Gurobi MIP Algorithms

GUROBI
OPTIMIZATION

Mixed Integer Programming

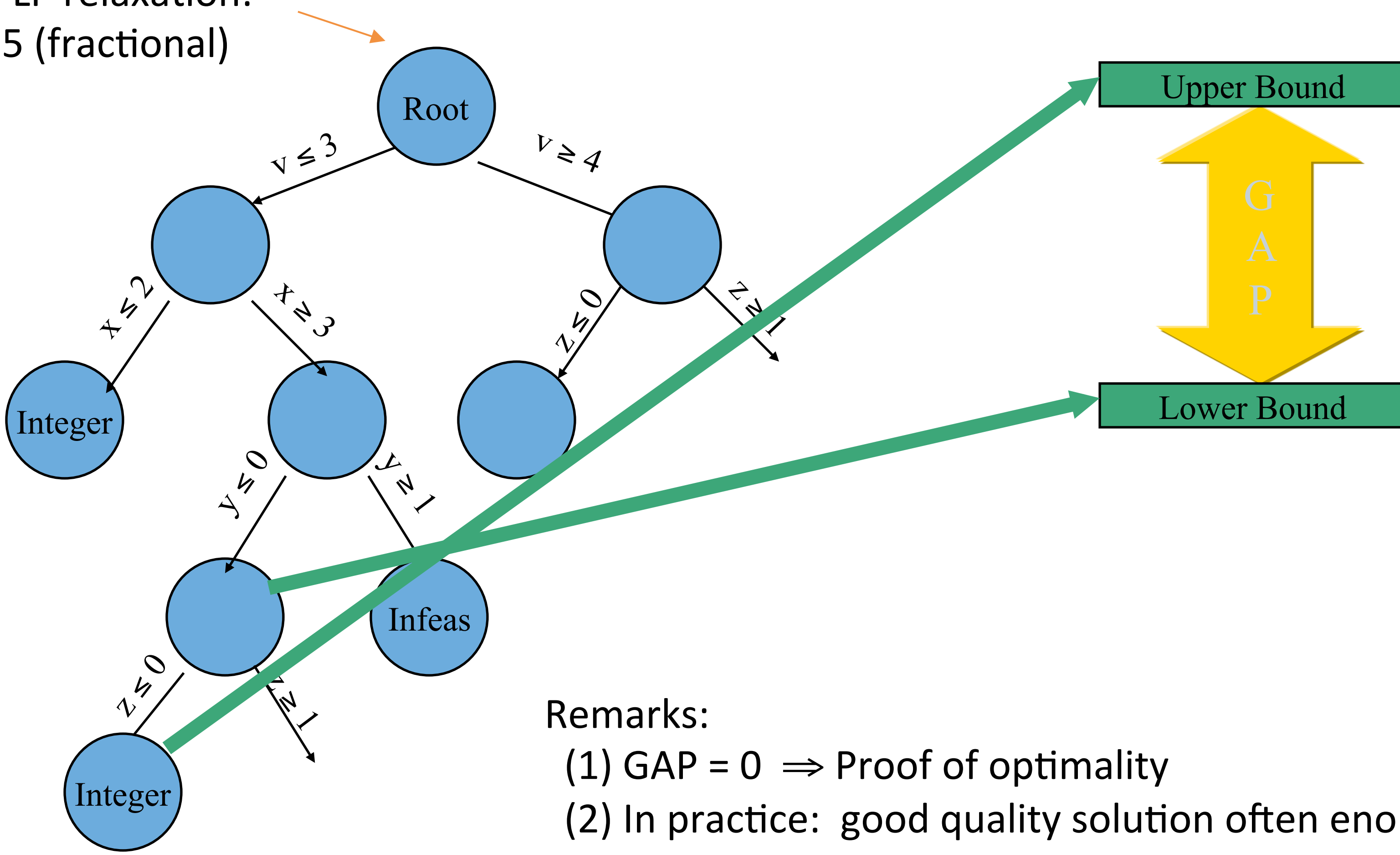
A *mixed-integer program* (MIP) is an optimization problem of the form

$$\begin{aligned} \underset{x}{\text{minimize}} \quad & \sum_{j=1}^{n} c_j x_j \\ \text{subject to} \quad & \sum_{j=1}^{n} A_{ij} x_j = b_i, \quad i = 1, \ldots, m, \\ & \ell_j \le x_j \le u_j, \quad j = 1, \ldots, n, \\ & \text{some or all } x_j \text{ integer} \end{aligned}$$

MIP solution framework:

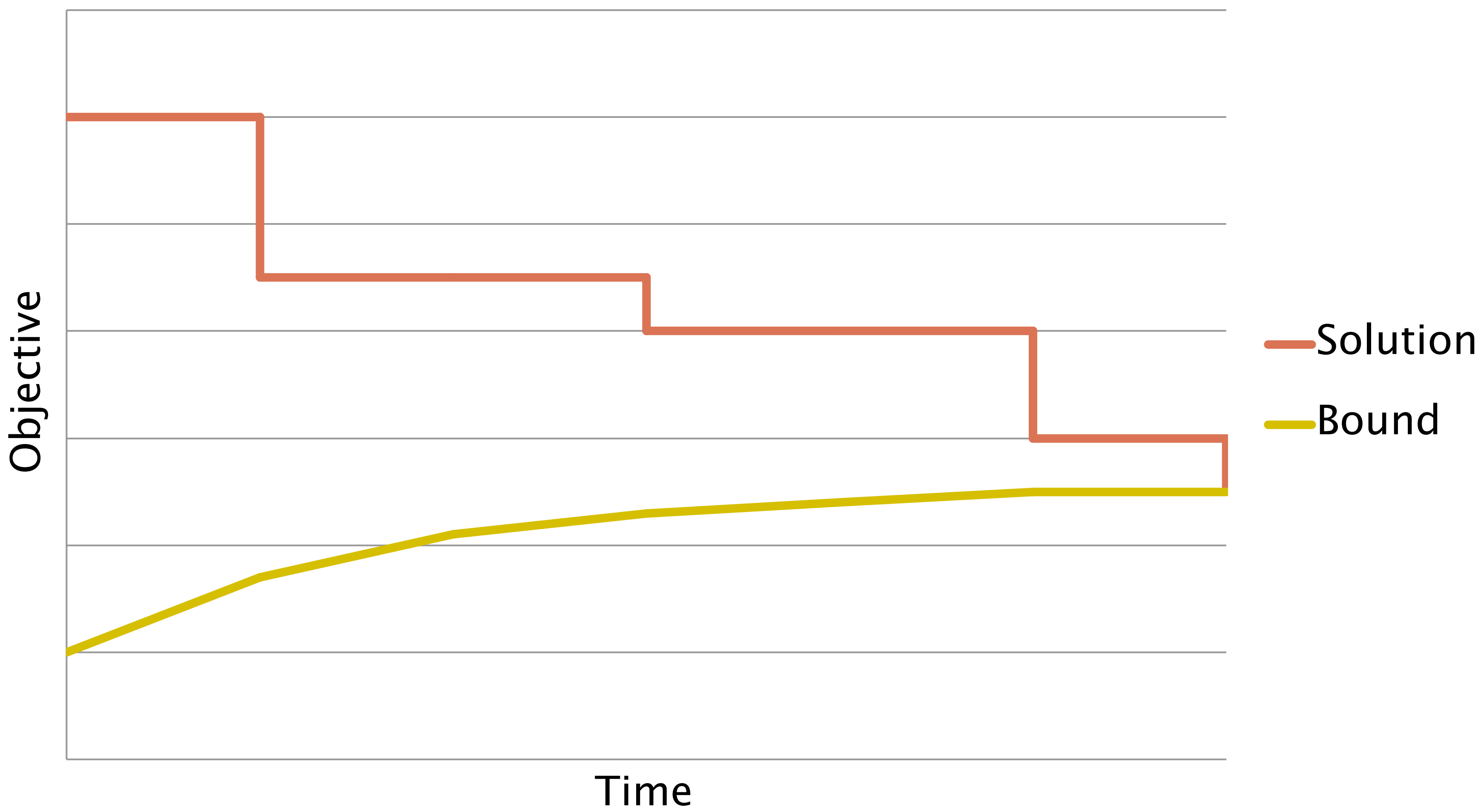
LP based Branch-and-Bound

Solve LP relaxation:
v=3.5 (fractional)



Remarks:
(1) GAP = 0 $\Rightarrow$ Proof of optimality
(2) In practice: good quality solution often enough

Solving a MIP Model



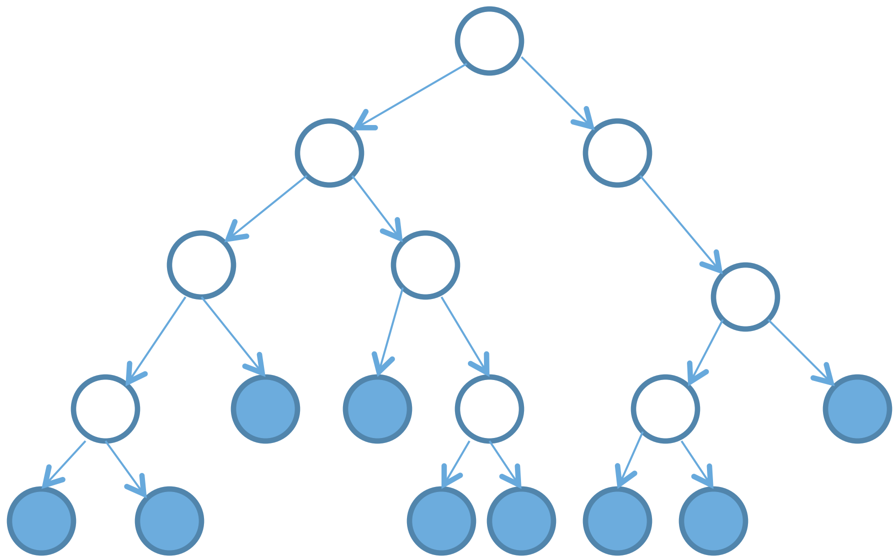
MIP Building Blocks

MIP Building Blocks

- Solve continuous relaxations
  - Ignoring integrality
  - Gives a bound on the optimal integral objective
- Branching variable selection
  - Crucial for limiting search tree size
- Cutting planes
  - Cut off relaxation solutions
- Primal heuristics
  - Find integer feasible solutions
- Presolve
  - Tighten formulation and reduce problem size

MIP Building Blocks

- Branch and bound



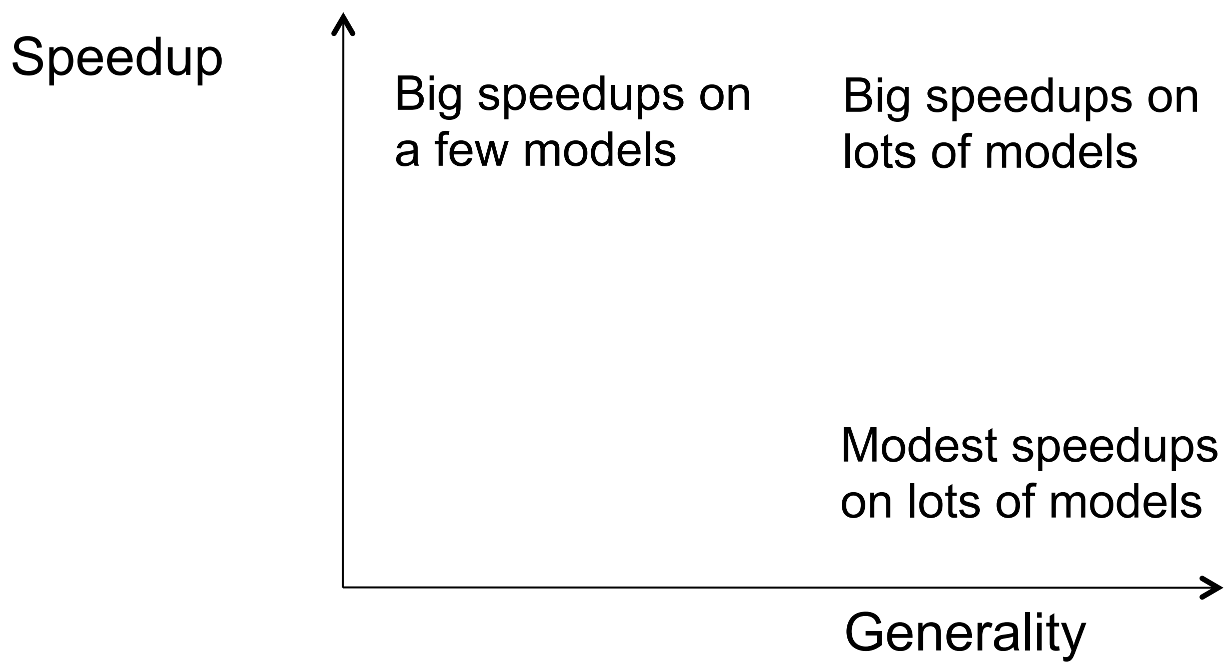
MIP Building Blocks

- Parallel branch-and-cut
  - Explore the MIP search tree using multiple processors
  - Deterministic parallel behavior

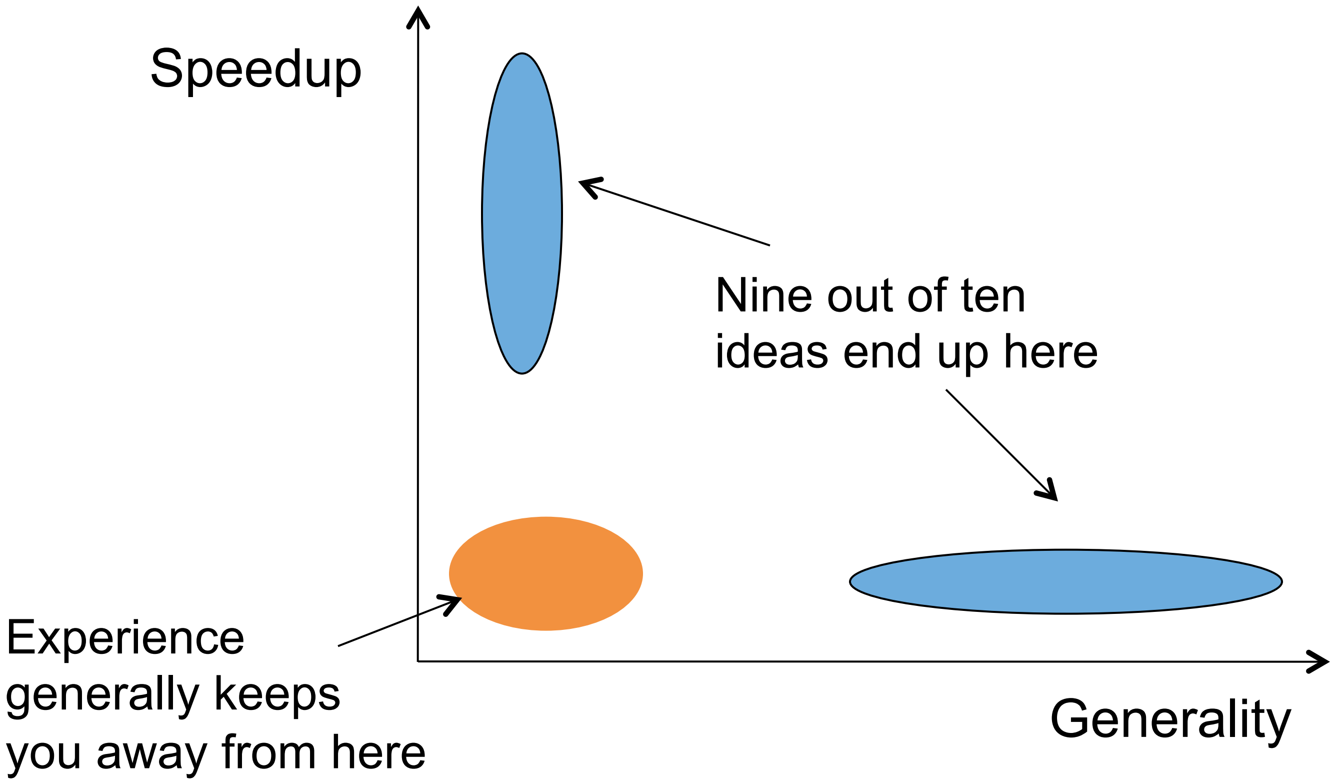
Building A Better MIP Solver

Improving a MIP Solver

- Improvements can be plotted on two axes:



New Ideas

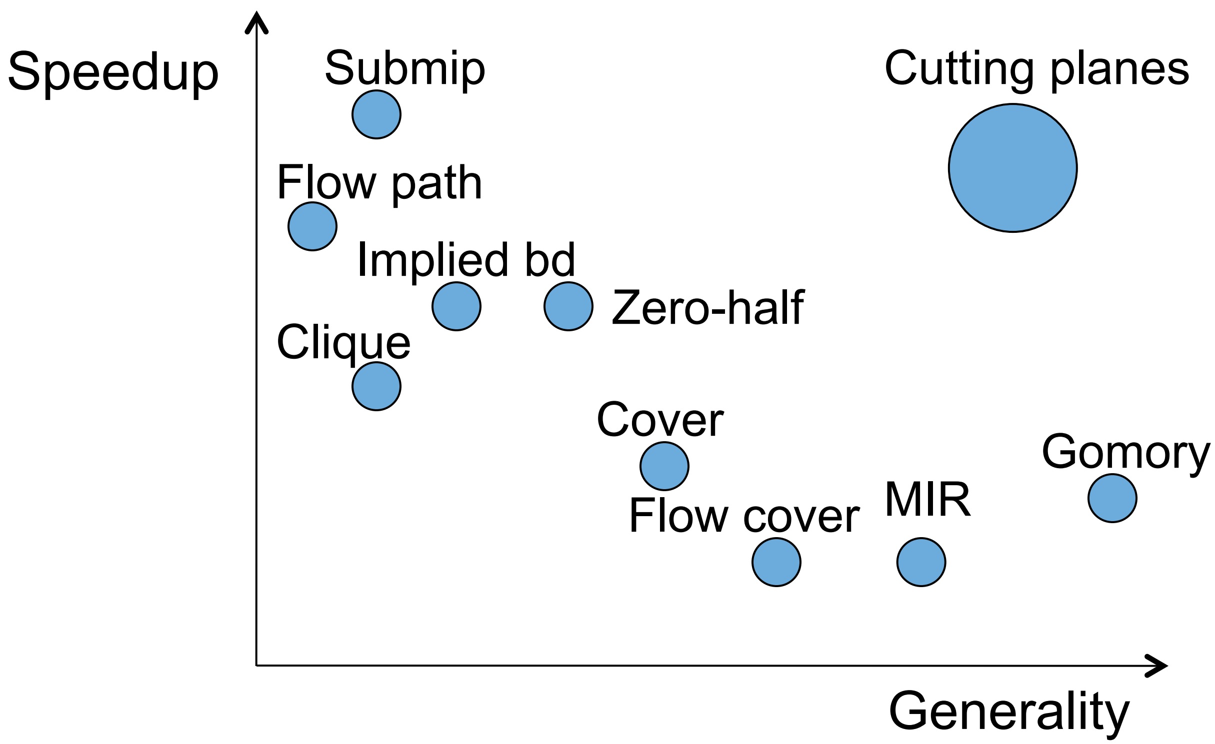


GUROBI OPTIMIZATION

Improvements Near the Axes

- Improvements near the axes are quite important as well
- Consider MIP cutting planes
  - General 'cutting plane' label clearly in the top-right
  - Considering cutting planes individually…

Cutting Planes



Presolve

- MIP presolve reduction types
  - Reduce the problem size (similar to LP)
    - Aggregation
    - Remove redundant constraints
  - Tighten formulation
    - Coefficient reduction

      $7x + y + z \leq 8$, x, y and z are binary

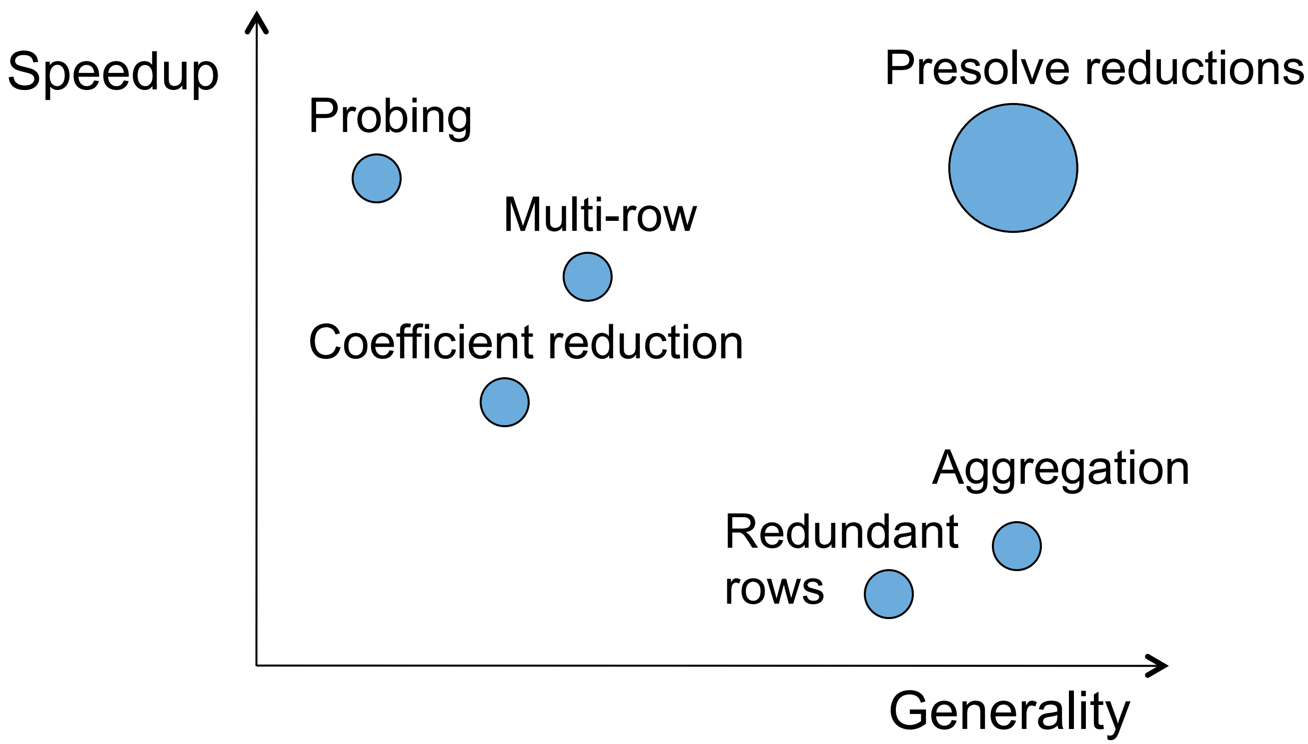
      …can be reduced to…

      $x + y + z \leq 2$
    - Probing

      $b = 0 \text{ -> } x = 0, \; b = 1 \text{ -> } x = 0 \;\Rightarrow\; x = 0$

- General vs. individual reductions (similar to cuts)
  - General 'presolve' label clearly in the top-right
  - Individual reductions …

Presolve Reductions



GUROBI OPTIMIZATION

Near the Axes – An Example

Disjoint Subtrees

- Basic principle of branching:
  - Feasible regions for child nodes after a branch should be disjoint

- Not always the case

- Simple example – integer complementarity:
  - $x \leq 10\ b$
  - $y \leq 10\ (1-b)$
  - 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

- Recognizes domain overlap
- Adjusts variable bounds in subtrees to remove it
- Huge win on a few models
  - neos859080 time drops from 10000+ seconds to 0.1s
- Not very general
  - Affects less than 1 in 10 models
  - Performance impact is small on most models

Parallel MIP

Need Deterministic Behavior

- Non-deterministic parallel behavior:
  - Multiple runs with the same inputs can give different results
    - Big difference in run-time
    - Different optimal solutions

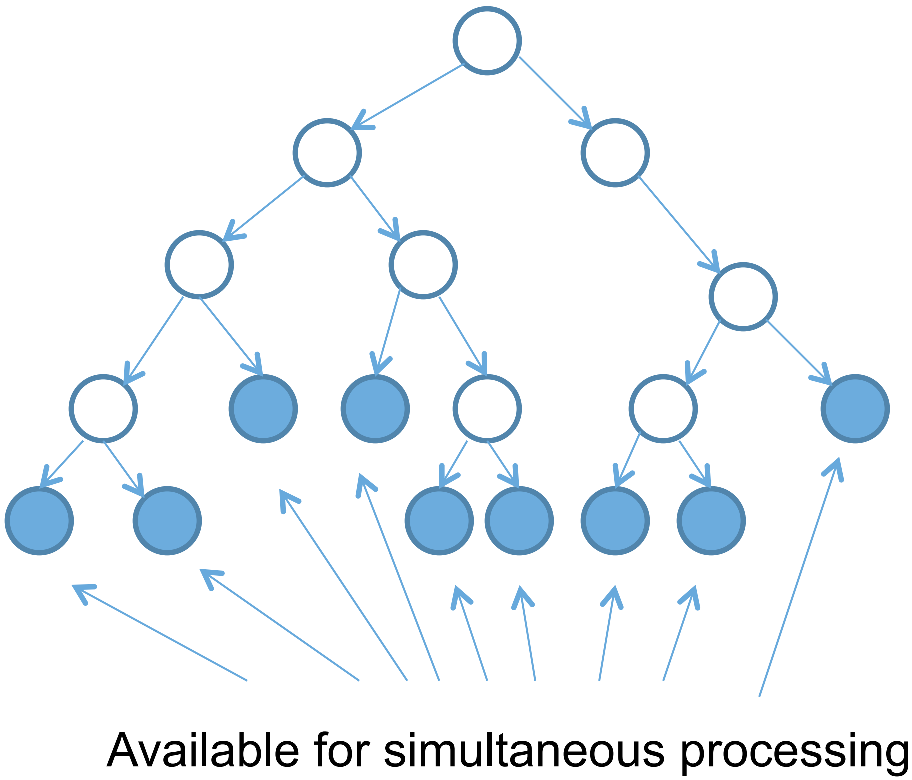
- "Insanity: doing the same thing over and over again and expecting different results"
  – Albert Einstein

- Conclusion:
  - Non-deterministic parallel behavior will drive you insane!

Building Blocks

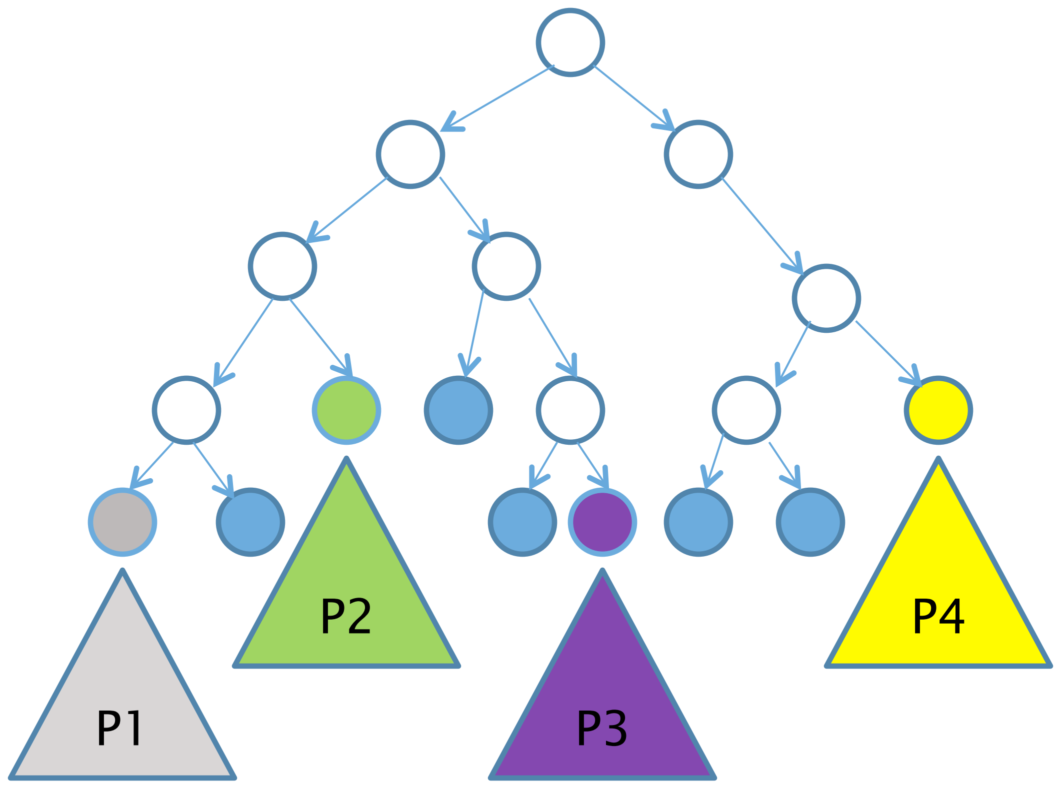
Building Blocks

- Parallel MIP is parallel branch-and-bound:



Deterministic Parallel MIP

- One subtree per processor:

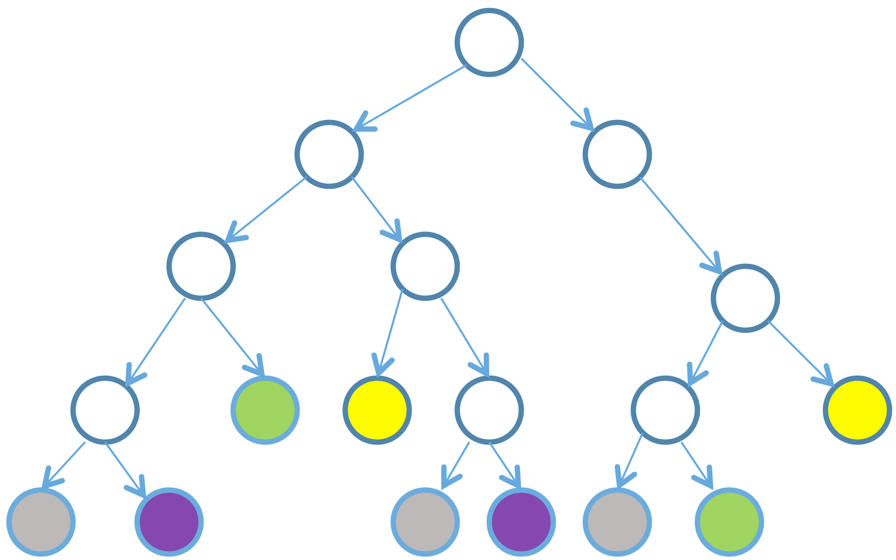


Subtree Partitioning

- Problem: hard to predict subtree difficulty
  - Subtree may quickly prove to be uninteresting
    - Poor relaxation objectives
      - May want to abandon it
    - Pruned quickly
      - Leaves processor idle
  - Majority of work may be in one subtree

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

Branch Variable Selection

Pseudo-Costs

- Given a relaxation solution x*
  - Branching candidates:
    - Integer variables x_j that take fractional values
      - xj=0.5 produces two child nodes (x=0 or x=1)
  - Need to pick a variable to branch on
    - Choice is crucial in determining the size of the overall search tree

Pseudo-Costs

- What's a good branching variable?
  - Superb: fractional variable infeasible in both branch directions
  - Great: infeasible in one direction
  - Good: both directions move the objective

- Expensive to predict which branches lead to infeasibility or big objective moves
  - Strong branching
    - Truncated LP solve for every possible branch at every node
    - Rarely cost-effective
  - Need a quick estimate

Pseudo-Costs

- Use historical data to predict impact of a branch:
  - Record $\text{cost}_x = \Delta_{obj} / \Delta_x$ for each branch
    - Need a scheme for infeasible branches too
  - Store results in a pseudo-cost table
    - Two entries per integer variable
      - Average (or max) down cost
      - Average (or max) up cost
  - Use table to predict cost of a future branch

Pseudo-Cost Initialization

- What do you do when there is no history?
  - E.g., at the root node
- Initialize pseudo-costs [Linderoth & Savelsbergh, 1999]
  - Always compute up/down cost (using strong branching) for new fractional variables
    - Initialize pseudo-costs for every fractional variable at root
- Reliability branching [Achterberg, Koch, & Martin, 2002]
  - Don't rely on historical data until pseudo-cost for a variable has been recomputed r times

Pseudo-Cost Adjustment

- Gurobi MIP solver adjusts pseudo-costs in several ways
  - Implied pseudo-cost bounds
  - Ancestor adjustment

Implied Pseudo-Cost Bounds

- Consider constraint:
  - $\sum x_i = 1$
- Computed objective bounds:
  - $x_1=1$ -> obj ≥ 100
  - $x_2=0$ -> obj ≥ 110
- Stronger bound for $x_1=1$:
  - $x_1=1$ -> $x_2=0$ -> obj ≥ 110
- In general:
  - If x=a -> y=b, then $obj_{x=a} \geq obj_{y=b}$
- Often violated:
  - Strong branching uses an iteration limit

Implied Pseudo-Cost Bounds

- Consider $\sum x_i = 1$ again

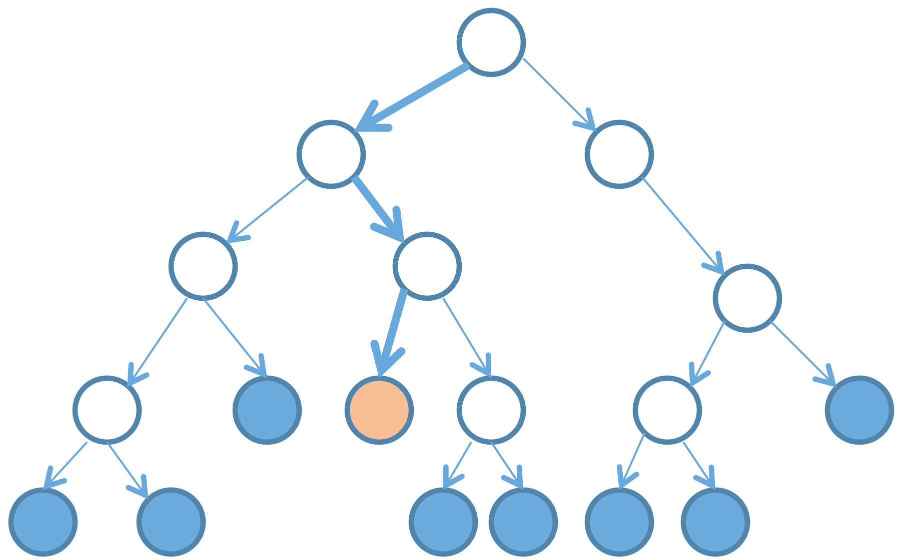
- Computed objective bounds:
  - $x_1=0$ -> obj $\geq$ 100
  - $x_j=1$ -> obj $\geq$ 110 for all j != 1

- Stronger bound for $x_1=0$:
  - $x_1=0$ -> $x_j=1$ for some j != 1 -> obj $\geq$ 110

- In general:
  - If $x_i=a$ -> $x_j=b$ for some j != i
  - Then $obj_{xi=a} \geq \min(obj_{xj=b})$

Ancestor Adjustment

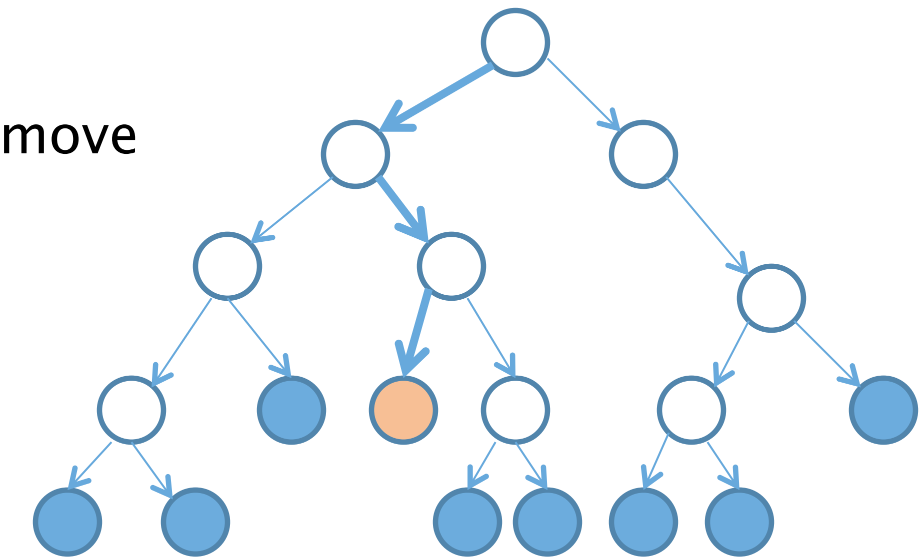
- Objective move isn't entirely the result of most recent branch variable
  - Depends on ancestors as well
  - Particularly true for infeasible nodes



- Adjust pseudo-costs for ancestor branches

Ancestor Adjustment

- Need an adjustment strategy
- Empirically, adjustment should decrease as you move up the tree
- Our approach:
  - Exponential backoff
  - ½ for parent, ¼ for grandparent, etc.



Thank You