Goal: Integrate modeling and solution methods

- There is an underlying similarity in several solution methods
  - MILP, constraint programming, Benders decomposition, global optimization, SAT (propositional satisfiability), local search
- They use a **search-infer-relax** algorithmic framework
Search-Infer-Relax

- **Search**: Enumerate problem restrictions.
  - Branching tree nodes, Benders subproblems, local search neighborhoods, etc.
- **Infer**: Deduce constraints from current restriction
  - Nogoods, cutting planes, filtering, etc.
- **Relax**: Solve relaxation of current restriction
  - LP, Lagrangean, domain store, Benders master, etc.
Conventional solution methods

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

- **Constraint Programming solver**
  - **Search**: Branching
  - **Inference**: Filtering
  - **Relaxation**: Domain store

- **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 solver (propositional satisfiability)**
  • **Search:** Branching
  • **Inference:** Conflict clauses
  • **Relaxation:** Same as restriction

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

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

**Guides**
- Separating cut

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

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

**Inference**

**Relaxation**

**Restriction**
Integration Principles

• Integrate MILP, constraint programming, global optimization in a **unified** approach.
Integration Principles

- Integrate MILP, constraint programming, global optimization in a *unified* approach.
- **Low-level** integration with *high-level* modeling.
Integration Principles

- Integrate MILP, constraint programming, global optimization in a **unified** approach.
- **Low-level** integration with **high-level** modeling.
- Succinct modeling with **meta-constraints**.
  - Model communicates **problem structure** to the solver.
Integration Principles

- Integrate MILP, constraint programming, global optimization in a **unified** approach.
- **Low-level** integration with **high-level** modeling.
- Succinct modeling with **meta-constraints**.
  - Model communicates **problem structure** to the solver.
- General **search-infer-relax** solution algorithm.
  - Enumerate problem restrictions.
  - Branching or logic-based Benders.
  - Underlying search/inference and search/relaxation dualities.
Integration Principles

• Integrate MILP, constraint programming, global optimization in a **unified** approach.
• **Low-level** integration with **high-level** modeling.
• Succinct modeling with **meta-constraints**.
  • Model communicates **problem structure** to the solver.
• General **search-infer-relax** solution algorithm.
  • Enumerate problem restrictions.
  • Branching or logic-based Benders.
  • Underlying search/inference and search/relaxation dualities
• **Constraint-based** control.
  • Filtering, relaxation, branching.
Evolution of SIMPL

• Basic Framework
  – JNH. *Logic-based methods for optimization*, *CP* 1994
Evolution of SIMPL


Evolution of SIMPL

- **Search**
  - JNH. *Unifying local and exhaustive search*. *ENC 2005*.

- **Modeling**
Evolution of SIMPL

• Theory and background
Evolution of SIMPL

- Nogood-based search (Logic-based Benders)
  - JNH. *Planning and scheduling to minimize tardiness*. CP 2005.
Evolution of SIMPL

- **Relaxation methods**
  - JNH. *Convex programming methods for global optimization*, *COCOS 2003*
A taste of the underlying theory

- Inference duality.
  - LP, Lagrangean, surrogate duals
- Nogood-based search
  - Logic-based and classical Benders
- Example: SAT
  - DPL
  - DPL + conflict clauses
  - Partial order dynamic backtracking
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
- Provides nogoods
  - Including Benders cuts, conflict clauses
  - Nogood = conditions under which proof is still valid
- Directs the search
  - **All** search is nogood-based search
  - Next restriction defined by nogood
- Postoptimality analysis
  - Result of altering premises of proof
Inference dual

Primal

\[ \text{min } f(x) \]
\[ x \in S \]

Feasible set

Dual

\[ \text{max } \nu \]
\[ x \in S \Rightarrow \nu \geq f(x) \]
\[ P \in \mathcal{P} \]

Follows using proof \( P \)

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

Primal

\[
\begin{align*}
& \text{min } cx \\
& Ax \geq b \\
& x \geq 0
\end{align*}
\]

Dual

\[
\begin{align*}
& \max v \\
& \left\{ \begin{array}{l} 
Ax \geq b \\
x \geq 0 \end{array} \right\} \quad \Rightarrow \\
& \quad cx \geq v
\end{align*}
\]

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

Ax \geq b \Rightarrow cx \geq v \quad \text{iff} \quad \lambda Ax \geq \lambda b \quad \text{dominates} \quad cx \geq v \\
\text{for some } \lambda \geq 0 \\
\lambda A \leq c \quad \text{and} \quad \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 v ) ( = \max \min {f(x) - \lambda g(x)} ) (_{\lambda \geq 0, x \in S} )</td>
</tr>
<tr>
<td>( g(x) \geq 0 )</td>
<td>( g(x) \geq b \Rightarrow f(x) \geq v )</td>
</tr>
<tr>
<td>( x \in S )</td>
<td>( P \in \mathcal{P} ) ( \leftarrow ) Nonnegative linear combination + domination</td>
</tr>
<tr>
<td>( g(x) \geq 0 \Rightarrow f(x) \geq v ) (_{x \in S} )</td>
<td>( \lambda g(x) \geq 0 ) \text{dominates} ( f(x) - \nu \geq 0 ) for some ( \lambda \geq 0 )</td>
</tr>
</tbody>
</table>

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

Or \( \nu \leq \min \{f(x) - \lambda g(x)\} \)

\( \lambda g(x) \leq f(x) - \nu \) for all \( x \in S \)

- Inference method is **incomplete**
Example: Surrogate dual

Primal

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

Dual

\[
\begin{align*}
\max \nu & \\
g(x) \geq b & \Rightarrow f(x) \geq \nu \\
\nu P \in P & \text{ Nonnegative linear combination + implication}
\end{align*}
\]

\[
\lambda g(x) \geq 0 \text{ implies } f(x) \geq \nu \text{ for some } \lambda \geq 0
\]

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

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

• 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 $x$

Subproblem results from fixing $x$

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

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
Example: Classical Benders

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

Subproblem results from fixing $x$

\[
\begin{align*}
\min & \quad f(x) + cy \\
\text{s.t.} & \quad g(x) + Ay \geq b \\
& \quad x \in D_x, \quad y \geq 0
\end{align*}
\]

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

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 $\overline{X}$ Benders cuts
Example: SAT

- Solve SAT by chronological backtracking + unit clause rule = DPL.
  - Chronological = fixed branching order.
- To get nogood, solve subproblem at current node.
  - Solve 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
DPL with chronological backtracking

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

$(x_1, \ldots, x_5) = (0, \ldots, 0)$ creates the infeasibility.
DPL 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, which proves infeasibility.

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

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

Generate nogood.
DPL 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>$(0,0,0,0,0,\cdot)$</td>
<td>$x_1 \vee x_2 \vee x_3 \vee x_4 \vee x_5$</td>
</tr>
<tr>
<td>1</td>
<td>$x_1 \vee x_2 \vee x_3 \vee x_4 \vee x_5$</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Conflict clause appears as nogood induced by solution of $R_k$. 
### Relaxation Solution of 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>$x_1 \lor x_2 \lor x_3 \lor x_4 \lor x_5$</td>
</tr>
<tr>
<td>1</td>
<td>$x_1 \lor x_2 \lor x_3 \lor x_4 \lor x_5$</td>
<td>$(0,0,0,0,1,\cdot)$</td>
<td>$x_1 \lor x_2 \lor x_3 \lor x_4 \lor \overline{x}_5$</td>
</tr>
</tbody>
</table>

- **DPL with chronological backtracking**: Consists of processed nogoods
- **Go to solution that solves relaxation, with priority to 0**
DPL 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>$x_1 \vee x_2 \vee x_3 \vee x_4 \vee x_5$</td>
<td>(0,0,0,0,-)</td>
<td>$x_1 \vee x_2 \vee x_3 \vee x_4 \vee x_5$</td>
</tr>
<tr>
<td>1</td>
<td>$x_1 \vee x_2 \vee x_3 \vee x_4 \vee x_5$</td>
<td>(0,0,0,0,1,-)</td>
<td>$x_1 \vee x_2 \vee x_3 \vee x_4 \vee \overline{x}_5$</td>
</tr>
</tbody>
</table>

Process nogood set with parallel resolution

parallel-absorbs

$x_1 \vee x_2 \vee x_3 \vee x_4 \vee x_5$

$x_1 \vee x_2 \vee x_3 \vee x_4 \vee \overline{x}_5$
DPL 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></td>
<td></td>
</tr>
</tbody>
</table>

Process nogood set with
**parallel resolution**

**parallel-absorbs**

- $x_1 \lor x_2 \lor x_3 \lor x_4 \lor x_5$
- $x_1 \lor x_2 \lor x_3 \lor x_4 \lor \overline{x_5}$
DPL 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

- Nogoods = conflict clauses.
  - Nogoods rule out only branches that play a role in unit clause refutation.
DPL with conflict clauses

$x_1 = 0$

$x_2 = 0$

$x_3 = 0$

$x_4 = 0$

$x_5 = 0$

Branch to here. Unit clause rule proves infeasibility.

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

\[ x_1 = 0 \]
\[ x_2 = 0 \]
\[ x_3 = 0 \]
\[ x_4 = 0 \]
\[ x_5 = 0 \]

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></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></td>
<td></td>
</tr>
</tbody>
</table>
DPL 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>
</tbody>
</table>
DPL 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>(0,0,0,0,0,·)</td>
<td>$x_1 \lor x_5$</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>$x_1 \lor x_5$</td>
<td>(0,0,0,0,1,·)</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 $x_1 \lor x_2$

$x_2 \lor \overline{x_5}$ parallel-absorbs $x_1 \lor x_5$

$x_2 \lor \overline{x_5}$
DPL with conflict clauses

<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..)</td>
<td>$x_1 \lor x_5$</td>
</tr>
<tr>
<td>1</td>
<td>$x_1 \lor x_5$</td>
<td>(0,0,0,0,1,0..)</td>
<td>$x_2 \lor \overline{x_5}$</td>
</tr>
<tr>
<td>2</td>
<td>$x_1 \lor x_2$</td>
<td>(0,1,0,0,0,0..)</td>
<td>$x_1 \lor \overline{x_2}$</td>
</tr>
<tr>
<td>3</td>
<td>$x_1$</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

$x_1 \lor x_2$ parallel-resolve to yield $x_1$
DPL with conflict clauses

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

\[ x_2 = 0 \]

\[ x_3 = 0 \quad x_1 \lor \bar{x}_2 \]

\[ x_4 = 0 \quad x_1 \lor \bar{x}_5 \]

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

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

\[ k \quad \text{Relaxation } R_k \quad \text{Solution of } R_k \quad \text{Nogoods} \]

<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>$x_1 \lor x_5$</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>$x_2 \lor \bar{x}_5$</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>$(1,\cdot,\cdot,\cdot,\cdot)$</td>
<td>$\bar{x}_1$</td>
</tr>
<tr>
<td>4</td>
<td>$\emptyset$</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Search terminates
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
**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></td>
<td></td>
</tr>
</tbody>
</table>

- Other variables are penultimate
- Arbitrarily choose one variable to be last
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>(1,0,0,0,0,0,0)</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></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></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>$(0,0,0,0,0,\cdot)$</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>$\overline{x_5}$</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.

Parallel-resolution:

- $x_5 \lor x_1$
- $x_5 \lor \overline{x_1}$ to yield $x_5$

Choice of last variable is arbitrary but must be consistent with partial order implied by previous choices.
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>(0,0,0,0,0,0,0)</td>
<td>$x_5 \lor x_1$</td>
</tr>
<tr>
<td>1</td>
<td>$x_1 \lor x_5$</td>
<td>(1,0,0,0,0,0,0)</td>
<td>$x_5 \lor \overline{x}_1$</td>
</tr>
<tr>
<td>2</td>
<td>$x_5$</td>
<td>(0,0,0,0,1,0,0)</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>(0,0,0,0,0,0,0)</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>$x_1 \lor x_5$</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>$x_5$</td>
<td>$(\cdot,0,\cdot,1,\cdot)$</td>
<td>$\overline{x}_5 \lor x_2$</td>
</tr>
<tr>
<td>3</td>
<td>$\left{ x_5 \right}$</td>
<td>$(\cdot,1,\cdot,1,\cdot)$</td>
<td>$\overline{x}_2$</td>
</tr>
<tr>
<td>4</td>
<td>$\emptyset$</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Search terminates**

**Must conform**
Examples, with results from SIMPL

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

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

\[
\max \sum_{i} f_i(x_i) \quad \text{Each } f_i \text{ is a piecewise linear semicontinuous function}
\]

\[
\sum_{i} x_i \leq C
\]
Production Planning

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

**MILP model**

\[
\begin{align*}
\text{min} & \quad \sum_{ik} \lambda_{ik} c_{ik} + \mu_{ik} d_{ik} \\
\sum_i x_i & \leq C \\
x_i & = \sum_k \lambda_{ik} L_{ik} + \mu_{ik} U_{ik}, \quad \text{all } i \\
\sum_k \lambda_{ik} + \mu_{ik} & = 1, \quad \text{all } i \\
0 & \leq \lambda_{ik} \leq y_{ik}, \quad \text{all } i, k \\
0 & \leq \mu_{ik} \leq y_{ik}, \quad \text{all } i, k \\
\sum_k y_{ik} & = 1, \quad \text{all } i \\
y_{ik} & \in \{0,1\}, \quad \text{all } i, k
\end{align*}
\]
Production Planning

**MILP model**

\[
\begin{align*}
\min & \sum_{ik} \lambda_{ik} c_{ik} + \mu_{ik} d_{ik} \\
\sum_i x_i & \leq C \\
x_i &= \sum_k \lambda_{ik} L_{ik} + \mu_{ik} U_{ik}, \quad \text{all } i \\
\sum_k \lambda_{ik} + \mu_{ik} &= 1, \quad \text{all } i \\
0 &\leq \lambda_{ik} \leq y_{ik}, \quad \text{all } i, k \\
0 &\leq \mu_{ik} \leq y_{ik}, \quad \text{all } i, k \\
\sum_k y_{ik} &= 1, \quad \text{all } i \\
y_{ik} &\in \{0,1\}, \quad \text{all } i, k
\end{align*}
\]

= 1 if \( x_i \) is in interval \( k \)

**SOS2 branching not useful**
Production Planning

Integrated model

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

Metaconstraint

(global constraint in CP)

Production Planning

Semicontinuous piecewise linear function $f(x)$

Tight linear relaxation
Production Planning

Semiconcious piecewise linear function $f(x)$

Tighter relaxation after branching

Value of $x$ in solution of current linear relaxation
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 }
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 }

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. CONSTRANTS
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.
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-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

<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

**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>

**Nodes**

<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

---

Slide 69
Product configuration

Unit cost of producing attribute \( j \)

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

MILP model

\[
\min \sum_j c_j v_j
\]

\[
v_j = \sum_{tk} A_{ijk} q_{ik}, \text{ all } j
\]

\( L_j \leq v_j \leq U_j, \text{ all } j \)

\( q_{ik} \leq M_i x_{ik}, \text{ all } i, k \)

\( \sum_k x_{ik} = 1, \text{ all } i \)

Quantity of component \( i \) installed
Product configuration

Unit cost of producing attribute $j$

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

MILP model

Minimize $\sum_j c_j v_j$

$V_j = \sum_{tk} A_{ijk} q_{ik}$, all $j$

$L_j \leq v_j \leq U_j$, all $j$

$q_{ik} \leq M_i x_{ik}$, all $i, k$

$\sum_k x_{ik} = 1$, all $i$

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

Quantity of component $i$ installed
Product configuration

Unit cost of producing attribute $j$

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

MILP model

$$\text{min } \sum_j c_j v_j$$

$$v_j = \sum_{tk} A_{ijk} q_{ik}, \text{ all } j$$

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

$$q_{ik} \leq M_i x_{ik}, \text{ all } i, k$$

$$\sum_k x_{ik} = 1, \text{ all } i$$

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

Quantity of component $i$ installed

1 if type $k$ of component $i$ is used
Product configuration

Unit cost of producing attribute $j$

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

Integrated model

$$\min \sum_{j} c_j v_j$$

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

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

Quantity of component $i$ installed

Product configuration

Unit cost of producing attribute $j$

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

\[
\min \sum_j c_j v_j
\]

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

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

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

Integrated model

Quantity of component $i$ installed

Product configuration

Unit cost of producing attribute $j$

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

$$\min \sum_j c_j v_j$$

$$v_j = \sum_{i,k} A_{ijk} q_{ij}, \text{ all } j$$

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

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

Integrated model

Product configuration

Linear inequality
metaconstraint

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

Product configuration

Indexed linear metaconstraint

\[ \min \sum_j c_j v_j \]

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

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

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

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

\[ \text{element}\left(t_i, (q_i, A_{ij_1}, \ldots, q_i A_{ijn}), z_i\right), \text{ all } i, j \]

\[ \min \sum_j c_j v_j \]

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

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

*This is rewritten as*

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

Propagation

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

\[ \text{element} \left( t_i, (q_i, A_{ij_1}, \ldots, q_i A_{ij_n}), z_i \right), \text{ all } i, j \]

This is propagated by:
(a) using specialized filters for element constraints of this form,
(b) adding knapsack cuts for the valid inequalities:

\[ \sum_i \max_{k \in D_{ti}} \{ A_{ijk} \} q_i \geq v_j, \text{ all } j \]

\[ \sum_i \min_{k \in D_{ti}} \{ A_{ijk} \} q_i \leq v_j, \text{ all } j \]

and (c) propagating the knapsack cuts.

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

Relaxation

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

This is relaxed as

\[
v_j \leq v_j \leq \bar{v}_j
\]
This is relaxed by relaxing this and adding the knapsack cuts.

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

element \((t_i, (q_i, A_{ij_1}, \ldots, q_i A_{ijn}), z_i), \text{ all } i, j\)

\[ \min \sum_j c_j v_j \]

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

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

This is relaxed as

\[ v_j \leq v_j \leq \bar{v}_j \]
Product configuration

Relaxation

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

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

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

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

01. OBJECTIVE
02. \( \text{minimize } \sum_j c[j] * v[j] \)
03. CONSTRAINTS
04. usage means {
05. \[v[j] = \sum_i q[i] * a[i][j][t[i]] \text{forall } j\]
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 }

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]] 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:reducost }

SIMPL model

LP relaxation is convex hull of disjunction.
CP relaxation propagates bounds.
Product configuration

01. OBJECTIVE
02. \[ \text{minimize} \ \sum_j c[j] \cdot v[j] \]
03. CONSTRAINTS
04. usage means \{
05. \[ v[j] = \sum_i q[i] \cdot a[i][j][t[i]] \] for all j
06. relaxation = \{ lp, cp \}
07. \text{inference} = \{ \text{knapsack} \} \}
08. quantities means \{
09. \[ q[1] \geq 1 \Rightarrow q[2] = 0 \]
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

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 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

Logical constraint on quantities
Product configuration

01. OBJECTIVE
02. minimize \( \sum_j c[j] \times v[j] \)
03. CONSTRAINTS
04. usage means {
05. \( v[j] = \sum_i q[i] \times a[i][j][t[i]] \) for all \( j \)
06. relaxation = \{ lp, cp \}
07. inference = \{ knapsack \} }
08. quantities means {
09. \( q[1] \geq 1 \Rightarrow q[2] = 0 \)
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

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]] 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 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: \( q_i = \) nearest integer \( q'_i \), \( q_i < q'_i \), \( q_i > q'_i \)
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 most violated logical constraint on $t_i$ variables (omitted)
Product configuration

01. OBJECTIVE

02. \[ \text{minimize } \sum j \text{ of } c[j] \times v[j] \]

03. CONSTRAINTS

04. usage means {

05. \[ v[j] = \sum i \text{ of } q[i] \times a[i][j][t[i]] \quad \text{forall } j \]

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 \}

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

### Example

Assign 5 jobs to 2 machines.
Schedule jobs assigned to each machine without overlap.

<table>
<thead>
<tr>
<th>Job</th>
<th>Release time $r_j$</th>
<th>Deadline $d_j$</th>
<th>Processing time $p_{Aj}$</th>
<th>Processing time $p_{Bj}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0</td>
<td>9</td>
<td>1</td>
<td>5</td>
</tr>
<tr>
<td>2</td>
<td>0</td>
<td>9</td>
<td>3</td>
<td>6</td>
</tr>
<tr>
<td>3</td>
<td>2</td>
<td>7</td>
<td>3</td>
<td>7</td>
</tr>
<tr>
<td>4</td>
<td>2</td>
<td>9</td>
<td>4</td>
<td>6</td>
</tr>
<tr>
<td>5</td>
<td>4</td>
<td>7</td>
<td>2</td>
<td>5</td>
</tr>
</tbody>
</table>
Machine scheduling

**MILP continuous-time model**

from Jain & Grossmann

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

\[r_j \leq s_j \leq d_j - \sum_i p_{ij} x_{ij}, \quad \text{all } j\]

\[\sum_i x_{ij} = 1, \quad \text{all } j\]

\[y_{jj'} + y_{jj} \leq 1, \quad \text{all } j' > j\]

\[y_{jj'} + y_{jj} + x_{ij} + x_{ij'} \leq 2, \quad \text{all } j' > j, i' \neq i\]

\[y_{jj'} + y_{jj} \geq x_{ij} + x_{ij'} - 1, \quad \text{all } j' > j, i\]

\[s_{jj'} \geq s_j + \sum_i p_{ij} x_{ij} - M(1 - y_{jj'}), \quad \text{all } j' \neq j\]

\[\sum_j p_{ij} x_{ij} \leq \max_j d_j - \min_j r_j, \quad \text{all } i\]

\[x_{ij} \in \{0,1\}, \quad y_{jj'} \in \{0,1\}, \text{ all } j' \neq j\]
Machine scheduling

MILP continuous-time model

from Jain & Grossmann

\[
\begin{align*}
\text{min} & \sum_{ij} c_{ij} x_{ij} \\
r_j & \leq s_j \leq d_j - \sum_i p_{ij} x_{ij}, \quad \text{all } j \\
\sum_i x_{ij} & = 1, \quad \text{all } j \\
y_{jj'} + y_{jj} & \leq 1, \quad \text{all } j' > j \\
y_{jj'} + y_{jj} + x_{ij} + x_{i'j'} & \leq 2, \quad \text{all } j' > j, i' \neq i \\
y_{jj'} + y_{jj} & \geq x_{ij} + x_{i'j'} - 1, \quad \text{all } j' > j, i \\
s_j & \geq s_j + \sum_i p_{ij} x_{ij} - M(1 - y_{jj'}), \quad \text{all } j' \neq j \\
\sum_j p_{ij} x_{ij} & \leq \max_j \{d_j\} - \min_j \{r_j\}, \quad \text{all } i \\
x_{ij} & \in \{0,1\}, \quad y_{jj'} \in \{0,1\}, \text{ all } j' \neq j
\end{align*}
\]

= 1 if job j assigned to machine i

= 1 of job j precedes j'

job start time
Machine scheduling

Integrated model

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

\[
 r_j \leq s_j \leq d_j - p_{x_{ij}}, \text{ all } j
\]

Disjunctive \((s_j | x_j = i), (p_{ij} | x_j = i)\), all \(i\)

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 model

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

\[ r_j \leq s_j \leq d_j - p_{x_{ij}}, \text{ all } j \]

disjunctive\((s_j \mid x_j = i), (p_{ij} \mid x_j = i)\), all \(i\)
Machine scheduling

Integrated model

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

\[ r_j \leq s_j \leq d_j - p_{x_{ij}}, \text{ all } j \]

\[ \text{disjunctive}\left((s_j \mid x_j = i), (p_{ij} \mid x_j = i)\right), \text{ all } i \]

Indexed linear metaconstraint

Disjunctive scheduling metaconstraint
Machine scheduling

Integrated model

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

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

\[
\begin{align*}
\min & \quad M \\
M & \geq s_j + p_{x_j}, \text{ all } j \text{ with } \bar{x}_j = i \\
r_j & \leq s_j \leq d_j - p_{x_j}, \text{ all } j \text{ with } \bar{x}_j = i \\
\text{disjunctive}(s_j | \bar{x}_j = i), (p_{ij} | \bar{x}_j = i)
\end{align*}
\]
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 relaxation, formulated as an MILP:

$$\min \sum_{ij} c_{ij} x_{ij}$$

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

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

$$x_{A2} + x_{A3} + x_{A5} \leq 2$$

$$x_{ij} \in \{0,1\}$$
Machine scheduling

The master problem is a relaxation, formulated as an MILP:

$$\min \sum_{ij} c_{ij} x_{ij}$$

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

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

$$x_{A2} + x_{A3} + x_{A5} \leq 2$$

$$x_{ij} \in \{0, 1\}$$

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

Also for cumulative scheduling.
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 forall j;
06.   relaxation = { ip:master } }
07. xy means {
08.   x[i][j] = 1 <=> y[i][j] = 1 forall i, j;
09.   relaxation = { cp } }
10. tbounds means {
11.   x[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

01. OBJECTIVE
02. \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. \textbf{relaxation} = \{ \text{ip:master} \} \}
07. xy means {
08. x[i][j] = 1 \iff y[i][j] = 1 \forall i, j;
09. relaxation = \{ cp \} \}
10. tbounds means {
11. x[j] \leq t[j] \forall j;
12. t[j] \leq d[j] - p[y[j]][j] \forall j;
13. relaxation = \{ \text{ip:master, cp} \} \}
14. machinecap means {
15. \text{cumulative}\{ t[j], p[i][j], 1 \} \forall j \mid x[i][j] = 1, 1 \forall i;
16. relaxation = \{ cp:\text{subproblem, ip:master} \}
17. inference = \{ \text{feasibility} \} \}
18. SEARCH
19. type = \{ benders \}

SIMPL model

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

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

SIMPL model

Definition of \(x_{ij}\) variables for MILP master problem
Machine scheduling

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

SIMPL model

CP-based propagation
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 forall j;
06.      relaxation = { ip:master } }
07.  xy means {
08.      x[i][j] = 1 <=> y[i][j] = 1 forall i, j;
09.      relaxation = { cp } }
10.  tbounds means {
11.      x[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 }

SIMPL model

Time window constraints recognized as indexed linear system
Machine scheduling

SIMPL model

01. OBJECTIVE
02. min sum i,j of c[i][j] * x[i][j];
03. CONSTRANIS
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[i][j] = 1 forall i, j;
09. relaxation = { cp } }
10. tbounds means {
11. t[j] <= d[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

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

```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[i][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 }
```

**SIMPL model**

The CP problem goes into the Benders subproblem.
A relaxation of the constraint goes into the master.
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[i][j] = 1 forall i, j;
09. relaxation = { cp } }
10. tbounds means {
11. x[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[i][j] = 1 forall i, j;
09.   relaxation = { cp } }
10. tbounds means {
11.   x[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>Sec.</td>
</tr>
<tr>
<td>3</td>
<td>2</td>
<td>1</td>
<td>0.01</td>
</tr>
<tr>
<td>7</td>
<td>3</td>
<td>1</td>
<td>0.02</td>
</tr>
<tr>
<td>12</td>
<td>3</td>
<td>499</td>
<td>0.98</td>
</tr>
<tr>
<td>15</td>
<td>5</td>
<td>529</td>
<td>2.6</td>
</tr>
<tr>
<td>20</td>
<td>5</td>
<td>250,047</td>
<td>369</td>
</tr>
<tr>
<td>22</td>
<td>5</td>
<td>&gt; 27.5 mil.</td>
<td>&gt; 48 hr</td>
</tr>
<tr>
<td>25</td>
<td>5</td>
<td>&gt; 5.4 mil.</td>
<td>&gt; 19 hr*</td>
</tr>
</tbody>
</table>

*out of memory
Machine scheduling

Benders cut for minimum makespan, with a cumulative scheduling subproblem

\[ M \geq \min_{i \in J_i} \left( \sum_{j \in J_i} p_{ij} (1 - x_{ij}) + \max_{j \in J_i} \{d_j\} - \min_{j \in J_i} \{d_j\} \right) \]

- 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.
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.
• LP, surrogate, Lagrangean, and superadditive duals are special cases of inference duality and relaxation duality.
  • Whence the prevalence of relaxation and inference dualities in problem solving.
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

\( v_i = \) elongation of bar

\( s_i = \) force along bar

\( d_j = \) node displacement

\( h_i = \) length of bar \( i \)

\( A_i = \) cross-sectional area of bar

\( p_j = \) load along d.f. \( j \)
Truss Structure Design

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

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

Constraints can be imposed for multiple loading conditions
Truss Structure Design

Introducing new variables linearizes the problem but makes it much larger.

\[ \text{min } \sum_i h_i \sum_k A_{ik} y_{ik} \]

\[ \text{s.t. } \sum_i \cos \theta_{ij} s_i = p_j, \text{ all } j \]

\[ \sum_j \cos \theta_{ij} d_j = \sum_k v_{ik}, \text{ all } i \]

\[ \frac{E_i}{h_i} \sum_k A_{ik} v_{ik} = s_i, \text{ all } i \]

\[ v_i^L \leq v_i \leq v_i^U, \text{ all } i \]

\[ d_j^L \leq d_j \leq d_j^U, \text{ all } j \]

\[ \sum_k y_{ik} = 1, \text{ all } i \]
Truss Structure Design

**Integrated approach**

- Use the **original model** (don’t introduce new variables)
- Branch by **splitting** the range of areas $A_i$
- Generate a **quasi-relaxation**, which is linear and much smaller than MILP model.

Truss Structure Design

**Theorem** (JNH 2005)

If $g(x,y)$ is semihomogeneous in $x \in R^n$ 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)**

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 $g(x,y) \leq 0$:

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

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)

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

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

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

\[g(0, y) = 0 \quad \text{for all } y\]
Truss Structure Design

**Theorem (JNH 2005)**

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)

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

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

Bounds on \( x \)
Theorem (JNH 2005)

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

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

\[
\frac{E_i}{h_i} A_i v_i = s_i \quad \text{has the form } g(x,y) = 0 \text{ with } g \text{ semihomogenous in } x
\]
because we can write it

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

with \( x = (A_i, s_i) \), \( y = v_i \).
Truss Structure Design

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

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

In general, we can have a **metaconstraint** to represent the semihomogeneous constraint $g(x,y) \leq 0$ and generate a quasi-relaxation.

Since a bilinear constraint $xy = \alpha$ is always semihomogeneous, we will use a **bilinear** metaconstraint with a quasi-relaxation option.
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. hookes means {
11. E[i]/h[i] * A[i] * v[i,l] = s[i,l] forall i
12. relaxation = { lp:quasi }
13. SEARCH
14. type = { hh:best.dive }
15. branching = { hookes:quasicut, A:splitup }
Truss Structure Design

SIMPL model

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

15. branching = { hooke: first : quasicut, A: splitup }
Truss Structure Design

SIMPL model

Then branch on $A_i$ in-domain constraint.

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

Take upper branch first.

Slide 143

```
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,j] = p[j,i] forall j,i
06. relaxation = { lp }
07. compatibility means {
08. sum j of b[i,j]*d[j,i] = v[i,j] forall i,j
09. relaxation = { lp }
10. hooke means {
11. E[i]/h[i]*A[i]*v[i,j] = s[i,j] forall i
12. relaxation = { lp:quasi }
13. SEARCH
14. type = { hh:best.dive }
15. branching = { hooke:first:quasicut, A:splitup }
```
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
Current Version of SIMPL

- To download:
  - Click the link to SIMPL on John Hooker’s website.
  - See readme file for complete instructions.
  - Download executable and associated files
- Operational on GNU/Linux only
- Requires subsidiary solvers
  - CPLEX (version 9, 10, or 11)
  - Eclipse (any version 5.8.80 or later), free download
- Download problem instances
  - Including all reported in this talk.
Proposal for SCIP/SIMPL cooperation

- A unique opportunity
- What SCIP/SIMPL can offer
- Primary tasks
A unique opportunity

• Next generation of general-purpose solvers
  • High-level modeling that integrates CP, MIP, GO, LS.
  • Efficient code
  • Exploits program micro-structure through meta-constraints
  • Non-commercial

• SCIP brings efficient algorithmic tools
  • Advanced MIP technology.

• SIMPL brings experience in integrated problem-solving
  • High-level modeling
SCIP offers…

- …highly efficient noncommercial LP solver
  - SIMPL currently uses CPLEX.
- …well-developed MIP code.
  - SIMPL has rudimentary MIP features.
- …many cutting planes
  - SIMPL implements few.
- …constraint handles
  - For user-supplied constraints
SIMPL offers…

• …high-level modeling language
  • SCIP uses ZIMPL language.
• …built-in integration of CP, MIP, GO.
  • SCIP leaves this largely to user.
• …some CP filters (needs more)
  • SCIP leaves it to user to write constraint handles.
• …nogood-based search (logic-based Benders)
  • Not available in SCIP
Primary tasks

- Implement library of metaconstraints
  - Design high-level modeling language or user interface
  - Filters
  - Relaxations / convexification / rules for pooling relaxations
  - Cutting planes / disjunctive MIP formulations
  - Constraint-based branching choices
- Implement single branching scheme for CP & MIP
  - Combine propagation and relaxation
- Implement other search schemes
  - Nogood-based search
  - Continuous global optimization
  - Local search?
  - Perhaps integrate these with CP/MIP branching