Integrating Solution Methods through Duality

John Hooker

Microsoft Tractability Workshop
Cambridge, UK, July 2010
Outline

• What makes a problem easy
• Exploiting structure with primal-dual-dual methods
• Simple example
• A closer look at inference duality
• Examples, with computational results
What makes a problem easy

- A problem is easy when we know the answer.
What makes a problem easy

- A problem is **easy** when we **know the answer**.
  - An **epistemic** concept of tractability.
  - Tractability is relative to the **solution method**.
What makes a problem easy

- A problem is **easy** when we **know the answer**.
  - An **epistemic** concept of tractability.
  - Tractability is relative to the **solution method**.

- **Pigeon hole principle**
  - Resolution proof – exponential
  - Cutting plane proof – polynomial
  - Counting argument – trivial
What makes a problem easy

- No need to view some problems as inherently hard.
  - “Easy” problems can be hard if we don’t exploit structure (e.g., linear programming).
  - “Hard” problems can be solved if we exploit structure (we do this for a living).
What makes a problem easy

- No need to view some problems as inherently hard.
  - “Easy” problems can be hard if we don’t exploit structure (e.g., linear programming).
  - “Hard” problems can be solved if we exploit structure (we do this for a living).

- A good solution methods knows more about the problem.
  - That is, it exploits problem structure.
Two ways to exploit structure

- **Inference** and **relaxation**.
  - Used throughout **optimization**
  - Provides a principle for **unifying** and **integrating** solution methods.
Two ways to exploit structure

- **Inference and relaxation.**
  - Used throughout **optimization**
  - Provides a principle for **unifying** and **integrating** solution methods.

- Well-chosen inference and relaxation techniques can **accelerate solution.**
  - Techniques can be drawn from **existing methods**, which results in **integration**.
  - **New** techniques can be invented for the problem.
We can integrate:

- **MIP**
  - Mixed integer programming.

- **CP**
  - Constraint programming.

- **GO**
  - Global optimization
  - Nonlinear, nonconvex
  - Discrete and/or continuous variables

- **SAT**
  - Propositional satisfiability

- **LS**
  - Local search, metaheuristics
Two ways to exploit structure

- **Inference**
  - Use knowledge of problem structure to reveal *hidden information*.
  - This can exclude *unpromising* areas of the search space.
Two ways to exploit structure

- **Relaxation**
  - Use knowledge of problem structure to design a larger but simpler search space.
  - Solution of relaxation may be near solution of original problem.
Two dualities

• Duality of search and inference.
  • Search looks for a certificate of feasibility.
  • The inference dual looks for a certificate (proof) of infeasibility (or optimality).

• Duality of search and relaxation.
  • Search enumerates restrictions of the problem.
  • The relaxation dual enumerates (parameterized) relaxations of the problem.
Primal-dual-dual methods

- Primal methods.
  - Enumerate possible solutions (in general, problem restrictions).

- Dual methods.
  - Enumerate proofs or relaxations.

- Primal-dual methods.
  - Solve the primal and a dual simultaneously.

- Primal-dual-dual methods.
  - Solve the primal and both duals simultaneously.
  - Strategy of most successful optimization methods.
Classical solution methods

- **CP solver**
  - **Search**: Branching
  - **Inference**: Filtering
  - **Relaxation**: Domain store

- **MILP solver**
  - **Search**: Branching
  - **Inference**: Cutting planes, presolve, reduced cost variable fixing
  - **Relaxation**: LP

- **Benders**
  - **Search**: Enumerate subproblems.
  - **Inference**: Benders cuts
  - **Relaxation**: Master problem
Classical solution methods

- **Global optimization**
  - **Search:** Enumerate boxes
  - **Inference:** Domain reduction, dual-based variable bounding
  - **Relaxation:** Convexification

- **SAT**
  - **Search:** Branching, dynamic backtracking, etc.
  - **Inference:** Conflict clauses
  - **Relaxation:** Same as restriction

- **Local search**
  - **Search:** Enumerate neighborhoods.
  - **Inference:** Tabu list, etc.
  - **Relaxation:** Same as restriction
Interaction

Relaxation

Strengthens
- Fixes variables
- Reduces domains
- Adds IP cuts
- Adds conflict clause
- Adds Benders cuts
- Shrinks box
- Creates neighborhood

Activates
- IP cut
- Filtering/propagation
- Reduced domain
- Benders cut
- Subproblem dual

Guides
- Separating cut

Defines
- Identifies next branch
- Fractional variable
- Nonsingleton domain
- Violated constraint
- Defines subproblem
- Solution of master
- Defines neighborhood
- Center on previous solution

Restriction
Simple example: Freight transfer

- Transport 42 tons of freight using 8 trucks, which come in 4 sizes...

<table>
<thead>
<tr>
<th>Truck size</th>
<th>Number available</th>
<th>Capacity (tons)</th>
<th>Cost per truck</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>3</td>
<td>7</td>
<td>90</td>
</tr>
<tr>
<td>2</td>
<td>3</td>
<td>5</td>
<td>60</td>
</tr>
<tr>
<td>3</td>
<td>3</td>
<td>4</td>
<td>50</td>
</tr>
<tr>
<td>4</td>
<td>3</td>
<td>3</td>
<td>40</td>
</tr>
</tbody>
</table>
Problem formulation

\[
\begin{align*}
\text{min} & \quad 90x_1 + 60x_2 + 50x_3 + 40x_4 \\
7x_1 + 5x_2 + 4x_3 + 3x_4 & \geq 42 \\
x_1 + x_2 + x_3 + x_4 & \leq 8 \\
x_i & \in \{0, 1, 2, 3\}
\end{align*}
\]

<table>
<thead>
<tr>
<th>Truck type</th>
<th>Number available</th>
<th>Capacity (tons)</th>
<th>Cost per truck</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>3</td>
<td>7</td>
<td>90</td>
</tr>
<tr>
<td>2</td>
<td>3</td>
<td>5</td>
<td>60</td>
</tr>
<tr>
<td>3</td>
<td>3</td>
<td>4</td>
<td>50</td>
</tr>
<tr>
<td>4</td>
<td>3</td>
<td>3</td>
<td>40</td>
</tr>
</tbody>
</table>
Inference: Bounds propagation

Standard CP technique

\[
\begin{align*}
\text{min} & \quad 90x_1 + 60x_2 + 50x_3 + 40x_4 \\
7x_1 + 5x_2 + 4x_3 + 3x_4 & \geq 42 \\
x_1 + x_2 + x_3 + x_4 & \leq 8 \\
x_i & \in \{0, 1, 2, 3\}
\end{align*}
\]

\[
x_1 \geq \left[ \frac{42 - 5 \cdot 3 - 4 \cdot 3 - 3 \cdot 3}{7} \right] = 1
\]
Bounds propagation

Standard CP technique

\[
\begin{align*}
\min \quad & 90x_1 + 60x_2 + 50x_3 + 40x_4 \\
7x_1 + 5x_2 + 4x_3 + 3x_4 & \geq 42 \\
x_1 + x_2 + x_3 + x_4 & \leq 8 \\
x_1 \in \{1,2,3\}, \quad & x_2, x_3, x_4 \in \{0,1,2,3\}
\end{align*}
\]

Reduced domain

\[
x_1 \geq \left\lfloor \frac{42 - 5 \cdot 3 - 4 \cdot 3 - 3 \cdot 3}{7} \right\rfloor = 1
\]
Bounds propagation

Standard CP technique

Reduced domain

In general:

Bounds propagation aims for **bounds consistency**

Domain filtering aims for **hyperarc consistency (GAC)**
Relaxation: Linear programming

\[
\begin{align*}
\min & \quad 90x_1 + 60x_2 + 50x_3 + 40x_4 \\
7x_1 + 5x_2 + 4x_3 + 3x_4 & \geq 42 \\
x_1 + x_2 + x_3 + x_4 & \leq 8 \\
0 \leq x_i \leq 3, & \quad x_1 \geq 1
\end{align*}
\]

Replace domains with bounds

This is a linear programming problem, which has a simplified search space (polyhedron).

Its optimal value provides a lower bound on optimal value of original problem.

The optimal solution may be close to, or equal to, an optimal solution of the original problem.
Inference: cutting planes (valid inequalities)

\[
\begin{align*}
\text{min } & \quad 90x_1 + 60x_2 + 50x_3 + 40x_4 \\
7x_1 + 5x_2 + 4x_3 + 3x_4 & \geq 42 \\
x_1 + x_2 + x_3 + x_4 & \leq 8 \\
0 & \leq x_i \leq 3, \quad x_1 \geq 1
\end{align*}
\]

We can create a **tighter** relaxation with the addition of **cutting planes**.
Inference: cutting planes (valid inequalities)

\[ \begin{align*}
\min & \quad 90x_1 + 60x_2 + 50x_3 + 40x_4 \\
& \quad 7x_1 + 5x_2 + 4x_3 + 3x_4 \geq 42 \\
& \quad x_1 + x_2 + x_3 + x_4 \leq 8 \\
& \quad 0 \leq x_i \leq 3, \quad x_1 \geq 1
\end{align*} \]

A cutting plane excludes ("cut off") solutions of the continuous relaxation…

but no feasible solutions of original problem.
Inference: cutting planes (valid inequalities)

\[
\begin{align*}
\text{min} & \quad 90x_1 + 60x_2 + 50x_3 + 40x_4 \\
7x_1 + 5x_2 + 4x_3 + 3x_4 & \geq 42 \\
x_1 + x_2 + x_3 + x_4 & \leq 8 \\
0 \leq x_i \leq 3, & \quad x_1 \geq 1
\end{align*}
\]

{1,2} is a packing

...because $7x_1 + 5x_2$ alone cannot satisfy the inequality, even with $x_1 = x_2 = 3$. 
Inference: cutting planes (valid inequalities)

\[
\begin{align*}
\min & \quad 90x_1 + 60x_2 + 50x_3 + 40x_4 \\
& \quad 7x_1 + 5x_2 + 4x_3 + 3x_4 \geq 42 \\
& \quad x_1 + x_2 + x_3 + x_4 \leq 8 \\
& \quad 0 \leq x_i \leq 3, \quad x_1 \geq 1
\end{align*}
\]

\{1,2\} is a packing

So, \[4x_3 + 3x_4 \geq 42 - (7 \cdot 3 + 5 \cdot 3)\] Knapsack cut

which implies \[x_3 + x_4 \geq \left\lfloor \frac{42 - (7 \cdot 3 + 5 \cdot 3)}{\max\{4,3\}} \right\rfloor = 2\]
Cutting planes (valid inequalities)

\[
\begin{align*}
\text{min} \quad & 90x_1 + 60x_2 + 50x_3 + 40x_4 \\
& 7x_1 + 5x_2 + 4x_3 + 3x_4 \geq 42 \\
x_1 + x_2 + x_3 + x_4 \leq 8 \\
& 0 \leq x_i \leq 3, \quad x_1 \geq 1
\end{align*}
\]

<table>
<thead>
<tr>
<th>Maximal Packings</th>
<th>Knapsack cuts</th>
</tr>
</thead>
<tbody>
<tr>
<td>{1,2}</td>
<td>(x_3 + x_4 \geq 2)</td>
</tr>
<tr>
<td>{1,3}</td>
<td>(x_2 + x_4 \geq 2)</td>
</tr>
<tr>
<td>{1,4}</td>
<td>(x_2 + x_3 \geq 3)</td>
</tr>
</tbody>
</table>

Knapsack cuts corresponding to nonmaximal packings can be nonredundant.
Continuous relaxation with cuts

\[
\begin{align*}
\text{min} & \quad 90x_1 + 60x_2 + 50x_3 + 40x_4 \\
7x_1 + 5x_2 + 4x_3 + 3x_4 & \geq 42 \\
x_1 + x_2 + x_3 + x_4 & \leq 8 \\
0 \leq x_i & \leq 3, \quad x_1 \geq 1 \\
\end{align*}
\]

\[
\begin{align*}
x_3 + x_4 & \geq 2 \\
x_2 + x_4 & \geq 2 \\
x_2 + x_3 & \geq 3 \\
\end{align*}
\]

Knapsack cuts

Optimal value of 523.3 is a lower bound on optimal value of original problem.
Primal-dual-dual method

**Search:** branching

**Inference:** Bounds propagation, cutting planes

**Relaxation:** LP, domain store
Primal-dual-dual method

Propagate bounds and solve relaxation. Since solution of relaxation is infeasible, branch.

\begin{align*}
  x_1 &\in \{1,2,3\} \\
  x_2 &\in \{0,1,2,3\} \\
  x_3 &\in \{0,1,2,3\} \\
  x_4 &\in \{0,1,2,3\} \\
  x &= (2\frac{1}{3},3,2\frac{2}{3},0) \\
  \text{value} &= 523\frac{1}{3}
\end{align*}
Primal-dual-dual method

Branch on a variable with nonintegral value in the relaxation.

\[ x_1 \in \{1, 2\} \]
\[ x_1 = 3 \]
\[ x_2 \in \{0123\} \]
\[ x_3 \in \{0123\} \]
\[ x_4 \in \{0123\} \]
\[ x = (2\frac{1}{3}, 3, 2\frac{2}{3}, 0) \]
\[ \text{value} = 523\frac{1}{3} \]
Primal-dual-dual method

Propagate bounds and solve relaxation.

Since relaxation is infeasible, backtrack.

\[
x_1 \in \{123\}
x_2 \in \{0123\}
x_3 \in \{0123\}
x_4 \in \{0123\}
\]
\[
x = (2\frac{1}{3}, 3, 2\frac{2}{3}, 0)
\]
\[
\text{value} = 523\frac{3}{4}
\]

\[
x_1 \in \{1, 2\}
\]
\[
x_1 = 3
\]
Propagate bounds and solve relaxation. Branch on nonintegral variable.
Primal-dual-dual method

Branch again.

\[ x_1 \in \{123\} \]
\[ x_2 \in \{0123\} \]
\[ x_3 \in \{0123\} \]
\[ x_4 \in \{0123\} \]
\[ x = (2\frac{1}{3},3,2\frac{2}{3},0) \]
\[ \text{value} = 523\frac{3}{4} \]

infeasible relaxation

\[ x_1 \in \{ 12 \} \]
\[ x_2 \in \{ 23 \} \]
\[ x_3 \in \{ 123 \} \]
\[ x_4 \in \{ 123 \} \]
\[ x = (3,2.6,2,0) \]
\[ \text{value} = 526 \]

\[ x_1 = 3 \]
\[ x_2 = 3 \]
\[ x_3 = 3 \]

\[ x_1 \in \{ 1,2 \} \]
\[ x_2 \in \{ 0,1,2 \} \]
\[ x_3 \in \{ 1,2 \} \]
\[ x_4 \in \{ 0123 \} \]
\[ x = (3,2,2\frac{3}{4},0) \]
\[ \text{value} = 527\frac{1}{2} \]
Primal-dual-dual method

Solution of relaxation is integral and therefore feasible in the original problem.

This becomes the incumbent solution.
Primal-dual-dual method

Solution is nonintegral, but we can backtrack because value of relaxation is no better than incumbent solution.

Slide 37
Primal-dual-dual method

Another feasible solution found.

No better than incumbent solution, which is optimal because search has finished.
A closer look at inference duality

- Formulation of inference dual
- LP, Lagrangean, surrogate duals are special cases
A closer look at inference duality

- **Formal definition** of inference dual
  - LP, Lagrangean, surrogate duals are special cases

- Nogood-based search
  - Solution of inference dual provides **nogoods** to guide search.
  - **Benders cuts** and **conflict clauses** in SAT are examples.
A closer look at inference duality

- **Formal definition** of inference dual
  - LP, Lagrangean, surrogate duals are special cases

- Nogood-based search
  - Solution of inference dual provides *nogoods* to guide search.
  - *Benders cuts* and *conflict clauses* in SAT are examples.

- Example: SAT
  - DPLL
  - DPLL + conflict clauses
  - Partial order dynamic backtracking
A closer look at inference duality

- Recognition of inference dual structure can lead to massive speedups.
  - For example, **logic-based Benders** decomposition
  - As in **machine scheduling** example to follow.
Inference duality

- **All optimization duals** are inference duals
  - Also **relaxation duals**

- Solution of inference dual is **proof** of optimality
  - Primal in **co-NP** when dual is in **NP**

- **Postoptimality analysis**
  - Result of altering premises of proof
Inference dual

**Primal**

\[
\min_{x \in S} f(x)
\]

**Dual**

\[
\max_{\nu} \nu \\
\text{s.t. } x \in S \implies \nu \geq f(x)
\]

\[
P \in \mathcal{P}
\]

- Dual is defined relative to an **inference method**.
- Strong duality applies if inference method is **complete**.

Follows using proof \( P \).
Example: LP dual

<table>
<thead>
<tr>
<th>Primal</th>
<th>Dual</th>
</tr>
</thead>
<tbody>
<tr>
<td>(\text{min } cx)</td>
<td>(\text{max } v)</td>
</tr>
<tr>
<td>(Ax \geq b)</td>
<td>({Ax \geq b} \Rightarrow cx \geq v)</td>
</tr>
<tr>
<td>(x \geq 0)</td>
<td>(\lambda A \leq c)</td>
</tr>
</tbody>
</table>

\[ P \in \mathcal{P} \leftarrow \text{Nonnegative linear combination + domination} \]

\[ Ax \geq b \Rightarrow cx \geq v \text{ iff } \lambda Ax \geq \lambda b \text{ dominates } cx \geq v \text{ for some } \lambda \geq 0 \]

\(\lambda A \leq c\) and \(\lambda b \geq v\)

- Inference method is **complete** (assuming feasibility) due to Farkas Lemma.
- So we have **strong duality** (assuming feasibility).
Example: Lagrangean dual

<table>
<thead>
<tr>
<th>Primal</th>
<th>Dual</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\min f(x)$</td>
<td>$\max \nu$</td>
</tr>
<tr>
<td>$g(x) \geq 0$</td>
<td>$\max v = \max \min {f(x) - \lambda g(x)}$</td>
</tr>
<tr>
<td>$x \in S$</td>
<td>$g(x) \geq b \implies f(x) \geq \nu$</td>
</tr>
<tr>
<td>$P \in \mathcal{P}$</td>
<td>$\lambda g(x) \geq 0 \quad \text{dominates} \ f(x) - \nu \geq 0$ for some $\lambda \geq 0$</td>
</tr>
</tbody>
</table>

Nonnegative linear combination + domination

$g(x) \geq 0 \implies f(x) \geq \nu$ iff $\lambda g(x) \leq f(x) - \nu$ for all $x \in S$

That is, $\nu \leq f(x) - \lambda g(x)$ for all $x \in S$

Or $\nu \leq \min_{x \in S} \{f(x) - \lambda g(x)\}$

• Inference method is incomplete

Slide 46
Example: Surrogate dual

Primal

\[ \begin{align*}
\min f(x) \\
g(x) \geq 0 \\
x \in S
\end{align*} \]

Dual

\[ \begin{align*}
\max v \\
g(x) \geq b \Rightarrow f(x) \geq v \\
P \in \mathcal{P}
\end{align*} \]

\[ \begin{align*}
g(x) \geq 0 \Rightarrow f(x) \geq v \\
\lambda g(x) \geq 0 \text{ implies } f(x) \geq v \\
\text{for some } \lambda \geq 0
\end{align*} \]

Any \( x \in S \) with \( \lambda g(x) \geq 0 \) satisfies \( f(x) \geq v \)

So, \( \min \{ f(x) \mid \lambda g(x) \leq 0, x \in S \} \geq v \)

\[ \text{• Inference method is incomplete} \]
Nogood-based search

- All search is nogood-based search
  - Each solution examined generates a nogood.
  - Next solution must satisfy current nogood set.

- Nogoods are derived by solving inference dual of the subproblem.
  - Subproblem is normally defined by fixing variables to current values in the search.
Example: Logic-based Benders

Partition variables $x, y$ and search over values of $x$

$$\min_{(x, y) \in S} f(x, y)$$

Subproblem results from fixing $x$

$$\min_{(\bar{x}, y) \in S} f(\bar{x}, y)$$

Let proof $P$ be solution of subproblem dual for $x = \bar{x}$

Let $B(P, x)$ be lower bound obtained by $P$ for given $x$.

Add Benders cut $v \geq B(P, x)$ to master problem:

$$\min v$$

Solve master problem for next $\bar{x}$ Benders cuts
Classical Benders

Partition variables $x, y$ and search over values of $x$

Subproblem results from fixing $x$

$$\min f(x) + cy$$

$$g(x) + Ay \geq b$$

$$x \in D_x, \ y \geq 0$$

$$\min f(x^k) + cy$$

$$Ay \geq b - g(x^k) \quad (\lambda)$$

$$y \geq 0$$

Let proof $\lambda$ be solution of subproblem dual for $X = \bar{X}$

Let $B(\lambda, x) = f(x) + \lambda(b - g(x))$ be bound obtained by $\lambda$ for given $x$.

Add Benders cut $v \geq B(\lambda, x)$ to master problem: $\min v$

Solve master problem for next $\bar{X}$

Benders cuts
Example: SAT

- Solve SAT by chronological backtracking + unit clause rule = DPLL.
  - Chronological = fixed branching order.
Example: SAT

- Solve SAT by chronological backtracking + unit clause rule = DPLL.
  - Chronological = fixed branching order.

- To get nogood, solve **inference dual** at current node.
  - Solve dual with unit clause rule
  - Nogood identifies branches that create infeasibility.
  - Simplest scheme: nogood rules out path to current leaf node.
Example: SAT

- Solve SAT by chronological backtracking + unit clause rule = DPLL.
  - Chronological = fixed branching order.

- To get nogood, solve **inference dual** at current node.
  - Solve dual with unit clause rule
  - Nogood identifies branches that create infeasibility.
  - Simplest scheme: nogood rules out path to current leaf node.

- Process nogood set with **parallel resolution**
  - Nogood set is a **relaxation** of the problem.
Example: SAT

- Solve SAT by chronological backtracking + unit clause rule = DPLL.
  - Chronological = fixed branching order.

- To get nogood, solve **inference dual** at current node.
  - Solve dual with unit clause rule
  - Nogood identifies branches that create infeasibility.
  - Simplest scheme: nogood rules out path to current leaf node.

- Process nogood set with **parallel resolution**
  - Nogood set is a **relaxation** of the problem.

- Solve relaxation without branching
  - Select solution with preference for 0
The problem

Find a satisfying solution.

\begin{align*}
&x_1 \lor x_5 \lor x_6 \\
&x_1 \lor x_5 \lor \overline{x}_6 \\
&x_2 \lor \overline{x}_5 \lor x_6 \\
&x_2 \lor \overline{x}_5 \lor \overline{x}_6 \\
&\overline{x}_1 \lor x_3 \lor x_4 \\
&\overline{x}_2 \lor x_3 \lor x_4 \\
&\overline{x}_1 \lor \overline{x}_3 \\
&\overline{x}_1 \lor \overline{x}_4 \\
&\overline{x}_2 \lor \overline{x}_3 \\
&\overline{x}_2 \lor \overline{x}_4
\end{align*}
DPLL with chronological backtracking

Branch to here.
Solve subproblem with unit clause rule, which proves infeasibility.

\( (x_1, \ldots, x_5) = (0, \ldots, 0) \) creates the infeasibility.
DPLL with chronological backtracking

\[ x_1 = 0 \]

\[ x_2 = 0 \]

\[ x_3 = 0 \]

\[ x_4 = 0 \]

\[ x_5 = 0 \]

Branch to here.

Solve subproblem with unit clause rule, which proves infeasibility.

\((x_1, \ldots x_5) = (0, \ldots, 0)\) creates the infeasibility.

Generate nogood.

\[ x_1 \lor x_2 \lor x_3 \lor x_4 \lor x_5 \]

Slide 57
DPLL with chronological backtracking

Consists of processed nogoods

Conflict clause appears as nogood induced by solution of $R_k$.

<table>
<thead>
<tr>
<th>$k$</th>
<th>Relaxation $R_k$</th>
<th>Solution of $R_k$</th>
<th>Nogoods</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>$x_1 \lor x_2 \lor x_3 \lor x_4 \lor x_5$</td>
<td>$(0,0,0,0,0,\cdot)$</td>
<td>$x_1 \lor x_2 \lor x_3 \lor x_4 \lor x_5$</td>
</tr>
<tr>
<td>1</td>
<td>$x_1 \lor x_2 \lor x_3 \lor x_4 \lor x_5$</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

$x_1 = 0$

$x_2 = 0$

$x_3 = 0$

$x_4 = 0$

$x_5 = 0$

$x_1 \lor x_2 \lor x_3 \lor x_4 \lor x_5$
**Relaxation Solution of Nogoods**

<table>
<thead>
<tr>
<th>Relaxation $R_k$</th>
<th>Solution of $R_k$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$0$</td>
<td>$(0,0,0,0,0,1)$</td>
</tr>
<tr>
<td>$1$</td>
<td>$(0,0,0,0,1,1)$</td>
</tr>
</tbody>
</table>

DPLL with chronological backtracking

Consists of processed nogoods

Go to solution that solves relaxation, with priority to $0$
Slide 60

DPLL with chronological backtracking

Consists of processed nogoods

<table>
<thead>
<tr>
<th>k</th>
<th>Relaxation $R_k$</th>
<th>Solution of $R_k$</th>
<th>Nogoods</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>$x_1 \lor x_2 \lor x_3 \lor x_4 \lor x_5$</td>
<td>$(0,0,0,0,0,\cdot)$</td>
<td>$x_1 \lor x_2 \lor \bar{x}_3 \lor \bar{x}_4 \lor x_5$</td>
</tr>
<tr>
<td>1</td>
<td>$x_1 \lor x_2 \lor x_3 \lor x_4 \lor x_5$</td>
<td>$(0,0,0,0,1,\cdot)$</td>
<td>$x_1 \lor x_2 \lor x_3 \lor x_4 \lor \bar{x}_5$</td>
</tr>
</tbody>
</table>

Process nogood set with parallel resolution

parallel-absorbs

$x_1 \lor x_2 \lor x_3 \lor x_4 \lor x_5$

$x_1 \lor x_2 \lor x_3 \lor x_4 \lor \bar{x}_5$
DPLL with chronological backtracking

Consists of processed nogoods

<table>
<thead>
<tr>
<th>$k$</th>
<th>Relaxation $R_k$</th>
<th>Solution of $R_k$</th>
<th>Nogoods</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td></td>
<td>(0,0,0,0,0,0)</td>
<td>$x_1 \lor x_2 \lor x_3 \lor x_4 \lor x_5$</td>
</tr>
<tr>
<td>1</td>
<td>$x_1 \lor x_2 \lor x_3 \lor x_4 \lor x_5$</td>
<td>(0,0,0,0,1,0)</td>
<td>$x_1 \lor x_2 \lor x_3 \lor x_4 \lor \overline{x}_5$</td>
</tr>
<tr>
<td>2</td>
<td>$x_1 \lor x_2 \lor x_3 \lor x_4$</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Process nogood set with **parallel resolution**

**parallel-absorbs**

Slide 61
DPLL with chronological backtracking

Consists of processed nogoods

<table>
<thead>
<tr>
<th>$k$</th>
<th>Relaxation $R_k$</th>
<th>Solution of $R_k$</th>
<th>Nogoods</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td></td>
<td>(0, 0, 0, 0, 0, ·)</td>
<td>$x_1 \lor x_2 \lor x_3 \lor x_4 \lor x_5$</td>
</tr>
<tr>
<td>1</td>
<td>$x_1 \lor x_2 \lor x_3 \lor x_4 \lor x_5$</td>
<td>(0, 0, 0, 0, 1, ·)</td>
<td>$x_1 \lor x_2 \lor x_3 \lor x_4 \lor \overline{x_5}$</td>
</tr>
<tr>
<td>2</td>
<td>$x_1 \lor x_2 \lor x_3 \lor x_4$</td>
<td>(0, 0, 0, 1, 0, ·)</td>
<td></td>
</tr>
</tbody>
</table>

Solve relaxation again, continue.
So backtracking is nogood-based search with parallel resolution
Example: SAT + conflict clauses

- Use stronger nogoods = conflict clauses.
  - Nogoods rule out only branches that play a role in unit clause refutation.
Branch to here. Unit clause rule proves infeasibility. 

$(x_1, x_5) = (0, 0)$ is only premise of unit clause proof.

DPLL with conflict clauses
DPLL with conflict clauses

Relaxation Solution of Nogoods

$k$ Relaxation $R_k$  Solution of $R_k$  Nogoods

0  (0,0,0,0,0,·)  $x_1 \lor x_5$

1  $x_1 \lor x_5$

Conflict clause appears as nogood induced by solution of $R_k$. 

Slide 65
Consists of processed nogoods

<table>
<thead>
<tr>
<th>$k$</th>
<th>Relaxation $R_k$</th>
<th>Solution of $R_k$</th>
<th>Nogoods</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td></td>
<td>$(0,0,0,0,0,\cdot)$</td>
<td>$x_1 \lor x_5$</td>
</tr>
<tr>
<td>1</td>
<td>$x_1 \lor x_5$</td>
<td>$(0,0,0,0,1,\cdot)$</td>
<td>$x_2 \lor \bar{x}_5$</td>
</tr>
</tbody>
</table>

$$x_1 = 0$$

$$x_2 = 0$$

$$x_3 = 0$$

$$x_4 = 0$$

$$x_5 = 0$$

$$x_5 = 1$$

$$x_1 \lor x_5$$

$$x_2 \lor \bar{x}_5$$
DPLL with conflict clauses

Consists of processed nogoods

<table>
<thead>
<tr>
<th>$k$</th>
<th>Relaxation $R_k$</th>
<th>Solution of $R_k$</th>
<th>Nogoods</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td></td>
<td>$(0,0,0,0,0,\cdot)$</td>
<td>$x_1 \lor x_5$</td>
</tr>
<tr>
<td>1</td>
<td>$x_1 \lor x_5$</td>
<td>$(0,0,0,0,1,\cdot)$</td>
<td>$x_2 \lor \overline{x_5}$</td>
</tr>
<tr>
<td>2</td>
<td>$x_1 \lor x_2$</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

$X_1 \lor X_5$

parallel-resolve to yield $X_1 \lor X_2$

$X_2 \lor \overline{X_5}$

$X_1 \lor X_2$

parallel-absorbs

$X_1 \lor X_5$

$X_2 \lor \overline{X_5}$
DPLL with conflict clauses

\[
\begin{align*}
&x_1 = 0 \\
x_2 = 0 \\
x_3 = 0 \\
x_4 = 0 \\
x_5 = 0 \\
x_1 \lor x_5 \\
x_2 \lor \overline{x}_5 \\
x_1 \lor x_2
\end{align*}
\]

<table>
<thead>
<tr>
<th>(k)</th>
<th>Relaxation (R_k)</th>
<th>Solution of (R_k)</th>
<th>Nogoods</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>(x_1 \lor x_5)</td>
<td>((0,0,0,0,0,\cdot))</td>
<td>(x_1 \lor x_5)</td>
</tr>
<tr>
<td>1</td>
<td>(x_1 \lor x_5)</td>
<td>((0,0,0,0,1,\cdot))</td>
<td>(x_2 \lor \overline{x}_5)</td>
</tr>
<tr>
<td>2</td>
<td>(x_1 \lor x_2)</td>
<td>((0,1,\cdot,\cdot,\cdot,\cdot))</td>
<td>(x_1 \lor \overline{x}_2)</td>
</tr>
<tr>
<td>3</td>
<td>(x_1)</td>
<td>(\cdot)</td>
<td>(\cdot)</td>
</tr>
</tbody>
</table>

Parallel-resolve to yield \(x_1\)
DPLL with conflict clauses

\[ x_1 = 0 \quad x_1 = 1 \]

\[ x_2 = 0 \]

\[ x_3 = 0 \]

\[ x_4 = 0 \]

\[ x_5 = 0 \quad x_5 = 1 \]

\[ x_1 \lor x_5 \quad x_2 \lor \bar{x}_5 \quad x_1 \lor x_2 \]

\[ \emptyset \]

Search terminates

<table>
<thead>
<tr>
<th>( k )</th>
<th>Relaxation ( R_k )</th>
<th>Solution of ( R_k )</th>
<th>Nogoods</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>( x_1 \lor x_5 )</td>
<td>( (0,0,0,0,0,0,\cdot) )</td>
<td>( x_1 \lor x_5 )</td>
</tr>
<tr>
<td>1</td>
<td>( x_1 \lor x_5 )</td>
<td>( (0,0,0,0,1,\cdot) )</td>
<td>( x_2 \lor \bar{x}_5 )</td>
</tr>
<tr>
<td>2</td>
<td>( x_1 \lor x_2 )</td>
<td>( (0,1,\cdot,\cdot,\cdot,\cdot) )</td>
<td>( x_1 \lor \bar{x}_2 )</td>
</tr>
<tr>
<td>3</td>
<td>( x_1 )</td>
<td>( (1,\cdot,\cdot,\cdot,\cdot,\cdot) )</td>
<td>( \bar{x}_1 )</td>
</tr>
<tr>
<td>4</td>
<td>( \emptyset )</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Example: SAT + partial order dynamic backtracking

- Solve relaxation by selecting a solution that *conforms* to nogoods.
  - Conform = takes opposite sign than in nogoods.
  - More freedom than in branching.
Partial Order Dynamic Backtracking

<table>
<thead>
<tr>
<th>$k$</th>
<th>Relaxation $R_k$</th>
<th>Solution of $R_k$</th>
<th>Nogoods</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td></td>
<td>(0, 0, 0, 0, 0, ·)</td>
<td>$x_5 \lor x_1$</td>
</tr>
<tr>
<td>1</td>
<td>$x_1 \lor x_5$</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Arbitrarily choose one variable to be last
<table>
<thead>
<tr>
<th>( k )</th>
<th>Relaxation ( R_k )</th>
<th>Solution of ( R_k ) Nogoods</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>( x_1 )</td>
<td>( x_5 )</td>
</tr>
<tr>
<td>1</td>
<td>( x_1 \lor x_5 )</td>
<td></td>
</tr>
</tbody>
</table>

Arbitrarily choose one variable to be last

Other variables are penultimate
Partial Order Dynamic Backtracking

<table>
<thead>
<tr>
<th>$k$</th>
<th>Relaxation $R_k$</th>
<th>Solution of $R_k$</th>
<th>Nogoods</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>$x_1 \lor x_5$</td>
<td>$(0,0,0,0,0,\cdot)$</td>
<td>$x_5 \lor x_1$</td>
</tr>
<tr>
<td>1</td>
<td>$x_1 \lor x_5$</td>
<td>$(1,\cdot,\cdot,0,\cdot)$</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Since $x_5$ is penultimate in at least one nogood, it must conform to nogoods.

It must take value opposite its sign in the nogoods.

$x_5$ will have the same sign in all nogoods where it is penultimate.

This allows more freedom than chronological backtracking.
**Partial Order Dynamic Backtracking**

<table>
<thead>
<tr>
<th>$k$</th>
<th>Relaxation $R_k$</th>
<th>Solution of $R_k$</th>
<th>Nogoods</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>$x_1 \lor x_5$</td>
<td>$(0,0,0,0,0,\cdot)$</td>
<td>$x_5 \lor x_1$</td>
</tr>
<tr>
<td>1</td>
<td>$x_1 \lor x_5$</td>
<td>$(1,\cdot,\cdot,0,\cdot)$</td>
<td>$x_5 \lor \overline{x_1}$</td>
</tr>
<tr>
<td>2</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Choice of last variable is arbitrary but must be consistent with partial order implied by previous choices.

Since $x_5$ is penultimate in at least one nogood, it must conform to nogoods.

It must take value opposite its sign in the nogoods.

$x_5$ will have the same sign in all nogoods where it is penultimate.

This allows more freedom than chronological backtracking.
Partial Order Dynamic Backtracking

<table>
<thead>
<tr>
<th>$k$</th>
<th>Relaxation $R_k$</th>
<th>Solution of $R_k$</th>
<th>Nogoods</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td></td>
<td>(0,0,0,0,0,0)</td>
<td>$x_5 \lor x_1$</td>
</tr>
<tr>
<td>1</td>
<td>$x_1 \lor x_5$</td>
<td>(1,0,0,0,0,0)</td>
<td>$x_5 \lor \overline{x_1}$</td>
</tr>
<tr>
<td>2</td>
<td>$x_5$</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Since $x_5$ is penultimate in at least one nogood, it must conform to nogoods. It must take value opposite its sign in the nogoods.

$x_5$ will have the same sign in all nogoods where it is penultimate.

This allows more freedom than chronological backtracking.

Choice of last variable is arbitrary but must be consistent with partial order implied by previous choices.

Parallel-resolve to yield $x_5$
### Partial Order Dynamic Backtracking

<table>
<thead>
<tr>
<th>$k$</th>
<th>Relaxation $R_k$</th>
<th>Solution of $R_k$</th>
<th>Nogoods</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td></td>
<td>$(0,0,0,0,0,\cdot)$</td>
<td>$x_5 \lor x_1$</td>
</tr>
<tr>
<td>1</td>
<td>$x_1 \lor x_5$</td>
<td>$(1,\cdot,\cdot,0,\cdot)$</td>
<td>$x_5 \lor \overline{x}_1$</td>
</tr>
<tr>
<td>2</td>
<td>$x_5$</td>
<td>$(\cdot,0,\cdot,1,\cdot)$</td>
<td>$\overline{x}_5 \lor x_2$</td>
</tr>
<tr>
<td>3</td>
<td>${x_5, \overline{x}_5 \lor x_2}$</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

$x_5$ does not parallel-resolve with $\overline{x}_5 \lor x_2$ because $x_5$ is not last in both clauses
Partial Order Dynamic Backtracking

<table>
<thead>
<tr>
<th>k</th>
<th>Relaxation $R_k$</th>
<th>Solution of $R_k$</th>
<th>Nogoods</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td></td>
<td>(0,0,0,0,0)</td>
<td>$x_5 \lor x_1$</td>
</tr>
<tr>
<td>1</td>
<td>$x_1 \lor x_5$</td>
<td>(1,0,0,0,0)</td>
<td>$x_5 \lor \overline{x}_1$</td>
</tr>
<tr>
<td>2</td>
<td>$x_5$</td>
<td>(0,0,1,0,0)</td>
<td>$\overline{x}_5 \lor x_2$</td>
</tr>
<tr>
<td>3</td>
<td>${x_5}$</td>
<td>(0,1,1,1,1)</td>
<td>$\overline{x}_2$</td>
</tr>
<tr>
<td>4</td>
<td>$\emptyset$</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Examples, with computational results

- Production planning.
  - Semicontinuous piecewise linear functions
- Product configuration.
  - Variable indices
- Machine scheduling.
  - Logic-based Benders
- Truss structure design.
  - Global optimization.

- Solved by SIMPL, a prototype integrated solver.
Production Planning

Maximize profit, which is a piecewise linear function of output.

\[
\max \sum_i f_i(x_i) \quad \text{subject to} \quad \sum_i x_i \leq C
\]

Each \( f_i \) is a piecewise linear semicontinuous function.
Production Planning

Semicontinuous piecewise linear function $f(x)$
Production Planning

Integrated approach

• *Search:* **Branch** on variables

• *Relaxation:* Use a specialized **convex hull relaxation** for piecewise linear functions (no need for 0-1 model).

• *Inference:* **Bounds propagation**.

**Production Planning**

**Integrated model**

\[
\text{max } \sum_{i} u_i \\
\sum_{i} x_i \leq C \\
piecewise(x_i, u_i, L_i, U_i, c_i, d_i), \text{ all } i
\]

**Metaconstraint**

*(global constraint in CP)*
Production Planning

Semicontinuous piecewise linear function \( f(x) \)

Tight linear relaxation
Production Planning

Semicontinuous piecewise linear function $f(x)$

Value of $x$ in solution of current linear relaxation

Tighter relaxation after branching
SIMPL model

01. OBJECTIVE
02. maximize sum i of u[i]
03. CONSTRAINTS
04. capacity means {
05. sum i of x[i] <= C
06. relaxation = { lp, cp } }
07. piecewisectr means {
08. piecewise(x[i],u[i],L[i],U[i],c[i],d[i]) forall i
09. relaxation = { lp, cp } }
10. SEARCH
11. type = { bb:bestdive }
12. branching = { piecewisectr:most }
Production Planning

SIMPL model

01. OBJECTIVE
02.  maximize sum i of u[i]
03. CONSTRAINTS
04.  capacity means {
05.     sum i of x[i] <= C
06.     relaxation = { lp, cp } }
07.  piecewisectr means {
08.     piecewise(x[i],u[i],L[i],U[i],c[i],d[i]) for all i
09.     relaxation = { lp, cp } }
10. SEARCH
11.  type = { bb:bestdive }
12.  branching = { piecewisectr:most }

Recognized as a linear system.
Production Planning

**SIMPL model**

01. OBJECTIVE
02. maximize sum i of u[i]
03. CONSTRAINTS
04. capacity means {
05. sum i of x[i] <= C
06. relaxation = { lp, cp }
07. piecewisectr means {
08. piecewise(x[i],u[i],L[i],U[i],c[i],d[i]) forall i
09. relaxation = { lp, cp }
10. SEARCH
11. type = { bb:bestdive }
12. branching = { piecewisectr:most }

Is its own LP relaxation.
CP relaxation propagates bounds.
Production Planning

SIMPL model

01. OBJECTIVE
02. maximize sum i of u[i]
03. CONSTRAINTS
04. capacity means {
05. sum i of x[i] <= C
06. relaxation = { lp, cp } }
07. piecewisectr means {
08. piecewise(x[i],u[i],L[i],U[i],c[i],d[i]) forall i
09. relaxation = { lp, cp } }
10. SEARCH
11. type = { bb:bestdive }
12. branching = { piecewisectr:most }

Piecewise linear metaconstraint.
Production Planning

**SIMPL model**

01. OBJECTIVE
02. maximize sum i of u[i]
03. CONSTRAINTS
04. capacity means {
05. sum i of x[i] <= C
06. relaxation = { lp, cp } }
07. piecewisectr means {
08. piecewise(x[i],u[i],L[i],U[i],c[i],d[i]) forall i
09. relaxation = { lp, cp } }
10. SEARCH
11. type = { bb:bestdive }
12. branching = { piecewisectr:most }

LP relaxation is convex hull.
CP relaxation propagates bounds.
**SIMPL model**

01. OBJECTIVE
02. maximize sum i of u[i]
03. CONSTRAINTS
04. capacity means {
05. sum i of x[i] <= C
06. relaxation = { lp, cp } }
07. piecewisectr means {
08. piecewise(x[i],u[i],L[i],U[i],c[i],d[i]) forall i
09. relaxation = { lp, cp } }
10. SEARCH
11. type = { bb:bestdive }
12. branching = { piecewisectr:most }

Branch-and-bound search.
Dive to leaf node from node with best lower bound.
Production Planning

SIMPL model

01. OBJECTIVE
02. maximize sum i of u[i]
03. CONSTRAINTS
04. capacity means {
05. sum i of x[i] <= C
06. relaxation = { lp, cp } }
07. piecewisectr means {
08. piecewise(x[i],u[i],L[i],U[i],c[i],d[i]) forall i
09. relaxation = { lp, cp } }
10. SEARCH
11. type = { bb:bestdive }
12. branching = { piecewisectr:most }

Branch on piecewise constraint with greatest violation.
## Production Planning

### Computational Results (seconds)

Hand-coded integrated method was comparable to CPLEX 9

<table>
<thead>
<tr>
<th>No. Products</th>
<th>MILP CPLEX 9</th>
<th>MILP CPLEX 11</th>
<th>SIMPL</th>
</tr>
</thead>
<tbody>
<tr>
<td>50</td>
<td>9.5</td>
<td>2.4</td>
<td>0.43</td>
</tr>
<tr>
<td>60</td>
<td>55</td>
<td>1.8</td>
<td>1.1</td>
</tr>
<tr>
<td>70</td>
<td>99</td>
<td>2.6</td>
<td>0.82</td>
</tr>
<tr>
<td>80</td>
<td>61</td>
<td>4.6</td>
<td>1.25</td>
</tr>
<tr>
<td>90</td>
<td>422</td>
<td>6.2</td>
<td>1.7</td>
</tr>
<tr>
<td>100</td>
<td>4458</td>
<td>4.4</td>
<td>2.8</td>
</tr>
</tbody>
</table>
Production Planning

CPLEX has become orders of magnitude faster, but still slower than SIMPL

<table>
<thead>
<tr>
<th>No. Products</th>
<th>MILP CPLEX 9</th>
<th>MILP CPLEX 11</th>
<th>SIMPL</th>
</tr>
</thead>
<tbody>
<tr>
<td>50</td>
<td>9.5</td>
<td>2.4</td>
<td>0.43</td>
</tr>
<tr>
<td>60</td>
<td>55</td>
<td>1.8</td>
<td>1.1</td>
</tr>
<tr>
<td>70</td>
<td>99</td>
<td>2.6</td>
<td>0.82</td>
</tr>
<tr>
<td>80</td>
<td>61</td>
<td>4.6</td>
<td>1.25</td>
</tr>
<tr>
<td>90</td>
<td>422</td>
<td>6.2</td>
<td>1.7</td>
</tr>
<tr>
<td>100</td>
<td>4458</td>
<td>4.4</td>
<td>2.8</td>
</tr>
</tbody>
</table>
SIMPL’s advantage grows with the problem size

**Seconds**

<table>
<thead>
<tr>
<th>No. Products</th>
<th>MILP CPLEX 9</th>
<th>MILP CPLEX 11</th>
<th>SIMPL</th>
</tr>
</thead>
<tbody>
<tr>
<td>300</td>
<td>82</td>
<td>376</td>
<td>19</td>
</tr>
<tr>
<td>300</td>
<td>701</td>
<td>372</td>
<td>19</td>
</tr>
<tr>
<td>600</td>
<td>3515</td>
<td>4509</td>
<td>39</td>
</tr>
<tr>
<td>600</td>
<td>214</td>
<td>9416</td>
<td>131</td>
</tr>
</tbody>
</table>
SIMPL’s advantage grows with the problem size

<table>
<thead>
<tr>
<th>No. Products</th>
<th>MILP CPLEX 9</th>
<th>MILP CPLEX 11</th>
<th>SIMPL</th>
</tr>
</thead>
<tbody>
<tr>
<td>300</td>
<td>82</td>
<td>376</td>
<td>19</td>
</tr>
<tr>
<td>300</td>
<td>701</td>
<td>372</td>
<td>19</td>
</tr>
<tr>
<td>600</td>
<td>3515</td>
<td>4509</td>
<td>39</td>
</tr>
<tr>
<td>600</td>
<td>214</td>
<td>9416</td>
<td>131</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>No. Products</th>
<th>MILP CPLEX 9</th>
<th>MILP CPLEX 11</th>
<th>SIMPL</th>
</tr>
</thead>
<tbody>
<tr>
<td>300</td>
<td>10,164</td>
<td>101,756</td>
<td>73</td>
</tr>
<tr>
<td>300</td>
<td>43,242</td>
<td>128,333</td>
<td>58</td>
</tr>
<tr>
<td>600</td>
<td>363,740</td>
<td>646,907</td>
<td>74</td>
</tr>
<tr>
<td>600</td>
<td>7,732</td>
<td>1,297,071</td>
<td>214</td>
</tr>
</tbody>
</table>
Product configuration

Choose what type of each component, and how many
Product configuration

Integrated approach

• **Search:** Branch on variables.

• **Relaxation:** Generate a specialized linear relaxation for CP-based global constraints in the model.

• **Inference:** Apply specialized filtering algorithms to global constraints, and generate knapsack cuts.
Product configuration

Amount of attribute $j$ produced
($< 0$ if consumed): memory, heat, power, weight, etc.

Unit cost of producing attribute $j$

\[
\min \sum_{j} c_j v_j
\]

Quantity of component $i$ installed

\[
v_j = \sum_{ik} A_{ijt} q_i, \text{ all } j
\]

\[
L_j \leq v_j \leq U_j, \text{ all } j
\]
Product configuration

Unit cost of producing attribute $j$

Amount of attribute $j$ produced ($< 0$ if consumed): memory, heat, power, weight, etc.

Integrated model

\[ \min \sum_j c_j v_j \]

\[ v_j = \sum_{ik} A_{ij} q_i, \text{ all } j \]

\[ L_j \leq v_j \leq U_j, \text{ all } j \]

Amount of attribute $j$ produced by type $t_i$ of component $i$

Quantity of component $i$ installed
Product configuration

Amount of attribute $j$ produced ($< 0$ if consumed): memory, heat, power, weight, etc.

Unit cost of producing attribute $j$

\[ \min \sum_{j} c_j v_j \]

\[ v_j = \sum_{ik} A_{ij} q_i, \text{ all } j \]

\[ L_j \leq v_j \leq U_j, \text{ all } j \]

Amount of attribute $j$ produced by type $t_i$ of component $i$

$t_i$ is a variable index

Integrated model
Product configuration

Linear inequality
metaconstraint

$$\min \sum_j c_j v_j$$

$$v_j = \sum_{ik} q_i A_{ikt}, \text{ all } j$$

$$L_j \leq v_j \leq U_j, \text{ all } j$$
Indexed linear metaconstraint

\[
\begin{align*}
\min \sum_{j} c_j v_j \\
V_j &= \sum_{ik} q_{ik} A_{ij}, \text{ all } j \\
L_j &\leq v_j \leq U_j, \text{ all } j
\end{align*}
\]
Product configuration

Propagation

\[ \min \sum_{j} c_j v_j \]

\[ v_j = \sum_{ik} q_i A_{ijt_i}, \text{ all } j \]

\[ L_j \leq v_j \leq U_j, \text{ all } j \]

This is propagated in the usual way
**Product configuration**

**Propagation**

\[ v_j = \sum_i z_i, \text{ all } j \]

\[ \text{element}(t_i, (q_i, A_{ij1}, \ldots, q_i A_{ijn}), z_i), \text{ all } i, j \]

\[ \min \sum_j c_j v_j \]

\[ v_j = \sum_{i,k} q_i A_{ijt_i}, \text{ all } j \]

\[ L_j \leq v_j \leq U_j, \text{ all } j \]

*This is rewritten as*

*This is propagated in the usual way*
Product configuration

Propagation

\[ v_j = \sum_i z_i, \text{ all } j \]

element \((t_i, (q_i, A_{ij1}, \ldots, q_i A_{ijn}), z_i)\), all \(i, j\)

This is propagated by (a) using specialized **filters** for **element** constraints of this form…
Product configuration

Propagation

\[ v_j = \sum_i z_i, \text{ all } j \]

element \((t, (q_i, A_{ij1}, \ldots, q_i A_{ijn}), z_i)), \text{ all } i, j\)

This is propagated by

(a) using specialized filters for \textit{element} constraints of this form,
(b) adding \textbf{knapsack cuts} for the valid inequalities:

\[
\sum_{i \in D_{t_i}} \max \{ A_{ijk} \} q_i \geq v_j, \text{ all } j
\]

\[
\sum_{i \in D_{t_i}} \min \{ A_{ijk} \} q_i \leq \overline{v}_j, \text{ all } j
\]

and (c) propagating the knapsack cuts.

\([v_j, \overline{v}_j]\) is current domain of \(v_j\)
Product configuration

Relaxation

\[
\begin{align*}
\min & \sum_j c_j v_j \\
v_j &= \sum_{ik} q_i A_{ijt_i}, \text{ all } j \\
L_j &\leq v_j \leq U_j, \text{ all } j
\end{align*}
\]

This is relaxed as

\[
\underline{v}_j \leq v_j \leq \overline{v}_j
\]
Product configuration

Relaxation

\[ v_j = \sum_i z_i, \text{ all } j \]

\[ \text{element} \left( t_i, (q_i, A_{ij1}, \ldots, q_i A_{ijn}), z_i \right), \text{ all } i, j \]

\[ \min \sum_j c_j v_j \]

\[ v_j = \sum_{ik} q_i A_{ijt_i}, \text{ all } j \]

\[ L_j \leq v_j \leq U_j, \text{ all } j \]

This is relaxed by relaxing \textit{this} and adding the knapsack cuts.

This is relaxed as

\[ \underline{v}_j \leq v_j \leq \overline{v}_j \]
Product configuration

Relaxation

\[ v_j = \sum_i z_i, \text{ all } j \]

Element \( (t_i, (q_i, A_{i1}, \ldots, q_i A_{in}), z_i) \), all \( i, j \)

This is relaxed by writing each element constraint as a disjunction of linear systems and writing a convex hull relaxation of the disjunction:

\[ z_i = \sum_{k \in D_{t_i}} A_{ijk} q_{ik}, \quad q_i = \sum_{k \in D_{t_i}} q_{ik} \]
Product configuration

01. OBJECTIVE
02. minimize sum j of c[j]*v[j]
03. CONSTRAINTS
04. usage means {
05. \[ v[j] = \text{sum } i \text{ of } q[i]*a[i][j][t[i]] \text{ forall } j \]
06. relaxation = { lp, cp }
07. inference = { knapsack }
08. quantities means {
09. \[ q[1] >= 1 \Rightarrow q[2] = 0 \]
10. relaxation = { lp, cp }

15. SEARCH
16. type = { bb:bestdive }
17. branching = { quantities, t:most, q:least:triple, types:most }
18. inference = { q:redcost }

SIMPL model

Recognized as indexed linear system
Product configuration

01. OBJECTIVE
02. minimize sum j of c[j]*v[j]
03. CONSTRAINTS
04. usage means {
05. v[j] = sum i of q[i]*a[i][j][t[i]] forall j
06. relaxation = { lp, cp }  \textcolor{red}{\textbf{SIMPL model}}
07. inference = { knapsack } }
08. quantities means {
09. q[1] >= 1 => q[2] = 0  \textcolor{red}{\textbf{LP relaxation is convex hull of disjunction.}}
10. relaxation = { lp, cp } }
15. SEARCH
16. type = { bb:bestdive }  \textcolor{red}{\textbf{CP relaxation propagates bounds.}}
17. branching = { quantities, t:most, q:least:triple, types:most }
18. inference = { q:redcost }

Slide 111
Product configuration

01. OBJECTIVE
minimize sum j of c[j]*v[j]

03. CONSTRAINTS

04. usage means {

05. v[j] = sum i of q[i]*a[i][j][t[i]] forall j

06. relaxation = { lp, cp }

07. inference = { knapsack } }

08. quantities means {

09. q[1] >= 1 => q[2] = 0

10. relaxation = { lp, cp } }

15. SEARCH

16. type = { bb:bestdive }

17. branching = { quantities, t:most, q:least:triple, types:most }

18. inference = { q:redcost }

SIMPL model

Generate knapsack cuts from associated valid inequalities.
Product configuration

01. OBJECTIVE
02. minimize sum j of c[j]*v[j]
03. CONSTRAINTS
04. usage means {
05. \[ v[j] = \sum i \cdot a[i][j][t[i]] \text{ for all } j \]
06. relaxation = \{ lp, cp \}
07. inference = \{ knapsack \}
08. quantities means {
09. \[ q[1] \geq 1 \implies q[2] = 0 \]
10. relaxation = \{ lp, cp \}
15. SEARCH
16. type = \{ bb:bestdive \}
17. branching = \{ quantities, t:most, q:least:triple, types:most \}
18. inference = \{ q:redcost \}

Logical constraint on quantities
Product configuration

01. OBJECTIVE
02. minimize sum j of c[j]*v[j]
03. CONSTRAINTS
04. usage means {
05. v[j] = sum i of q[i]*a[i][j][t[i]] forall j
06. relaxation = { lp, cp }
07. inference = { knapsack } }
08. quantities means {
09. q[1] >= 1 => q[2] = 0
10. relaxation = { lp, cp } }
15. SEARCH
16. type = { bb:bestdive }
17. branching = { quantities, t:most, q:least:triple, types:most }
18. inference = { q:redcost }

SIMPL model

First branch on violated logical constraint on $q_i$ variables
Product configuration

01. OBJECTIVE
02. minimize sum j of c[j]*v[j]
03. CONSTRAINTS
04. usage means {
05. v[j] = sum i of q[i]*a[i][j][t[i]] forall j
06. relaxation = { lp, cp }
07. inference = { knapsack } }
08. quantities means {
09. q[1] >= 1 => q[2] = 0
10. relaxation = { lp, cp } }
15. SEARCH
16. type = { bb:bestdive }
17. branching = { quantities, t:most, q:least:triple, types:most }
18. inference = { q:redcost }

SIMPL model

Then branch on most violated \( t_i \) in-domain constraint.

Violated when domain of \( t_i \) is not a singleton, or two or more associated \( q_{ik} \)s are positive.
Product configuration

01. OBJECTIVE
02. minimize sum j of c[j]*v[j]
03. CONSTRAINTS
04. usage means {
05. v[j] = sum i of q[i]*a[i][j][t[i]] for all j
06. relaxation = { lp, cp }
07. inference = { knapsack } }
08. quantities means {
09. q[1] >= 1 => q[2] = 0
10. relaxation = { lp, cp } }
15. SEARCH
16. type = { bb:bestdive }
17. branching = { quantities, t:most, q:least:triple, types:most }
18. inference = { q:redcost }

SIMPL model

Then branch on least violated $q_i$ in-domain constraint.
Create three branches:
$q_i = \text{nearest integer } q'_i$,
$q_i < q'_i$, $q_i > q'_i$
Product configuration

SIMPL model

01. OBJECTIVE

02. minimize sum j of c[j]*v[j]

03. CONSTRAINTS

04. usage means {

05. v[j] = sum i of q[i]*a[i][j][t[i]] forall j

06. relaxation = { lp, cp } 

07. inference = { knapsack } }

08. quantities means {

09. q[1] >= 1 => q[2] = 0

10. relaxation = { lp, cp } }

15. SEARCH

16. type = { bb:bestdive }

17. branching = { quantities, t:most, q:least:triple, types:most }

18. inference = { q:redcost }

Then branch on most violated logical constraint on $t_i$ variables (omitted)
Product configuration

01. OBJECTIVE
02.    minimize sum j of c[j]*v[j]
03. CONSTRAINTS
04.    usage means {
05.        v[j] = sum i of q[i]*a[i][j][t[i]] for all j
06.    relaxation = { lp, cp }
07.    inference = { knapsack } }
08.    quantities means {
09.        q[1] >= 1 => q[2] = 0
10.    relaxation = { lp, cp } }
15. SEARCH
16.    type = { bb:bestdive }
17.    branching = { quantities, t:most, q:least:triple, types:most }
18.    inference = { q:redcost }

SIMPL model

Reduced-cost variable fixing for q_i's
Product configuration

Computational results

SIMPL matches hand-coded integrated method, which was orders of magnitude faster than CPLEX.

Again, CPLEX has become much faster, now somewhat faster than SIMPL.

<table>
<thead>
<tr>
<th>Nodes</th>
<th>Sec.</th>
<th>Nodes</th>
<th>Sec.</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0.07</td>
<td>56</td>
<td>0.49</td>
</tr>
<tr>
<td>1</td>
<td>0.10</td>
<td>32</td>
<td>0.25</td>
</tr>
<tr>
<td>31</td>
<td>0.68</td>
<td>186</td>
<td>1.67</td>
</tr>
<tr>
<td>1</td>
<td>0.02</td>
<td>28</td>
<td>0.24</td>
</tr>
<tr>
<td>1</td>
<td>0.12</td>
<td>32</td>
<td>0.33</td>
</tr>
<tr>
<td>1</td>
<td>0.07</td>
<td>9</td>
<td>0.09</td>
</tr>
<tr>
<td>10</td>
<td>0.18</td>
<td>35</td>
<td>0.30</td>
</tr>
<tr>
<td>1</td>
<td>0.10</td>
<td>32</td>
<td>0.25</td>
</tr>
<tr>
<td>1</td>
<td>0.05</td>
<td>28</td>
<td>0.22</td>
</tr>
<tr>
<td>1</td>
<td>0.12</td>
<td>14</td>
<td>0.13</td>
</tr>
</tbody>
</table>
Machine scheduling

• Assign jobs to machines, and schedule the machines assigned to each machine within time windows.

• The objective is to minimize processing cost.
Machine scheduling

### Job Data

<table>
<thead>
<tr>
<th>Job</th>
<th>Release time</th>
<th>Deadline</th>
<th>Processing time</th>
</tr>
</thead>
<tbody>
<tr>
<td>$j$</td>
<td>$r_j$</td>
<td>$d_j$</td>
<td>$p_{A_j}$</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>9</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>0</td>
<td>9</td>
<td>3</td>
</tr>
<tr>
<td>3</td>
<td>2</td>
<td>7</td>
<td>3</td>
</tr>
<tr>
<td>4</td>
<td>2</td>
<td>9</td>
<td>4</td>
</tr>
<tr>
<td>5</td>
<td>4</td>
<td>7</td>
<td>2</td>
</tr>
</tbody>
</table>

### Example

Assign 5 jobs to 2 machines.

Schedule jobs assigned to each machine without overlap.
Machine scheduling

**Integrated model**

\[
\begin{align*}
\text{min} & \quad \sum_j c_{x_{ij}} \\
\text{s.t.} & \quad r_j \leq s_j \leq d_j - p_{x_{ij}}, \text{ all } j \\
\text{disjunctive} & \quad (s_j \mid x_j = i), (p_{ij} \mid x_j = i), \text{ all } i
\end{align*}
\]

- Start time of job \( j \)
- Time windows
- Jobs cannot overlap
- Machine assigned to job \( j \)
Machine scheduling

Integrated approach

• *Search*: Enumerate subproblems (defined by assigning jobs to machines)

• *Relaxation*: Enumerate master problems (which assign jobs to machines)

• *Inference*: Generate nogoods (logic-based Benders cuts), which are added to master problem.
Machine scheduling

**Integrated approach**

- Assign the jobs in the **master problem**, to be solved by **MILP**.
- Schedule the jobs in the **subproblem**, to be solved by **CP**.

Machine scheduling

Integrated approach

- Assign the jobs in the **master problem**, to be solved by **MILP**.
- Schedule the jobs in the **subproblem**, to be solved by **CP**.

The subproblem decouples into a separate scheduling problem on each machine.

In this problem, the subproblem is a feasibility problem.

Machine scheduling

Integrated approach

• Assign the jobs in the **master problem**, to be solved by **MILP**.

• Schedule the jobs in the **subproblem**, to be solved by **CP**.

The subproblem decouples into a separate scheduling problem on each machine.

In this problem, the subproblem is a feasibility problem.

• Solve **inference dual** of subproblem to generate **nogoods** (logic-based Benders cuts), which are added to master problem.

Machine scheduling

**Integrated model**

\[
\begin{align*}
\text{min } & \sum_j c_{x_j} \\
\text{s.t. } & r_j \leq s_j \leq d_j - p_{x_j}, \text{ all } j \\
& \text{disjunctive} \left( (s_j | x_j = i), (p_{ij} | x_j = i) \right), \text{ all } i
\end{align*}
\]

Indexed linear metaconstraint
Machine scheduling

**Integrated model**

\[
\min \sum_{j} c_{x_j} \\
\text{subject to:} \quad r_j \leq s_j \leq d_j - p_{x_j}, \quad \text{all } j \\
\quad \text{disjunctive}\left((s_j | x_j = i), (p_{ij} | x_j = i)\right), \quad \text{all } i
\]
Machine scheduling

Integrated model

\[
\begin{align*}
\text{min } & M \\
M & \geq s_j + p_{x_j}, \text{ all } j \\
r_j & \leq s_j \leq d_j - p_{x_j}, \text{ all } j \\
\text{disjunctive} & \left( (s_j | x_j = i), (p_{ij} | x_j = i) \right), \text{ all } i
\end{align*}
\]

For a fixed assignment \( \bar{x} \) the subproblem on each machine \( i \) is

\[
\begin{align*}
\text{min } & M \\
M & \geq s_j + p_{x_j}, \text{ all } j \text{ with } \bar{x}_j = i \\
r_j & \leq s_j \leq d_j - p_{x_j}, \text{ all } j \text{ with } \bar{x}_j = i \\
\text{disjunctive} & \left( (s_j | \bar{x}_j = i), (p_{ij} | \bar{x}_j = i) \right)
\end{align*}
\]
**Machine scheduling**

**Logic-based Benders approach**

Suppose we assign jobs 1, 2, 3, 5 to machine A in iteration $k$.

We can prove that there is no feasible schedule.

**Edge finding** derives infeasibility by reasoning only with jobs 2, 3, 5. So these jobs alone create infeasibility.

So we have a Benders cut $\neg(x_2 = x_3 = x_5 = A)$.
Machine scheduling

Logic-based Benders approach

We want the master problem to be an MILP, which is good for assignment problems.

So we write the Benders cut \( \neg (x_2 = x_3 = x_5 = A) \)

Using 0-1 variables:

\[
x_{A2} + x_{A3} + x_{A5} \leq 2
\]

= 1 if job 5 is assigned to machine A
Machine scheduling

The master problem is a \textit{relaxation}, formulated as an MILP:

\[
\min \sum_{ij} c_{ij} x_{ij}
\]

\[
\sum_{j=1}^{5} p_{Aj} x_{Aj} \leq 9, \text{ etc.}
\]

\[
\sum_{j=1}^{5} p_{ Bj} x_{Bj} \leq 9, \text{ etc.}
\]

\[
x_{A2} + x_{A3} + x_{A5} \leq 2
\]

\[
x_{ij} \in \{0,1\}
\]

Constraints derived from time windows

Benders cut from machine A
The master problem is a relaxation, formulated as an MILP:

\[
\begin{align*}
\min & \sum_{ij} c_{ij} x_{ij} \\
\text{s.t.} & \sum_{j=1}^{5} p_{A_j} x_{A_j} \leq 9, \text{ etc.} \\
& \sum_{j=1}^{5} p_{B_j} x_{B_j} \leq 9, \text{ etc.} \\
& x_{A2} + x_{A3} + x_{A5} \leq 2 \\
& x_{ij} \in \{0, 1\}
\end{align*}
\]

Benders cuts have been developed for min makespan and min tardiness (subproblem is an optimization problem)

Also for cumulative scheduling.
Machine scheduling

SIMPL model

01. OBJECTIVE
02. \[ \text{min} \sum_{i,j} c[i][j] \cdot x[i][j]; \]
03. CONSTRAINTS
04. assign means {
05. \[ \sum_{i} x[i][j] = 1 \text{ for all } j; \]
06. relaxation = { ip:master } }
07. xy means {
08. \[ x[i][j] = 1 \iff y[j] = 1 \text{ for all } i, j; \]
09. relaxation = { cp } }
10. tbounds means {
11. \[ r[j] \leq t[j] \text{ for all } j; \]
12. \[ t[j] \leq d[j] - p[y[j]][j] \text{ for all } j; \]
13. relaxation = { ip:master, cp } }
14. machinecap means {
15. \[ \text{cumulative} \{ t[j], p[i][j], 1 \} \text{ for all } j | x[i][j] = 1, 1 \text{ for all } i; \]
16. relaxation = { cp:subproblem, ip:master }
17. inference = { feasibility } }
18. SEARCH
19. type = { benders }
Machine scheduling

SIMPL model

01. OBJECTIVE
02. \[
    \text{min } \sum_{i,j} c[i][j] \cdot x[i][j];
\]
03. CONSTRAINTS
04. assign means {
05. \[
    \text{sum } i \text{ of } x[i][j] = 1 \text{ forall } j;
\]
06. relaxation = { ip:master }
07. xy means {
08. \[
    x[i][j] = 1 \iff y[j] = 1 \text{ forall } i, j;
\]
09. relaxation = { cp }
10. tbounds means {
11. \[
    r[j] \leq t[j] \text{ forall } j;
\]
12. \[
    t[j] \leq d[j] - p[y[j]][j] \text{ forall } j;
\]
13. relaxation = { ip:master, cp }
14. machinecap means {
15. \[
    \text{cumulative(\{ } t[j], p[i][j], 1 \text{ \} forall } j | x[i][j] = 1, 1) \text{ forall } i;
\]
16. relaxation = { cp:subproblem, ip:master }
17. inference = { feasibility }
18. SEARCH
19. type = { benders }

MILP relaxation of the constraint (which is the constraint itself) goes into master problem
Machine scheduling

SIMPL model

01. OBJECTIVE
02. \( \text{min} \ \sum_{i,j} c[i][j] \times x[i][j] \);
03. CONSTRAINTS
04. assign means {
05. \( \sum_{i} x[i][j] = 1 \) forall \( j \);
06. relaxation = { ip: master } }
07. \( \text{xy means} \{ \)
08. \( x[i][j] = 1 \iff y[j] = 1 \) forall \( i, j \);
09. relaxation = { cp } }
10. tbounds means {
11. \( r[j] \leq t[j] \) forall \( j \);
12. \( t[j] \leq d[j] - p[y[j]][j] \) forall \( j \);
13. relaxation = { ip: master, cp } }
14. machinecap means {
15. cumulative({ \( t[j], p[i][j], 1 \) } forall \( j \mid x[i][j] = 1, 1 \) forall \( i \));
16. relaxation = { cp: subproblem, ip: master }
17. inference = { feasibility } }
18. SEARCH
19. type = { benders }
Machine scheduling

**SIMPL model**

```
01. OBJECTIVE
02. min sum i,j of c[i][j] * x[i][j];
03. CONSTRAINTS
04. assign means {
05.    sum i of x[i][j] = 1 forall j;
06.    relaxation = { ip:master }
07.    xy means {
08.      x[i][j] = 1 <= y[j] = 1 forall i, j;
09.      relaxation = { cp }
10.    }
11.    tbounds means {
12.      r[j] <= t[j] forall j;
13.      relaxation = { ip:master, cp }
14.    }
15.    machinecap means {
16.      cumulative({ t[j], p[i][j], 1 } forall j | x[i][j] = 1, 1) forall i;
17.      relaxation = { cp:subproblem, ip:master }
18.      inference = { feasibility }
19.    }
20. SEARCH
21.    type = { benders }
```
Machine scheduling

01. OBJECTIVE
02. \[ \min \sum_{i,j} c[i][j] \cdot x[i][j] ; \]
03. CONSTRAINTS
04. assign means {
05. \[ \sum_{i} x[i][j] = 1 \text{ forall } j ; \]
06. relaxation = \{ ip:master \} \}
07. xy means {
08. \[ x[i][j] = 1 \iff y[j] = 1 \text{ forall } i, j ; \]
09. relaxation = \{ cp \} \}
10. tbounds means {
11. \[ r[j] \leq t[j] \text{ forall } j ; \]
12. \[ t[j] \leq d[j] - p[y[j]][j] \text{ forall } j ; \]
13. relaxation = \{ ip:master, cp \} \}
14. machinecap means {
15. \[ \text{cumulative}\{ t[j], p[i][j], 1 \} \text{ forall } j \mid x[i][j] = 1, 1 \text{ forall } i ; \]
16. relaxation = \{ cp:subproblem, ip:master \}
17. inference = \{ feasibility \} \}
18. SEARCH
19. type = \{ benders \}
Machine scheduling

SIMPL model

01. OBJECTIVE
02. min sum i,j of c[i][j] * x[i][j];
03. CONSTRAINTS
04. assign means {
05. sum i of x[i][j] = 1 forall j;
06. relaxation = { ip:master } }
07. xy means {
08. x[i][j] = 1 <=> y[j] = 1 forall i, j;
09. relaxation = { cp } }
10. tbounds means {
11. r[j] <= t[j] forall j;
13. relaxation = { ip:master, cp } }
14. machinecap means {
15. cumulative({ t[j], p[i][j], 1 } forall j | x[i][j] = 1, 1) forall i;
16. relaxation = { cp:subproblem, ip:master }
17. inference = { feasibility } }
18. SEARCH
19. type = { benders }

MILP formulation goes into master problem
CP-based propagation
Machine scheduling

SIMPL model

```plaintext
01. OBJECTIVE
02.  min sum i,j of c[i][j] * x[i][j];
03. CONSTRAINTS
04.  assign means {
05.      sum i of x[i][j] = 1 forall j;
06.      relaxation = { ip:master } }
07.  xy means {
08.      x[i][j] = 1 <=> y[j] = 1 forall i, j;
09.      relaxation = { cp } }
10.  tbounds means {
11.      r[j] <= t[j] forall j;
13.      relaxation = { ip:master, cp } }
14.  machincap means {
15.      cumulative({ t[j], p[i][j], 1 } forall j | x[i][j] = 1, 1) forall i;
16.      relaxation = { cp:subproblem, ip:master }
17.      inference = { feasibility } }
18. SEARCH
19.  type = { benders }
```

Disjunctive scheduling constraint written as special case of cumulative scheduling constraint (resource consumption = 1, capacity = 1)
Machine scheduling

01. OBJECTIVE
02. min sum i,j of c[i][j] * x[i][j];
03. CONSTRAINTS
04. assign means {
05. sum i of x[i][j] = 1 for all j;
06. relaxation = { ip: master } }
07. xy means {
08. x[i][j] = 1 <-> y[j] = 1 for all i, j;
09. relaxation = { cp } }
10. tbounds means {
11. r[j] <= t[j] for all j;
13. relaxation = { ip: master, cp } }
14. machinecap means {
15. cumulative({ t[j], p[i][j], 1 } for all j | x[i][j] = 1, 1) for all i;
16. relaxation = { cp: subproblem, ip: master } }
17. inference = { feasibility } }
18. SEARCH
19. type = { benders }

The CP problem goes into the Benders subproblem.
A relaxation of the constraint goes into the master
Machine scheduling

SIMPL model

01. OBJECTIVE
02. min sum i,j of c[i][j] * x[i][j];

03. CONSTRAINTS
04. assign means {
05.  sum i of x[i][j] = 1 forall j;
06.  relaxation = { ip:master } }
07. xy means {
08.  x[i][j] = 1 <=> y[j] = 1 forall i, j;
09.  relaxation = { cp } }
10. tbounds means {
11.  r[j] <= t[j] forall j;
13.  relaxation = { ip:master, cp } }
14. machinecap means {
15.  cumulative({ t[j], p[i][j], 1 } forall j | x[i][j] = 1, 1) forall i;
16.  relaxation = { cp:subproblem, ip:master }
17.  inference = { feasibility } }
18. SEARCH
19.  type = { benders }
Machine scheduling

SIMPL model

01. OBJECTIVE
02. min sum i,j of c[i][j] * x[i][j];
03. CONSTRAINTS
04. assign means {
05. sum i of x[i][j] = 1 forall j;
06. relaxation = { ip:master } }
07. xy means {
08. x[i][j] = 1 <=> y[j] = 1 forall i, j;
09. relaxation = { cp } }
10. tbounds means {
11. r[j] <= t[j] forall j;
13. relaxation = { ip:master, cp } }
14. machinecap means {
15. cumulative( { t[j], p[i][j], 1 } forall j | x[i][j] = 1, 1) forall i;
16. relaxation = { cp:subproblem, ip:master }
17. inference = { feasibility } }
18. SEARCH
19. type = { benders }

Benders-based search, where problem restrictions are Benders subproblems and problem relaxations are master problems.
Machine scheduling

**Computational results – Long processing times**

<table>
<thead>
<tr>
<th>Jobs</th>
<th>Machines</th>
<th>MILP (CPLEX 11)</th>
<th>SIMPL Benders</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>Nodes</td>
<td>Sec.</td>
</tr>
<tr>
<td>3</td>
<td>2</td>
<td>1</td>
<td>0.00</td>
</tr>
<tr>
<td>7</td>
<td>3</td>
<td>1</td>
<td>0.00</td>
</tr>
<tr>
<td>12</td>
<td>3</td>
<td>3,351</td>
<td>6.6</td>
</tr>
<tr>
<td>15</td>
<td>5</td>
<td>2,779</td>
<td>8.8</td>
</tr>
<tr>
<td>20</td>
<td>5</td>
<td>33,321</td>
<td>882</td>
</tr>
<tr>
<td>22</td>
<td>5</td>
<td>352,309</td>
<td>10,563</td>
</tr>
</tbody>
</table>

SIMPL results are similar to original hand-coded results.
## Machine scheduling

### Computational results – Short processing times

<table>
<thead>
<tr>
<th>Jobs</th>
<th>Machines</th>
<th>MILP (CPLEX 11)</th>
<th>SIMPL Benders</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>Nodes</td>
<td>Iter.</td>
</tr>
<tr>
<td>3</td>
<td>2</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>7</td>
<td>3</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>12</td>
<td>3</td>
<td>499</td>
<td>1</td>
</tr>
<tr>
<td>15</td>
<td>5</td>
<td>529</td>
<td>2</td>
</tr>
<tr>
<td>20</td>
<td>5</td>
<td>250,047</td>
<td>6</td>
</tr>
<tr>
<td>22</td>
<td>5</td>
<td>&gt; 27.5 mil.</td>
<td>9</td>
</tr>
<tr>
<td>25</td>
<td>5</td>
<td>&gt; 5.4 mil.</td>
<td>17</td>
</tr>
</tbody>
</table>

*out of memory
Machine scheduling

Benders cut for minimum makespan, with a cumulative scheduling subproblem

$$M \geq M_i^* - \left( \sum_{j \in J_i} p_{ij} (1 - x_{ij}) + \max_{j \in J_i} \{d_j\} - \min_{j \in J_i} \{d_j\} \right)$$

- Minimum makespan on machine $i$ for jobs currently assigned
- Jobs currently assigned to machine $i$
Logic-based Benders Decomposition

• In general, Benders cuts are obtained by solving the inference dual of the subproblem.
  • The dual solution is a proof of optimality.
  • LP dual is a special case, where the proof is encoded by dual multipliers.
Logic-based Benders Decomposition

• In general, Benders cuts are obtained by solving the inference dual of the subproblem.
  • The dual solution is a proof of optimality.
  • LP dual is a special case, where the proof is encoded by dual multipliers.

• The Benders cut states conditions on the master problem variables under which the proof remains valid.
  • Classical Benders cut is a special case.
Truss Structure Design

Select size of each bar (possibly zero) to support the load while minimizing weight.

10-bar cantilever truss

Total 8 degrees of freedom
Truss Structure Design

Notation

\[ v_i = \text{elongation of bar} \]

\[ s_i = \text{force along bar} \]

\[ h_i = \text{length of bar } i \]

\[ A_i = \text{cross-sectional area of bar} \]

\[ p_j = \text{load along d.f. } j \]

\[ d_j = \text{node displacement} \]
**Truss Structure Design**

\[
\begin{align*}
\text{min} & \quad \sum_i h_i A_i & \{ \text{Minimize total weight} \} \\
\text{s.t.} & \quad \sum_i \cos \theta_{ij} s_i = p_j, \text{ all } j & \{ \text{Equilibrium} \} \\
& \quad \sum_j \cos \theta_{ij} d_j = v_i, \text{ all } i & \{ \text{Compatibility} \} \\
& \quad \frac{E_i}{h_i} A_i v_i = s_i, \text{ all } i & \{ \text{Hooke's law} \} \\
& \quad v_i^L \leq v_i \leq v_i^U, \text{ all } i & \{ \text{Elongation bounds} \} \\
& \quad d_j^L \leq d_j \leq d_j^U, \text{ all } j & \{ \text{Displacement bounds} \} \\
& \quad \bigvee_k (A_i = A_{ik}) & \{ \text{Logical disjunction} \}
\end{align*}
\]

Area must be one of several discrete values \( A_{ik} \)

Constraints can be imposed for multiple loading conditions
Introducing new variables linearizes the problem but makes it much larger.

**MILP model**

\[
\begin{align*}
\text{min } & \sum_i h_i \sum_k A_{ik} y_{ik} \\
\text{s.t. } & \sum_i \cos \theta_{ij} s_i = p_j, \text{ all } j \\
& \sum_j \cos \theta_{ij} d_j = \sum_k v_{ik}, \text{ all } i \\
& \frac{E_i}{h_i} \sum_k A_{ik} v_{ik} = s_i, \text{ all } i \\
& v_i^L \leq v_i \leq v_i^U, \text{ all } i \\
& d_j^L \leq d_j \leq d_j^U, \text{ all } j \\
& \sum_k y_{ik} = 1, \text{ all } i
\end{align*}
\]
Truss Structure Design

Integrated approach

• Search: Branch by splitting the range of areas $A_i$ (no need for 0-1 variables).

• Relaxation: Generate a quasi-relaxation, which is linear and much smaller than MILP model.

• Inference: Use logic cuts.

Truss Structure Design

**Theorem (JNH 2005)**

Suppose we minimize $cx$ subject to $g(x,y) \leq 0$.

If $g(x,y)$ is semihomogeneous in $x \in \mathbb{R}^n$ and concave in scalar $y$, then the following is a **quasi-relaxation** of $g(x,y) \leq 0$:

\[
\begin{align*}
g(x^1, y_L) + g(x^2, y_U) &\leq 0 \\
\alpha x^L &\leq x^1 \leq \alpha x^U \\
(1 - \alpha) x^L &\leq x^2 \leq (1 - \alpha) x^U \\
x &= x^1 + x^2
\end{align*}
\]
Truss Structure Design

Theorem (JNH 2005)

Suppose we minimize $cx$ subject to $g(x,y) \leq 0$.

If $g(x,y)$ is semihomogeneous in $x \in \mathbb{R}^n$ and concave in scalar $y$, then the following is a quasi-relaxation of $g(x,y) \leq 0$:

$$g(x^1, y_L) + g(x^2, y_U) \leq 0$$
$$\alpha x^L \leq x^1 \leq \alpha x^U$$
$$(1 - \alpha) x^L \leq x^2 \leq (1 - \alpha) x^U$$
$$x = x^1 + x^2$$

Its optimal value is a lower bound on the optimal value of the original problem, if cost is a function of $x$ alone.
Theorem (JNH 2005)

Suppose we minimize $cx$ subject to $g(x,y) \leq 0$.

If $g(x,y)$ is semihomogeneous in $x \in \mathbb{R}^n$ and concave in scalar $y$, then the following is a quasi-relaxation of $g(x,y) \leq 0$:

\[
g(x^1, y_L) + g(x^2, y_U) \leq 0 \\
\alpha x^L \leq x^1 \leq \alpha x^U \\
(1 - \alpha) x^L \leq x^2 \leq (1 - \alpha) x^U \\
x = x^1 + x^2
\]

\[
g(\alpha x, y) \leq \alpha g(x, y) \quad \text{for all } x, y \text{ and } \alpha \in [0,1] \\
g(0, y) = 0 \quad \text{for all } y
\]
Theorem (JNH 2005)

Suppose we minimize $cx$ subject to $g(x,y) \leq 0$.

If $g(x,y)$ is semihomogeneous in $x \in \mathbb{R}^n$ and concave in scalar $y$, then the following is a quasi-relaxation of $g(x,y) \leq 0$:

$$g(x^1, y_L) + g(x^2, y_U) \leq 0$$

$$\alpha x^L \leq x^1 \leq \alpha x^U$$

$$(1-\alpha) x^L \leq x^2 \leq (1-\alpha) x^U$$

$x = x^1 + x^2$

Bounds on $y$
Theorem (JNH 2005)

Suppose we minimize $cx$ subject to $g(x,y) \leq 0$.

If $g(x,y)$ is semihomogeneous in $x \in \mathbb{R}^n$ and concave in scalar $y$, then the following is a quasi-relaxation of $g(x,y) \leq 0$:

$$g(x^1, y_L) + g(x^2, y_U) \leq 0$$

$$\alpha x^L \leq x^1 \leq \alpha x^U$$

$$(1 - \alpha) x^L \leq x^2 \leq (1 - \alpha) x^U$$

$$x = x^1 + x^2$$

Bounds on $x$
Theorem (JNH 2005)

Suppose we minimize $cx$ subject to $g(x,y) \leq 0$.

If $g(x,y)$ is semihomogeneous in $x \in \mathbb{R}^n$ and concave in scalar $y$, then the following is a **quasi-relaxation** of $g(x,y) \leq 0$:

$$g(x^1, y_L) + g(x^2, y_U) \leq 0$$
$$\alpha x^L \leq x^1 \leq \alpha x^U$$
$$(1-\alpha)x^L \leq x^2 \leq (1-\alpha)x^U$$
$$x = x^1 + x^2$$

\[
\frac{E_i}{h_i} A_i v_i = s_i
\]

has the form $g(x,y) = 0$ with $g$ semihomogenous in $x$ because we can write it

\[
\frac{E_i}{h_i} A_i v_i - s_i = 0
\]

with $x = (A_i, s_i), \ y = v_i$. 

Slide 159
Truss Structure Design

So we have a quasi-relaxation of the truss problem:

\[
\begin{align*}
\min & \quad \sum_i h_i [A_i^L y_i + A_i^U (1 - y_i)] \\
\text{s.t.} & \quad \sum_i \cos \theta_{ij} s_i = p_j, \text{ all } j \\
& \quad \sum_j \cos \theta_{ij} d_j = v_{i0} + v_{i1}, \text{ all } i \\
& \quad \frac{E_i}{h_i} (A_i^L v_{i0} + A_i^U v_{i1}) = s_i, \text{ all } i \\
& \quad v_i^L y_i \leq v_{i0} \leq v_i^U y_i, \text{ all } i \\
& \quad v_i^L (1 - y_i) \leq v_{i1} \leq v_i^U (1 - y_i), \text{ all } i \\
& \quad d_j^L \leq d_j \leq d_j^U, \text{ all } j \\
& \quad 0 \leq y_i \leq 1, \text{ all } i
\end{align*}
\]

- Hooke’s law is linearized
- Elongation bounds split into 2 sets of bounds
Truss Structure Design

Logic cuts

\( v_{i0} \) and \( v_{i1} \) must have same sign in a feasible solution.

If not, we branch by adding logic cuts

\[
\begin{align*}
v_{i0}, v_{i1} & \leq 0, \\
v_{i0}, v_{i1} & \geq 0
\end{align*}
\]
Truss Structure Design

SIMPL model

Recognized as linear systems

01. OBJECTIVE
02. maximize sum i of c[i]*h[i]*A[i]
03. CONSTRAINTS
04. equilibrium means {
05.   sum i of b[i,j]*s[i,l] = p[j,l] forall j,l
06.   relaxation = { lp } }
07. compatibility means {
08.   sum j of b[i,j]*d[j,l] = v[i,l] forall i,l
09.   relaxation = { lp } }
10. hook means {
11.   E[i]/h[i]*A[i]*v[i,l] = s[i,l] forall i
12.   relaxation = { lp:quasi } }
13. SEARCH
14.   type = { bb:bestdive }
15.   branching = { hook:first:quasicut, A:splitup }
Truss Structure Design

SIMPL model

01. OBJECTIVE
maximize sum i of c[i]*h[i]*A[i]

03. CONSTRAINTS
04.  equilibrium means {
05.  sum i of b[i,j]*s[i,l] = p[j,l] forall j,l
06.  relaxation = { lp } }
07.  compatibility means {
08.  sum j of b[i,j]*d[j,l] = v[i,l] forall i,l
09.  relaxation = { lp } }
10.  hooke means {
11.    E[i]/h[i]*A[i]*v[i,l] = s[i,l] forall i
12.    relaxation = { lp:quasi } }

13. SEARCH
14.  type = { bb:bestdive }
15.  branching = { hooke:first:quasicut, A:splitup }

Recognized as bilinear system
Truss Structure Design

SIMPL model

01. OBJECTIVE
02. maximize sum i of c[i]*h[i]*A[i]
03. CONSTRAINTS
04. equilibrium means {
05. sum i of b[i,j]*s[i,1] = p[j,1] forall j,1
06. relaxation = { lp } }
07. compatibility means {
08. sum j of b[i,j]*d[j,1] = v[i,1] forall i,1
09. relaxation = { lp } }
10. hooke means {
11. E[i]/h[i]*A[i]*v[i,1] = s[i,1] forall i
12. relaxation = { lp:quasi } }
13. SEARCH
14. type = { bb:bestdive }
15. branching = { hooke:first:quasicut, A:splitup }

Generate quasi-relaxation for semihomogenous function
Truss Structure Design

SIMPL model

01. OBJECTIVE
02. maximize sum i of c[i]*h[i]*A[i]
03. CONSTRAINTS
04. equilibrium means {
05. sum i of b[i,j]*s[i,l] = p[j,l] forall j,l
06. relaxation = { lp } }
07. compatibility means {
08. sum j of b[i,j]*d[j,l] = v[i,l] forall i,l
09. relaxation = { lp } }
10. hooke means {
11. E[i]/h[i]*A[i]*v[i,l] = s[i,l] forall i
12. relaxation = { lp:quasi } }
13. SEARCH
14. type = { bb:bestdive }
15. branching = { hooke:first:quasicut, A:splitup }

Branch first on violated logic cuts for quasi-relaxation
Truss Structure Design

SIMPL model

01. OBJECTIVE
02. maximize sum i of c[i]*h[i]*A[i]
03. CONSTRAINTS
04. equilibrium means {
05. sum i of b[i,j]*s[i,l] = p[j,l] forall j,l
06. relaxation = { lp } }
07. compatibility means {
08. sum j of b[i,j]*d[j,l] = v[i,l] forall i,l
09. relaxation = { lp } }
10. hooke means {
11. E[i]/h[i]*A[i]*v[i,l] = s[i,l] forall i
12. relaxation = { lp:quasi } }
13. SEARCH
14. type = { bb:bestdive }
15. branching = { hooke:first:quasicut, A:splitup }

Then branch on $A_i$ in-domain constraint.

Violated when $A_i$ is not one of the discrete bar sizes.

Take upper branch first.
Truss Structure Design

10-bar cantilever truss

Load
## Truss Structure Design

### Computational results (seconds)

<table>
<thead>
<tr>
<th>No. bars</th>
<th>Loads</th>
<th>BARON</th>
<th>CPLEX 11</th>
<th>Hand coded</th>
<th>SIMPL</th>
</tr>
</thead>
<tbody>
<tr>
<td>10</td>
<td>1</td>
<td>5.3</td>
<td>0.40</td>
<td>0.03</td>
<td>0.08</td>
</tr>
<tr>
<td>10</td>
<td>1</td>
<td>3.8</td>
<td>0.26</td>
<td>0.02</td>
<td>0.07</td>
</tr>
<tr>
<td>10</td>
<td>1</td>
<td>8.1</td>
<td>0.83</td>
<td>0.16</td>
<td>0.49</td>
</tr>
<tr>
<td>10</td>
<td>1</td>
<td>8.8</td>
<td>1.2</td>
<td>0.22</td>
<td>0.63</td>
</tr>
<tr>
<td>10</td>
<td>2</td>
<td>24</td>
<td>4.9</td>
<td>0.64</td>
<td>1.84</td>
</tr>
<tr>
<td>10</td>
<td>2*</td>
<td>327</td>
<td>146</td>
<td>145</td>
<td>65</td>
</tr>
<tr>
<td>10</td>
<td>2*</td>
<td>2067</td>
<td>1087</td>
<td>600</td>
<td>651</td>
</tr>
</tbody>
</table>

*plus displacement bounds
Truss Structure Design

25-bar problem
Truss Structure Design

72-bar problem
## Truss Structure Design

### Computational results (seconds)

<table>
<thead>
<tr>
<th>No. bars</th>
<th>Loads</th>
<th>BARON</th>
<th>CPLEX 11</th>
<th>Hand coded</th>
<th>SIMPL</th>
</tr>
</thead>
<tbody>
<tr>
<td>25</td>
<td>2</td>
<td>3,302</td>
<td>44</td>
<td>44</td>
<td>20</td>
</tr>
<tr>
<td>72</td>
<td>2</td>
<td>3,376</td>
<td>208</td>
<td>33</td>
<td>28</td>
</tr>
<tr>
<td>90</td>
<td>2</td>
<td>21,011</td>
<td>570</td>
<td>131</td>
<td>92</td>
</tr>
<tr>
<td>108</td>
<td>2</td>
<td>&gt; 24 hr*</td>
<td>3208</td>
<td>1907</td>
<td>1720</td>
</tr>
<tr>
<td>200</td>
<td>2</td>
<td>&gt; 24 hr*</td>
<td>&gt; 24 hr*</td>
<td>&gt; 24 hr**</td>
<td>&gt; 24 hr***</td>
</tr>
</tbody>
</table>

* no feasible solution found

** best feasible solution has cost 32,748

*** best feasible solution has cost 32,700
Summary

- We can understand intractability as an epistemic notion.
  - Ignorance of the solution space.
  - No need to view certain problems as inherently hard.

- To defeat intractability, use methods with knowledge of problem structure.
  - In optimization, inference and relaxation techniques exploit structure...
  - ...in the context of primal-dual-dual methods.
Summary

• We can understand intractability as an epistemic notion.
  • Relaxation: Enumerate **master problems** (which assign jobs to machines)

• **Inference:** Generate **nogoods** (logic-based Benders cuts), which are added to master problem.