The Latest Advances in Mixed-Integer Programming Solvers

Robert E. Bixby
Gurobi Optimization & Rice University
Ed Rothberg, Zonghao Gu
Gurobi Optimization

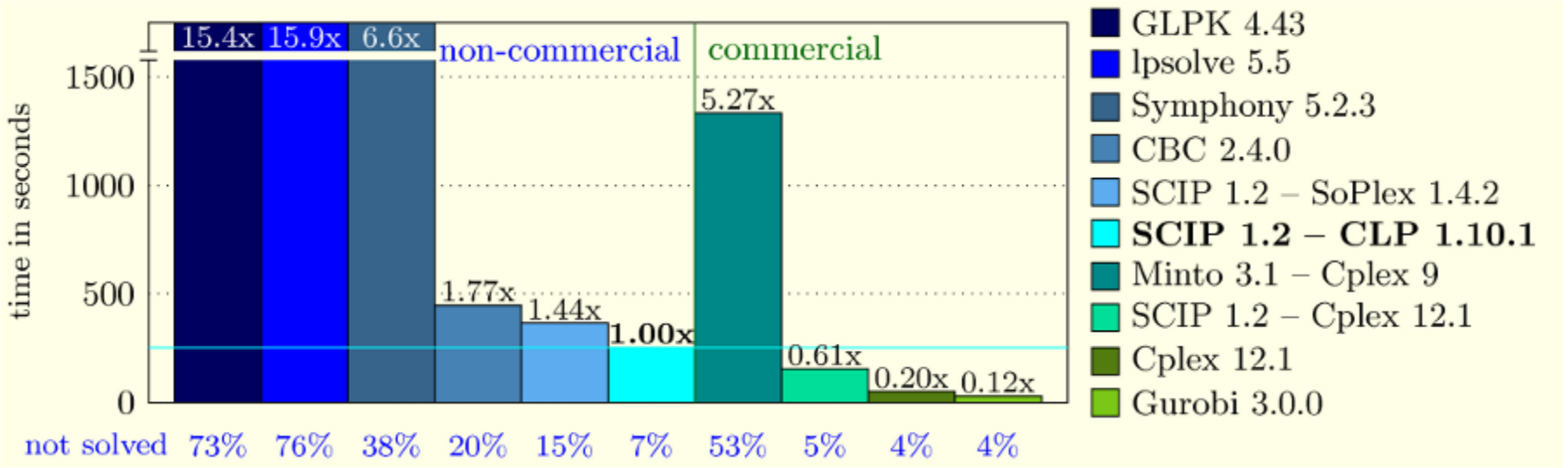
Overview

- Two-decades of Progress: 1988-2008
  - Linear Programming
  - Mixed-Integer Programming
- Recent developments: 2008-Present
  - Redesigning MIP
    - Tree of Trees
    - Parallel MIP
  - The bag of tricks
    - Branching
    - Heuristics
    - Cutting planes
    - An example reduction – Disjoint subtrees
    - Domination
      - Symmetry testing
- Where are we now
  - Performance summary

Acknowledgement

- I will focus on two solvers
  - CPLEX
    - 1988 – 2008
  - Gurobi
    - 2009 – Present
- There are other very good solvers
  - XPRESS
  - SCIP
    - Alexander Martin, Tobias Achterberg, ZIB

Publicly Available Table
http://scip.zib.de/



Two Decades of Progress

Linear Programming

The Foundation

Progress in LP: 1988—2004

(Operations Research, Jan 2002, pp. 3—15, updated in 2004)

- Algorithms (*machine independent*):

  Primal *versus* best of Primal/Dual/Barrier 3,300x

- Machines (workstations →PCs): 1,600x
- NET: Algorithm × Machine 5,300,000x

(2 months/5300000 ~= 1 second)

Progress in LP

- Algorithm comparison – 2004 (CPLEX)
  - Dual simplex vs. primal: Dual 2.70x faster
  - Dual simplex vs. barrier: Dual 1.06x faster
- Algorithm comparison – Today (Gurobi)
  - Modest progress in Dual since 2004 (1.4x)
  - In practice LP is viewed as a solved problem
  - But, …. a word of warning
    - 2% of MIP models are blocked by linear programming

Mixed-Integer Programming Part I

Two Decades of Progress:

1988-2008

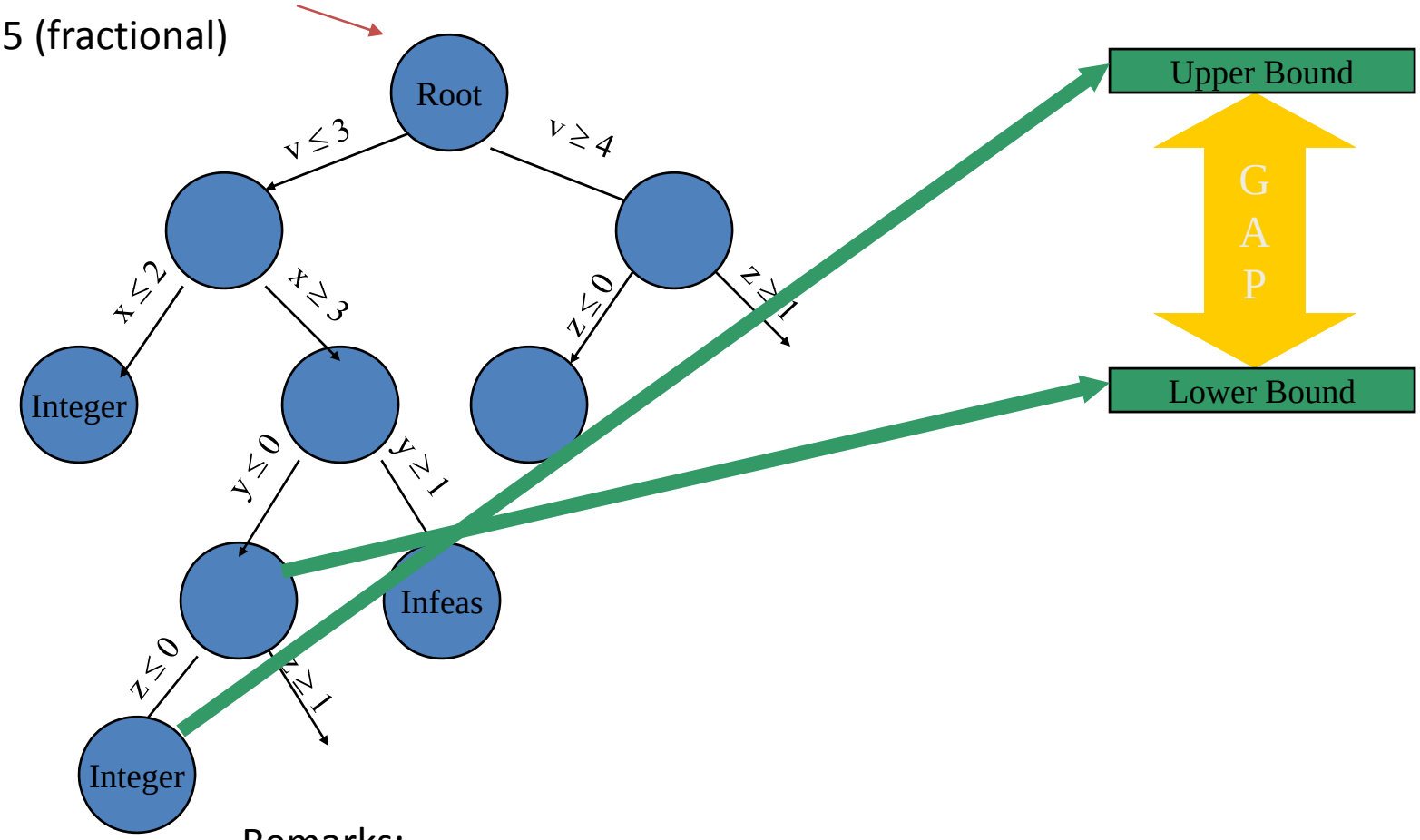
A Definition

A *mixed-integer program* (MIP) is an optimization problem of the form

$$
\begin{aligned}
&\textit{Minimize} && c^T x \\
&\textit{Subject to} && Ax = b \\
& && l \le x \le u \\
&\text{some or all } x_j \text{ integer}
\end{aligned}
$$

MIP solution framework:
LP based Branch-and-Bound

Solve LP relaxation:
v=3.5 (fractional)



Remarks:

(1) GAP = 0 $\Rightarrow$ Proof of optimality

(2) In practice: Often good enough to have good Solution

Electric Power Generation:

Unit Commitment, Power Dispatch

A Historical Comment

Electrical Power Industry, ERPI GS-6401, June 1989:
Mixed-integer programming (MIP) is a powerful modeling tool, "They are, however, theoretically complicated and computationally cumbersome"

In Other Words: MIP is an interesting "toy", but it just isn't going to work it practice.

And Example Unit-Commitment Model

California 7-Day Model

`UNITCAL_7:` 48939 constraints, 25755 variables (2856 binary)

Reported Results 1999 – machine unknown

2 Day model: 8 hours, no progress

7 Day model: 1 hour to solve initial LP

Desktop PC -- ran full 7-day model

CPLEX 6.5 (1999): 22 minutes, optimal

California 7-Day Model

```
Gurobi Optimizer version 3.0.0

Read MPS format model from file unitcal_7.mps.bz2
Optimize a model with 48939 Rows, 25755 Columns and 127595 NonZeros
Presolved: 38804 Rows, 19960 Columns, 105627 Nonzeros

Root relaxation: objective 1.945018e+07, 18340 iterations, 0.60 seconds

    Nodes    |    Current Node    |     Objective Bounds      |     Work
 Expl Unexpl |  Obj  Depth IntInf | Incumbent    BestBd   Gap | It/Node Time

     0     0 1.9450e+07    0  721          - 1.9450e+07     -      -    2s
     0     0 1.9596e+07    0  559          - 1.9596e+07     -      -   16s
     0     0 1.9598e+07    0  487          - 1.9598e+07     -      -   20s
H    0     0                     2.066856e+07 1.9598e+07  5.18%     -   25s
    13    11 1.9669e+07    6  217 2.0669e+07 1.9602e+07  5.16%   649   30s
    36    28 1.9668e+07    9  219 2.0669e+07 1.9605e+07  5.15%   707   35s
H   93    84                     1.998399e+07 1.9605e+07  1.90%   342   37s
   100    74 1.9678e+07   17  204 1.9984e+07 1.9606e+07  1.89%   321   43s
*  271   111              54    1.972042e+07 1.9606e+07  0.58%   170   43s
H  417    29                     1.964604e+07 1.9606e+07  0.21%   129   43s
   858   178 1.9629e+07   11  141 1.9637e+07 1.9609e+07  0.14%  96.9   50s
H  924   187                     1.963578e+07 1.9609e+07  0.13%  94.6   51s
H  987   221                     1.963563e+07 1.9611e+07  0.12%  93.9   53s
  1024   237 1.9630e+07   19  107 1.9636e+07 1.9611e+07  0.12%  93.5   56s
H 1034   237                     1.963556e+07 1.9611e+07  0.12%  92.8   56s
  1144   288 1.9630e+07   14  501 1.9636e+07 1.9611e+07  0.12%  89.0   63s
  1147   290 1.9617e+07   13  595 1.9636e+07 1.9611e+07  0.12%  88.8   71s
  1153   294 1.9626e+07   17  491 1.9636e+07 1.9611e+07  0.12%  88.3   80s
  1163   303 1.9611e+07   16  547 1.9636e+07 1.9611e+07  0.12%   120  156s
  1170   303 1.9624e+07   20  488 1.9636e+07 1.9611e+07  0.12%   123  166s
  1245   294 1.9621e+07   30  399 1.9636e+07 1.9619e+07  0.09%   131  185s
  1673   261 1.9623e+07   35  120 1.9636e+07 1.9623e+07  0.07%   117  190s

Cutting planes:
  Gomory: 20
  Cover: 31
  Implied bound: 553
  Clique: 61
  MIR: 71
  Flow cover: 416

Explored 2167 nodes (274011 simplex iterations) in 194.37 seconds
Optimal solution found (tolerance 1.00e-04)
```

Computational History: 1950 –1998

- **1954 Dantzig, Fulkerson, S. Johnson:** 42 city TSP
  - Solved to optimality using LP and cutting planes
- **1957 Gomory**
  - Cutting plane algorithms
- **1960 Land, Doig, 1965 Dakin**
  - B&B
- **1971 MPSX/370**
- **1972 UMPIRE**
  - LP-based B&B
  - MIP became commercially viable

- **1972 – 1998** Good B&B remained the state-of-the-art in commercial codes, in spite of ….
  - Edmonds, polyhedral combinatorics
  - 1973 Padberg, cutting planes
  - 1973 Chvátal, revisited Gomory
  - 1974 Balas, disjunctive programming
  - 1983 Crowder, Johnson, Padberg: PIPX, pure 0/1 MIP
  - 1987 Van Roy and Wolsey: MPSARX, mixed 0/1 MIP
  - TSP, Grötschel, Padberg, …

1998 … A New Generation of MIP Codes

- Linear programming
  - Stable, robust dual simplex
- Variable/node selection
  - Influenced by traveling salesman problem
- Primal heuristics
  - 12 different tried at root
  - Retried based upon success
- Node presolve
  - Fast, incremental bound strengthening (very similar to Constraint Programming)
- Presolve – numerous small ideas
  - Probing in constraints:

    $\sum x_j \leq (\sum u_j)\, y,\ y = 0/1$

    ➔ $x_j \leq u_j y$ (for all j)
- Cutting planes
  - Gomory, mixed-integer rounding (MIR), knapsack covers, flow covers, cliques, GUB covers, implied bounds, zero-half cuts, path cuts

Mining the Theoretical Backlog

CPLEX 6.5

Progress: MIP

Which Single Feature Helps Most?

(After CPLEX 6.5 < 1000 seconds, Before CPLEX 6.5 unsolvable)

- Cuts 53.7x
- Presolve 10.8x
- Variable selection 2.9x
- No heuristics 1.4x
- No node presolve 1.3x

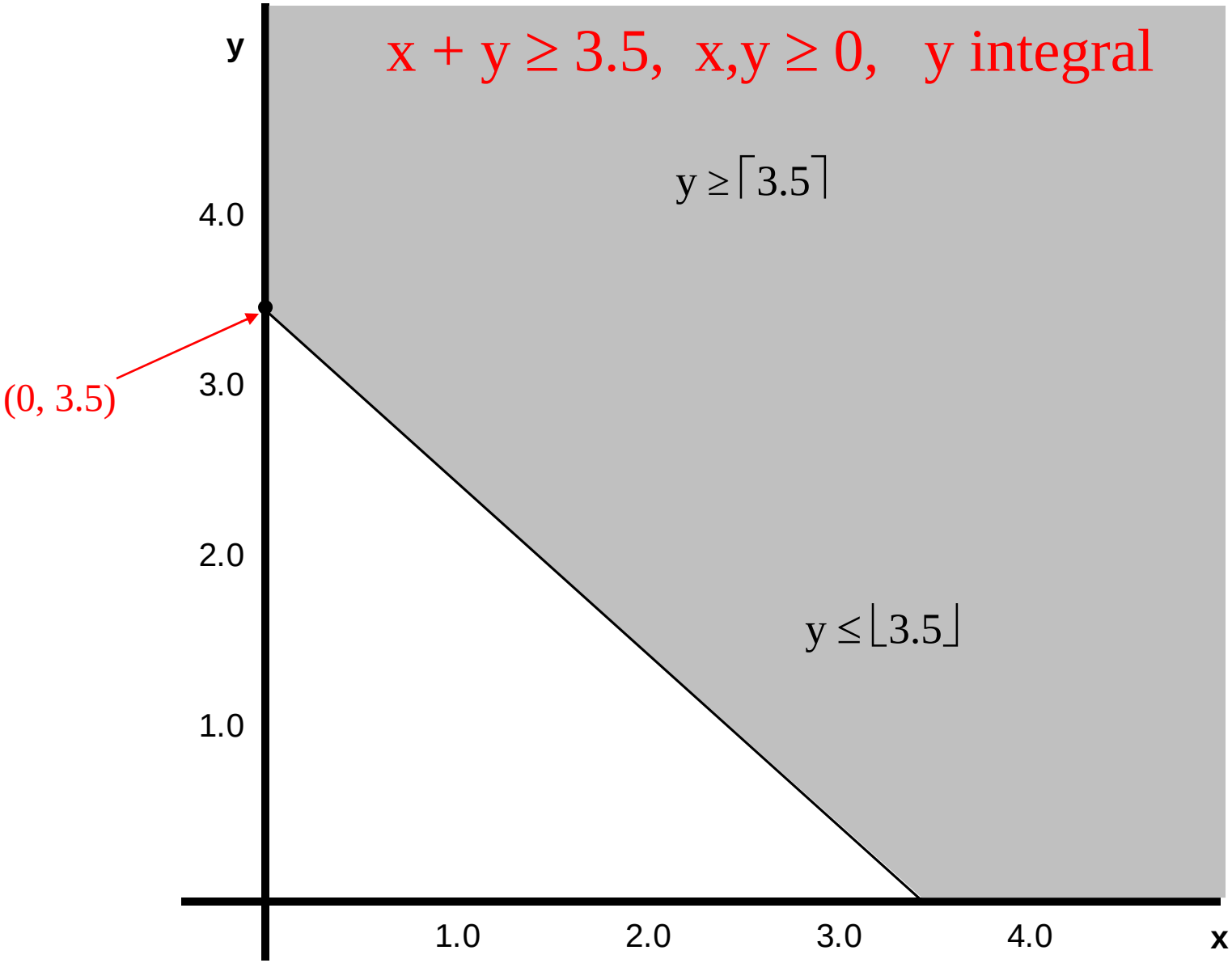
Progress: MIP

Removing Single Cuts

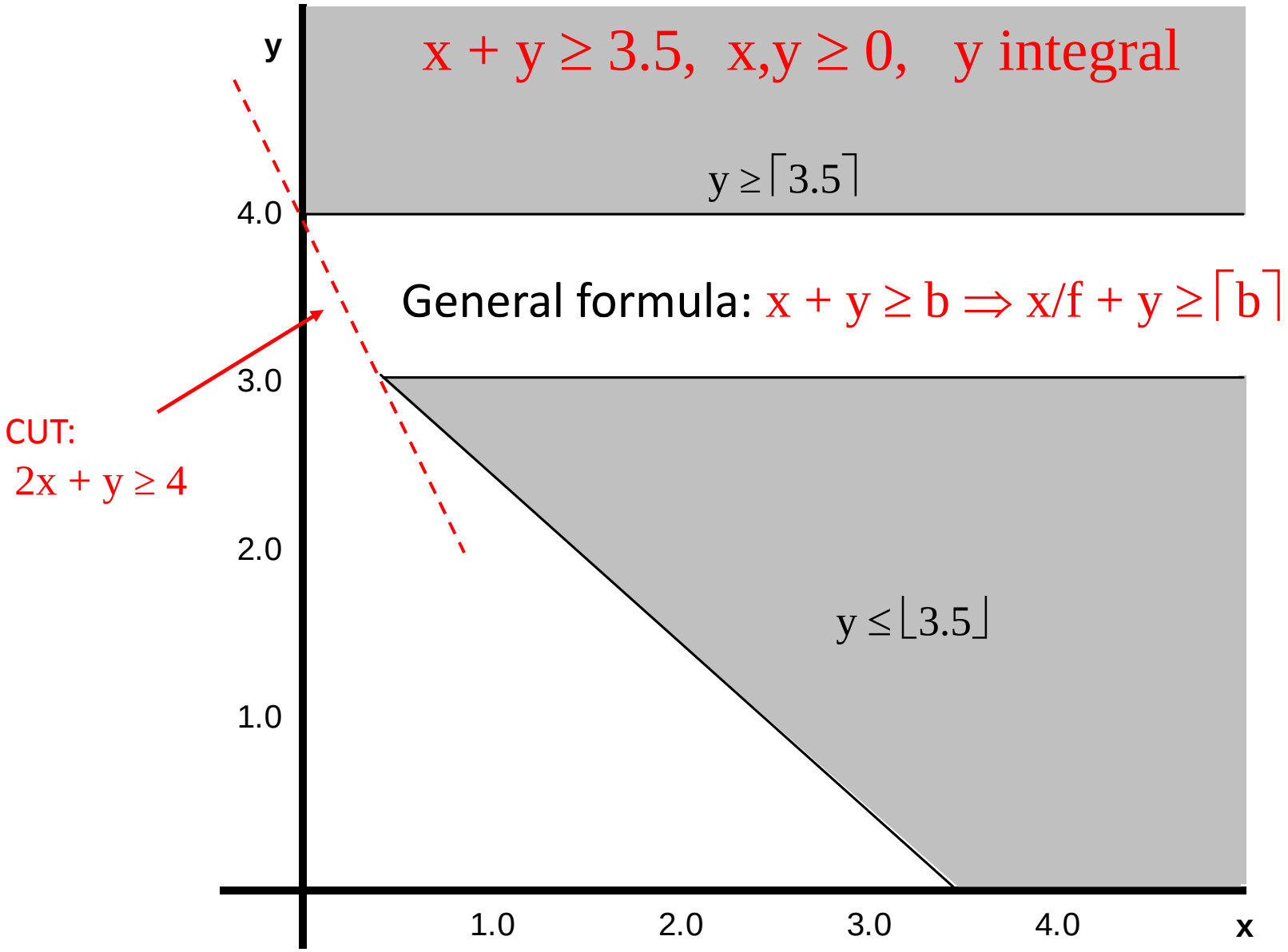
• Gomory mixed-integer	2.52x
• Mixed-integer rounding	1.83x
• Knapsack cover	1.40x
• Flow cover	1.22x
• Implied bound	1.19x
• Path	1.04x
• Clique	1.02x
• GUB cover	1.02x

The BEST of the cuts:
MIR & Gomory Mixed Cuts

I. Mixed-Integer Rounding Cut



I. Mixed-Integer Rounding Cut



II. Gomory Mixed Cut

- Given $y, x_j \in Z_+$, and
  $y + \sum a_{ij}x_j = d = \lfloor d \rfloor + f,\ f > 0$
- Rounding: Where $a_{ij} = \lfloor a_{ij} \rfloor + f_j$, define
  $t = y + \sum(\lfloor a_{ij} \rfloor x_j : f_j \le f) + \sum(\lceil a_{ij} \rceil x_j : f_j > f) \in Z$
- Then
  $\sum(f_j\, x_j : f_j \le f) + \sum(f_j - 1)x_j : f_j > f) = d - t$
- Disjunction:
  $t \le \lfloor d \rfloor \Rightarrow \sum(f_j x_j : f_j \le f) \ge f$
  $t \ge \lceil d \rceil \Rightarrow \sum((1-f_j)x_j : f_j > f) \ge 1-f$
- Combining:
  $\sum((f_j/f)x_j : f_j \le f) + \sum([(1-f_j)/(1-f)]x_j : f_j > f) \ge 1$

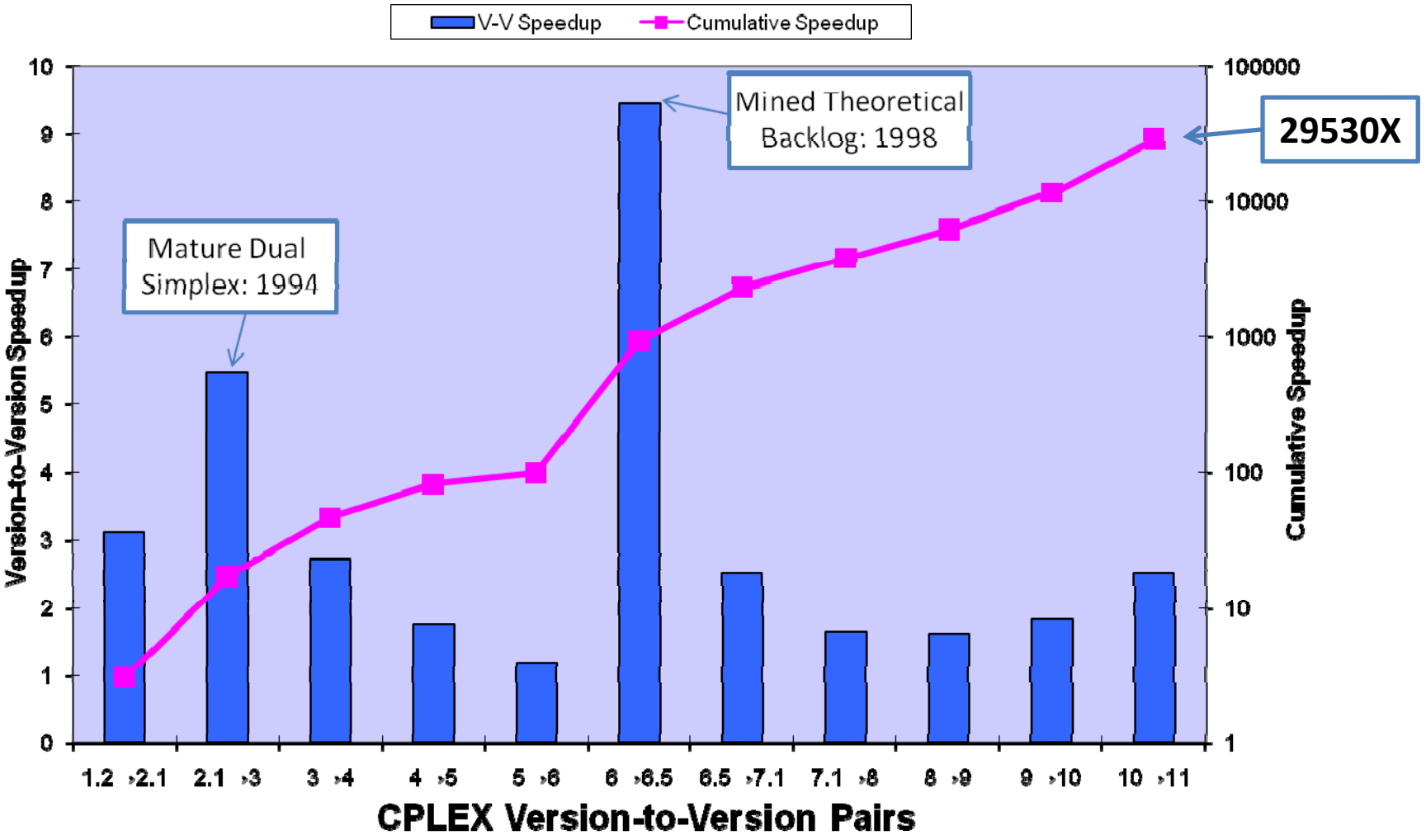
Computing Gomory Mixed Cuts

1. Make a an ordered list of fractional variables based upon Driebeek penalties.
2. Take the first 100. Compute corresponding tableau rows. Reject if coefficient range too big.
3. Add to LP.
4. Repeat twice.
5. Computed only at root. Slack cuts purged at end of root computation.

Some Test Results

- Test set: **1852 real-world MIPs**
  - Full library
    - 2791 MIPs
  - Removed:
    - 559 "Easy" MIPs
    - 348 "Duplicates"
    - 22 "Hard" LPs (0.8%)
- Parameter settings
  - Pure defaults
  - 30000 second time limit
- Versions Run
  - CPLEX 1.2 (1991) -- CPLEX 11.0 (2007)

CPLEX MIP Performance Improvements



Recent Developments

Mixed-Integer Programming

Part II

Gurobi Optimization

- Gurobi Optimization, Inc.
  - Founders: **Gu**, **Ro**thberg, **Bi**xby
  - Began Code development March 2008
- The Gurobi Optimizer:
  - LP simplex and Deterministic Parallel MIP
  - Version 1.0 – May 2009
  - Version 2.0 – October 2009
  - Version 3.0 – April 2010

Redesigning the MIP Solver

Key Ingredients

- For an effective MIP solver, you need …
  - Heuristics
    - Explore solutions near the relaxation quickly
    - Find feasible solutions at nodes other than leaf nodes
  - Parallelism
  - Presolve
    - Tighten formulation before starting branch & bound
    - Once you start branching, mistakes replicate
  - Branch-variable selection
  - Cutting planes
  - LP dual-simplex solver

A Fresh Look

A Fresh Look

- Start from a clean slate
  - With the benefit of 20+ years of experience
- Things have changed:
  - Two examples:
    - "Sub-MIP" as a pervasive approach
    - Ubiquitous parallel processing

Sub-MIP As A Paradigm

- Key recent insight for heuristics:
  - Can use MIP solver recursively as a heuristic
  - Solve a related model:
    - Hopefully smaller and simpler
  - Examples:
    - Local cuts [Applegate, Bixby, Chvatal & Cook, 2001]
    - Local branching [Fischetti & Lodi, 2003]
    - RINS [Danna, Rothberg, Le Pape, 2005]
    - Solution polishing [Rothberg, 2007]

RINS

- Relaxation Induced Neighborhood Search
  - Given two “solutions”:
    - x^*: any integer feasible solution (not optimal)
    - x^R: optimal relaxation solution (not integer feasible)
  - Fix variables that agree
  - Solve the result as a MIP
    - Possibly requiring early termination
- Extremely effective heuristic
  - Often finds solutions that no other technique finds

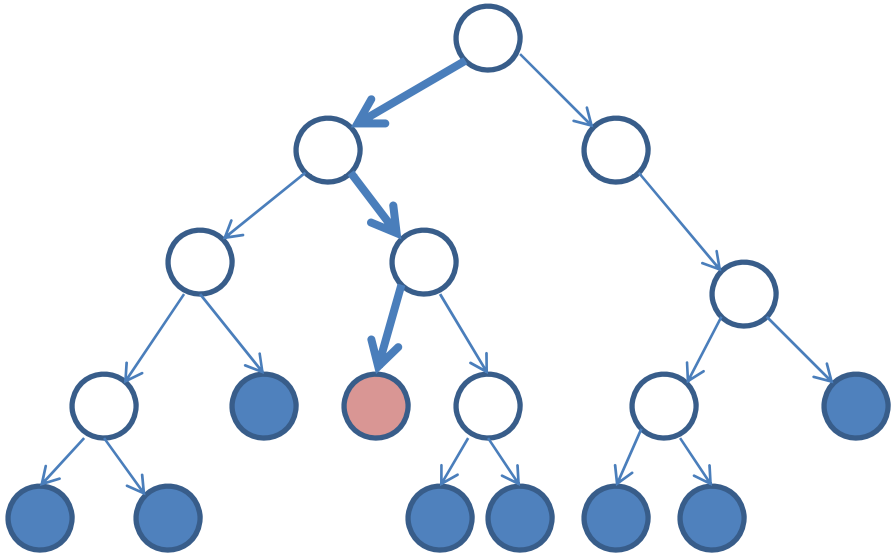
Why Is RINS So Effective?

- MIP models often involve a hierarchy of decisions
  - Some much more important than others
- Fixing variables doesn't just make the problem smaller
  - Often changes the nature of the problem
    - Extreme case:
      - Problem decomposes into multiple, simple problems
    - More general case:
      - Resolving few key decisions can have a dramatic effect
  - Strategies that worked well for the whole problem may not work well for RINS sub-MIP
    - More effective to treat it as a brand new MIP

Rethinking MIP Tree Search

Branch-and-Bound

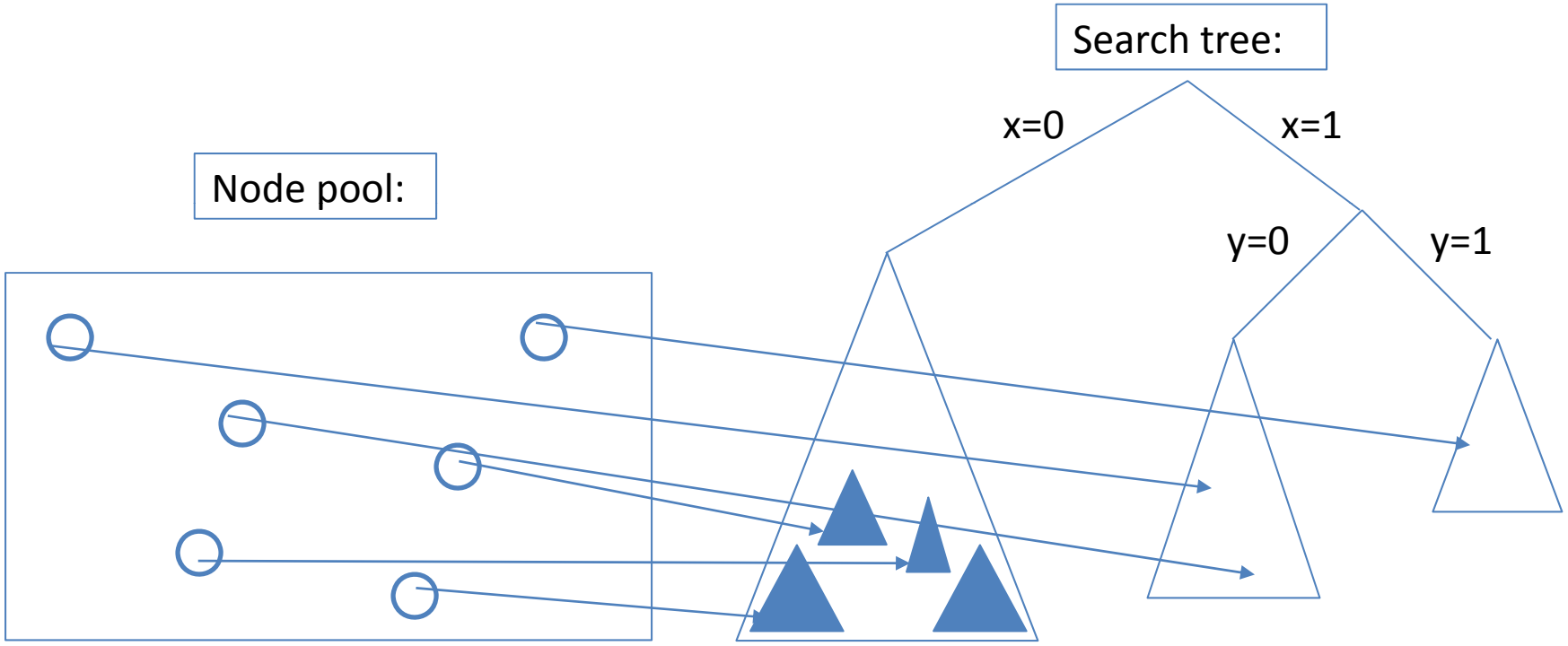
- Each node in branch-and-bound is a new MIP
  - Original model, plus several variable fixings
  - Can view search tree as a *tree-of-trees*
  - As in RINS, nature of sub-MIP can change dramatically



Tree of Trees

- Gurobi MIP search tree manager built to handle multiple related trees
  - Can transform any node into the root node of a new tree
- Maintains a pool of nodes from all trees
  - No need to dedicate the search to a single sub-tree

Tree of trees



Tree of Trees

- Each tree has its own relaxation and its own strategies...
  - Presolved model for each subtree
  - Cuts specific to that subtree
  - Pseudo-costs for that subtree only
  - Symmetry detection on that submodel
  - Etc.
- Captures structure that is often not visible in the original model

Parallel MIP

Why Parallel?

- Microprocessor trends have changed
- Transistors are:
  - Still getting smaller
  - But not faster
- Implications:
  - New math for CPUs: more transistors = more cores
  - *Sequential software won't be getting significantly faster in the foreseeable future*
- Gurobi MIP solver built for parallel from the ground up
  - Sequential is just a special case

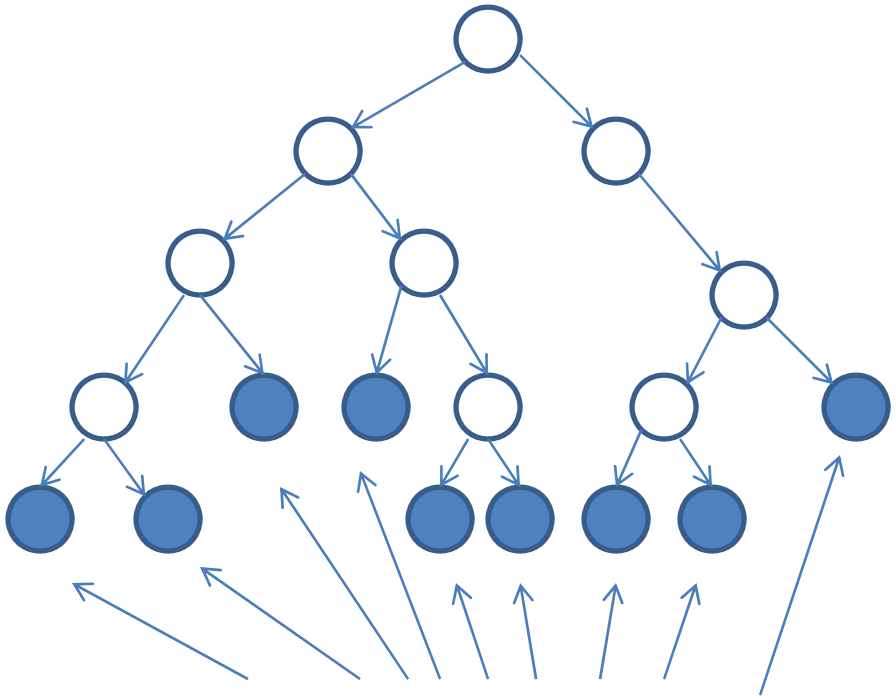
Need Deterministic Behavior

- Non-deterministic parallel behavior:
  - Multiple runs with the same inputs can give different results
- "Insanity: doing the same thing over and over again and expecting different results."
  - Albert Einstein
- Conclusion: non-deterministic parallel behavior will drive you insane

Building Blocks

Building Blocks

- Parallel MIP is parallel branch-and-bound:



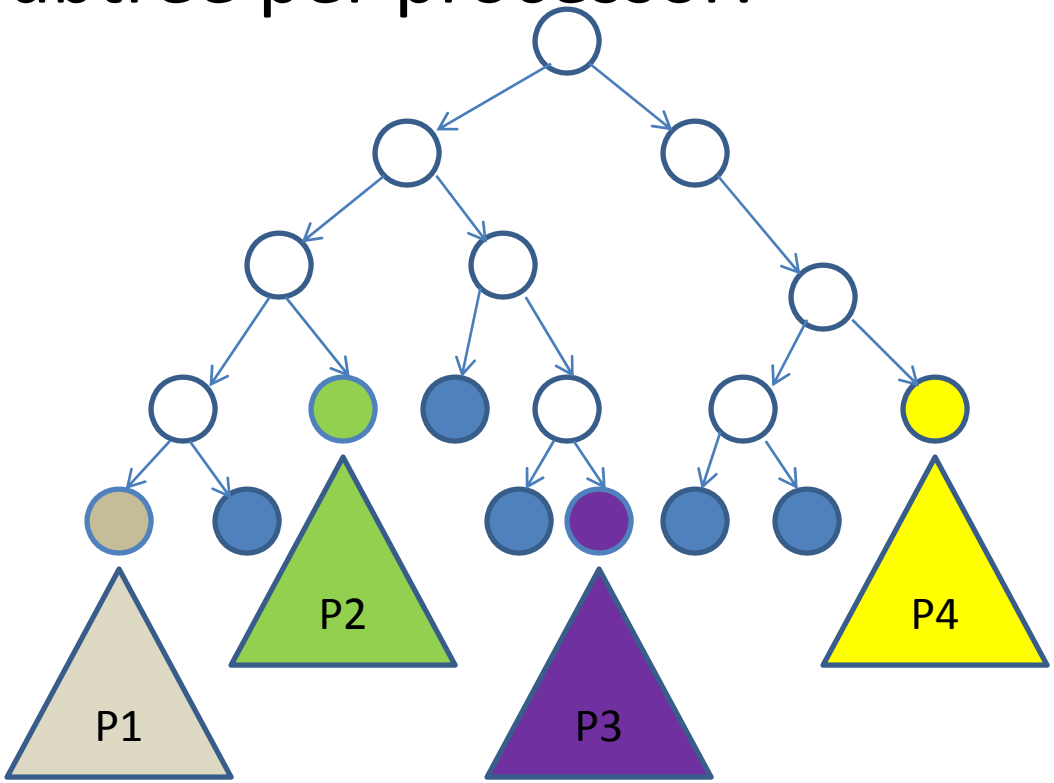
Available for simultaneous processing

Deterministic Parallel MIP

- Multiple phases
- In each phase, on each processor:
  - Explore nodes assigned to processor
  - Report back results
    - New active nodes
    - New solutions
    - New cuts
    - Etc.
- One approach to node assignment:
  - Assign a subtree to each processor
  - Limit amount of exploration in each phase

Deterministic Parallel MIP

- One subtree per processor:

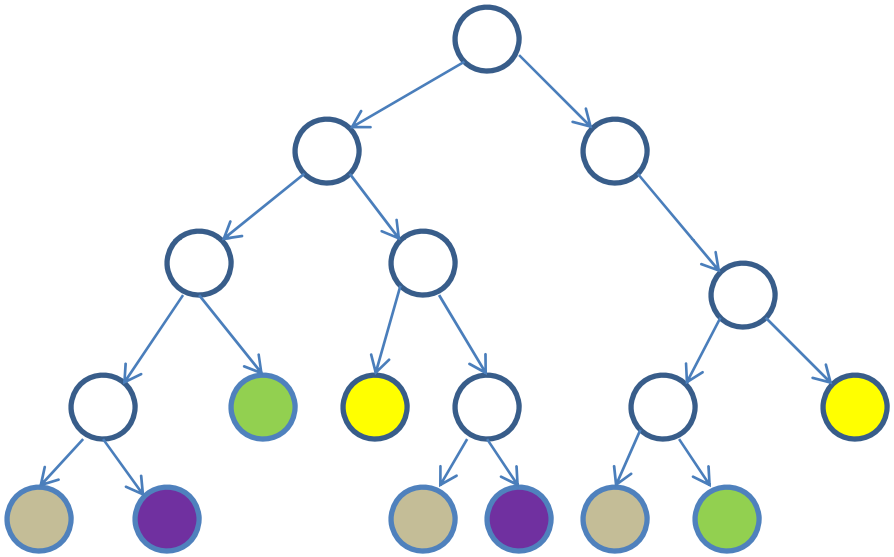


Subtree Partitioning

- Problem:
  - Subtree may quickly prove to be uninteresting
    - Poor relaxation objectives
      - May want to abandon it
    - Pruned quickly
      - Leaves processor idle

More Global Partitioning

- Node coloring: assign a color to every node



  - Processor can only process nodes of the appropriate color
  - New child node same color as parent node
  - Perform periodic re-coloring

More Dynamic Node Processing

- Allows much more flexibility
  - Processor can choose from among many nodes of the appropriate color
- *Deterministic priority queue* data structure required to support node coloring
  - Single global view of active nodes
  - Support notion of node color
    - Processor only receives node of the appropriate color
  - Efficient, frequent node reallocation

Parallel Performance: Realistic Expectations

Realistic Expectations

- Major constraint: sequential phases
  - Presolve
    - Mean for our test set: 3% of runtime
  - Root node
    - Lots of things happening…
      - Solving relaxations, cuts, heuristics, etc.
    - Lots of opportunities for parallelism?
    - Majority of time still in simplex:
      - Mean for our test set: 84% of root runtime
  - Many models solve at the root:
    - Of 55 models in Mittelmann MIP optimality test set
      - 25% solve in fewer than 20 nodes
  - Significant drag on potential improvement

Realistic Expectations

- Multi-core chips share resources
  - Cache
  - Memory access
- Performance doesn't scale perfectly with cores
- Our conclusion: 2x improvement for p=4 is about the most we can expect (3x seems a reasonable guess for p=8)

The Bag of Tricks

Overview

- Two-decades of Progress: 1988-2008
  - Linear Programming
  - Mixed-Integer Programming
- Recent developments: 2008-Present
  - Redesigning MIP
    - Tree of Trees
    - Parallel MIP
  - The bag of tricks
    - Branching
    - Heuristics
    - Cutting planes
    - An example reduction – Disjoint subtrees
    - Domination
      - Symmetry testing
- Where are we now
  - Performance summary

Branching

Branching

- Variable branching
  - Max fractional value
    - Worse than random
  - Shadow costs (similar to pseudo costs)
  - Strong branching
    - Traveling salesman problem
  - “Modern” pseudo costs
    - Keep estimate of effect on the objective of branching on a variable – up estimate, down estimate
    - Improve by initializing using strong branching
    - Improve further by applying strong branching multiple times: reliability branching

Branching

- SOS branching
  - Pseudo-cost branching for SOS sets
    - Compute variable pseudo costs by fixing to zero and solving LP
    - Find a split for an infeasible SOS set, x_1, …, x_k with relaxation solution x*
    - Compute pseudo costs for left and right sets by using sum pcost[i]* x*[i]
    - Combine them
    - Pick the set with maximal combined value

Heuristics

Summary of Heuristics

- 5 heuristics prior to solving root LP
  - 5 different variable orders, fix variables in this order
- 15 heuristics within tree (9 primary, several variations)
  - RINS, RINS diving, rounding, fix and dive (LP), fix and dive (Presolve), fix and dive (simple), Lagrangian approach, pseudo costs, Hail Mary (set objective to 0)
- 3 solution improvement heuristics
  - Applied whenever a new integer feasible is found
- Key tool:
  - Bound strengthening

Managing the Heuristics

- Available parameters
  - **Heuristics = fraction of time spent on heuristics (default = 0.05)**
  - Submipnodes = number of nodes explored in a submip (default = 500)
  - RINS = frequency of application of RINS.
- Non default heuristic: Feasibility pump (for finding in initial feasible solutions)

Cutting Planes

Cutting Planes

- Gomory
  - Gu ISMP 2006 , strengthen by lifting in GUBs
- Flow covers
  - Strengthened by lifting before un-transforming (Gu thesis)
- MIR
- Knapsack covers
  - From Gu's thesis (separation and lifting)
- Clique
- Implied bound
- Flow paths
  - Results fed into flow covers
- GUB covers
- Zero-half

More Cutting Planes
Gurobi 3.0

- New cutting planes
  - Network cuts
  - Submip cuts
- Improvement of existing cut routines
  - Aggregation for MIR and flow covers
  - Cut filter

Network Cuts

- Related to multi-commodity flow cuts
- Finding network structure
  - Use a simple heuristic to find a set of network rows
  - Identify associated fixed-charge indicator variables
- Separation
  - Sort arcs based on relaxation values
  - Use the order to construct a spanning tree (forest)
  - Repeat
    - Remove a non-leaf arc from the spanning tree splits network into two parts
    - Aggregate each of the two parts
    - Look for violated flow-cover cut
- Performance
  - 10% speedup on models with network structure.

SUBMIP Cuts

- Solve submip to generate cuts
  - Expensive
  - Applied dynamically for some more difficult models (not typically applied in defaults)
- Main idea
  - Quite different from ideas that solve a sub-MIP to separate
  - Objective: generate cuts from using a point different from the one from the LP relaxation

Cut Changes Summary

- Overall performance improvement
  - 20% speedup on our internal model set

Disjoint Subtrees

Disjoint Subtrees

- Basic principle of branching:
  - Feasible regions for child nodes after a branch should be disjoint
- Not always the case
- Simple example – integer complementarities:
  - $x \leq 10\ b$
  - $y \leq 10\ (1\text{-}b)$
  - x, y non-negative ints, $x \leq 10$, $y \leq 10$, b binary
  - Branch on b: x=y=0 feasible in both children

Recognizing Subtree Overlap

- Problem arises when sole purpose of branching variable is to bound other variables
  - Otherwise, b=0/b=1 split is typically sufficient to make the subtrees disjoint
- Recognizing overlap:
  - Constraints involving branching variable must be redundant after branch
  - Domains of remaining variables must overlap

Removing Overlap

- Simplest way to remove overlap:
  - Modify variable bound in one subtree
- Integer complementarities example:
  - $x \leq 10\ b$
  - $y \leq 10\ (1-b)$
  - Branch on b: x=y=0 feasible in both children
- b=0 child: $x = 0,\ 10 \geq y \geq 0$
- b=1 child: $y = 0,\ 10 \geq x \geq 1$

Performance Impact

- Overlap present in several models
  - 35 out of 510 models in our test set
- Performance impact can be huge
  - Model neos859080 goes from 10000+ seconds to 0.01s
  - Makes it tough to quote mean improvements over a small set
- Median improvement for affected models is ~1%

MIP Domination

MIP Domination

- MIP domination
  - A feasible solution X is (strictly) dominated by a feasible solution Y, if Y has objective value as good as (better than) X.
  - Suppose for any feasible solution X with Xj > a, there exists another feasible solution Y with Yj <= a such that Y dominates X. Then we need only consider xj <= a.
- Use of domination information
  - Reduce or simplify MIP models
  - Avoid unnecessary search

Domination Techniques

- Presolve reductions
- Dominated nodes detection
- Symmetry breaking

A Simple Presolve Reduction

- Consider

```
Min 5 x1 + 4 x2 + 11 x3 + 2 y1 + 2 y2
s.t.  x1 +   x2 +  3 x3 +      +   y2 >= 7
    2 x1 + 2 x2 +  2 x3 +   y1         >= 10
    x1, x2, x3, y1, y2 >= 0
    x1, x2, x3 are integers
```

- Parallel columns
  - x1 and x2 are parallel and x1 is dominated because of its objective coefficient

Dual Presolve Reductions

- Consider

```
Min 5 x1 + 4 x2 + 11 x3 + 2 y1 + 2 y2
s.t.  x1 +   x2 +  3 x3         +   y2 >= 7
    2 x1 + 2 x2 +  2 x3 +   y1         >= 10
    x1, x2, x3, y1, y2 >= 0
    x1, x2, x3 are integers
```

- Dual bound tightening
  - Consider the dual of any relaxation at a B&B node
  - Let d1 and d2 be dual variables for two constraints
  - Use dual constraints corresponding to y1 and y2, we can tighten dual bounds to d1 <= 2 and d2 <= 2
  - Reduced cost for x3 >= 11 – 3 * 2 – 2 * 2 = 1, so x3 can be fixed to 0

Another Presolve Reduction

- Consider
  - a x + b y = c
  - x, y are integer variables
  - a, b and c are integers, a > 1
  - Assume gcd(a,b) = 1
    - Otherwise a Euclidean reduction is possible
  - Observation: Then x(mod b) and y(mod a) are constants.
- Reduction
  - Substitute y = a z + d (d is easy to compute).
  - z has a smaller search space than y
- General application
  - Can easily be extended to general “all integer” constraints.

Dominated Nodes

- 0-1 knapsack example

```
Min 5 x1 + 6 x2 + 7 x3 + 9 x4 + sum wj xj
    3 x1 + 4 x2 + 5 x3 + 6 x4 + sum aj xj <= b
```

- At a node
  - (x1, x2, x3, x4) = (1, 0, 0, 1)
- Let
  - cost(x1, x2, x3, x4) = 5 x1 + 6 x2 + 7 x3 + 9 x4
  - a(x1, x2, x3, x4) = 3 x1 + 4 x2 + 5 x3 + 6 x4
- Then
  - cost(0,1,1,0) < cost(1,0,0,1)
  - a(0,1,1,0)= a(1,0,0,1)
  - The node is dominated

Dominated Nodes

- Consider a general MIP

```
Min  e^T x + f^T y + g^T z
s.t. A x + B y + C z <= b
     A, B and C are matrices
     x, y, z are variable vectors
     x are binary
     some y, z are integer or binary
```

- At a node
  - x is fixed to x* (via branching)
  - y represent extra “freedom” beyond x
- Note just for simplicity, we assume
  - All constraints are inequalities
  - Variables x are binary

Dominated Nodes

- Let

```
s(y*) =  Min  e^T x + f^T y
          s.t. A x + B y <= A x* + B y*
               x are binary
               some y are integer or binary
```

- If $s(y^*) < e^T x^* + f^T y^*$ for all y*, then the node is dominated
- Two alternatives
  - If |x| is small and y is empty, computing s(.) is often cheap. However $s(.) < e^T x^*$ is rare in those cases.
  - With non-empty y, where y has special properties, we can sometimes solve for s(y*).

Dominated Nodes

- Fixed charge sub-networks
  - Binary variables indicate whether arcs are open
  - x are the binary variables branched to one
  - Suppose the support of x contains a cycle.
  - Then using y defined by this cycle, we can conclude that the node is dominated.

Symmetry

- Definition
  - Given a MIP

    Min { $c^T x$ | A x <= b, integrality conditions on x}
  - Let

    α: a column permutation of A

    β: a row permutation of A

    PC: a set of all column permutations

    PR: a set of all row permutations
  - A symmetry group is defined as

    G = {α in PC | there exists β in PR, such that

    (β ,α)(A) = A, α(c) = c and β(b) = b}

Exploiting Symmetry

- Find symmetry group
- Use to improve MIP search

Symmetry

- Finding the Symmetry group
  - Considerable published work on graph automorphisms
    - Several computer programs are available, e.g. NAUTY and SAUCY
  - Easy to translate MIP symmetry problem into graph automorphisms (Puget 2005).

Symmetry

- Exploit in MIP search
  - Several papers over the last 10 years
    - Adding cuts: Rothberg (2000)
    - Fathom symmetric nodes: Margot (2002)
    - Orbit branching: Ostrowski, Linderoth, Rossi, and Smriglio (2007).
  - Commercial MIP software
    - Introduced in CPLEX 9
    - Substantially improved in CPLEX 10

Symmetry

- Gurobi 3.0
  - Implemented symmetry detection directly using the matrix
  - Apply orbital branching plus several additional ideas
  - 28% of models in our test set have symmetry
  - Performance is affected on 50% of those with symmetry
  - Many unsolvable models become solvable
  - 25% geometric speedup on the whole set (including those without symmetry)

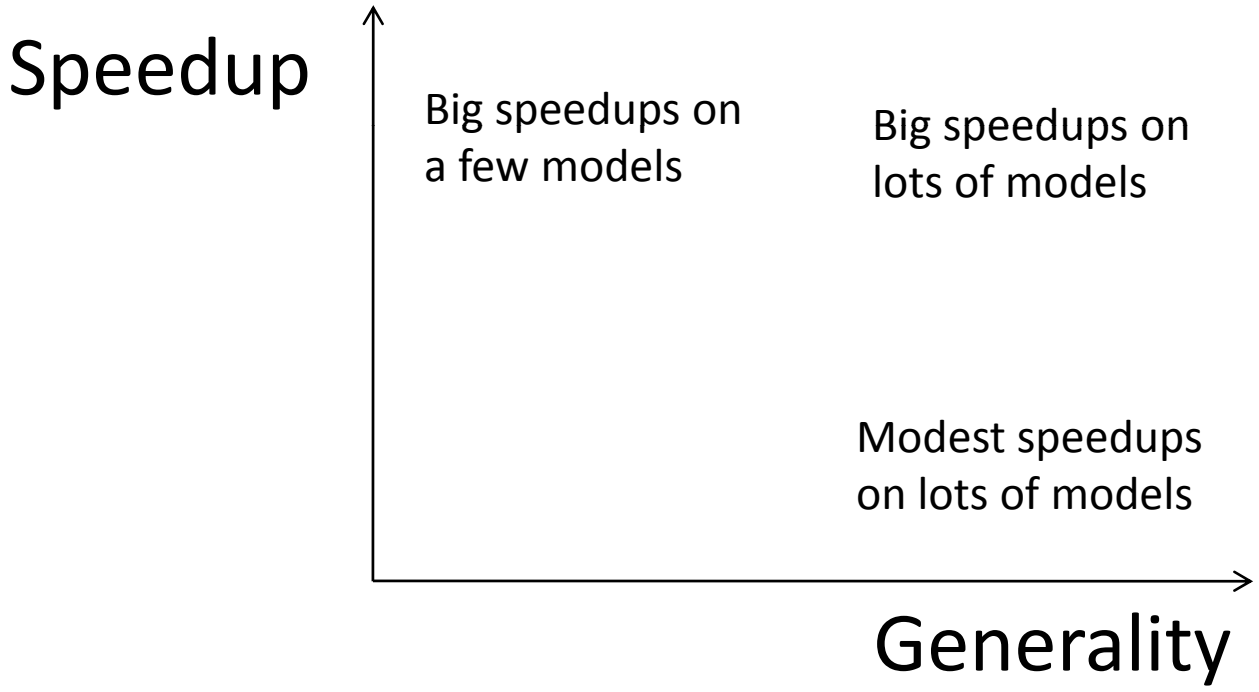
Where We Are Now

MIP Performance

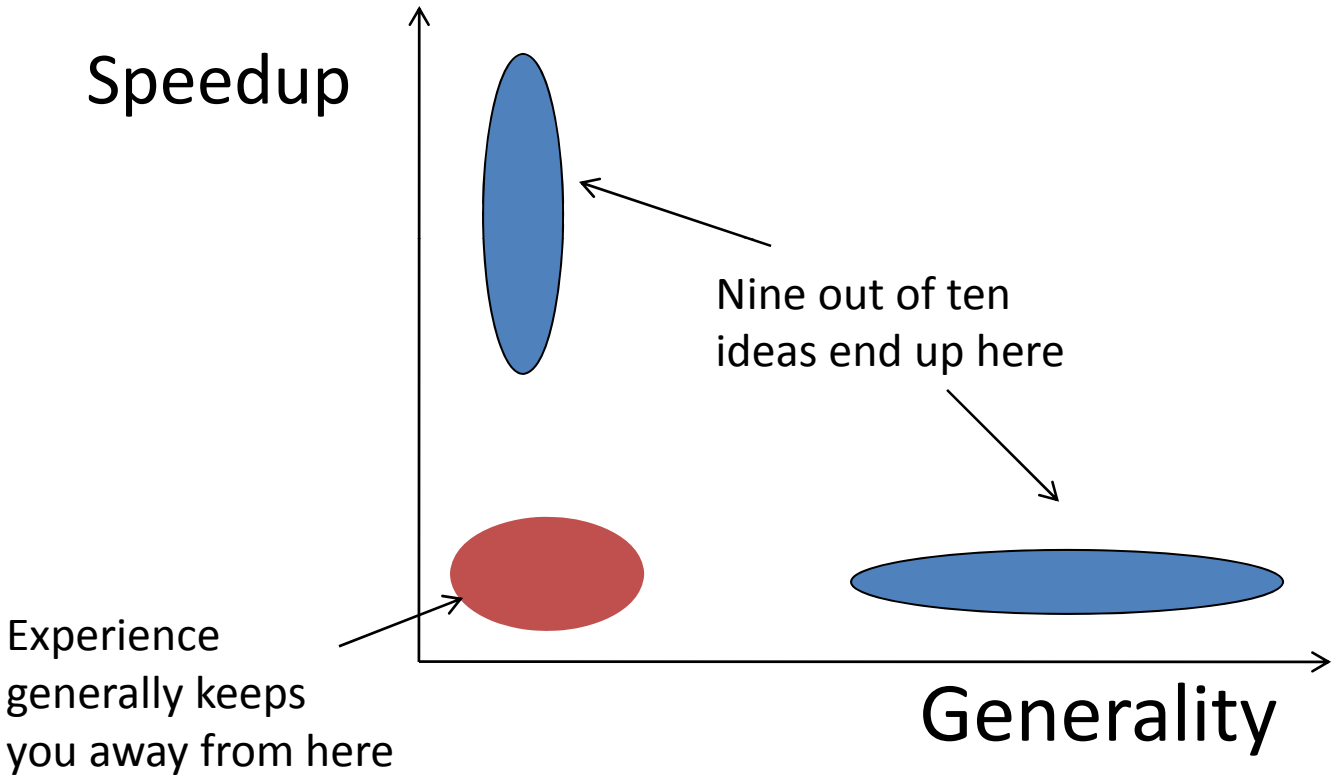
A Framework for Viewing MIP Improvements

Improving a MIP Solver

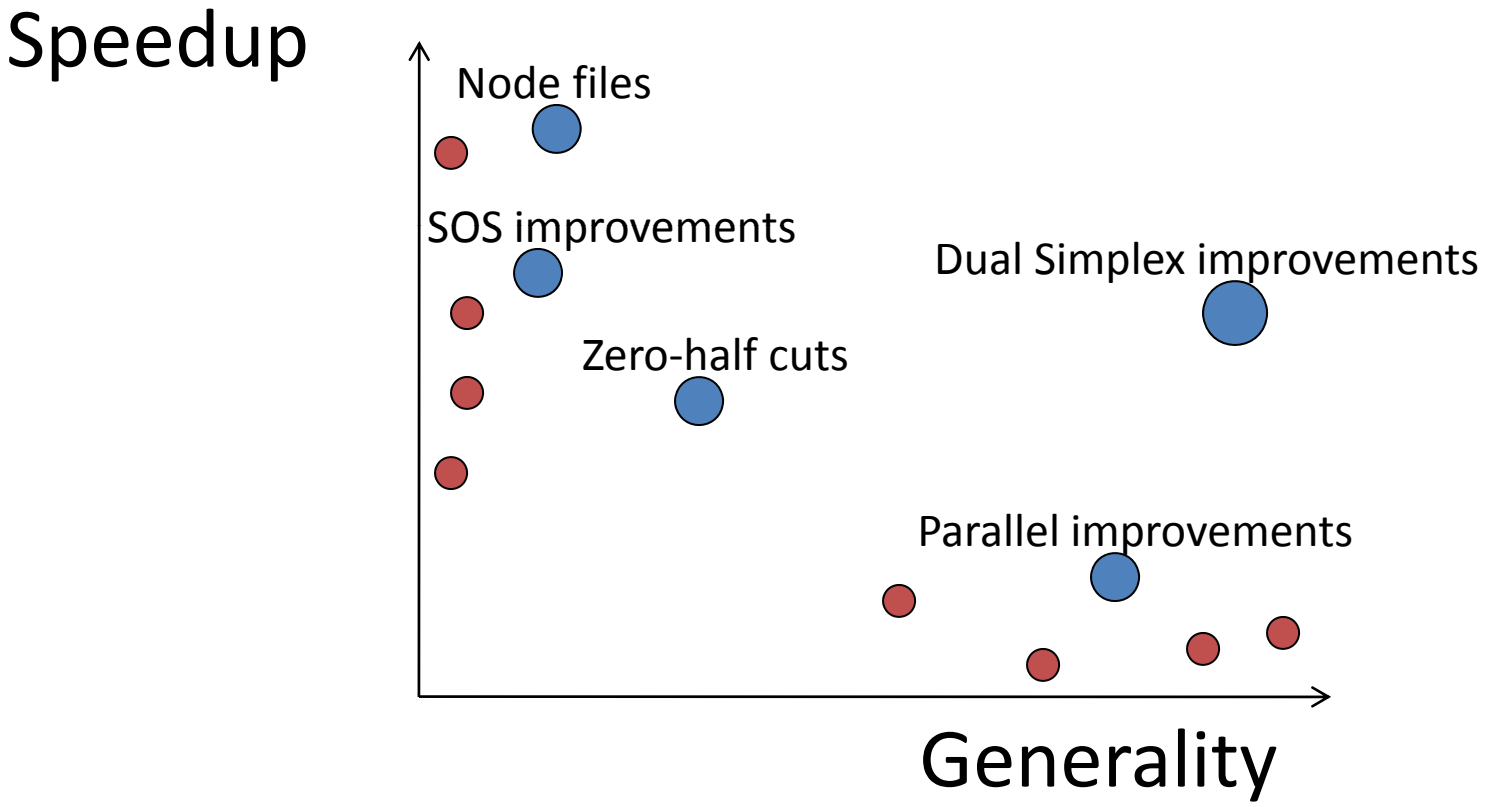
- Improvements can be plotted on two axes:



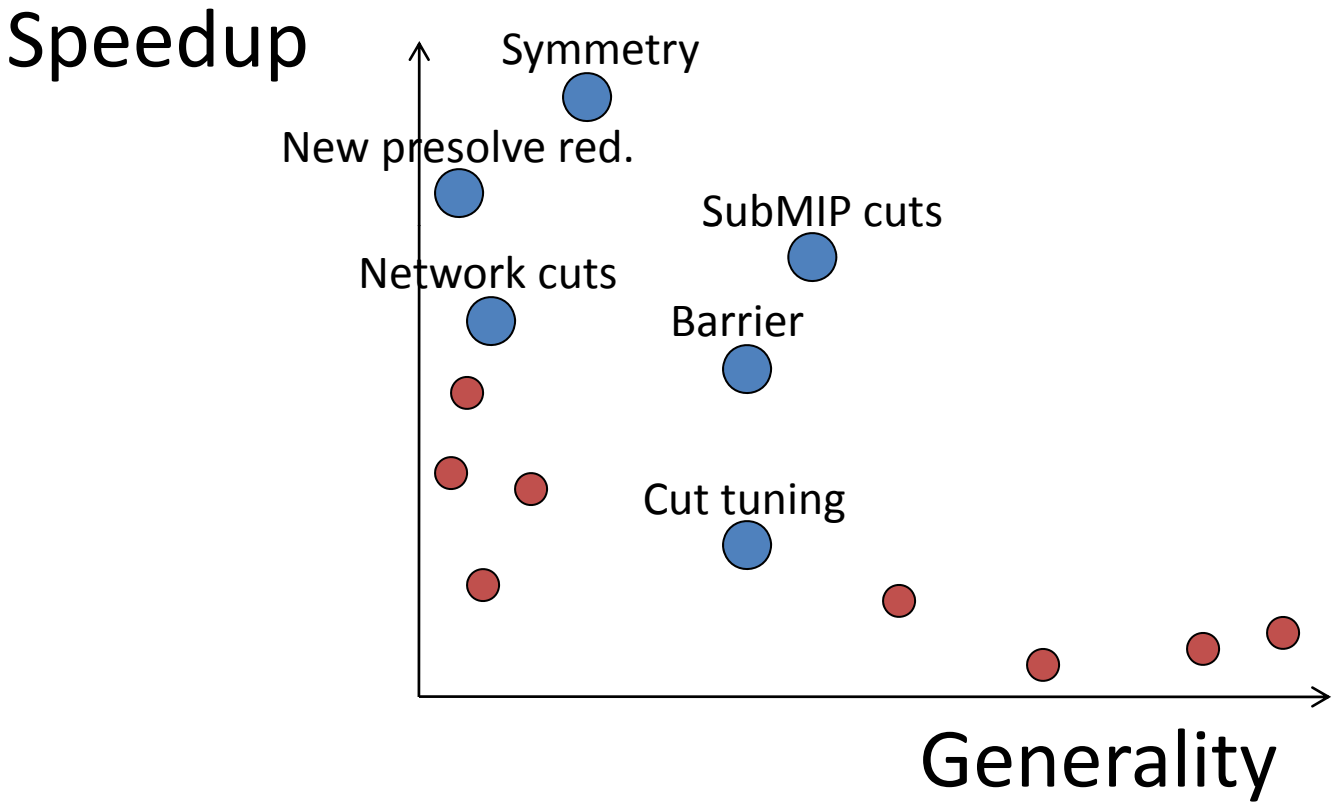
New Ideas


Speedup
Nine out of ten
ideas end up here
Experience
generally keeps
you away from here
Generality

Gurobi 2.0


Speedup
Node files
SOS improvements
Zero-half cuts
Dual Simplex improvements
Parallel improvements
Generality

Gurobi 3.0


Speedup
Symmetry
New presolve red.
SubMIP cuts
Network cuts
Barrier
Cut tuning
Generality

MIP Performance Benchmarks

- Performance test sets:
  - Mittelmann optimality test set:
    - 55 models, varying degrees of difficulty
    - http://plato.asu.edu/ftp/milpc.html
  - Mittelmann feasibility test set:
    - 33 models, difficult to find feasible solutions
    - http://plato.asu.edu/ftp/feas_bench.html
  - Mittelmann infeasibility test set:
    - 11 models, objective is to prove infeasibility
    - http://plato.asu.edu/ftp/infeas.html
  - Our own broader test set:
    - A set of 2458 models
- Test platform:
  - Q9450 (2.66 GHz, quad-core system)

MIP Performance – Gurobi Internal Test Set

- Gurobi V2.0 vs. V1.0 (p=4)
  - 2340 total models in test set
    - 1309 solved by both in < 1 second (removed)
    - 650 solved by at least one in < 10000 seconds
    - 381 solved by neither in < 10000 seconds
- Gurobi V3.0 vs. V2.0 (p=4)
  - 2458 total models in test set
    - 1350 solved by both in < 1 second (removed)
    - 794 solved by at least one in < 10000 seconds
    - 314 solved by neither in < 10000 seconds

MIP Performance – Gurobi Internal Test Set

- Gurobi V2.0 versus V1.0 (P=4)

Time	# Models	Speedup
> 1s	650	1.7x
> 10s	410	1.9x
> 100s	210	2.2x
> 1000s	59	3.9x

- Gurobi V3.0 versus V2.0 (P=4) (bigger test set)

Time	# Models	Speedup
> 1s	794	1.6x
> 10s	521	2.0x
> 100s	295	2.9x
> 1000s	144	6.7x

MIP Performance – Public Benchmarks

- Gurobi 3.0 vs. CPLEX 12.1:

	P=1	P=4
Optimality	1.75X	1.87X
Feasibility	4.76X	-
Infeasibility	-	4.09X

Parallel Performance

Parallel Performance (V2.0 data)

- Parallel speedups (Gurobi P=1 vs P=4):

	P=4 speedup
>1s	1.54
>10s	1.64
>100s	1.79

Thank You