Solving Mixed Integer Programs in Practice
Linear Programming

• A linear program (LP) is an optimization problem of the form

\[
\begin{align*}
\text{minimize} \quad & \sum_{j=1}^{n} c_j x_j \\
\text{subject to} \quad & \sum_{j=1}^{n} A_{ij} x_j = b_i, \quad i = 1, \ldots, m, \\
& l_j \leq x_j \leq u_j, \quad j = 1, \ldots, n,
\end{align*}
\]

• Why do we care about this problem?
  • Some applications (e.g., blending in the oil industry)
  • Work horse for mixed integer programming solvers
Mixed Integer Programming

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

\[
\begin{align*}
\text{minimize} \quad & \sum_{j=1}^{n} c_j x_j \\
\text{subject to} \quad & \sum_{j=1}^{n} A_{ij} x_j = b_i, \quad i = 1, \ldots, m, \\
& \ell_j \leq x_j \leq u_j, \quad j = 1, \ldots, n,
\end{align*}
\]

some or all \( x_j \) integer

• Why do we care about this problem?
Applications of Mixed Integer Programming

- Accounting
- Advertising
- Agriculture
- Airlines
- ATM provisioning
- Compilers
- Defense
- Electrical power
- Energy
- Finance
- Food service
- Forestry
- Gas distribution
- Government
- Internet applications
- Logistics/supply chain
- Medical
- Mining
- National research labs
- Online dating
- Portfolio management
- Railways
- Recycling
- Revenue management
- Semiconductor
- Shipping
- Social networking
- Sports betting
- Sports scheduling
- Statistics
- Steel Manufacturing
- Telecommunications
- Transportation
- Utilities
- Workforce scheduling
- ...
MIP Application Types

• Static MIP
  • Formulate problem
  • Solve it with a black-box MIP algorithm
  • Read solution
  • Potentially adjust problem and iterate
  • most frequent use of MIP in practical applications

• Branch-and-cut
  • Problem has too many constraints to formulate in static fashion
    • e.g., classical TSP model: exponentially many sub-tour elimination constraints
  • Construct partial problem
  • Add violated constraints on demand

• Branch-and-price
  • Problem has too many variables to formulate in static fashion
    • e.g., many public transport and airline problems are solved via B&P
  • Start with subset of variables
  • Pricing: add variables that may improve solution on the fly
  • Usually needs problem specific branching rule that is compatible with pricing
  • Heuristic variant: column generation
    • Only apply pricing for the root LP, then solve static MIP with resulting set of variables
MIP Building Blocks

• Presolve
  • Tighten formulation and reduce problem size

• Solve continuous relaxations
  • Ignoring integrality
  • Gives a bound on the optimal integral objective

• Cutting planes
  • Cut off relaxation solutions

• Branching variable selection
  • Crucial for limiting search tree size

• Primal heuristics
  • Find integer feasible solutions
MIP Building Blocks

• Presolve
  • Tighten formulation and reduce problem size

• Solve continuous relaxations
  • Ignoring integrality
  • Gives a bound on the optimal integral objective

• Cutting planes
  • Cut off relaxation solutions

• Branching variable selection
  • Crucial for limiting search tree size

• Primal heuristics
  • Find integer feasible solutions
LP Presolve

• Goal
  • Reduce the problem size
    • Speedup linear algebra during the solution process

• Example

\[
\begin{align*}
x + y + z & \leq 5 \quad (1) \\
u - x - z & = 0 \quad (2)
\end{align*}
\]

\[
\begin{align*}
x + y + z & \leq 3, \text{ so (1) is redundant}
\end{align*}
\]

• Reductions
  • Redundant constraint
    • (3) ⇒ \( x + y + z \leq 3 \), so (1) is redundant
  • Substitution
    • (2) and (4) ⇒ \( u \) can be substituted with \( x + z \)
MIP Presolve

• Goals:
  • Reduce problem size
    • Speed-up linear algebra during the solution process
  • Strengthen LP relaxation
  • Identify problem sub-structures
    • Cliques, implied bounds, networks, disconnected components, ...

• Similar to LP presolve, but more powerful:
  • Exploit integrality
    • Round fractional bounds and right hand sides
    • Lifting/coefficient strengthening
    • Probing
  • Does not need to preserve duality
    • We only need to be able to uncrush a primal solution
    • Neither a dual solution nor a basis needs to be uncrushed
MIP Presolve

- Goals:
  - Reduce problem size
    - Speed-up linear algebra during the solution process
  - Strengthen LP relaxation
  - Identify problem sub-structures
    - Cliques, implied bounds, networks, disconnected components, ...

- Similar to LP presolve, but more powerful:
  - Exploit integrality
    - Round fractional bounds and right hand sides
    - Lifting/coefficient strengthening
    - Probing
  - Does not need to preserve duality
    - We only need to be able to uncrush a primal solution
    - Neither a dual solution nor a basis needs to be uncrushed

<table>
<thead>
<tr>
<th>model</th>
<th>without presolve</th>
<th>with presolve</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>rows</td>
<td>cols</td>
</tr>
<tr>
<td>roll3000</td>
<td>2291</td>
<td>1166</td>
</tr>
<tr>
<td>neos-787933</td>
<td>1897</td>
<td>236376</td>
</tr>
</tbody>
</table>
Single-Row Reductions

- **Clean-up rows**
  - Discard empty rows
  - Discard redundant inequalities: \( \sup\{A_r x \leq b_r \} \)
  - Remove coefficients with tiny impact \(|a_{ij} \cdot (u_j - l_j)|\)

- **Bound strengthening**
  - \( a_{rj} > 0, \ s := b_r - \inf\{A_r x\} \Rightarrow x_j \leq l_j + s/a_{rj} \)
  - \( a_{rj} < 0, \ s := b_r - \inf\{A_r x\} \Rightarrow x_j \geq u_j + s/a_{rj} \)

- **Coefficient strengthening for inequalities**
  - \( j \in I, \ a_{ij} > 0, \ t := b_r - \sup\{A_r x\} + a_{rj} > 0 \)

  \[ \Rightarrow a_{rj} := a_{ij} - t, \ b_r := b_r - u_j t \]
Single-Row Reductions – Performance

benchmark data based on Gurobi 5.6
Single-Column Reductions

- Remove fixed variables and empty columns
  - If $|u_j - l_j| \leq \epsilon$, fix to some value in $[l_j, u_j]$ and move terms to rhs
  - Choice of value can be very tricky for numerical reasons

- Round bounds of integer variables

- Strengthen semi-continuous and semi-integer variables

- Dual fixing, substitution, and bound strengthening
  - Variable $x_j$ does not appear in equations
  - $c_j \geq 0, A_{ij} \geq 0 \Rightarrow x_j := l_j$
  - $c_j \geq 0, A_{ij} \geq 0$ except for $a_{ij} < 0$,
    $z = 0 \rightarrow$ row $i$ redundant,
    $z = 1 \rightarrow x_j = u_j$
  - $c_j \geq 0$, all rows $i$ with $a_{ij} < 0$ redundant for $x_j \geq t \Rightarrow x_j \leq \max\{l_j, t\}$
Single-Column Reductions – Performance

- FIXED VARS: 2.7%
- SEMI-CONT/INT: 0.5%
- DUAL STRENGTHENING: 7.0%
- COMPL. SLACK: 3.1%
- AGGREGATE: 29.8%

Benchmark data based on Gurobi 5.6
Multi-Row Reductions

- **Parallel rows**
  - Search for pairs of rows such that coefficient vectors are parallel to each other
  - Discard the dominated row, or merge two inequalities into an equation

- **Sparsify**
  - Add equations to other rows in order to cancel non-zeros
  - Can also add inequalities with explicit slack variables

- **Multi-row bound and coefficient strengthening**
  - Like single-row version, but use other rows to get tighter bound on infimum and supremum \(\Rightarrow\) tighter bounds, better coefficients

- **Clique merging**
  - Merge multiple cliques into larger single clique, e.g.:
    \[
    x_1 + x_2 \leq 1 \\
    x_1 + x_3 \leq 1 \\
    x_2 + x_3 \leq 1
    \]
  - with binary variables \(x_1, x_2, x_3\) can be merged into
    \[
    x_1 + x_2 + x_3 \leq 1
    \]
Multi-Row Reductions

PARALLEL ROWS: 1.7%
SPARSIFY: 6.3%
BOUND/COEFF STR.: 17.9%
CLIQUE MERGING: 16.1%

Benchmark data based on Gurobi 5.6

Copyright 2017, Gurobi Optimization, Inc.
Multi-Column Reductions

• Fix redundant penalty variables
  • Penalty variables: \( \text{support}(A_j) = 1 \)
  • Multiple penalty variables in a single constraint
    • Some can be fixed if others can accomplish all that is needed

• Parallel columns (say, columns 1 and 2): \( A_1 = sA_2 \)
  • \( u_2 = \infty, c_1 \geq sc_2, 2 \notin l \) or \( |s| = 1, \{1,2\} \subseteq l \): \( x_1 := l_1 \)
  • \( l_2 = -\infty, c_1 \leq sc_2, 2 \notin l \) or \( |s| = 1, \{1,2\} \subseteq l \): \( x_1 := u_1 \)
  • \( c_1 = sc_2, 1,2 \notin l \) or \( |s| = 1, \{1,2\} \subseteq l \): \( x_1' := x_1 + sx_2 \)
  • Detection algorithm: two level hashing plus sorting

• Dominated columns: \( A_1 \geq sA_2 \), only inequalities
  • \( u_2 = \infty, c_1 \geq sc_2, 2 \notin l \) or \( |s| = 1, \{1,2\} \subseteq l \): \( x_1 := l_1 \)
  • Detection algorithm: essentially pair-wise comparison
    • Can be very expensive: needs work limit
Multi-Column Reductions – Performance

benchmark data based on Gurobi 5.6
Full Problem Reductions

- Symmetric variable substitution
  - Integer variables in same orbit can be aggregated if the involved symmetries do not overlap
  - Continuous variables in same orbit can always be aggregated
  - Issue: symmetry detection can sometimes be time consuming!

- Probing
  - Tentatively fix binary $x = 0$ and $x = 1$, propagate fixing to get domain reductions for other variables
    - $x = 0 \rightarrow y \leq u_0, x = 1 \rightarrow y \leq u_1 \Rightarrow y \leq \max\{u_0, u_1\}$ (bound strength.)
    - $x = 0 \rightarrow y = l_y, x = 1 \rightarrow y = u_y \Rightarrow y := l_y + (u_y-l_y) \cdot x$ (substitution)
    - $ax \leq b, x = 1 \rightarrow ay \leq d < b \Rightarrow ay + (b-d) \cdot x \leq b$ (lifting)

- Sequence dependent
  - Can be very time consuming
    - Needs specialized data structures and algorithms

- Implied Integer Detection
  - Primal version: $ax + y = b$, $x$ integer variables, $a \in \mathbb{Z}^n$, $b \in \mathbb{Z} \Rightarrow y$ integer
  - Dual version:
    - One of the inequalities for $y$ will be tight, but do not know which
    - If all those inequalities lead to primal version of implied integer detection, $y$ is implied integer
Full Problem Reductions

benchmark data based on Gurobi 5.6
MIP Building Blocks

• Presolve
  • Tighten formulation and reduce problem size

• Solve continuous relaxations
  • Ignoring integrality
  • Gives a bound on the optimal integral objective

• Cutting planes
  • Cut off relaxation solutions

• Branching variable selection
  • Crucial for limiting search tree size

• Primal heuristics
  • Find integer feasible solutions
Primal and Dual LP

- Primal Linear Program:
  \[ \begin{align*}
  \text{min} & \quad c^T x \\
  \text{s.t.} & \quad Ax = b \\
  & \quad x \geq 0
  \end{align*} \]

- Weighted combination of constraints (y) and bounds (z) yields
  \[ y^T Ax + z^T x \geq y^T b \quad \text{(with } z \geq 0) \]

- Dual Linear Program:
  \[ \begin{align*}
  \text{max} & \quad y^T b \\
  \text{s.t.} & \quad y^T A + z^T = c^T \\
  & \quad z \geq 0
  \end{align*} \]

**Strong Duality Theorem:**

\[ c^T x^* = y^*^T b \]

(if primal and dual are both feasible)
Simplex Algorithm

• **Phase 1**: find some feasible vertex solution
Simplex Algorithm

- **Pricing:** find directions in which objective improves and select one of them
Simplex Algorithm

- **Ratio test**: follow outgoing ray until next vertex is reached
Simplex Algorithm

• Iterate until no more improving direction is found
MIP – LP Relaxation

MIP-optimal solutions

LP-optimal solutions

objective
MIP – LP Relaxation

No feasible solutions can be better than an LP optimum
MIP Building Blocks

- Presolve
  - Tighten formulation and reduce problem size

- Solve continuous relaxations
  - Ignoring integrality
  - Gives a bound on the optimal integral objective

- Cutting planes
  - Cut off relaxation solutions

- Branching variable selection
  - Crucial for limiting search tree size

- Primal heuristics
  - Find integer feasible solutions
MIP – Cutting Planes

fractional LP-optimal solution

objective
MIP – Cutting Planes

new LP-optimal solution

fractional LP-optimal solution

objective
MIP – Cutting Planes

![Diagram of cutting planes](image)

Objective
MIP – Cutting Planes

objective
MIP – Cutting Planes

No feasible solutions can be better than an LP optimum
Cutting Planes – Overview

• General-purpose cutting planes
  • Gomory mixed integer cuts
  • Mixed Integer Rounding (MIR) cuts
  • Flow cover cuts
  • Lift-and-project (L&P) cuts
  • Zero-half and mod-k cuts
  • ...

• Structural cuts
  • Implied bound cuts
  • Knapsack cover cuts
  • GUB cover cuts
  • Clique cuts
  • Multi-commodity-flow (MCF) cuts
  • Flow path cuts
  • ...

Copyright 2017, Gurobi Optimization, Inc.
Mixed Integer Rounding Cuts

- Consider $S := \{(x, y) \in \mathbb{Z} \times \mathbb{R}_{\geq 0} \mid x - y \leq b\}$.

Then, $x - \frac{1}{1 - f_0} y \leq \lfloor b \rfloor$

is valid for $S$ with $f_0 := b - \lfloor b \rfloor$.

- Example: $x - y \leq 2.5$
- MIR cut: $x - 2y \leq 2$
Mixed Integer Rounding Cuts

• Consider \( S := \{(x,y) \in \mathbb{Z} \times \mathbb{R}_{\geq 0} \mid x - y \leq b\} \).

Then,  
\[
x - \frac{1}{1 - f_0} y \leq \lfloor b \rfloor
\]

is valid for \( S \) with \( f_0 := b - \lfloor b \rfloor \).

• Consider \( S := \{(x,y) \in \mathbb{Z}^p \times \mathbb{R}^q_{\geq 0} \mid ax + dy \leq b\} \).

Then,  
\[
\sum \left( |a_i| + \frac{\max\{f_i - f_0, 0\}}{1 - f_0} \right) x_i + \sum \left( \frac{\min\{d_j, 0\}}{1 - f_0} \right) y_j \leq \lfloor b \rfloor
\]

is valid for \( S \) with \( f_i := a_i - \lfloor a_i \rfloor \), \( f_0 := b - \lfloor b \rfloor \).
Mixed Integer Rounding Cuts

• General idea:
  1. Choose non-negative multipliers $\lambda \in \mathbb{R}^m_{\geq 0}$
  2. Aggregated inequality $\lambda^T A x \leq \lambda^T b$ is valid for $P$ because $\lambda \geq 0$
  3. Apply MIR formula to aggregated inequality to produce cutting plane

• Cut separation procedure of Marchand and Wolsey (1998, 2001):
  1. Start with one constraint of the problem (do this for each one), call this the "current aggregated inequality"
  2. Apply MIR procedure to current aggregated inequality
     (a) Complement variables if LP solution is closer to upper bound
     (b) For each $a_j$ in constraint and each of $\delta \in \{1,2,4,8\}$ divide the constraint by $\delta |a_j|$ and apply MIR formula to resulting scaled constraint
     (c) Choose most violated cut from this set of MIR cuts
     (d) Check if complementing one more (or one less) variable yields larger violation
  3. If no violated cut was found (and did not yet reach aggregation limit):
     (a) Add another problem constraint to the current aggregated inequality such that a continuous variable with LP value not at a bound is canceled
     (b) Go to 2
Gomory Mixed Integer Cuts

• Just an alternative way to aggregate constraints
• Read them from an optimal simplex tableau:
  • Let i be a basis index with \( x_i^* \notin \mathbb{Z} \)
  • Choose \( \lambda^T = (A_B^{-1})_i \).
  • Resulting aggregated inequality: \( x_i + (A_B^{-1})_i A_N x_N \leq (A_B^{-1})_i b \)
• Apply MIR formula on resulting aggregated inequality
• In theory, always produces a violated cutting plane
• Practical issues:
  • Gomory Mixed Integer Cuts can be pretty dense
  • Numerics (in particular for higher rank cuts) can be very challenging
• But:
  • If done right, GMICs (together with MIRs) are currently the most important cutting planes in practice
Knapsack Cover Cuts

- A (binary) knapsack is a constraint $ax \leq b$ with
  - $a_i \geq 0$ the weight of item $i$, $i = 1, \ldots, n$
  - $b \geq 0$ the capacity of the knapsack
- An index set $C \subseteq \{1, \ldots, n\}$ is called a cover, if $\sum_{i \in C} a_i > b$
- A cover $C$ entails a cover inequality
  $$\sum_{i \in C} x_i \leq |C| - 1$$
- Interesting for cuts: minimal covers
  $$\sum_{i \in C} a_i > b \quad \text{and} \quad \sum_{i \in C'} a_i \leq b \quad \text{for all} \quad C' \subsetneq C$$
Knapsack Cover Cuts – Example

• Consider knapsack $3x_1 + 5x_2 + 8x_3 + 10x_4 + 17x_5 \leq 24$, $x \in \{0,1\}^5$
• A minimal cover is $C = \{1,2,3,4\}$
• Resulting cover inequality: $x_1 + x_2 + x_3 + x_4 \leq 3$

• Lifting
  • If $x_5 = 1$, then $x_1 + x_2 + x_3 + x_4 \leq 1$
  • Hence, $x_1 + x_2 + x_3 + x_4 + 2x_5 \leq 3$ is valid
  • Need to solve knapsack problem $\alpha_j := d_0 - \max\{dx | ax \leq b - a_j\}$ to find lifting coefficient for variable $x_j$
    • Use dynamic programming to solve knapsack problem
Cutting Planes – Performance

Achterberg and Wunderling: "Mixed Integer Programming: Analyzing 12 Years of Progress" (2013)
benchmark data based on CPLEX 12.5
MIP Building Blocks

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

P₁  P₂

objective
MIP – Branching

P\textsubscript{1} \hspace{1cm} P\textsubscript{2}

objective
MIP – Branching

\[ \text{P}_1 \quad \text{P}_2 \]

\[ \text{objective} \]
MIP – Branching

Another improvement in dual bound
LP based Branch-and-Bound

Solve LP relaxation: $v=3.5$ (fractional)
LP based Branch-and-Bound
LP based Branch-and-Bound
LP based Branch-and-Bound

Remarks:
(1) GAP = 0 ⇒ Proof of optimality
(2) In practice: good quality solution often enough
Solving a MIP Model

![Graph showing the solution and bound over time.](image-url)
Branching Variable Selection

• Given a relaxation solution $x^*$
  • Branching candidates:
    • Integer variables $x_j$ that take fractional values
      • $x_j = 3.7$ produces two child nodes ($x \leq 3$ or $x \geq 4$)
  • Need to pick a variable to branch on
    • Choice is crucial in determining the size of the overall search tree
Branching Variable Selection

• 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 $\text{cost}(x_j) = \Delta \text{obj} / \Delta x_j$ for each branch
  • Store results in a pseudo-cost table
    • Two entries per integer variable
      • Average down cost
      • Average up cost
  • Use table to predict cost of a future branch
Pseudo-Costs

- Use historical data to predict impact of a branch:
  - Record $\text{cost}(x_j) = \Delta \text{obj} / \Delta x_j$ for each branch
  - Store results in a pseudo-cost table
    - Two entries per integer variable
      - Average down cost
      - Average up cost
  - Use table to predict cost of a future branch

\[ c^* = 13 \]
\[ x^* = 2.7 \]
\[ \Delta \text{obj}/\Delta x = 7/0.7 = 10 \]
\[ c^* = 20 \]
\[ x \geq 2 \]
\[ \Delta \text{obj}/\Delta x = 6/0.3 = 20 \]

\[ c^* = 19 \]
\[ v \geq 4 \]
Pseudo-Costs

- Use historical data to predict impact of a branch:
  - Record $\text{cost}(x_j) = \Delta \text{obj} / \Delta x_j$ for each branch
  - Store results in a pseudo-cost table
    - Two entries per integer variable
      - Average down cost
      - Average up cost
  - Use table to predict cost of a future branch

\[
\begin{align*}
\text{pseudo costs:} & \\
\downcost(x) &= 10 \\
\upcost(x) &= 20 \\
\text{down estimate:} & \\
\hat{c}' &= 17 + 0.4 \cdot 10 = 21 \\
\text{up estimate:} & \\
\hat{c}' &= 17 + 0.6 \cdot 20 = 29
\end{align*}
\]
Pseudo-Costs 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, 2005]
  • Do not rely on historical data until pseudo-cost for a variable has been recomputed \( r \) times
Branching Rules – Performance

Achterberg and Wunderling: "Mixed Integer Programming: Analyzing 12 Years of Progress" (2013)  
benchmark data based on CPLEX 12.5  
Achterberg, Koch, and Martin: "Branching Rules Revisited" (2005)
MIP Building Blocks

- Presolve
  - Tighten formulation and reduce problem size

- Solve continuous relaxations
  - Ignoring integrality
  - Gives a bound on the optimal integral objective

- Cutting planes
  - Cut off relaxation solutions

- Branching variable selection
  - Crucial for limiting search tree size

- Primal heuristics
  - Find integer feasible solutions
Primal Heuristics

- Try to find good integer feasible solutions quickly
  - Better pruning during search due to better bound
  - Reach desired gap faster
  - Often important in practice: quality of solution after fixed amount of time

- Start heuristics
  - Try to find integer feasible solution, usually "close" to LP solution

- Improvement heuristics
  - Given integer feasible solution, try to find better solution
Primal Heuristics Explained on Twitter

Matteo Fischetti @MFischetti · 1 Std.
@AchterbergT  hmmm, challenging suggestion: write a full scientific paper in a tweet! I would be tempted...

Matteo Fischetti @MFischetti · 1 Std.
@AchterbergT e.g. #LocalBranching take a 0-1 MIP and a solution $x^*$, bound Hamming distance from $x^*$ through a linear cut, and solve again.

Matteo Fischetti @MFischetti · 1 Std.
@AchterbergT #RINS take a MIP, a feasible sol. $x_-$ and the LP relaxation sol. $x^*$, fix all components that agree, and solve as a MIP.

Matteo Fischetti @MFischetti · 1 Std.
@AchterbergT #FP Take MIP and LP sol. $x^*$, round it, solve LP again but minimizing Hamming distance from rounded sol. Repeat. Serve it warm.
Primal Heuristics – Examples

• Start heuristics
  • Rounding heuristics: round LP solution to integral values
    • Potentially, try to fix constraint infeasibilities
  • Fix-and-dive heuristics: fix variables, propagate, resolve LP
  • Feasibility pump: push LP solution towards integrality by modifying objective
  • RENS: Solve sub-MIP in neighborhood of LP solution

• Improvement heuristics
  • 1-Opt and 2-Opt: Modify one or two variables to get better objective
  • Local Branching: Solve sub-MIP in neighborhood of MIP solution
  • Mutation: Solve sub-MIP in neighborhood of MIP solution
  • Crossover: Solve sub-MIP in neighborhood of 2 or more MIP solutions
  • RINS: Solve sub-MIP in neighborhood of LP and MIP solution
Primal Heuristics – Performance

Berthold: "Primal Heuristics for Mixed Integer Programs" (2006)
benchmark data based on SCIP 0.82b
Primal Heuristics – Measuring Performance

• Is time to optimality a good measure to assess impact of heuristics?
  • Goal of heuristics is to provide good solutions quickly
  • Faster progress in dual bound due to additional pruning is only secondary
  • Often important for practitioners:
    • Find any feasible solution quickly to validate that model is reasonable
    • Find good solution in reasonable time frame

• Primal gap: 
  \[ \gamma^p(\bar{x}) = \frac{|c^T x^* - c^T \bar{x}|}{\max\{|c^T x^*|, |c^T \bar{x}|\}} \]

• Primal gap function: 
  \[ p(t) = \begin{cases} 
  1, & \text{if no incumbent until time } t \\
  \gamma^p(\bar{x}(t)), & \text{with } \bar{x}(t) \text{ being incumbent at time } t 
  \end{cases} \]

• Primal integral: 
  \[ P(T) = \int_{t=0}^{T} p(t) dt \]
Primal Integral

Objective

P(T)

Time

Solution

Bound

\(c^T x^*\)
Primal Heuristics – Performance

Berthold (2014): "Heuristic algorithms in global MINLP solvers"
benchmark data based on SCIP 3.0.2
Putting It All Together
Branch-and-Cut

- Presolving
- Conflict Analysis
- Cutting Planes
- Node Selection
- Node Presolve
- LP Relaxation
- Heuristics
- Branching
Branch-and-Cut

Gurobi Optimizer version 6.0.0 (linux64)  
Copyright (c) 2014, Gurobi Optimization, Inc.

Read MPS format model from file /models/mip/roll3000.mps.bz2  
Reading time = 0.03 seconds

roll3000: 2295 rows, 1166 columns, 29386 nonzeros  
Optimize a model with 2295 rows, 1166 columns and 29386 nonzeros

Coefficient statistics:
Matrix range [2e-01, 3e+02]  
Objective range [1e+00, 1e+00]  
Bounds range [1e+00, 1e+09]  
RHS range [6e-01, 1e+03]

Presolve removed 1308 rows and 311 columns  
Presolve time: 0.08s  
Presolved: 987 rows, 855 columns, 19346 nonzeros

Variable types: 211 continuous, 644 integer (545 binary)

Root relaxation: objective 1.112003e+04, 1063 iterations, 0.03 seconds

<table>
<thead>
<tr>
<th>Nodes</th>
<th>Current Node</th>
<th>Objective Bounds</th>
<th>Work</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Expl Unexpl</td>
<td>Obj Depth IntInf</td>
<td>Incumbent BestBd Gap</td>
</tr>
<tr>
<td>0</td>
<td>0 11120.0279</td>
<td>0 154</td>
<td>- 11120.0279 - -</td>
</tr>
<tr>
<td>0</td>
<td>0 11526.8918</td>
<td>0 207</td>
<td>- 11526.8918 - -</td>
</tr>
<tr>
<td>0</td>
<td>0 11896.9710</td>
<td>0 190</td>
<td>- 11896.9710 - -</td>
</tr>
</tbody>
</table>
Branch-and-Cut

Which open node should be processed next?
Branch-and-Cut

Presolving → Node Selection → Conflict Analysis → Node Presolve → LP Relaxation

Presolved: 987 rows, 855 columns, 19346 nonzeros
Variable types: 211 continuous, 644 integer (545 binary)

Root relaxation: objective 1.112003e+04, 1063 iterations, 0.03 seconds

<table>
<thead>
<tr>
<th>Nodes</th>
<th>Current Node</th>
<th>Objective Bounds</th>
<th>Work</th>
</tr>
</thead>
<tbody>
<tr>
<td>Expl</td>
<td>Unexpl</td>
<td>Obj</td>
<td>Depth</td>
</tr>
<tr>
<td>0</td>
<td>0 11120.0279</td>
<td>0 154</td>
<td>-</td>
</tr>
<tr>
<td>0</td>
<td>0 11526.8918</td>
<td>0 207</td>
<td>-</td>
</tr>
<tr>
<td>0</td>
<td>0 11896.9710</td>
<td>0 190</td>
<td>-</td>
</tr>
<tr>
<td>...</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>H</td>
<td>327 218</td>
<td>13135.000000</td>
<td>12455.2162</td>
</tr>
<tr>
<td>H</td>
<td>380 264</td>
<td>13093.000000</td>
<td>12455.2162</td>
</tr>
<tr>
<td>H</td>
<td>413 286</td>
<td>13087.000000</td>
<td>12455.2162</td>
</tr>
<tr>
<td>1066</td>
<td>702 12356.2676</td>
<td>31 192</td>
<td>13087.000000</td>
</tr>
</tbody>
</table>
If node presolve or LP relaxation proves infeasibility of the current node:
- Analyze conflict to derive conflict constraint
- Can be used as cutting plane and for node presolve

See Jakob Witzig's talk on Wednesday
### Branch-and-Cut

**Presolving**

- Presolved: 987 rows, 855 columns, 19346 nonzeros
- Variable types: 211 continuous, 644 integer (545 binary)

**Root relaxation:**
- Objective: 1.112003e+04, 1063 iterations, 0.03 seconds

<table>
<thead>
<tr>
<th>Nodes</th>
<th>Current Node</th>
<th>Objective Bounds</th>
<th>Work</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Expl Unexpl</td>
<td>Obj</td>
<td>Depth</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>11120.0279</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>11526.8918</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>11896.9710</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>12151.4022</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>12278.3391</td>
<td>0</td>
</tr>
<tr>
<td>5485</td>
<td>634</td>
<td>12885.3652</td>
<td>52</td>
</tr>
</tbody>
</table>

**Cutting planes:**
- Learned: 4
- Gomory: 46
- Cover: 39
- Implied bound: 8
- Clique: 2
- MIR: 112
- Flow cover: 27
- GUB cover: 11
- Zero half: 91

Explored 6808 nodes (357915 simplex iterations) in 27.17 seconds
Thread count was 4 (of 8 available processors)
**Branch-and-Cut**

Presolved: 987 rows, 855 columns, 19346 nonzeros
Variable types: 211 continuous, 644 integer (545 binary)

Root relaxation: objective 1.112003e+04, 1063 iterations, 0.03 seconds

<table>
<thead>
<tr>
<th>Nodes</th>
<th>Current Node</th>
<th>Objective Bounds</th>
<th>Work</th>
</tr>
</thead>
<tbody>
<tr>
<td>Expl</td>
<td>Unexpl</td>
<td>Obj</td>
<td>Depth</td>
</tr>
</tbody>
</table>

- **Presolving:**
- **Conflict Analysis:**
- **Cutting Plan:**
- **Heuristics:**
- **Branching:**

Copyright 2017, Gurobi Optimization, Inc.
Presolved: 987 rows, 855 columns, 19346 nonzeros
Variable types: 211 continuous, 644 integer (545 binary)

Root relaxation: objective 1.112003e+04, 1063 iterations, 0.03 seconds

<table>
<thead>
<tr>
<th>Nodes</th>
<th>Current Node</th>
<th>Objective Bounds</th>
<th>Work</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Expl Unexpl</td>
<td>Obj Depth IntInf</td>
<td>Incumbent BestBd Gap</td>
</tr>
</tbody>
</table>
| 0     | 0 11120.0279 0 154 - 11120.0279 - - 0s 0 11526.8918 0 207 - 11526.8918 - - 0s 0 11896.9710 0 190 - 11896.9710 - - 0s ...
| H    | 0 12448.7684 0 181 15890.0000 12448.7684 21.7% - 0s 0 12956.2676 31 192 13087.0000 12629.5426 3.50% 37.2 5s 1097 724 12671.8285 8 147 13087.0000 12671.8285 3.17% 41.6 10s 1135 710 12732.5601 32 126 12890.0000 12727.1362 1.26% 44.6 15s 3416 887 12839.9880 46 136 12890.0000 12780.7059 0.85% 49.7 20s 5485 634 12885.3652 52 143 12890.0000 12829.0134 0.47% 54.5 25s ...

Branching
Performance Impact of MIP Solver Components (CPLEX 12.5 or SCIP)

- Presolving: 7.6x
- Conflict Analysis: 1.1x
- Cutting Planes: 3.8x
- LP Relaxation: 1.5x
- Node Selection: 1.3x
- Node Presolve: 1.2x
- Heuristics: 1.5x
- Branching: 4.6x

Achterberg and Wunderling: "Mixed Integer Programming: Analyzing 12 Years of Progress" (2013)
Parallelization

- Parallelization opportunities
  - Parallel probing during presolve
    - Almost no improvement
  - Use barrier or concurrent LP for initial LP relaxation solve
    - Only helps for large models
  - Run heuristics or other potentially useful algorithms in parallel to the root cutting plane loop
    - Moderate performance improvements: 20-25%
    - Does not scale beyond a few threads
  - Solve branch-and-bound nodes in parallel
    - Main speed-up for parallel MIP
    - Performance improvement depends a lot on shape of search tree
    - Typically scales relatively well up to 8 to 16 threads
Parallelization

- Parallelization issues
  - Determinism
  - Load balancing
  - CPU heat and memory bandwidth
    - Additional threads slow down main thread
  - Root node does not parallelize well
    - Sequential runtime of root node imposes limits on parallelization speed-up
    - Amdahl's law
  - A dive in the search tree cannot be parallelized
    - Parallelization only helps if significant number of dives necessary to solve model
benchmark data based on CPLEX 12.5, models with ≥ 100 seconds solve time
No Further Questions? Enjoy Your Coffee Break!