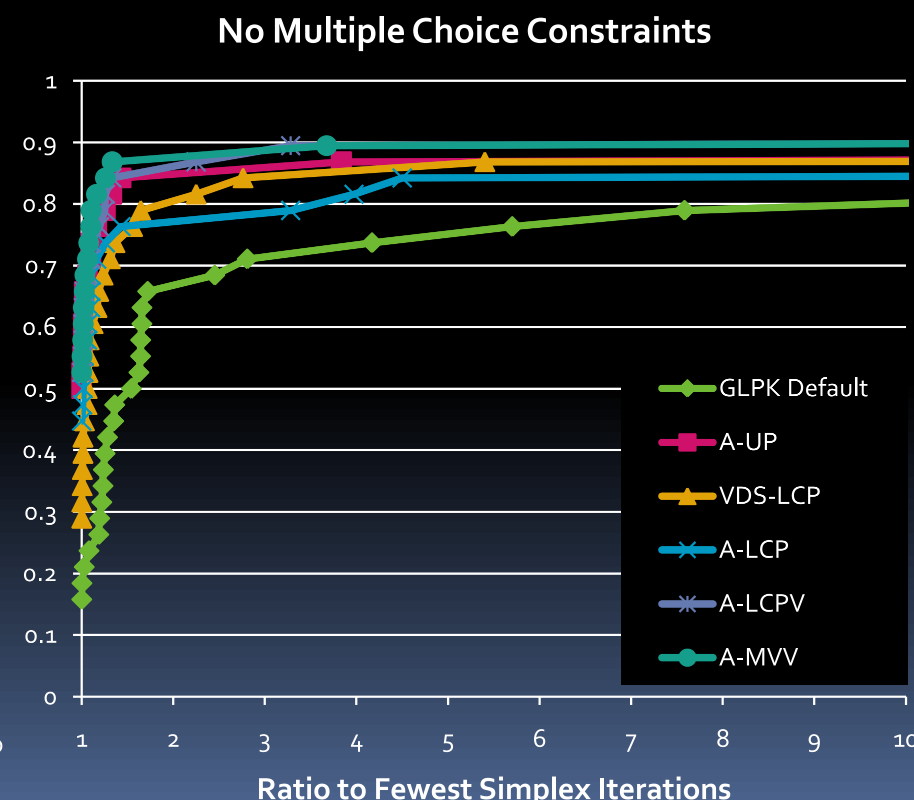
8. A-UP vs. VDS-LCP



9. Branching Up Revisited

- Why is it so good?
- Presence of multiple choice constraints?
  - 104 of 142 (73%) models have at least one





Prob. in Multiple Choice Constraints

# Variables	Cum. Prob. Up	Cum. Prob. Down
2	**0.158655254**	0.841344746
3	**0.078649604**	0.5
4	**0.041632258**	0.281851431
5	**0.022750132**	0.158655254
6	**0.012673659**	0.089856247

$x_1 + x_2 \leq 1$

# Variables	Equality Ratio Up	Equality Ratio Down
2	**0.188573417**	**0.188573417**
3	**0.085363401**	1
4	**0.043440797**	0.392469529
5	**0.023279749**	0.188573417
6	**0.012836343**	0.098727533

$x_1 + x_2 = 1$

10. Contributions

- Principle of branching to force change in the candidate variables leads to faster feasibility
  - *Surprise!* Branch to low-probability direction
- Presence of equalities, multiple choice constraints affects performance of heuristics
  - UP works well because it is more often the lower probability direction
- Extension of probability-based methods to equality constraints
- New branching methods (esp. VDS-LCP)

Outline

1. Introduction and Orientation (Lodi)

Part I: Achieving Integer-Feasibility Quickly

2. Classic Feasibility-Seeking Algorithms (Chinneck)
3. Active Constraint Variable Selection (Chinneck)
4. Branching to Force Change (Chinneck)
5. The Feasibility Pump (Lodi)

Part II: Reaching Optimality Quickly

6. New Node Selection Rules (Chinneck)
7. Local Branching and RINS (Lodi)

Part III: Analyzing Infeasible MIPs

8. Isolating Infeasible Subsystems (Chinneck)
9. Repairing MIP Infeasibility via Local Branching (Lodi)
10. Conclusions (Chinneck)

Faster MIP Solutions via New Node Selection Rules

Daniel T. Wojtaszek
John W. Chinneck
Systems and Computer Engineering
Carleton University
Ottawa, Canada

Branch and Bound

Main B&B algorithm design choices:

- How to choose the integer infeasible (*candidate*) variable to branch on at a node.
- How to choose the unexplored (active) node to solve next.
  - Triggering backtrack.
  - Which node to choose when backtracking.
  - *Theme*: using distributions and correlations to define heuristics

Outline

1. Triggering Backtrack
   - Feasibility Depth Extrapolation
   - Modified Best Projection Aspiration
2. Choosing Node When Backtracking
   - Modified Best Projection
   - Distribution-based Backtracking

   2.1 Active Node Search Threshold
3. Experiments
4. Conclusions

1. Triggering Backtrack

Typical methods:

- Proceed depth-first until:
  - A leaf node is reached
  - Current node no longer desirable:
    - No optimum descendents (compare to incumbent)
    - No feasible descendents.

- User-supplied aspiration value

Improved Backtrack Triggers

Goal:

- Faster MIP solutions

Method:

- Heuristics to trigger backtrack when all descendents:
    - *Unlikely* to be optimal or
    - *Unlikely* to be feasible

Predicting the Optimum Z

- Z^*: optimum objective function value
- Z^i: LP-relaxation objective function value at node *i*
- *Minimization assumed*

Concept:

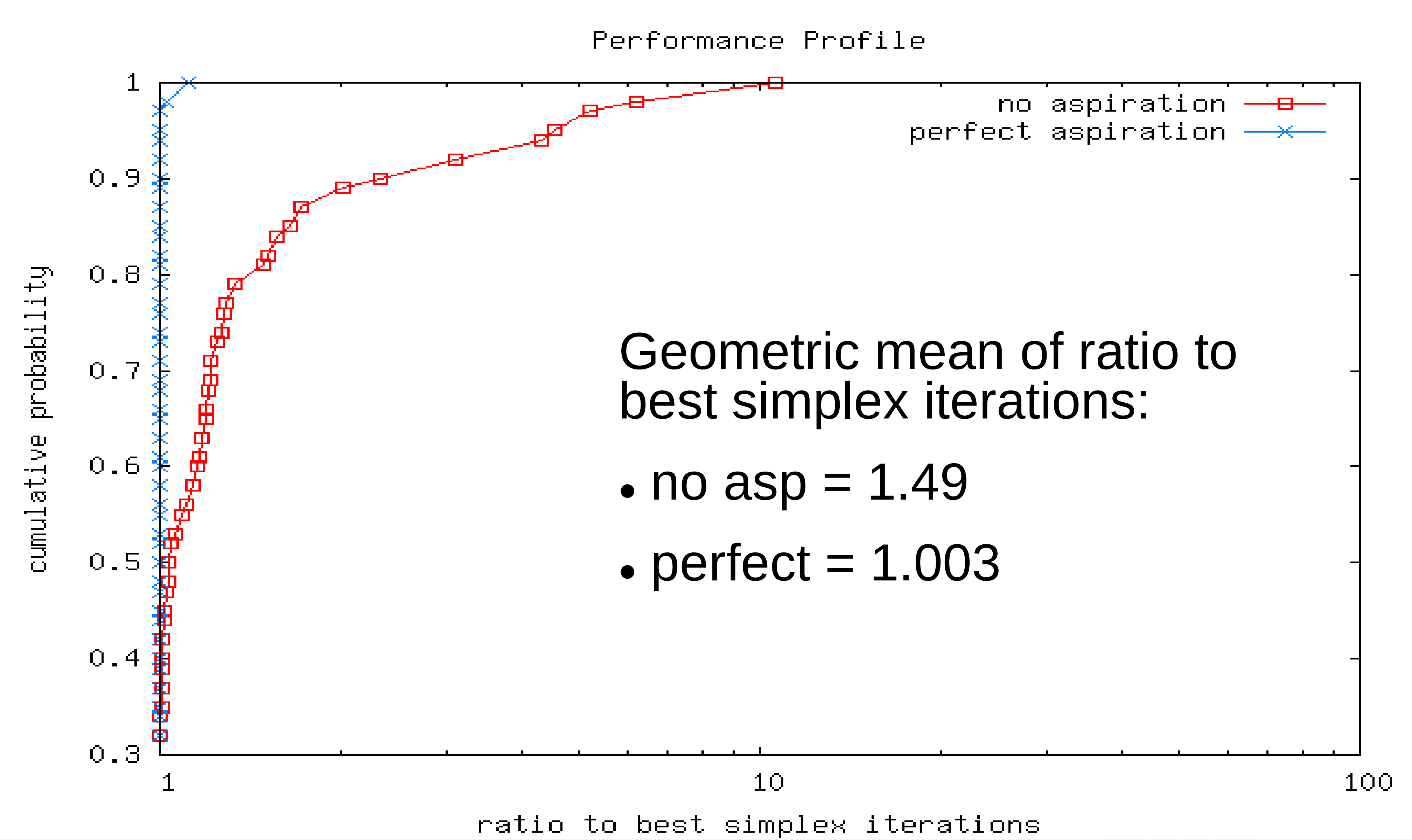
- If Z^* known in advance then trigger backtrack when node LP-relaxation value is worse
  - For minimization, trigger backtrack if $Z^i > Z^*$
- Can we estimate in advance an *aspiration value* Z^a that is close to Z^*?
  - Trigger backtrack if $Z^i > Z^a$

Proof of Concept

- Solve MIPs to find Z*
- Re-solve MIPs using $Z^a = Z^*$ to trigger backtrack

- Experimental setup:
  - Solver: GLPK 4.9
  - Default branching variable selection, backtracking node selection
  - Root node cuts: Gomory cuts
  - Test models: all MIPLIB/MIPLIB2003 that solve within 1 hour

Proof of Concept



Geometric mean of ratio to best simplex iterations:

- no asp = 1.49
- perfect = 1.003

Estimating Z*: State of the Art

Two methods normally used for node selection, ***not*** triggering backtrack:

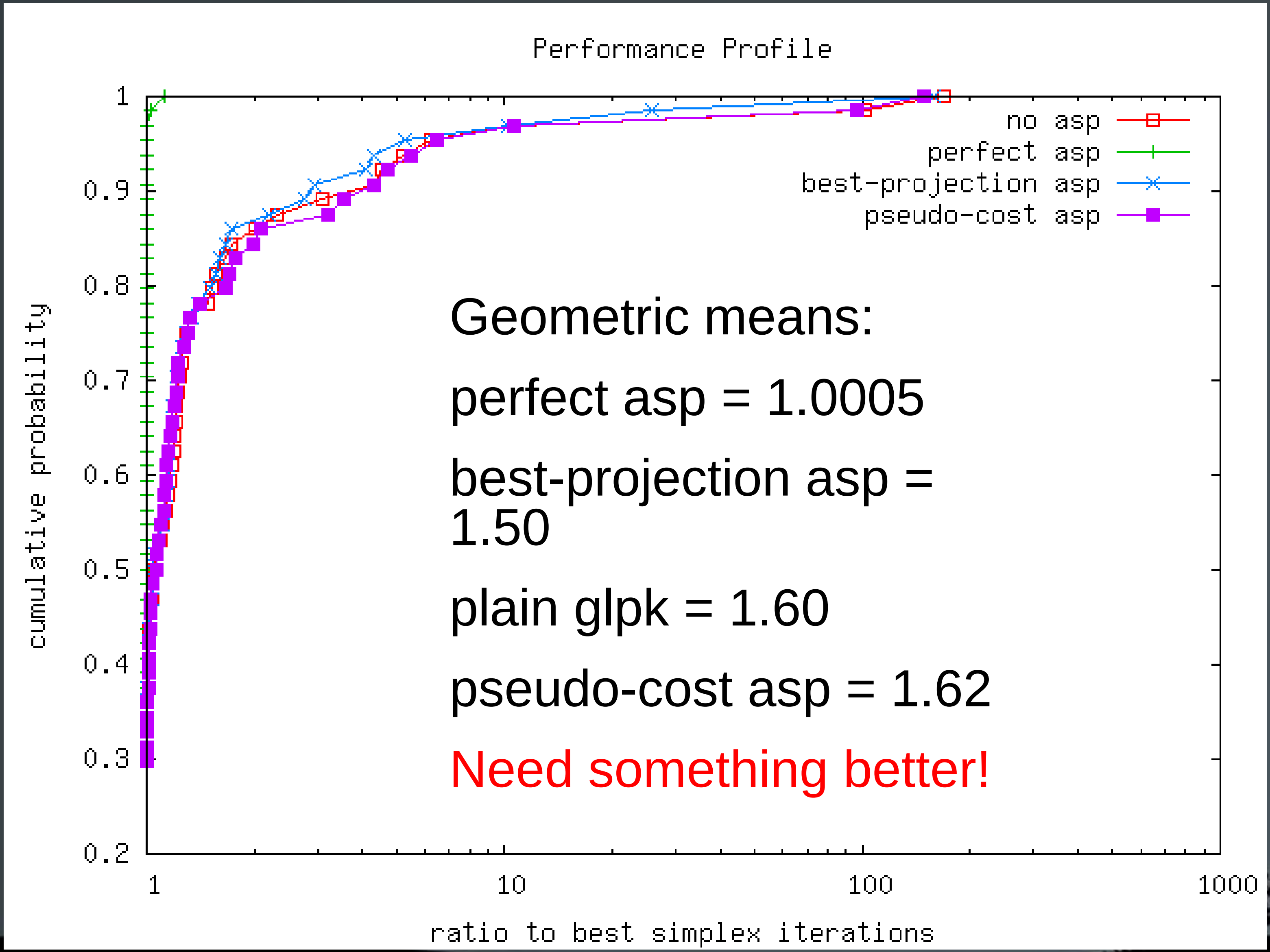
Pseudo-cost estimates:

- Note $\Delta Z/\Delta x$ at each branching. Project Z* based on this.

Best-projection estimates:

- Compare (improvement in Z between root LP-relaxation and incumbent) to (reduction in integer infeasibility)
- Project Z* based on this.

Using Available Estimators



Geometric means:

perfect asp = 1.0005

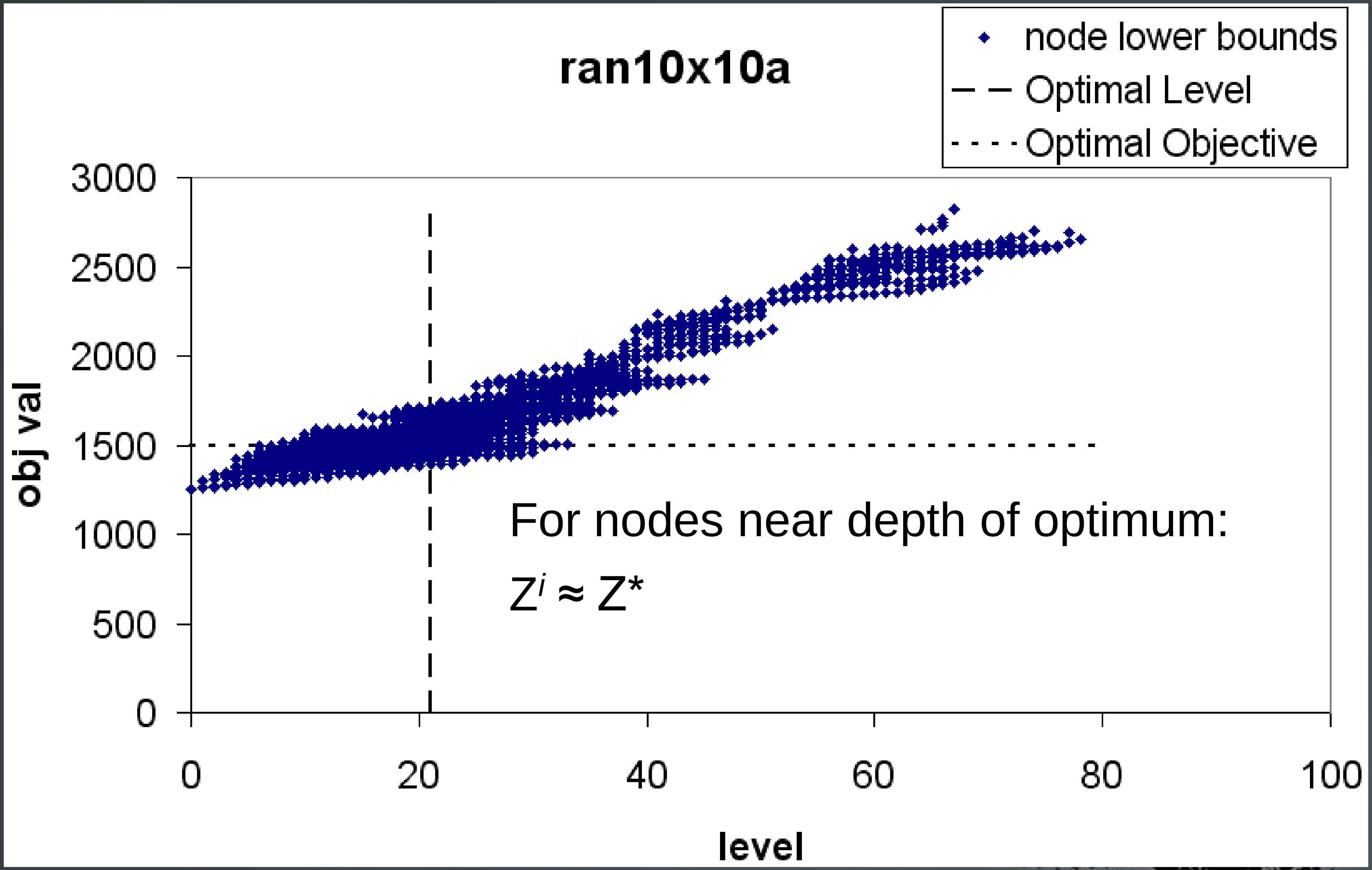
best-projection asp = 1.50

plain glpk = 1.60

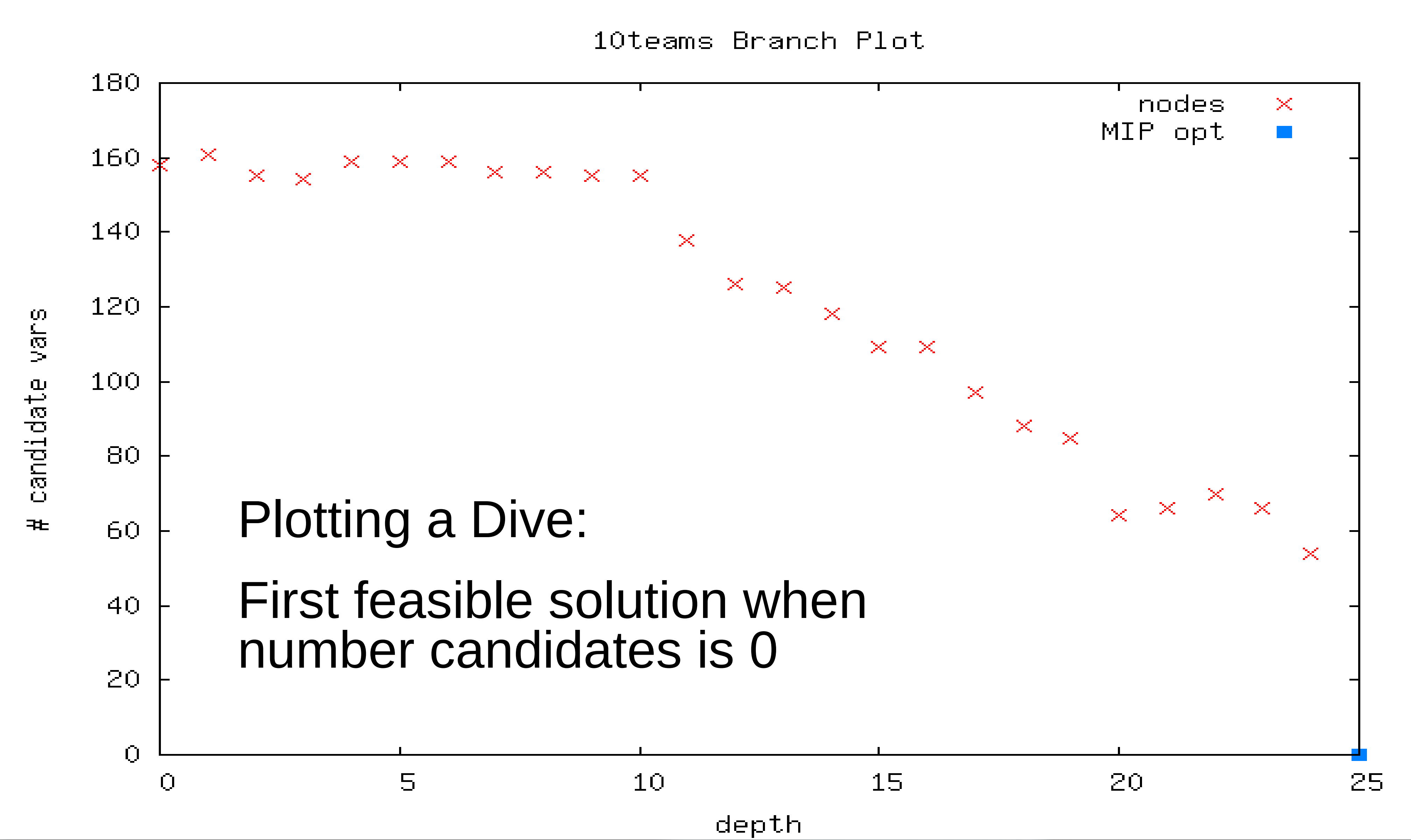
pseudo-cost asp = 1.62

Need something better!

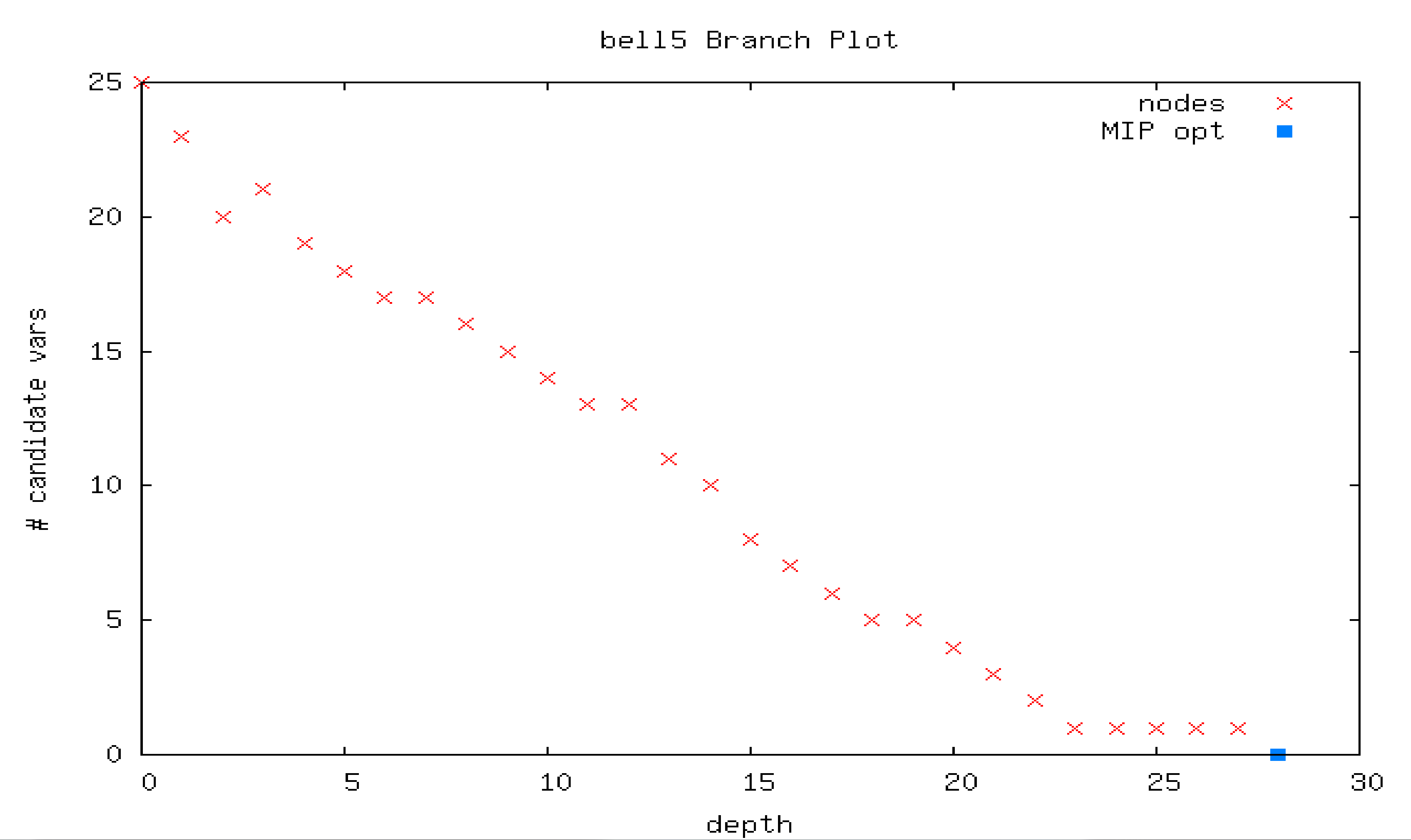
NEW: Using Depth Information



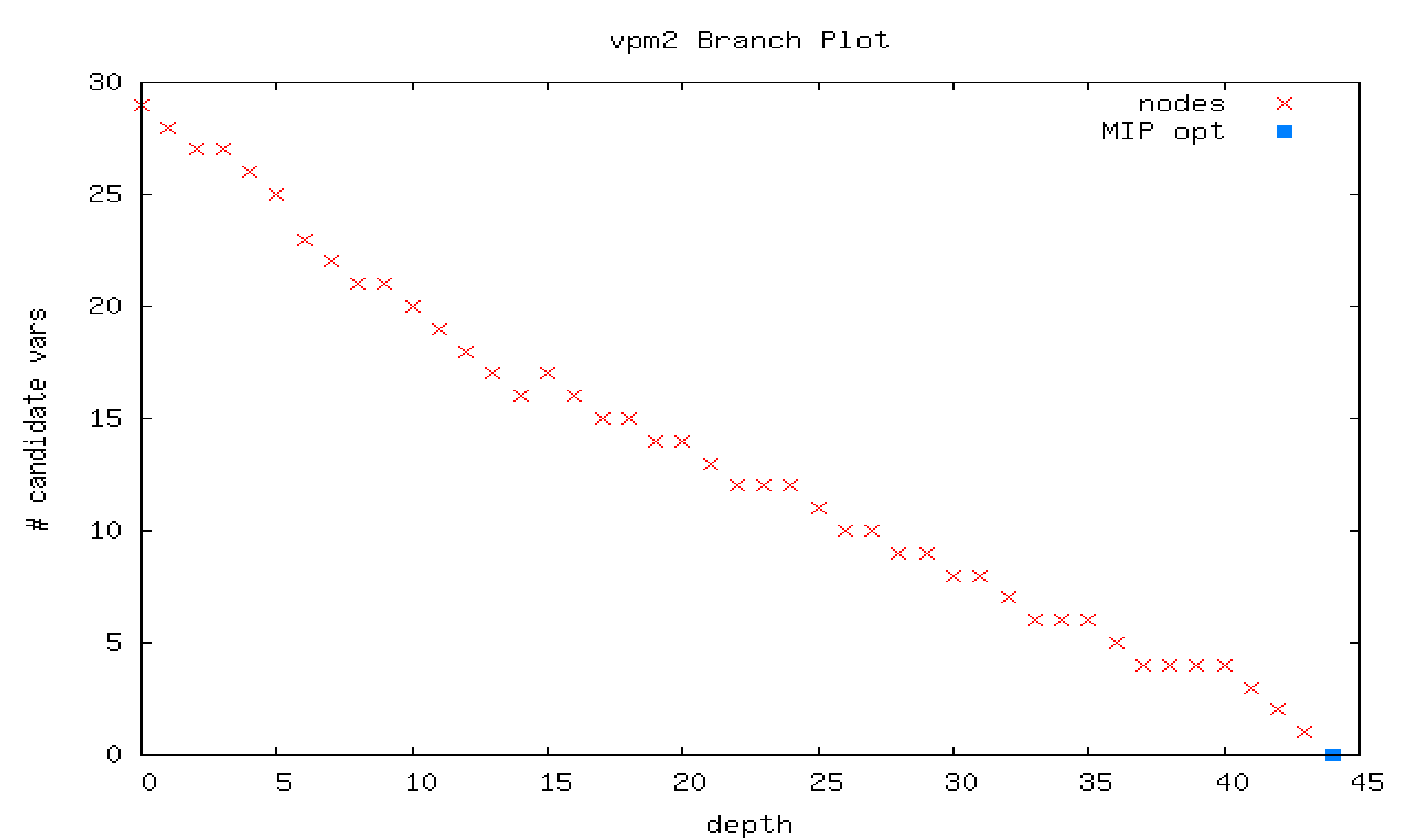
Can we predict depth of optimum?



Is There a Pattern?



Is There a Pattern?

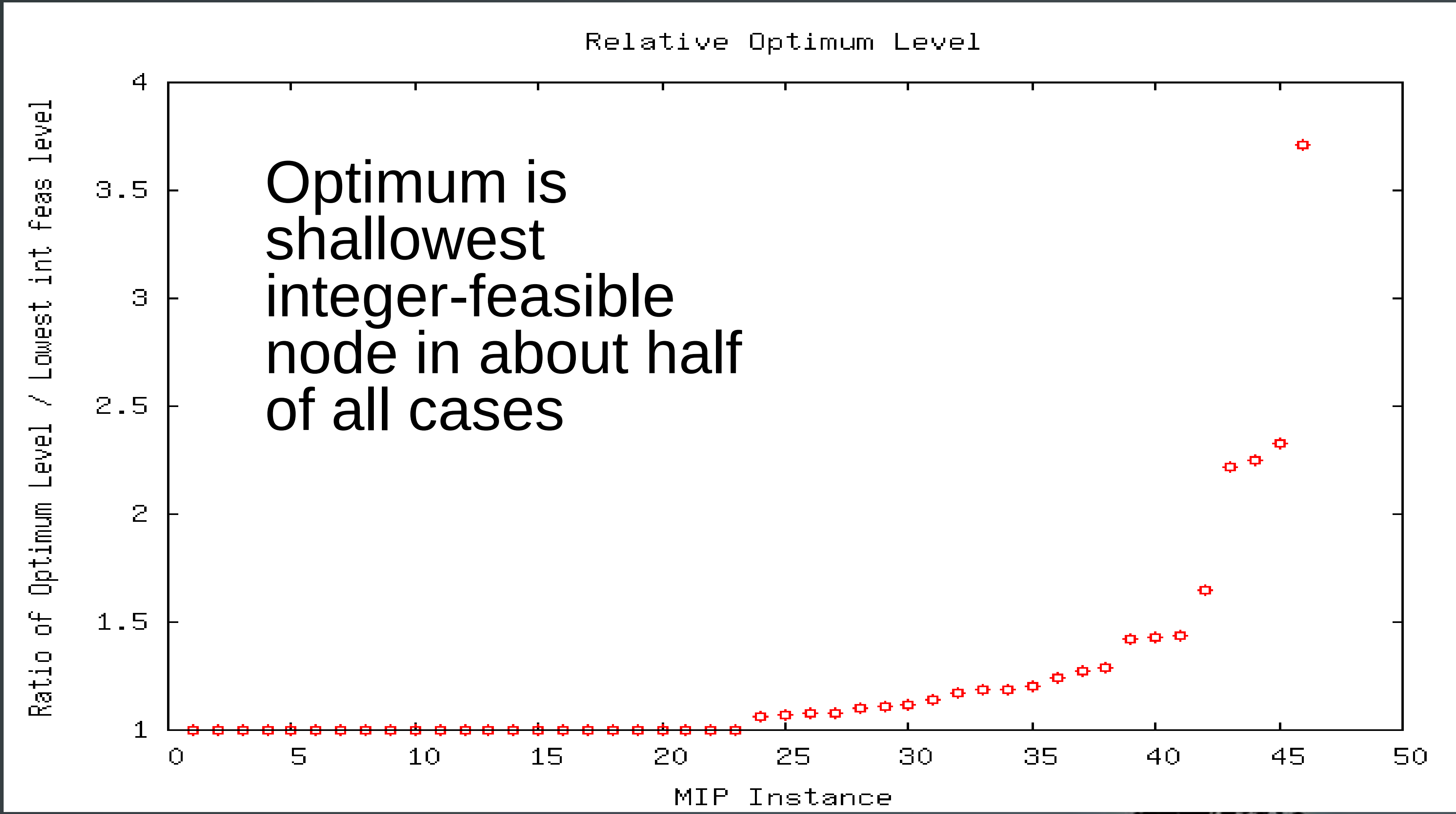


Reconciling multiple active nodes

- There are multiple active nodes in the tree
  - Each node provides a projected depth of first feasible solution
  - Which estimated depth should we use?

- *Is there a pattern*?

Observation: Optimum Depth



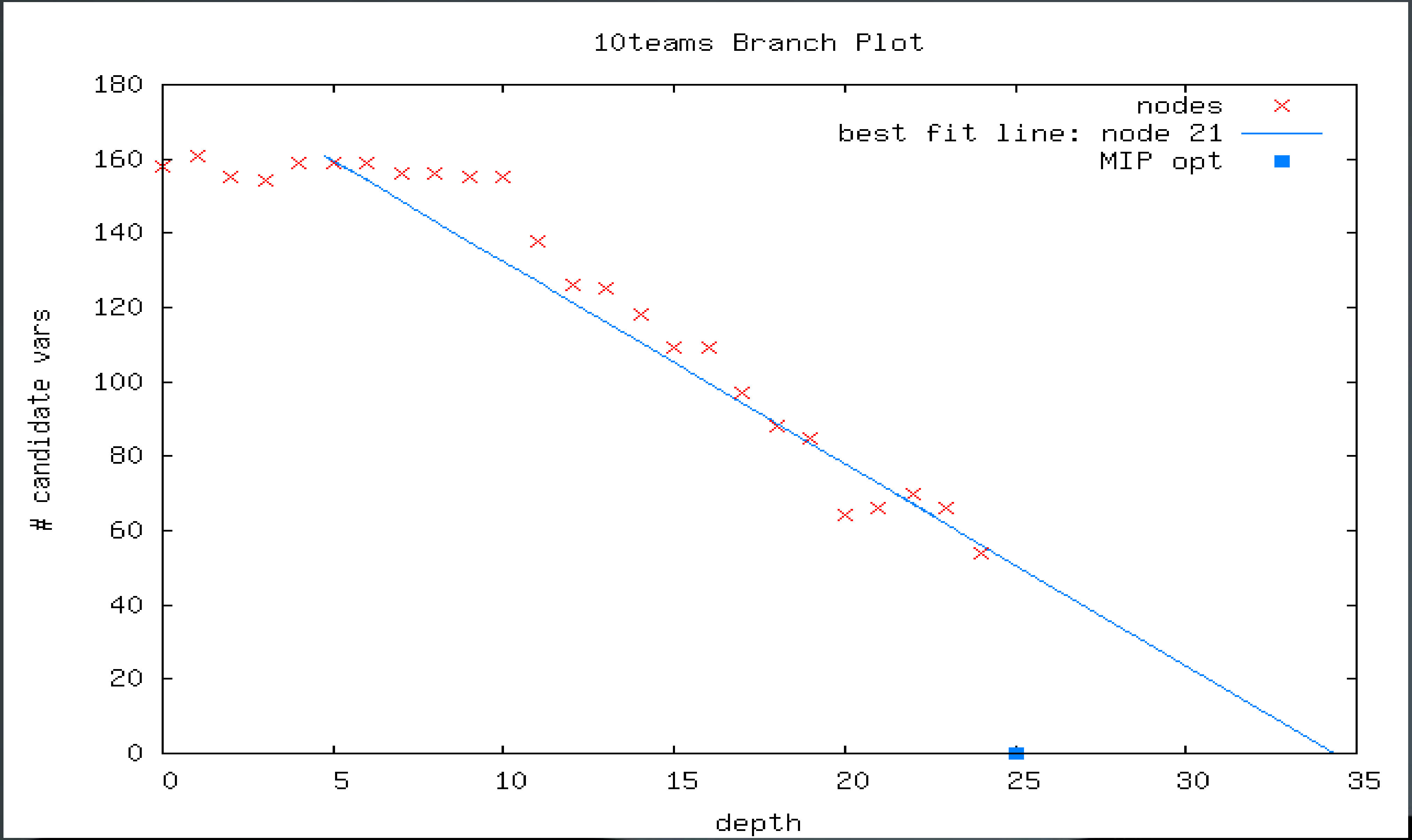
Linear Extrapolation to Estimate Z*

- 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)

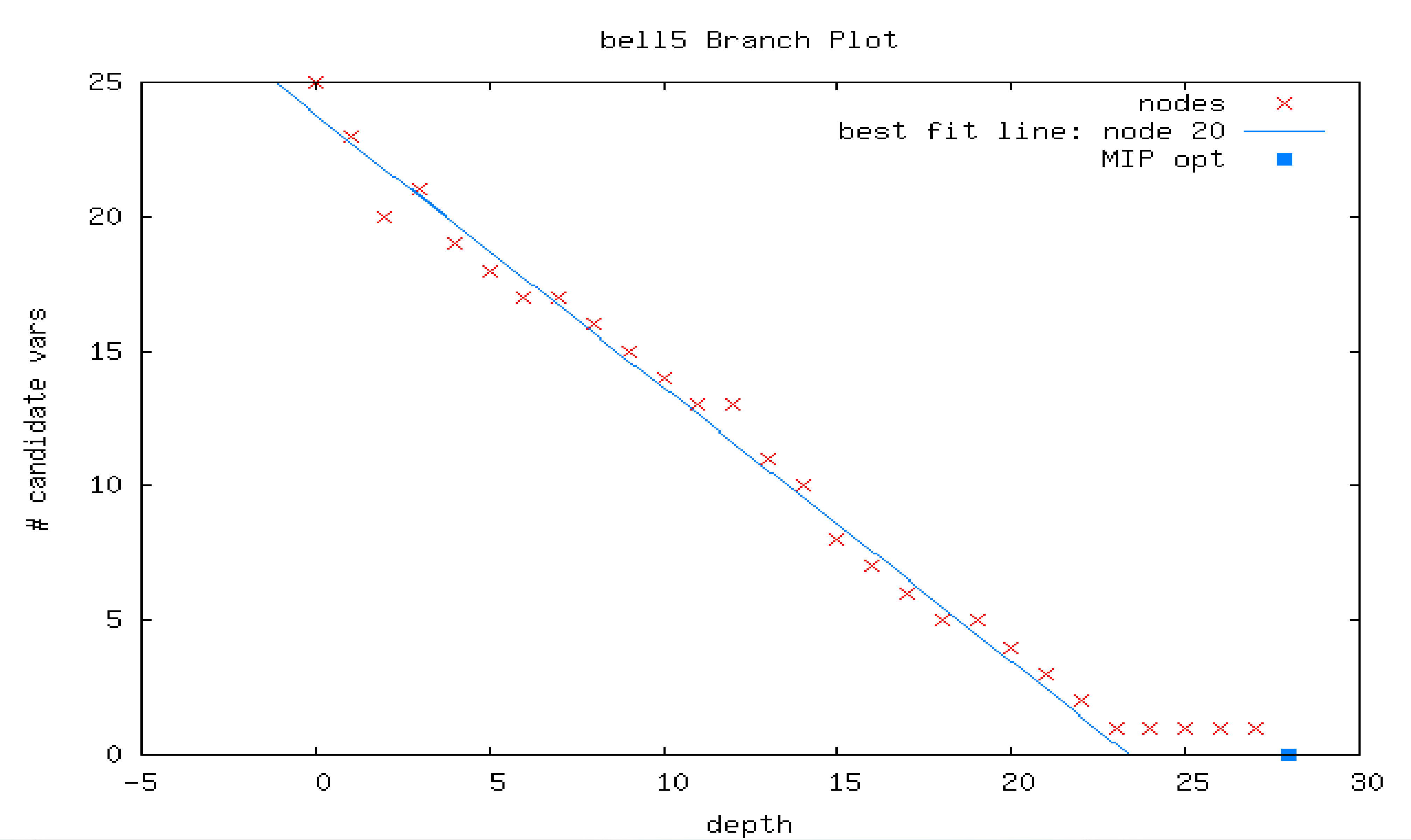
Notes

- 20 chosen empirically: enough data to extrapolate

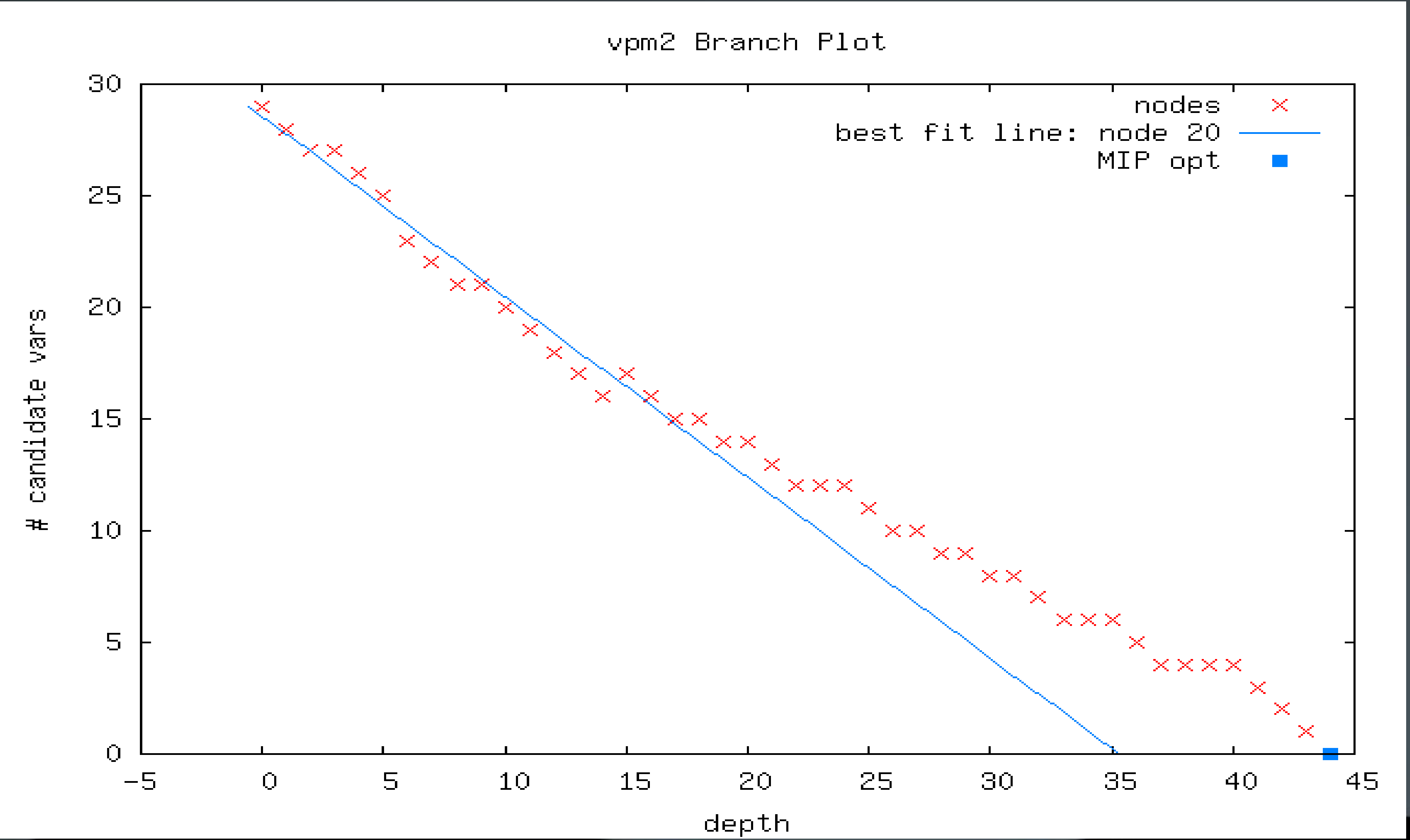
Linear Extrapolation



Linear Extrapolation



Linear Extrapolation



NEW: Modified Best Projection Aspiration

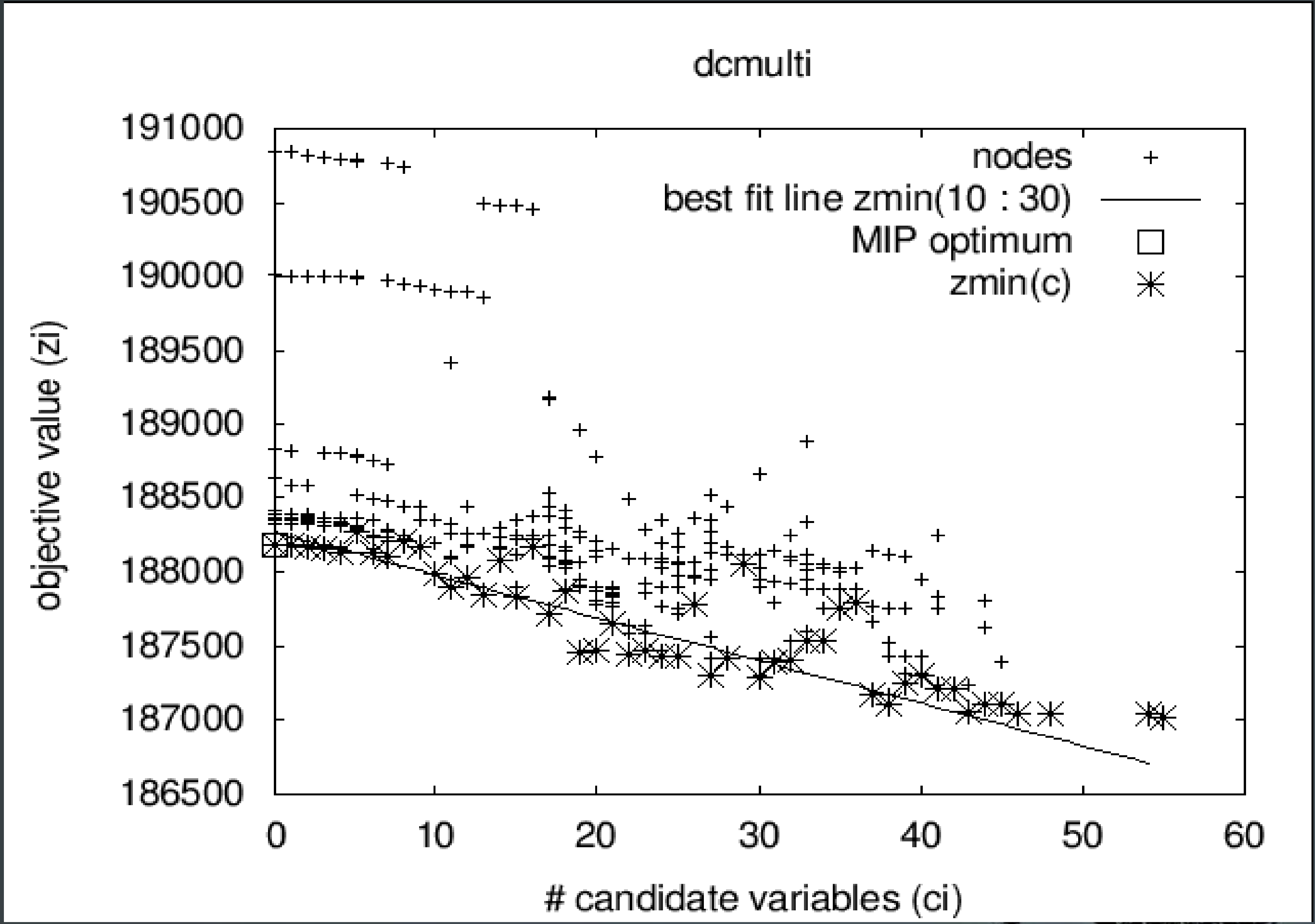
Usual best projection for node selection:

- $Z^a = Z^i + (Z^{inc} - Z^0)s^i/s^0$
  - s^i: sum of integer infeasibilities at node i
  - s^0: sum of integer infeasibilities at root node

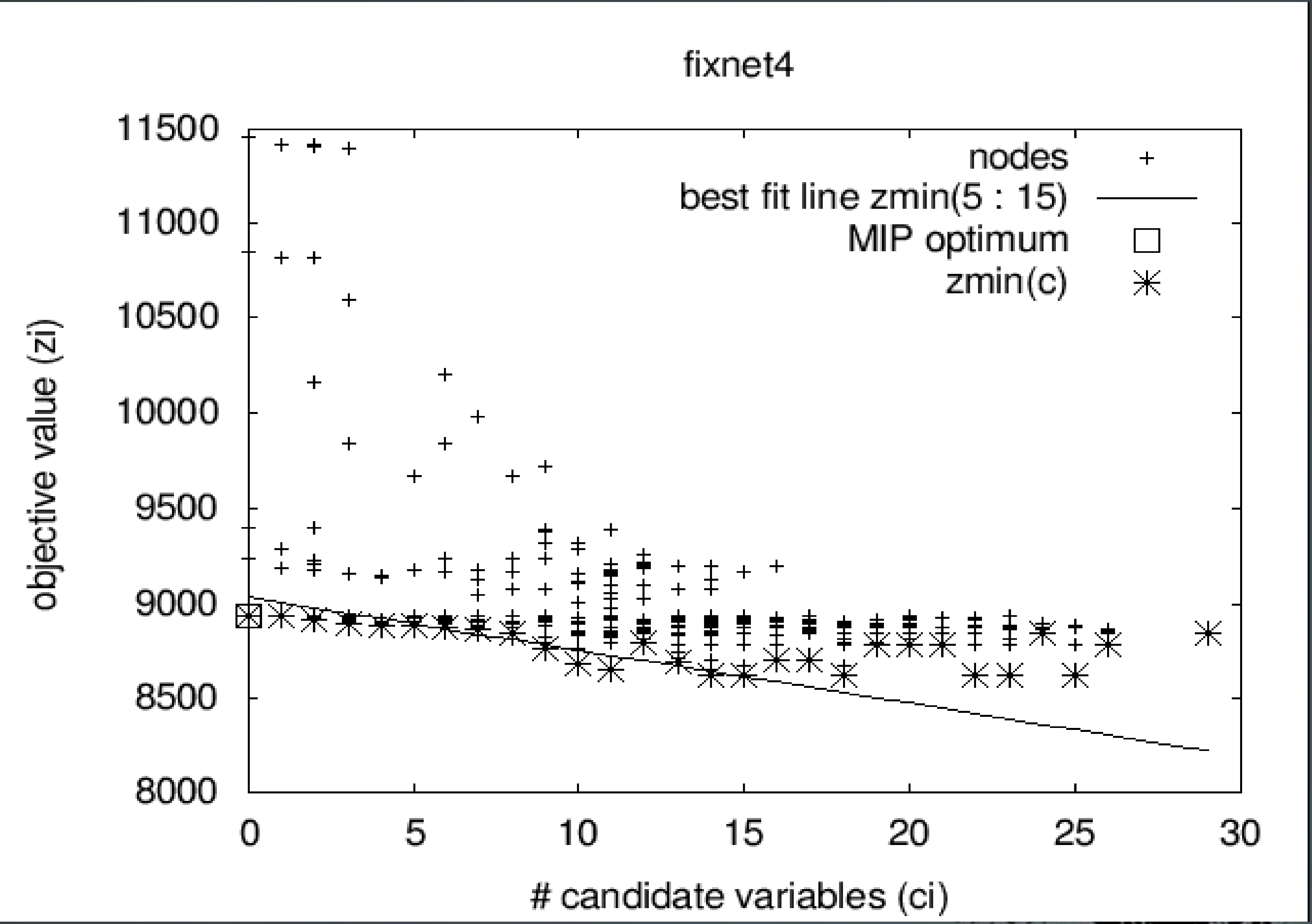
- Can we eliminate the need for an incumbent solution so this method can be applied at any node?

- *Is there a pattern?*

$Z^{min}(c)$: min Z at given C



Patterns in $Z^{min}(C)$



Modified Best Projection Aspiration

- $Z^a = Z^i + C^i[Z^{min}(C^{min})\text{-}Z^0]/(C^0\text{-}C^{min})$
  - C^i: number of candidate variables at node i
  - C^{min}: minimum number of candidate variables at any node

Notes:

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

2. Choosing Node when Backtracking

State of the Art:

- Choose node that is likely to have best objective function value:
  - Best-projection
  - Best-estimate (based on Pseudo-costs)
  - Best-bound
  - Depth-first backtrack to first active node
  - ...
- No method dominates

NEW: Distribution-based node selection

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

- C^i: number of candidate variables at node *i*
- Smaller Z^i and C^i both desirable
- Z^i tends to be *large* where C^i is *small*, and vice versa

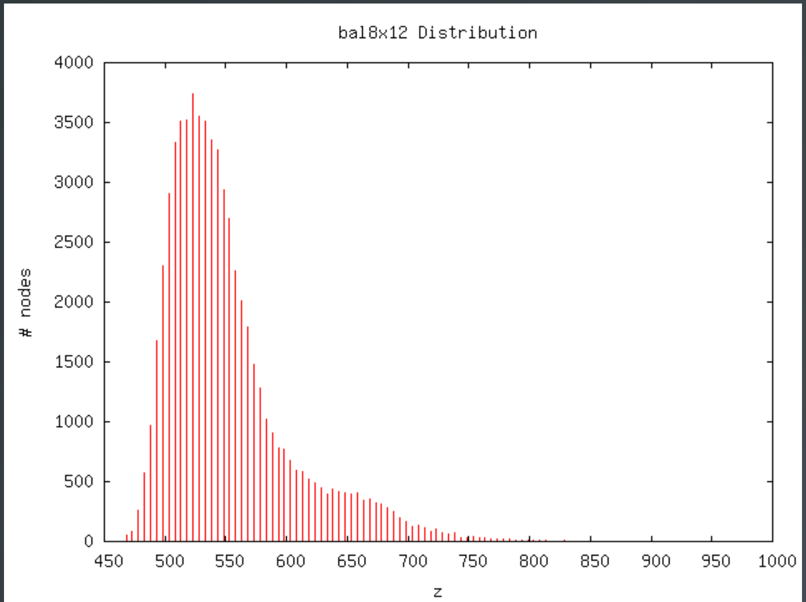
Ranges quite different: how to balance?

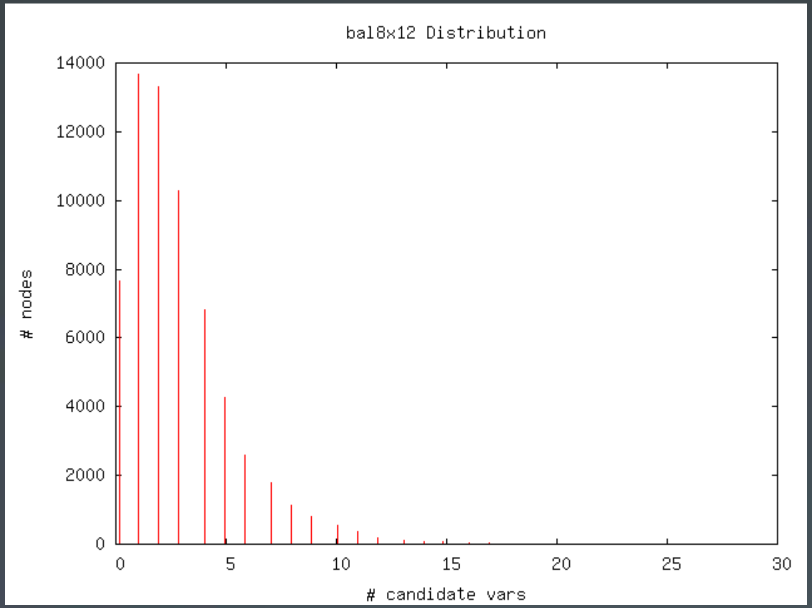
- Normalize ranges of Z^i and C^i assuming independent normal probability distributions
- Choose node *n* where $n = \arg\min_i P(Z \leq Z^i, C \leq C^i)$

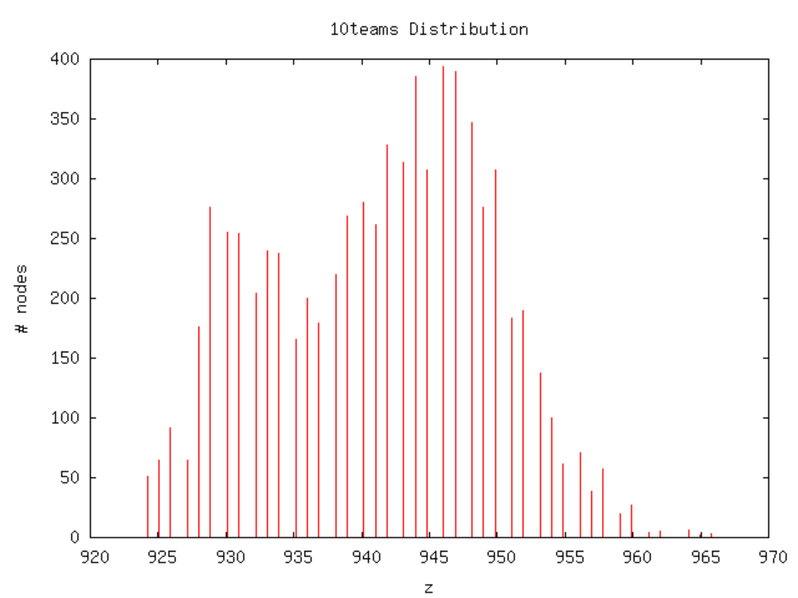
Notes on Distributions

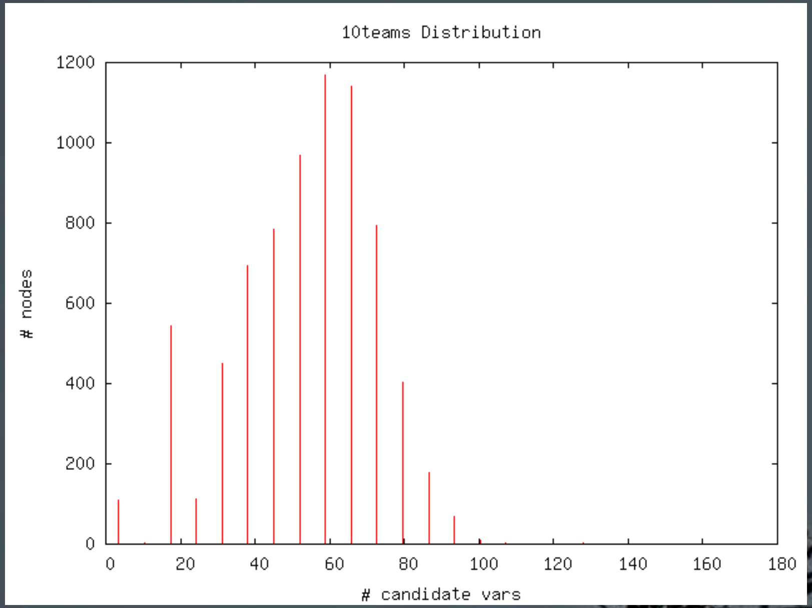
- Joint probability function of Z and C unknown.
- Single variable functions:
  - Assume Z and C are independent (*iffy!*)
  - $P(Z \leq Z^i, C \leq C^i) = F_Z(Z^i) \times F_C(C^i)$
- Functions tried in experiments:
  - Uniform
  - Rayleigh
  - ***Gaussian*** (best result)
    - Central Limit Theorem: sum of random varbs usually normal
    - Easy to update as nodes created

Example Distributions









Distribution Node Selection Algorithm

Distribution not helpful if:

- Standard deviation of Z is 0
- (Standard deviation of C) /depth is small, i.e. < 0.1 [empirical]

Algorithm:

- If standard deviation of Z or C too small then use default node selection method (best projection) and exit.
- For every active node *i:*
  - $F_{ZC}(Z^i,C^i) = F_Z(Z^i) * F_C(C^i)$
- Choose node *n* where $n = \arg\min_i F_{ZC}(Z^i,C^i)$

2.1 Active Node Search Threshold

Observations:

- Advanced node selection can take too much time
- Fewer iterations, fewer nodes, but more ***time***

- Node search time proportional to num. active nodes
- Too many active nodes?
  - Default to simple depth-first backtracking
- E.g.: mas76
  - Best-projection: 17,598 sec, ***3,186,117*** itns, ***1,177,063*** nodes
  - Depth-first: ***785*** sec, 5,691,683 itns, 2,165,073 nodes

Threshold

- R_t = (time for node selection)/(time for all else)
  - Cumulative time
- If $R_t > 0.1$, then switch to simple depth-first node selection
- Notes:
  - 0.1 is empirical
  - Fix-up if aspiration cut-off is being used

3. Experiments

Software:

- Solver: GLPK 4.9
- Branching variable selection: default
- Root node cuts: Gomory cuts

Hardware:

- CPU: Intel Core 2 6600 @ 2.4 GHz
- RAM: 4 GB
- OS: Linux 2.6.18

272 Test models:

- all instances from MIPLIB/MIPLIB2003
- all instances from CORAL
- exclude instances not solved within time limit by default GLPK

79 (legal) combinations of methods!

Backtracking node selection methods:

- Methods available in GLPK:
  - DEPF: Depth-first
  - BREF: Breadth-first
  - DEBP: Default best-projection
  - BESF: Best-First
- Methods added to GLPK:
  - BEES: Best-estimate
  - BFBE: BEES interleaved with BESF
- New methods
  - DIST: Distribution
  - MOBP: Modified Best-Projection

New Active node search threshold

- NOAN: No ANST (Default).
- ANST: Use ANST.

Backtrack triggering methods:

- Methods available in GLPK.
  - NONA: Non-aspiration backtracking: backtrack only from leaves (default).
- Methods added to GLPK:
  - ALLT: Perform backtracking node selection after every node solution.
  - DBPA: Default best-projection aspiration
  - PCAS: Pseudo-cost (best-estimate) aspiration
- New methods
  - LEXA: Linear feasibility depth extrapolation aspiration
  - MPAS: Modified best-projection aspiration

Prefiltering Experiment

- Try all 79 combinations of methods on a subset of faster-solving models
  - Select better methods for more extensive testing
- 79 Models:
  - those solved by default GLPK within 30 min
- Ranking is sum of:
  - Ranking by total time over all models (TR)
  - Ranking by ratio of geom. mean of avg ratio to best (RR)
  - Number of failed solutions (FAIL)

Best Methods

Rank	Configuration	FAIL	TR	RR
1	MOBP-MPAS-ANST	1	3	2
2	MOBP-MPAS-NOAN	1	4	3
3	MOBP-PCAS-ANST	1	1	7
4	DIST-ALLT-NOAN	2	7	1
5	DIST-MPAS-ANST	1	6	4
5	MOBP-PCAS-NOAN	1	2	8
7	MOBP-LEXA-ANST	2	8	6
39	DEBP-NONA-NOAN	0	36	44
64	BESF-NONA-NOAN	6	62	65
71	BREF-NONA-NOAN	6	71	71
76	DEPF-NONA-NOAN	7	76	76

Backtracking

MOBP: Modified Best Projection

DIST: Distribution

Triggering

MPAS: Modified Best Proj Asp

PCAS: Pseudocost Aspiration

ALLT: After Every Node

LEXA: Linear Extrapolation

ANST

ANST: Active Node Search Thresh.

NOAN: No ANST

Longer Experiments

- 7 top-ranked methods from prefiltering experiment
- Highest-ranked existing combination method
- GLPK default

- All 272 models
- One hour time limit
  - 9 weeks of computation

Overview of Results

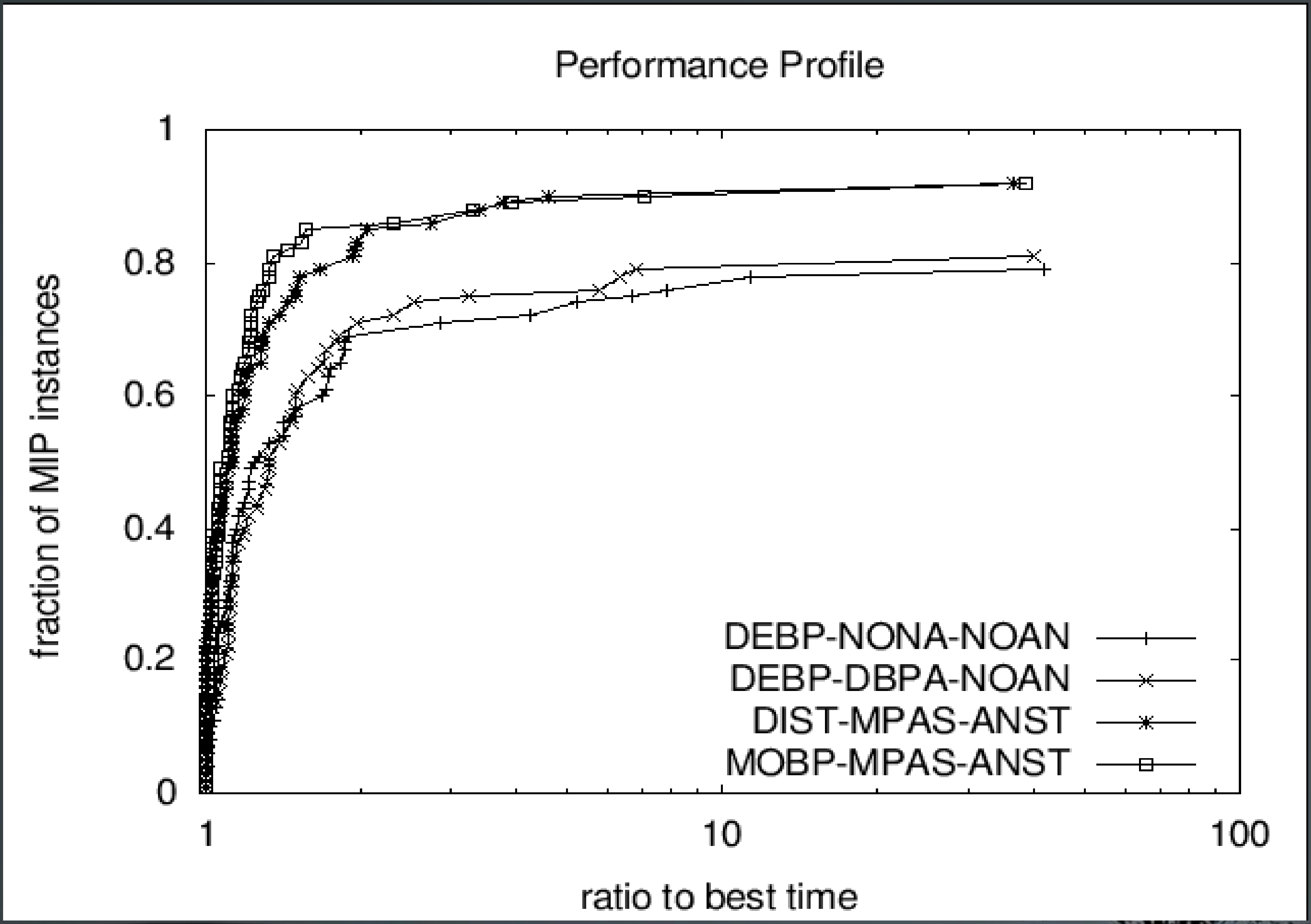
272 MIP instances total:

- 109 optimum found by at least 1 config
- 130 no optimum but at least one feasible soln found
- 33 no optimum and no feasible solutions found

Results

	AT LEAST 1 OPT			NO OPTIMUM		
Config	**Fail**	**TotTim**	**Mratio**	**Avgrank**	**Nfirsts**	**NINC**
MOBP-MPAS-ANST	6	86,654	1.33	4.54	8	75
DIST-MPAS-ANST	6	86,654	1.37	2.89	34	21
MOBP-PCAS-ANST	6	88,560	1.34	4.34	11	72
MOBP-LEXA-ANST	6	91,509	1.40	4.18	17	57
DIST-ALLT-NOAN	15	97,340	1.34	2.48	67	17
MOBP-MPAS-NOAN	13	99,799	1.43	4.62	7	81
MOBP-PCAS-NOAN	12	100,752	1.50	4.6	9	82
DEBP-DBPA-NOAN	14	109,240	1.75	3.74	21	15
DEBP-NONA-NOAN	15	109,579	1.78	3.47	25	15

Performance Profiles



Conclusions

- New methods very effective in speeding MIP solutions
- Best configurations:
  - MOBP-MPAS-ANST
  - DIST-MPAS-ANST
- Best configurations composed entirely of new methods

Outline

1. Introduction and Orientation (Lodi)

Part I: Achieving Integer-Feasibility Quickly

2. Classic Feasibility-Seeking Algorithms (Chinneck)
3. Active Constraint Variable Selection (Chinneck)
4. Branching to Force Change (Chinneck)
5. The Feasibility Pump (Lodi)

Part II: Reaching Optimality Quickly

6. New Node Selection Rules (Chinneck)
7. Local Branching and RINS (Lodi)

Part III: Analyzing Infeasible MIPs

8. Isolating Infeasible Subsystems (Chinneck)
9. Repairing MIP Infeasibility via Local Branching (Lodi)
10. Conclusions (Chinneck)

Analyzing Infeasible MIPs

John W. Chinneck
Systems and Computer Engineering, Carleton University, Ottawa, Canada

Outline

1. Analyzing Infeasible Math Programs
2. Extension to MIP
3. Special Methods for MIP
4. Empirical Tests
5. In Practice

1. Analyzing Infeasible Math Programs

General methods that also apply to MIPs

What is a "Constraint"?

Anything that restricts the solution space:

- A functional constraint: $3x_1 + 8x_2 \leq 12$
- A variable bound: $x_1 \geq 0$
- An integrality condition: x_1 is integer

Three Main Approaches

- Isolate an ***Irreducible Infeasible System (IIS)***
  - An infeasible set of constraints that becomes feasible if any constraint removed
  - *Main approach for MIPs*
- Find a ***Maximum Feasible Subset (Max FS)***
  - Maximum cardinality subset of constraints that is feasible
- Find “**best fix**” for infeasible constraints
  - Different matrix norms for measuring “best fix”

General Methods for Finding IISs

- Assume solver perfectly accurate in deciding feasibility status of a set of constraints
  - Reasonable assumption only for LP
- General methods for IIS isolation:
  - Deletion Filter
  - Additive Method
  - Elastic Filter
  - Additive/Deletion method

The Deletion Filter

INPUT: an infeasible set of constraints.

FOR each constraint in the set:

- Temporarily drop the constraint from the set.
- Test the feasibility of the reduced set:
- IF feasible THEN return dropped constraint to the set.
- ELSE (infeasible) drop the constraint permanently.

OUTPUT: constraints constituting a single IIS.

Deletion Filter: Example

IIS is {B,D,F} in {A,B,C,D,E,F,G}

- {B,C,D,E,F,G} infeasible. A deleted.
- {C,D,E,F,G} feasible. B reinstated.
- {B,D,E,F,G} infeasible. C deleted.
- {B,E,F,G} feasible. D reinstated.
- {B,D,F,G} infeasible. E deleted.
- {B,D,G} feasible. F reinstated.
- {B,D,F} infeasible. G deleted.

Output: the IIS {B,D,F}

Deletion Filter: Characteristics

- Returns *exactly one* IIS, even if there are multiple IISs in the model
- Which IIS?
  - IIS whose *first* member is *last* in the test list.
  - Consider {A,B,C,D,E,F,G,H,I,J,K}. IIS {G,I,K} found.
- Speed: isn't this slow?
  - *For LP:* time to isolate IIS usually a small fraction of time to find infeasibility initially
    - Due to advanced starts:
      each LP is very similar to the previous one
  - *For MIP and NLP:* slow

The Additive Method

Main insight:

- Add constraints one by one and test feasibility after each constraint is added.
- As soon as the tested set becomes infeasible, the last-added constraint ***must*** be part of an IIS

The Additive Method

C: ordered set of constraints in the infeasible model.
T: the current test set of constraints.
I: the set of IIS members identified so far.

INPUT: an infeasible set of constraints C.
Step 0: Set $T = I = \varnothing$.
Step 1: Set $T = I$.
FOR each constraint c_i in C:
Set $T = T \cup c_i$.
IF T infeasible THEN
Set $I = I \cup c_i$.
Go to Step 2.
Step 2: IF I feasible THEN go to Step 1.
OUTPUT: I is an IIS.

Additive Method: Example

IIS is {B,D,F} in {A,B,C,D,E,F,G}

- {A}, {A,B}, {A,B,C}, {A,B,C,D}, {A,B,C,D,E} all feasible.
- {A,B,C,D,E,F} infeasible: *I* = {F} is feasible.
- {F,A}, {F,A,B}, {F,A,B,C} all feasible.
- {F,A,B,C,D} infeasible: *I* = {F,D} is feasible.
- {F,D,A} feasible.
- {F,D,A,B} infeasible: *I* = {F,D,B} infeasible. Stop.

Output: the IIS {F,B,D}

Additive Method: Characteristics

- Returns *exactly one* IIS, even if there are multiple IISs in the model
- Which IIS?
  - IIS whose *last* member is *first* in the test list.
  - Consider {A,B,C,D,E,F,G,H,I,J,K}. IIS {B,E,J} found.
- Speed:
  - If IIS is small and early in the list of constraints, can use far fewer feasibility tests than deletion filter
  - *For LP:*
    speed similar to deletion filter due to basis re-use
  - *For MIP and NLP:* slow

Additive/Deletion Method

1. Apply additive method until first infeasible subset of constraints is found.
2. Apply deletion filter to subset.

- Consider {A,B,C,D,E,F,G,H,I,J,K}
  - Additive alone: 29 solutions
  - Additive/deletion: 19 solutions
  - Deletion alone: 11 solutions
- More efficient.

Dynamic reordering additive method

- If an intermediate test is feasible, scan all of the constraints past the current one and immediately add to *T* all those that are satisfied at the current test solution
  - Can avoid many model solutions

Speed-up: Grouping Constraints

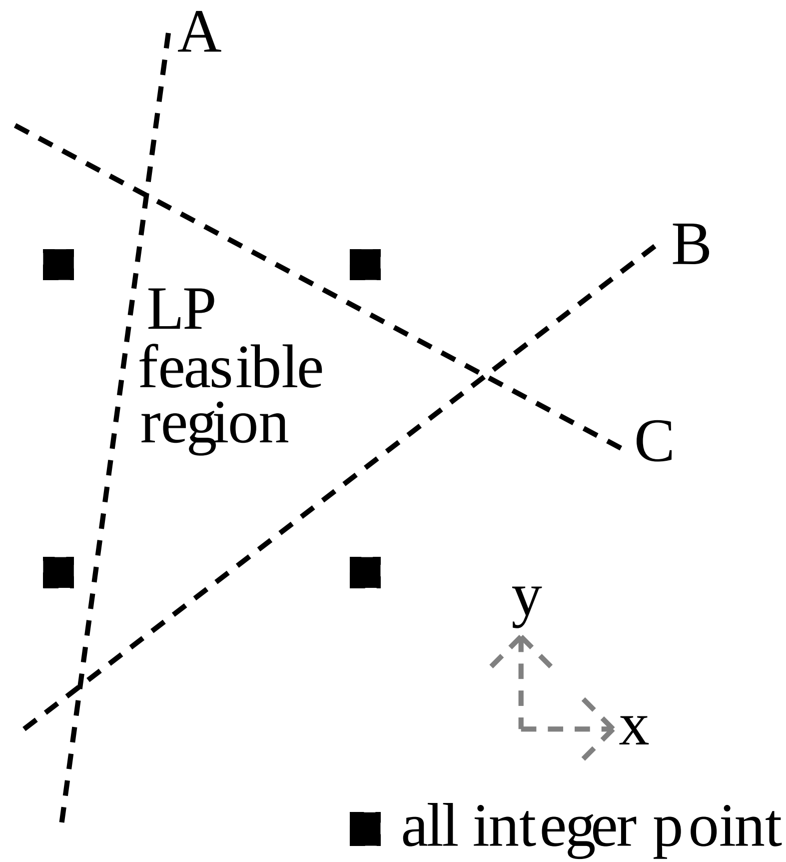
- Add/drop constraints in groups
  - In order, or by category
- *Deletion Filter:* back up and add singly if deleting a group causes feasibility
- *Additive Method*: back up and do singly if adding a group causes infeasibility

- Fixed group size? Adaptive group sizing?
- More recently: binary versions that split groups into halves in a combination of additive method and deletion filter

2. Extension to MIP

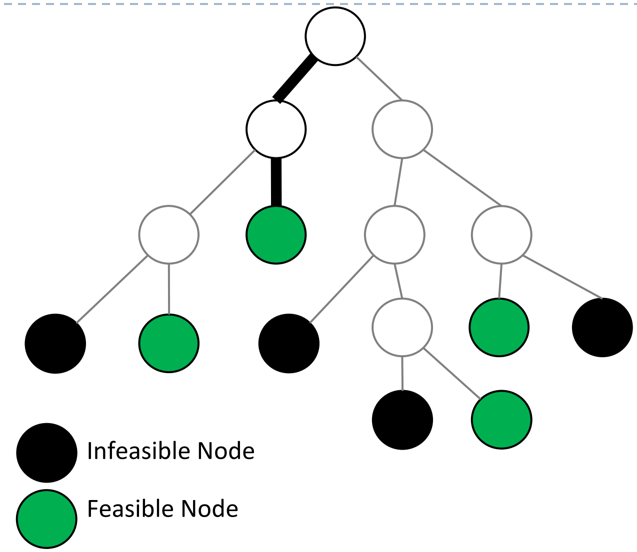
MIP Infeasibility

- Three classes of constraints:
  - Linear row constraints (LC)
  - Variable bounds (BD)
  - *Integer Restrictions (IR)*

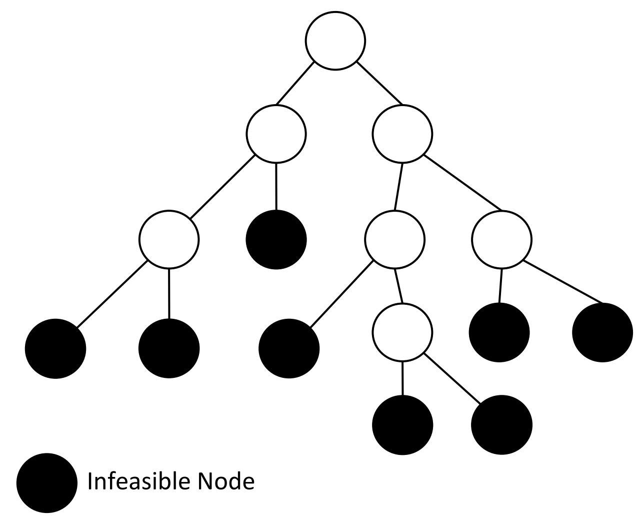


Proving (In)feasibility in MIPs

- Proving feasibility:
  - Find *any* feasible node in the search tree



- Proving infeasibility:
  - Expand entire tree until all leaves infeasible

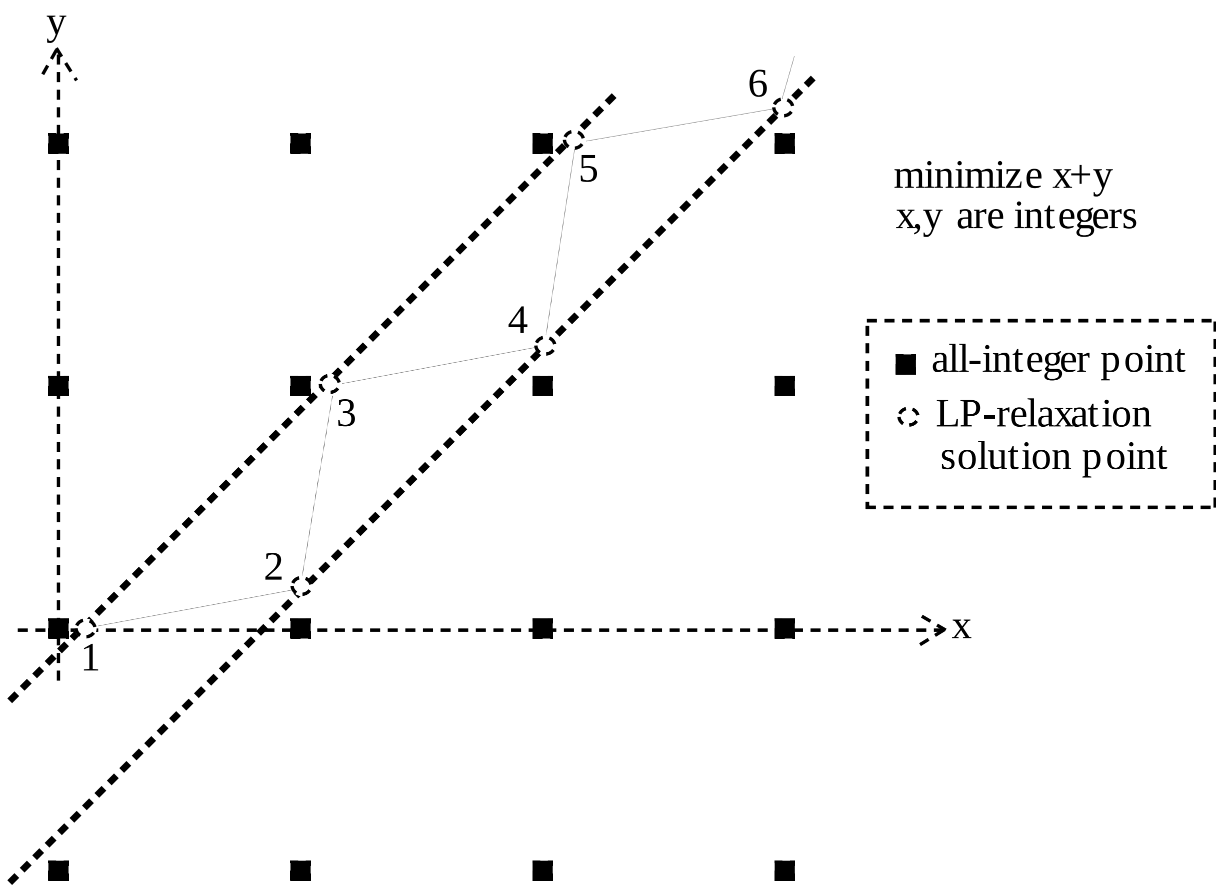


Analysis is Slow

- Methods rely on many MIP solutions, each having slightly different subsets of constraints
  - *Fast* for LP due to hot starts
  - *Slow* for MIPS

- Adjust methods to reduce the number of MIP solutions needed:
  - Also to deal with other nontermination

Difficulty: Nontermination in MIPs



Dealing with Non-termination

- If computation limit exceeded on subproblem:
  - Retain constraint and label it *dubious*
- Another approach:
  - Add *safety bounds* on variables
- May return *infeasible subsystem* (IS) instead of IIS if there are dubious constraints or safety bounds are active
- Non-termination can be frequent as constraints are removed by the IIS-finding algorithms

State of the Art in Infeasibility Analysis

- LP:
  - Very well developed theory
  - Fast in practice, implemented in most LP solvers
- MIP
  - Methods for general math programs adapted for MIPs
  - Relatively slow
- Infeasible MIPS:
  - Easy to analyze if caused by LP infeasibility
  - Interesting case is interaction with integer restrictions

Direct Application of Deletion Filter

- Nontermination may be frequent:
  - Preset computation limit for subproblems (max nodes in subproblem tree)
  - Constraint labelled *dubious*.
- Reducing incidence of nontermination:
  - Leave variable bounds in place as long as possible
  - Delete in this order: IR, LC, BD
- Slow but effective
  - Test in groups

Direct Application of Additive Method

- Assuming initial LP relaxation is feasible:
  - Start with LC, BD in test set T
  - Feed in the integer restrictions one by one
- Cannot directly identify dubious constraints
  - Test set always feasible or indeterminate, until infeasible
  - No indication of *which* constraint caused nontermination
    - In deletion filter, last constraint removed before nontermination can be labeled dubious
- *Dynamic reordering* can speed analysis

Additive/Deletion Method

- Can also be used
- Identifies dubious constraints during deletion filter

3. Special Methods for MIP

Useful Info in Original B&B Tree

- Recall:
  - intermediate nodes are LP-feasible, leaf nodes are LP-infeasible
- Theorem 1:
  - The IR set satisfied at any intermediate node cannot be the whole IR part of any IIS
- Theorem 2:
  - Mark LCs and BDs having nonzero shadow prices at *any* leaf node.
  - IR ∪ {marked LCs} ∪ {marked BDs} is infeasible
  - Can eliminate some LCs and BDs from consideration
- Theorem 3:
  - *Active IRs* are those actually used in the B&B tree branching
  - {active IRs}∪LC ∪BD is infeasible
  - Can eliminate some IRs from consideration

Using Info in Original Tree

- Eliminate LCs and BDs that are not sensitive in any leaf (Thm 2)
- Eliminate IRs not in active set (Thm 3)

- *Path set:* set of IRs used in branching on a root-to-leaf path.
  - Path sets good candidates for the IR set in an IIS
  - Use Thm 1 to eliminate candidate paths

Speed-ups

- Use alternative objective function (especially one which determines infeasibility faster)
  - e.g. Elastic varbs on constraints introduced during branching
  - Minimize sum of slacks
- Other MIP solver settings
  - Branch on most infeasible variable?
  - Depth-first vs. other node selection schemes?

4. Empirical Tests

Empirical Tests

- 20 infeasible test problems (hard to find!)
  - Avg. 238 LCs, 518 BDs, 80 IRs
- Software: Cplex 3.0 (!) and MINTO
  - Max 10,000 nodes in any subproblem
- Avg. initial solution:
  - 437 B&B tree nodes
  - 1719 LP iterations
  - Soln time: 6 secs

Results

Averages over 20 models.
Sun 10/30c computer, 36 MHz SPARC Sun 4 CPU, 33 Mbytes of memory

method	IISs (term)	dubious IR-LC-BD	IRs	LCs	BDs	nodes	LP itns	time
LC-IR-BD deletion	**5 (0)**	0-17-182	16	**132**	**290**	499,154	3,401,931	9:12:46
IR-LC-BD deletion	**5 (0)**	0-16-185	12	154	321	344,797	1,913,248	2:27:44
IR-LC-BD del, grp 4	**5 (0)**	0-16-186	12	153	311	189,561	**1,246,078**	**1:51:31**
Dyn. reorder add/del	3 (0)	**0-0-8**	**8**	135	309	124,512	1,487,991	2:25:21
Additive	4 (3)	NA	9	50	142	172,688	982,255	1:12:12
Dyn. reorder additive	4 (3)	NA	8	40	145	130,068	396,176	19:41
Initial tree w del filter (10 models)	1 (0)	0-168-475	6	209	520	61,330	Not recorded	0:58:43

Conclusions (1)

- Slow
  - Best method avg 1:51:31 vs. 6 sec for initial detection of infeasibility
- Effective. Best method eliminates:
  - 90% of IRs,
  - 43% of LCs
  - 40% of BDs)
- BUT: machine time is cheap, people time is expensive!

Conclusions (2)

- Best method (IIS fairly often, smallest IISs, fewest dubious constraints):
  - dynamic reordering additive/deletion method
- Fastest method:
  - info from original B&B tree
  - IR-LC-BD deletion filter
  - constraint grouping (fixed size)

Reference

O. Guieu and John W. Chinneck

Analyzing Infeasible Mixed-Integer and Integer Linear Programs

INFORMS Journal on Computing 11, pp. 63-77, 1999

5. In Practice

Cplex "Conflict Refiner" for MIPs

- May add/delete constraints in groups
- Can specify preferences for inclusion/exclusion of constraints or groups in the IIS
- Heuristics to try to find an infeasible subset more quickly, then apply detailed analysis

LINDO

- Similar capabilities in LINDO/LINGO

Outline

1. Introduction and Orientation (Lodi)

Part I: Achieving Integer-Feasibility Quickly

2. Classic Feasibility-Seeking Algorithms (Chinneck)
3. Active Constraint Variable Selection (Chinneck)
4. Branching to Force Change (Chinneck)
5. The Feasibility Pump (Lodi)

Part II: Reaching Optimality Quickly

6. New Node Selection Rules (Chinneck)
7. Local Branching and RINS (Lodi)

Part III: Analyzing Infeasible MIPs

8. Isolating Infeasible Subsystems (Chinneck)
9. Repairing MIP Infeasibility via Local Branching (Lodi)
10. Conclusions (Chinneck)

Conclusions

John W. Chinneck and Andrea Lodi

Conclusions

- Branch-and-bound/cut framework permits a very wide variety of heuristics
  - New developments mainly in heuristics
  - Heuristics can interact in unpredictable ways
- MIP heuristics are an active area of research
- Significant progress in recent years
  - Feasibility-seeking especially
- New commercial players
  - Microsoft: solver foundation
  - Gurobi

Future Research Directions

- Taking advantage of multiple cores
- Choosing the best heuristics dynamically
- General disjunctions