An Integrated Solver for Optimization Problems

Ionuț D. Aron\textsuperscript{1}  John N. Hooker\textsuperscript{2}  Tallys H. Yunes\textsuperscript{3}

\textsuperscript{1}Department of Computer Science, Brown University
\textsuperscript{2}Tepper School of Business, Carnegie Mellon University
\textsuperscript{3}Department of Management Science, University of Miami

June 6 2006
I’d Like To Thank the Program Committee for...

A “foolish” model depends on your vocabulary

Larger vocabulary ⇒ more natural models

Some things we once worried about are now automatic

Aron, Hooker and Yunes
An Integrated Solver for Optimization Problems
I’d Like To Thank the Program Committee for...

- Inviting me to be here
I’d Like To Thank the Program Committee for...

- Inviting me to be here
- Placing my talk right after Jon Lee’s
I’d Like To Thank the Program Committee for...

- Inviting me to be here
- Placing my talk right after Jon Lee’s
  - A “foolish” model depends on your vocabulary

Aron, Hooker and Yunes
An Integrated Solver for Optimization Problems
I’d Like To Thank the Program Committee for...

- Inviting me to be here
- Placing my talk right after Jon Lee’s
  - A “foolish” model depends on your vocabulary
  - Larger vocabulary $\Rightarrow$ more natural models
I’d Like To Thank the Program Committee for...

- Inviting me to be here
- Placing my talk right after Jon Lee’s
  - A “foolish” model depends on your vocabulary
  - Larger vocabulary $\Rightarrow$ more natural models
  - Some things we once worried about are now automatic
Outline

- Introduction and Motivation
- **SIMPL** Concepts
- 3 Modeling Examples
- Computational Experiments
- Future Work and Conclusion
Why Integrate?

Integration: combination of two or more solution techniques into a well-coordinated optimization algorithm. Recent research shows integration can sometimes significantly outperform traditional methods in:

▶ Planning and scheduling (jobs, crews, sports, etc.)
▶ Routing and transportation
▶ Engineering and network design
▶ Manufacturing
▶ Inventory management
▶ Etc.

Aron, Hooker and Yunes
An Integrated Solver for Optimization Problems
Why Integrate?

Integration: combination of two or more solution techniques into a well-coordinated optimization algorithm

- Planning and scheduling (jobs, crews, sports, etc.)
- Routing and transportation
- Engineering and network design
- Manufacturing
- Inventory management
- Etc.
Why Integrate?

Integration: combination of two or more solution techniques into a well-coordinated optimization algorithm

Recent research shows integration can sometimes significantly outperform traditional methods in:

- Planning and scheduling (jobs, crews, sports, etc.)
- Routing and transportation
- Engineering and network design
- Manufacturing
- Inventory management
- Etc.
Why Integrate?

Integration: combination of two or more solution techniques into a well-coordinated optimization algorithm

Recent research shows integration can sometimes significantly outperform traditional methods in
- Planning and scheduling (jobs, crews, sports, etc.)
- Routing and transportation
- Engineering and network design
- Manufacturing
- Inventory management
- Etc.
Integration Continued

We are mostly interested in combining traditional OR techniques (LP, MILP) with Constraint Programming. Some benefits of integration:

- Models are simpler, smaller, and more natural.
- It combines complementary strengths of different optimization techniques.
- Problem structure is more easily captured and exploited.

Aron, Hooker and Yunes
An Integrated Solver for Optimization Problems
We are mostly interested in combining traditional OR techniques (LP, MILP) with Constraint Programming.

Aron, Hooker and Yunes

An Integrated Solver for Optimization Problems
We are mostly interested in combining traditional OR techniques (LP, MILP) with Constraint Programming.

Some benefits of integration:
We are mostly interested in combining traditional OR techniques (LP, MILP) with Constraint Programming.

Some benefits of integration:
- Models are simpler, smaller and more natural.
Integration Continued

We are mostly interested in combining traditional OR techniques (LP, MILP) with Constraint Programming

Some benefits of integration:

- Models are simpler, smaller and more natural
- It combines complementary strengths of different optimization techniques
We are mostly interested in combining traditional OR techniques (LP, MILP) with Constraint Programming.

Some benefits of integration:

- Models are simpler, smaller and more natural.
- It combines complementary strengths of different optimization techniques.
- Problem structure is more easily captured and exploited.

Aron, Hooker and Yunes

An Integrated Solver for Optimization Problems
Constraint Programming (CP)

Originated from the AI and CS communities (80's)
Concerned with Constraint Satisfaction Problems (CSPs):

- Given variables $x_i \in D_i$
- Given constraints $c_i: D_1 \times \cdots \times D_n \rightarrow \{T, F\}$
- Assign values to variables to satisfy all constraints

Main Ideas:
- Constraints eliminate infeasible values: domain reduction
- Local inferences are shared: constraint propagation
Constraint Programming (CP)

- Originated from the AI and CS communities (80’s)

- Concerned with Constraint Satisfaction Problems (CSPs):
  - Given variables $x_i \in D_i$
  - Given constraints $c_i: D_1 \times \cdots \times D_n \rightarrow \{T, F\}$
  - Assign values to variables to satisfy all constraints

Main Ideas:
- Constraints eliminate infeasible values: domain reduction
- Local inferences are shared: constraint propagation
Constraint Programming (CP)

- Originated from the AI and CS communities (80’s)
- Concerned with **Constraint Satisfaction Problems** (CSPs):
  - Given variables $x_i \in D_i$
  - Given constraints $c_i : D_1 \times \cdots \times D_n \rightarrow \{T, F\}$
  - Assign values to variables to satisfy all constraints

Main Ideas:
- Constraints eliminate infeasible values: domain reduction
- Local inferences are shared: constraint propagation

Aron, Hooker and Yunes
An Integrated Solver for Optimization Problems
Constraint Programming (CP)

- Originated from the AI and CS communities (80’s)
- Concerned with **Constraint Satisfaction Problems (CSPs)**:
  - Given variables $x_i \in D_i$
  - Given constraints $c_i : D_1 \times \cdots \times D_n \rightarrow \{T, F\}$
  - Assign values to variables to satisfy all constraints

- **Main Ideas:**
  - Constraints eliminate infeasible values: **domain reduction**
  - Local inferences are shared: **constraint propagation**
When compared to MILP models, CP models usually have more informative (larger) variable domains:

- $\text{city} \in \{\text{Atlanta, Boston, Miami, San Francisco}\}$
- $\ell_i =$ location of facility $i$
- contrast with: $x_{ij} = 1$ if facility $i$ is placed in location $j$

More expressive constraints:

- allDifferent: all variables from a set assume distinct values
- element: implements variable indices ($C_x$)
- cumulative: job scheduling with resource constraints
- etc.

These are called global constraints.
When compared to MILP models, CP models usually have

More informative (larger) variable domains:

- city ∈ {Atlanta, Boston, Miami, San Francisco}
- ℓ_i = location of facility i
- contrast with: x_{ij} = 1 if facility i is placed in location j

More expressive constraints:

- alldifferent: all variables from a set assume distinct values
- element: implements variable indices (C x)
- cumulative: job scheduling with resource constraints
- etc.

These are called global constraints
When compared to MILP models, CP models usually have

- More **informative** (larger) variable domains:
  - city ∈ { Atlanta, Boston, Miami, San Francisco }
  - $\ell_i =$ location of facility $i$
  - **contrast with:** $x_{ij} = 1$ if facility $i$ is placed in location $j$
When compared to MILP models, CP models usually have

- More **informative** (larger) variable domains:
  - $\text{city} \in \{ \text{Atlanta, Boston, Miami, San Francisco} \}$
  - $\ell_i = \text{location of facility } i$
  - contrast with: $x_{ij} = 1$ if facility $i$ is placed in location $j$

- More **expressive** constraints:
  - alldifferent: all variables from a set assume distinct values
  - element: implements variable indices ($C_x$)
  - cumulative: job scheduling with resource constraints
  - etc.

These are called **global constraints**
When compared to MILP models, CP models usually have

- More **informative** (larger) variable domains:
  - city ∈ { Atlanta, Boston, Miami, San Francisco }
  - $\ell_i =$ location of facility $i$
  - contrast with: $x_{ij} = 1$ if facility $i$ is placed in location $j$

- More **expressive** constraints:
  - alldifferent: all variables from a set assume distinct values
  - element: implements variable indices ($C_x$)
  - cumulative: job scheduling with resource constraints
  - etc.
  - These are called **global constraints**
Existing modeling languages and programming libraries support integration to a greater or lesser extent: ECLiPSe, Radošek, Wallace and Rajian 99, OPL, Van Hentenryck, Lustig, Michel and Puget 99, Mosel, Colombani and Heipcke 02, SCIP, Achterberg 04.
Existing modeling languages and programming libraries support integration to a greater or lesser extent:
Existing modeling languages and programming libraries support integration to a greater or lesser extent:

- **ECLiPSe**, Rodošek, Wallace and Rajian 99
- **OPL**, Van Hentenryck, Lustig, Michel and Puget 99
- **Mosel**, Colombani and Heipcke 02
- **SCIP**, Achterberg 04
Some relevant concepts and techniques:

- Allowing information exchange among solvers: Rodošek, Wallace & Hajian 99
- Decomposition approaches: Benders 62, Eremin & Wallace 01, Hooker & Ottosson 03, Hooker & Yan 95, Jain & Grossmann 01
- Relaxation of global constraints as systems of linear inequalities: Hooker 00, Refalo 00, Williams & Yan 01, Yunes 02
- Propagation and variable fixing using relaxations: Focacci, Lodi & Milano 99
- Generation of cutting planes as a form of logical inference: Bockmayr & Kasper 98, Bockmayr & Eisenbrand 00

Aron, Hooker and Yunes
An Integrated Solver for Optimization Problems
Some relevant concepts and techniques:
Some relevant concepts and techniques:

- Allowing information exchange among solvers: Rodošek, Wallace & Hajian 99
Previous Work (continued)

Some relevant concepts and techniques:

- Allowing information exchange among solvers: Rodošek, Wallace & Hajian 99

- Decomposition approaches: Benders 62, Eremin & Wallace 01, Hooker & Ottosson 03, Hooker & Yan 95, Jain & Grossmann 01
Some relevant concepts and techniques:

- Allowing information exchange among solvers: Rodošek, Wallace & Hajian 99
- Decomposition approaches: Benders 62, Eremin & Wallace 01, Hooker & Ottosson 03, Hooker & Yan 95, Jain & Grossmann 01
- Relaxation of global constraints as systems of linear inequalities: Hooker 00, Refalo 00, Williams & Yan 01, Yunes 02
Some relevant concepts and techniques:

- Allowing information exchange among solvers: Rodošek, Wallace & Hajian 99
- Decomposition approaches: Benders 62, Eremin & Wallace 01, Hooker & Ottosson 03, Hooker & Yan 95, Jain & Grossmann 01
- Relaxation of global constraints as systems of linear inequalities: Hooker 00, Refalo 00, Williams & Yan 01, Yunes 02
- Propagation and variable fixing using relaxations: Focacci, Lodi & Milano 99
Some relevant concepts and techniques:

- Allowing information exchange among solvers: Rodošek, Wallace & Hajian 99
- Decomposition approaches: Benders 62, Eremin & Wallace 01, Hooker & Ottosson 03, Hooker & Yan 95, Jain & Grossmann 01
- Relaxation of global constraints as systems of linear inequalities: Hooker 00, Refalo 00, Williams & Yan 01, Yunes 02
- Propagation and variable fixing using relaxations: Focacci, Lodi & Milano 99
- Generation of cutting planes as a form of logical inference: Bockmayr & Kasper 98, Bockmayr & Eisenbrand 00
SIMPL Objectives

- High-level modeling language
  - Concise and easily understandable models
  - Natural specification of integrated models
  - Allow user to reveal problem structure to the solver

- Low-level integration
  - Increased effectiveness when underlying technologies interact at a micro level during the search

- Modularity, flexibility, extensibility, efficiency
  - Make it easy to add new types of constraints, relaxations, solvers and search strategies
SIMPL Objectives

- **High-level** modeling language
  - Concise and easily understandable models
  - Natural specification of integrated models
  - Allow user to reveal problem structure to the solver

- **Low-level** integration
  - Increased effectiveness when underlying technologies interact at a micro level during the search

- Modularity, flexibility, extensibility, efficiency
  - Make it easy to add new types of constraints, relaxations, solvers and search strategies

Aron, Hooker and Yunes
An Integrated Solver for Optimization Problems
SIMPL Objectives

- **High-level** modeling language
  - Concise and easily understandable models
  - Natural specification of integrated models
  - Allow user to reveal problem structure to the solver

- **Low-level** integration
  - Increased effectiveness when underlying technologies interact at a micro level during the search
SIMPL Objectives

- **High-level** modeling language
  - Concise and easily understandable models
  - Natural specification of integrated models
  - Allow user to reveal problem structure to the solver

- **Low-level** integration
  - Increased effectiveness when underlying technologies interact at a micro level during the search

- Modularity, flexibility, extensibility, efficiency
  - Make it easy to add new types of constraints, relaxations, solvers and search strategies
Main Idea Behind SIMPL
Main Idea Behind SIMPL

- **CP** and **MILP** are special cases of a general method, rather than separate methods to be combined.
Main Idea Behind SIMPL

- **CP** and **MILP** are special cases of a general method, rather than separate methods to be combined.

- Common solution strategy: **Search-Infer-Relax**
Main Idea Behind SIMPL

- **CP** and **MILP** are special cases of a general method, rather than separate methods to be combined.

- Common solution strategy: **Search-Infer-Relax**

- **Search** = enumeration of problem restrictions
The Ubiquity of Search, Inference, Relaxation

<table>
<thead>
<tr>
<th>Solution Method</th>
<th>Restriction</th>
<th>Inference</th>
<th>Relaxation</th>
</tr>
</thead>
<tbody>
<tr>
<td>MILP</td>
<td>Branch on fractional vars.</td>
<td>Cutting planes, preprocessing LP relaxation</td>
<td></td>
</tr>
<tr>
<td>CP</td>
<td>Split variable domains</td>
<td>Domain reduction, propagation</td>
<td></td>
</tr>
<tr>
<td>CGO</td>
<td>Split intervals</td>
<td>Interv. propag., lagr. mult. LP or NLP relaxation</td>
<td></td>
</tr>
<tr>
<td>Benders</td>
<td>Subproblem Benders cuts</td>
<td>Master problem</td>
<td></td>
</tr>
<tr>
<td>DPL</td>
<td>Branching Resolution and confl. clauses</td>
<td>Processed confl. clauses</td>
<td></td>
</tr>
<tr>
<td>Tabu Search</td>
<td>Current neighborhood</td>
<td>Tabu list</td>
<td>Same as restriction</td>
</tr>
<tr>
<td>Solution Method</td>
<td>Restriction</td>
<td>Inference</td>
<td>Relaxation</td>
</tr>
<tr>
<td>-----------------</td>
<td>-------------</td>
<td>-----------</td>
<td>------------</td>
</tr>
<tr>
<td>MILP</td>
<td>Branch on fractional vars.</td>
<td>Cutting planes, preprocessing</td>
<td>LP relaxation</td>
</tr>
<tr>
<td>CP</td>
<td>Split variable domains</td>
<td>Domain reduction, propagation</td>
<td>Current domains</td>
</tr>
<tr>
<td>CGO</td>
<td>Split intervals</td>
<td>Interv. propag., lagr. mult.</td>
<td>LP or NLP relaxation</td>
</tr>
<tr>
<td>Benders</td>
<td>Subproblem</td>
<td>Benders cuts (nogoods)</td>
<td>Master problem</td>
</tr>
<tr>
<td>DPL</td>
<td>Branching</td>
<td>Resolution and confl. clauses</td>
<td>Processed confl. clauses</td>
</tr>
<tr>
<td>Tabu Search</td>
<td>Current neighborhood</td>
<td>Tabu list</td>
<td>Same as restriction</td>
</tr>
</tbody>
</table>
Constraint-Based Control

Search:
- Each constraint has a branching module
- This module creates new problem restrictions

Infer:
- Each constraint has a filtering/inference module
- This module creates new constraints to tighten the relaxations

Relax:
- Each constraint has a relaxation module
- This module reformulates the constraint according to different relaxations (LP, MILP, CP, etc.)
Search: constraints **direct the search**

- Each constraint has a **branching module**
- This module creates new problem restrictions
**Constraint-Based Control**

- **Search**: constraints *direct the search*
  - Each constraint has a *branching module*
  - This module creates new problem restrictions

- **Infer**: constraints *drive the inference*
  - Each constraint has a *filtering/inference module*
  - This module creates new constraints to *tighten the relaxations*
Constraint-Based Control

- **Search**: constraints direct the search
  - Each constraint has a branching module
  - This module creates new problem restrictions

- **Infer**: constraints drive the inference
  - Each constraint has a filtering/inference module
  - This module creates new constraints to tighten the relaxations

- **Relax**: constraints create the relaxations
  - Each constraint has a relaxation module
  - This module reformulates the constraint according to different relaxations (LP, MILP, CP, etc.)
Example 1: Production Planning

Manufacture several products at a plant of limited capacity. Products made in one of several production modes (e.g. small scale, medium scale, etc.)

\[ x \] = quantity of a product

Only certain ranges of quantities are possible: gaps in the domain of \( x \)

Net income function \( f(x) \) is semi-continuous piecewise linear

Objective: maximize net income

Example 1: Production Planning

- Manufacture several products at a plant of limited capacity
Example 1: Production Planning

- Manufacture several products at a plant of limited capacity
- Products made in one of several production modes (e.g. small scale, medium scale, etc.)

Net income function $f(x)$ is semi-continuous piecewise linear.

Objective: maximize net income


Aron, Hooker and Yunes
Example 1: Production Planning

- Manufacture several products at a plant of limited capacity
- Products made in one of several production modes (e.g. small scale, medium scale, etc.)
- \( x = \) quantity of a product
Example 1: Production Planning

- Manufacture several products at a plant of limited capacity
- Products made in one of several production modes (e.g. small scale, medium scale, etc.)
- \( x = \) quantity of a product
- Only certain ranges of quantities are possible: gaps in the domain of \( x \)

Net income function \( f(x) \) is semi-continuous piecewise linear

Objective: maximize net income

Example 1: Production Planning

- Manufacture several products at a plant of limited capacity
- Products made in one of several production modes (e.g. small scale, medium scale, etc.)
- \( x = \) quantity of a product
- Only certain ranges of quantities are possible: gaps in the domain of \( x \)
- Net income function \( f(x) \) is semi-continuous piecewise linear
Example 1: Production Planning

- Manufacture several products at a plant of limited capacity
- Products made in one of several production modes (e.g. small scale, medium scale, etc.)
- $x =$ quantity of a product
- Only certain ranges of quantities are possible: gaps in the domain of $x$
- Net income function $f(x)$ is semi-continuous piecewise linear
- Objective: maximize net income
Example 1: Production Planning

- Manufacture several products at a plant of limited capacity
- Products made in one of several production modes (e.g. small scale, medium scale, etc.)
- \( x = \text{quantity of a product} \)
- Only certain ranges of quantities are possible: gaps in the domain of \( x \)
- Net income function \( f(x) \) is semi-continuous piecewise linear
- **Objective:** maximize net income

**Previous work:** Refalo (1999), Ottosson, Thorsteinsson and Hooker (1999, 2002)
Example 1: Production Planning
Shape of Net Income Function $f(x)$
Example 1: Production Planning: MILP
Example 1: Production Planning: MILP

- $x_i =$ quantity of product $i$ (continuous)
Example 1: Production Planning: MILP

- $x_i = \text{quantity of product } i$ (continuous)
- $y_{ik} = \text{whether or not product } i \text{ is made in mode } k$ (binary)
Example 1: Production Planning: MILP

- $x_i =$ quantity of product $i$ (continuous)
- $y_{ik} =$ whether or not product $i$ is made in mode $k$ (binary)
- $\lambda_{ik}, \mu_{ik} =$ weights for mode $k$ (convex combination)
Example 1: Production Planning: MILP

- $x_i =$ quantity of product $i$ (continuous)
- $y_{ik} =$ whether or not product $i$ is made in mode $k$ (binary)
- $\lambda_{ik}, \mu_{ik} =$ weights for mode $k$ (convex combination)

MILP Model
Example 1: Production Planning: MILP

- $x_i =$ quantity of product $i$ (continuous)
- $y_{ik} =$ whether or not product $i$ is made in mode $k$ (binary)
- $\lambda_{ik}, \mu_{ik} =$ weights for mode $k$ (convex combination)

**MILP Model**

$$\max \sum_{ik} \lambda_{ik} c_{ik} + \mu_{ik} d_{ik}$$

$$\sum_{k} \lambda_{ik} L_{ik} + \mu_{ik} U_{ik}, \forall i$$

$$\sum_{k} y_{ik} = 1, \forall i$$

$$0 \leq \lambda_{ik} \leq y_{ik}, \forall i, k$$

$$0 \leq \mu_{ik} \leq y_{ik}, \forall i, k$$

$$f(x_i)$$

**Graph:**

- $C_{ik}$
- $d_{ik}$
- $L_{ik}$
- $U_{ik}$
- $x_i$
Example 1: Production Planning: MILP

- $x_i =$ quantity of product $i$ (continuous)
- $y_{ik} =$ whether or not product $i$ is made in mode $k$ (binary)
- $\lambda_{ik}, \mu_{ik} =$ weights for mode $k$ (convex combination)

**MILP Model**

$$\max \sum_{ik} \lambda_{ik} c_{ik} + \mu_{ik} d_{ik}$$

$$\sum_i x_i \leq C$$

$$f(x_i)$$

$f(x_i)$ is a function of $x_i$. The graph shows a linear function with a slope and an intercept. The function is defined for $x_i$ within the range $L_{ik}$ to $U_{ik}$. The points $C_{ik}$ and $d_{ik}$ are shown on the graph, indicating specific values for the function at these points. The graph illustrates the relationship between $x_i$ and $f(x_i)$. The equation for the function is:

$$f(x_i) = \lambda_{ik} c_{ik} + \mu_{ik} d_{ik}$$
Example 1: Production Planning: MILP

- $x_i =$ quantity of product $i$ (continuous)
- $y_{ik} =$ whether or not product $i$ is made in mode $k$ (binary)
- $\lambda_{ik}, \mu_{ik} =$ weights for mode $k$ (convex combination)

MILP Model

$$\max \sum_{ik} \lambda_{ik}c_{ik} + \mu_{ik}d_{ik}$$

$$\sum_i x_i \leq C$$

$$x_i = \sum_k \lambda_{ik}L_{ik} + \mu_{ik}U_{ik}, \forall i$$

$$f(x_i)$$

$$d_{ik}$$

$$c_{ik}$$

$$L_{ik}$$

$$U_{ik}$$

$$x_i$$
Example 1: Production Planning: MILP

- $x_i =$ quantity of product $i$ (continuous)
- $y_{ik} =$ whether or not product $i$ is made in mode $k$ (binary)
- $\lambda_{ik}, \mu_{ik} =$ weights for mode $k$ (convex combination)

**MILP Model**

$$\max \sum_{ik} \lambda_{ik} c_{ik} + \mu_{ik} d_{ik}$$

$$\sum_{i} x_i \leq C$$

$$x_i = \sum_{k} \lambda_{ik} L_{ik} + \mu_{ik} U_{ik}, \forall i$$

$$\sum_{k} \lambda_{ik} + \mu_{ik} = 1, \forall i$$
Example 1: Production Planning: MILP

- \( x_i \) = quantity of product \( i \) (continuous)
- \( y_{ik} \) = whether or not product \( i \) is made in mode \( k \) (binary)
- \( \lambda_{ik}, \mu_{ik} \) = weights for mode \( k \) (convex combination)

MILP Model

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

![Graph showing the MILP model with a feasible region.]
Example 1: Production Planning: MILP

- \( x_i = \text{quantity of product } i \) (continuous)
- \( y_{ik} = \text{whether or not product } i \text{ is made in mode } k \) (binary)
- \( \lambda_{ik}, \mu_{ik} = \text{weights for mode } k \) (convex combination)

**MILP Model**

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

- $x_i =$ quantity of product $i$ (continuous)
- $y_{ik} =$ whether or not product $i$ is made in mode $k$ (binary)
- $\lambda_{ik}, \mu_{ik} =$ weights for mode $k$ (convex combination)

**MILP Model**

\[
\begin{align*}
\text{max} & \quad \sum_{ik} \lambda_{ik} c_{ik} + \mu_{ik} d_{ik} \\
\sum_i x_i & \leq C \\
x_i & = \sum_k \lambda_{ik} L_{ik} + \mu_{ik} U_{ik}, \; \forall \; i \\
\sum_k \lambda_{ik} + \mu_{ik} & = 1, \; \forall \; i \\
0 & \leq \lambda_{ik} \leq y_{ik}, \; \forall \; i, k \\
0 & \leq \mu_{ik} \leq y_{ik}, \; \forall \; i, k \\
\sum_k y_{ik} & = 1, \; \forall \; i
\end{align*}
\]
Example 1: Production Planning: Integrated
Example 1: Production Planning: Integrated

- $x_i = \text{quantity of product } i \text{ (continuous)}$
Example 1: Production Planning: Integrated

- $x_i = \text{quantity of product } i \ (\text{continuous})$
- $u_i = \text{net income from product } i \ (\text{continuous})$
Example 1: Production Planning: Integrated

- $x_i = \text{quantity of product } i$ (continuous)
- $u_i = \text{net income from product } i$ (continuous)

Integrated Model
Example 1: Production Planning: Integrated

- $x_i =$ quantity of product $i$ (continuous)
- $u_i =$ net income from product $i$ (continuous)

Integrated Model

$$\max \sum_i u_i$$
Example 1: Production Planning: Integrated

- $x_i =$ quantity of product $i$ (continuous)
- $u_i =$ net income from product $i$ (continuous)

**Integrated Model**

$$\text{max } \sum_i u_i$$
$$\sum_i x_i \leq C$$
Example 1: Production Planning: Integrated

- $x_i =$ quantity of product $i$ (continuous)
- $u_i =$ net income from product $i$ (continuous)

**Integrated Model**

$$\max \sum_i u_i$$

$$\sum_i x_i \leq C$$

piecewise$(x_i, u_i, L_i, U_i, c_i, d_i), \ \forall \ i$
Example 1: Production Planning: SIMPL Model

**SIMPL Model**

**OBJECTIVE**

max sum \( i \) of \( u_i \)

**CONSTRAINTS**

capacity means \( \sum_i x_i \leq C \)
relaxation = \{ lp, cp \}

income means \( \text{piecewise}(x_i, u_i, L_i, U_i, c_i, d_i) \) forall \( i \)
relaxation = \{ lp, cp \}

**SEARCH**

type = \{ bb:bestdive \}
branching = \{ income:most \}
Example 1: Production Planning: SIMPL Model

- $x_i = \text{quantity of product } i$ (continuous)
Example 1: Production Planning: SIMPL Model

- $x_i = \text{quantity of product } i$ (continuous)
- $u_i = \text{net income from product } i$ (continuous)
**Example 1: Production Planning: SIMPL Model**

- $x_i = \text{quantity of product } i$ (continuous)
- $u_i = \text{net income from product } i$ (continuous)

SIMPL Model
Example 1: Production Planning: SIMPL Model

- \( x_i \) = quantity of product \( i \) (continuous)
- \( u_i \) = net income from product \( i \) (continuous)
Example 1: Production Planning: SIMPL Model

- \( x_i \) = quantity of product \( i \) (continuous)
- \( u_i \) = net income from product \( i \) (continuous)

SIMPL Model

**OBJECTIVE**

\[
\max \sum_i u[i]
\]
Example 1: Production Planning: SIMPL Model

- \( x_i = \) quantity of product \( i \) (continuous)
- \( u_i = \) net income from product \( i \) (continuous)

SIMPL Model

**OBJECTIVE**

\[
\text{max } \sum_i u[i]
\]

**CONSTRAINTS**
Example 1: Production Planning: SIMPL Model

- $x_i =$ quantity of product $i$ (continuous)
- $u_i =$ net income from product $i$ (continuous)

SIMPL Model

**OBJECTIVE**

max sum $i$ of $u[i]$

**CONSTRAINTS**

capacity means 

$\sum i$ of $x[i] \leq C$

relaxation = { lp, cp }
Example 1: Production Planning: SIMPL Model

- \( x_i \) = quantity of product \( i \) (continuous)
- \( u_i \) = net income from product \( i \) (continuous)

SIMPL Model

**OBJECTIVE**

\[
\text{max} \sum \text{ of } u[i]
\]

**CONSTRAINTS**

capacity means { 
\[
\sum \text{ of } x[i] \leq C \\
\text{relaxation} = \{ \text{lp, cp} \}
\]

income means {
\[
\text{piecewise}(x[i], u[i], L[i], U[i], c[i], d[i]) \ \forall i \\
\text{relaxation} = \{ \text{lp, cp} \}
\]
SIMPL Model

**OBJECTIVE**

\[
\text{max } \sum i \text{ of } u[i]
\]

**CONSTRAINTS**

- **capacity** means \{ \\
  \sum i \text{ of } x[i] \leq C \\
  \text{relaxation} = \{ \text{lp, cp} \} \\
\}

- **income** means \{
  \text{piecewise}(x[i], u[i], L[i], U[i], c[i], d[i]) \text{ forall } i \\
  \text{relaxation} = \{ \text{lp, cp} \} \\
\}

**SEARCH**
Example 1: Production Planning: SIMPL Model

- $x_i = \text{quantity of product } i$ (continuous)
- $u_i = \text{net income from product } i$ (continuous)

SIMPL Model

**OBJECTIVE**
max sum $i$ of $u[i]$

**CONSTRAINTS**
capacity means {
  sum $i$ of $x[i] \leq C$
  relaxation = { lp, cp } }
income means {
  piecewise($x[i], u[i], L[i], U[i], c[i], d[i]$) forall $i$
  relaxation = { lp, cp } }

**SEARCH**
type = { bb:bestdive }
**Example 1: Production Planning: SIMPL Model**

- \( x_i \) = quantity of product \( i \) (continuous)
- \( u_i \) = net income from product \( i \) (continuous)

---

**SIMPL Model**

**OBJECTIVE**

\[
\text{max } \sum_i u[i]
\]

**CONSTRAINTS**

capacity means { 
  \[
  \sum_i x[i] \leq C
  \]
  relaxation = \{ lp, cp \}
}

income means {
  piecewise(x[i],u[i],L[i],U[i],c[i],d[i]) forall i
  relaxation = \{ lp, cp \}
}

**SEARCH**

- type = \{ bb:bestdive \}
- branching = \{ income:most \}
Example 1: Production Planning
Relaxation and Branching for piecewise
Example 1: Production Planning
Relaxation and Branching for piecewise
Example 1: Production Planning
Relaxation and Branching for piecewise

\[ f(x) \]

\( x \) value OK, \( y \) value OK: no problem
Example 1: Production Planning
Relaxation and Branching for piecewise

\[ f(x) \]

\[ x \text{ value not OK: split domain} \]
Example 1: Production Planning
Relaxation and Branching for piecewise

\[ f(x) \]

child 1

child 2
Example 1: Production Planning
Relaxation and Branching for piecewise

$f(x)$

$x$ OK, $y$ not OK: 3-way branch on $x$
Example 1: Production Planning
Relaxation and Branching for piecewise

\[ f(x) \]

\[ x \]

child 1

child 2

child 3
Example 1: Production Planning
Computational Results: Number of Search Nodes

<table>
<thead>
<tr>
<th>Number of Products</th>
<th>Nodes</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>10</td>
</tr>
<tr>
<td>10</td>
<td>20</td>
</tr>
<tr>
<td>100</td>
<td>30</td>
</tr>
<tr>
<td>1000</td>
<td>40</td>
</tr>
<tr>
<td>10000</td>
<td>50</td>
</tr>
<tr>
<td>100000</td>
<td></td>
</tr>
</tbody>
</table>

MILP

Integrated

Aron, Hooker and Yunes
Example 1: Production Planning
Computational Results: Number of Search Nodes

<table>
<thead>
<tr>
<th>Number of Products</th>
<th>Number of Nodes</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>10 MILP</td>
</tr>
<tr>
<td>10</td>
<td>20 Integrated</td>
</tr>
<tr>
<td>100</td>
<td>30 MILP</td>
</tr>
<tr>
<td>1000</td>
<td>40 Integrated</td>
</tr>
<tr>
<td>10000</td>
<td>50 MILP</td>
</tr>
<tr>
<td>100000</td>
<td>100 Integrated</td>
</tr>
</tbody>
</table>

Aron, Hooker and Yunes
An Integrated Solver for Optimization Problems
Example 1: Production Planning

Computational Results: CPU Time (s)
Example 1: Production Planning
Computational Results: CPU Time (s)

Aron, Hooker and Yunes
An Integrated Solver for Optimization Problems
Example 2: Product Configuration

A product (e.g. computer) is made up of several components (e.g. memory, CPU, etc.). Components come in different types:

\[ \text{Type } k \text{ of component } i \text{ uses/produces } a_{ijk} \text{ units of resource } j \]

\[ c_j = \text{unit cost of resource } j \]

Lower and upper bounds on resource usage/production.

Objective: minimize total cost.

Previous work: Thorsteinsson and Ottosson (2001)
Example 2: Product Configuration

- A product (e.g. computer) is made up of several components (e.g. memory, cpu, etc.)
Example 2: Product Configuration

- A **product** (e.g. computer) is made up of several **components** (e.g. memory, cpu, etc.)
- Components come in different **types**
Example 2: Product Configuration

- A **product** (e.g. computer) is made up of several **components** (e.g. memory, cpu, etc.)
- Components come in different **types**
- Type $k$ of component $i$ uses/produces $a_{ijk}$ units of **resource** $j$
Example 2: Product Configuration

- A **product** (e.g. computer) is made up of several **components** (e.g. memory, cpu, etc.)
- Components come in different **types**
- Type $k$ of component $i$ uses/produces $a_{ijk}$ units of **resource** $j$
- $c_j =$ unit cost of resource $j$
Example 2: Product Configuration

- A **product** (e.g. computer) is made up of several **components** (e.g. memory, CPU, etc.)
- Components come in different **types**
- Type $k$ of component $i$ uses/produces $a_{ijk}$ units of **resource** $j$
- $c_j = \text{unit cost of resource } j$
- Lower and upper bounds on resource usage/production
A product (e.g. computer) is made up of several components (e.g. memory, cpu, etc.)
Components come in different types
Type $k$ of component $i$ uses/produces $a_{ijk}$ units of resource $j$
$c_j = \text{unit cost of resource } j$
Lower and upper bounds on resource usage/production
Objective: minimize total cost
Example 2: Product Configuration

- A **product** (e.g. computer) is made up of several **components** (e.g. memory, cpu, etc.)
- Components come in different **types**
- Type $k$ of component $i$ uses/produces $a_{ijk}$ units of **resource** $j$
- $c_j = $ unit cost of resource $j$
- Lower and upper bounds on resource usage/production
- **Objective:** minimize total cost

**Previous work:** Thorsteinsson and Ottosson (2001)
Example 2: Product Configuration: MILP

- $x_{ik}$: whether or not type $k$ is chosen for component $i$ (binary)
- $q_{ik}$: # units of component $i$ of type $k$ to install (integer)
- $r_j$: amount of resource $j$ produced (continuous)

MILP Model

\[
\min \sum_j c_j r_j = \sum_{ik} a_{ijk} q_{ik}, \quad \forall j
\]

\[
L_j \leq r_j \leq U_j, \quad \forall j
\]

\[
q_{ik} \leq M_i x_{ik}, \quad \forall i, k
\]

\[
\sum_k x_{ik} = 1, \quad \forall i
\]
Example 2: Product Configuration: MILP

- \( x_{ik} \) = whether or not type \( k \) is chosen for component \( i \) (binary)

- \( q_{ik} \) = # units of component \( i \) of type \( k \) to install (integer)

- \( r_j \) = amount of resource \( j \) produced (continuous)

MILP Model

\[
\min \sum_j c_j r_j \\
L_j \leq r_j \leq U_j, \quad \forall j \\
q_{ik} \leq M_i x_{ik}, \quad \forall i, k \\
\sum_k x_{ik} = 1, \quad \forall i
\]
Example 2: Product Configuration: MILP

- $x_{ik} = \text{whether or not type } k \text{ is chosen for component } i$ (binary)
- $q_{ik} = \text{# units of component } i \text{ of type } k \text{ to install}$ (integer)
Example 2: Product Configuration: MILP

- $x_{ik} = \text{whether or not type } k \text{ is chosen for component } i \text{ (binary)}$
- $q_{ik} = \text{# units of component } i \text{ of type } k \text{ to install (integer)}$
- $r_j = \text{amount of resource } j \text{ produced (continuous)}$
Example 2: Product Configuration: MILP

- $x_{ik} =$ whether or not type $k$ is chosen for component $i$ (binary)
- $q_{ik} =$ # units of component $i$ of type $k$ to install (integer)
- $r_j =$ amount of resource $j$ produced (continuous)

MILP Model
Example 2: Product Configuration: MILP

- $x_{ik} = \text{whether or not type } k \text{ is chosen for component } i \text{ (binary)}$
- $q_{ik} = \# \text{ units of component } i \text{ of type } k \text{ to install (integer)}$
- $r_j = \text{amount of resource } j \text{ produced (continuous)}$

MILP Model

$$\min \sum_j c_j r_j$$
Example 2: Product Configuration: MILP

- $x_{ik} =$ whether or not type $k$ is chosen for component $i$ (binary)
- $q_{ik} =$ # units of component $i$ of type $k$ to install (integer)
- $r_j =$ amount of resource $j$ produced (continuous)

**MILP Model**

\[
\begin{align*}
\text{min} & \quad \sum_j c_j r_j \\
r_j &= \sum_{ik} a_{ijk} q_{ik}, \quad \forall \ j
\end{align*}
\]
Example 2: Product Configuration: MILP

- $x_{ik}$ = whether or not type $k$ is chosen for component $i$ (binary)
- $q_{ik}$ = # units of component $i$ of type $k$ to install (integer)
- $r_j$ = amount of resource $j$ produced (continuous)

**MILP Model**

$$
\begin{align*}
\text{min} & \sum_j c_j r_j \\
r_j &= \sum_{ik} a_{ijk} q_{ik}, \forall j \\
L_j &\leq r_j \leq U_j, \forall j
\end{align*}
$$
Example 2: Product Configuration: MILP

- $x_{ik} =$ whether or not type $k$ is chosen for component $i$ (binary)
- $q_{ik} =$ # units of component $i$ of type $k$ to install (integer)
- $r_j =$ amount of resource $j$ produced (continuous)

MILP Model

$$\min \sum_j c_j r_j$$
$$r_j = \sum_{ik} a_{ijk} q_{ik}, \forall j$$
$$L_j \leq r_j \leq U_j, \forall j$$
$$q_{ik} \leq M_i x_{ik}, \forall i, k$$
Example 2: Product Configuration: MILP

- $x_{ik} = \text{whether or not type } k \text{ is chosen for component } i$ (binary)
- $q_{ik} = \# \text{ units of component } i \text{ of type } k \text{ to install}$ (integer)
- $r_j = \text{amount of resource } j \text{ produced}$ (continuous)

**MILP Model**

$$\min \sum_j c_j r_j$$

$$r_j = \sum_{ik} a_{ijk} q_{ik}, \ \forall \ j$$

$$L_j \leq r_j \leq U_j, \ \forall \ j$$

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

$$\sum_k x_{ik} = 1, \ \forall \ i$$
Example 2: Product Configuration: Integrated
Example 2: Product Configuration: Integrated

- $q_i = \#$ units of component $i$ to install (integer)
Example 2: Product Configuration: Integrated

- $q_i = \#$ units of component $i$ to install (integer)
- $t_i = $ type chosen for component $i$ (discrete)
Example 2: Product Configuration: Integrated

- $q_i = \#$ units of component $i$ to install (integer)
- $t_i = $ type chosen for component $i$ (discrete)
- $r_j = $ amount of resource $j$ produced (continuous)
Example 2: Product Configuration: Integrated

- $q_i = \#$ units of component $i$ to install (integer)
- $t_i = \text{type chosen for component } i$ (discrete)
- $r_j = \text{amount of resource } j \text{ produced}$ (continuous)

Integrated Model
Example 2: Product Configuration: Integrated

- $q_i = \#$ units of component $i$ to install (integer)
- $t_i = \text{type chosen for component } i$ (discrete)
- $r_j = \text{amount of resource } j \text{ produced}$ (continuous)

Integrated Model

$$\min \sum_j c_j r_j$$
Example 2: Product Configuration: Integrated

- $q_i = \#$ units of component $i$ to install (integer)
- $t_i = \text{type chosen for component } i$ (discrete)
- $r_j = \text{amount of resource } j \text{ produced }$ (continuous)

Integrated Model

$$\min \sum_j c_j r_j$$
$$r_j = \sum_i a_{ij} t_i q_i, \ \forall \ j$$
Example 2: Product Configuration: Integrated

- $q_i = \#$ units of component $i$ to install \textit{(integer)}
- $t_i = \text{type chosen for component } i \text{ (discrete)}$
- $r_j = \text{amount of resource } j \text{ produced (continuous)}$

**Integrated Model**

$$
\begin{align*}
\min & \sum_j c_j r_j \\
\sum_i a_{ij} t_i q_i, & \forall j \\
L_j & \leq r_j \leq U_j, \forall j
\end{align*}
$$
Example 2: Product Configuration: Integrated

- $q_i = \# \text{ units of component } i \text{ to install (integer)}$
- $t_i = \text{ type chosen for component } i \text{ (discrete)}$
- $r_j = \text{ amount of resource } j \text{ produced (continuous)}$

**Integrated Model**

$$
\begin{align*}
\min & \sum_j c_j r_j \\
r_j &= \sum_i a_{ij} t_i q_i, \quad \forall j \\
L_j &\leq r_j \leq U_j, \quad \forall j
\end{align*}
$$

$z_{ij} = a_{ij} t_i q_i$
Example 2: Product Configuration: Integrated

- \( q_i = \# \) units of component \( i \) to install (integer)
- \( t_i = \) type chosen for component \( i \) (discrete)
- \( r_j = \) amount of resource \( j \) produced (continuous)

\[
\text{min} \sum_j c_j r_j \\
\sum_i a_{ij} t_i q_i \leq r_j \leq U_j, \forall j
\]

\( z_{ij} = a_{ij} t_i q_i \) and element \((t_i, (a_{ij1} q_i, \ldots, a_{ijn} q_i), z_{ij})\)
Example 2: Product Configuration: Integrated

- $q_i = \#$ units of component $i$ to install (integer)
- $t_i =$ type chosen for component $i$ (discrete)
- $r_j =$ amount of resource $j$ produced (continuous)

**Integrated Model**

\[
\min \sum_{j} c_j r_j \\
\text{s.t.} \quad r_j = \sum_{i} a_{ijt_i} q_i, \quad \forall \ j \\
L_j \leq r_j \leq U_j, \quad \forall \ j
\]

$z_{ij} = a_{ijt_i} q_i$ and element $(t_i, (a_{ij1} q_i, \ldots, a_{ijn} q_i), z_{ij})$

\[\bigvee_{k \in D_{t_i}} (z_{ij} = a_{ijk} q_i)\]

converted to

equivalent to

Aron, Hooker and Yunes: An Integrated Solver for Optimization Problems
Example 2: Product Configuration: Integrated

- \( q_i = \# \) units of component \( i \) to install \((\text{integer})\)
- \( t_i = \) type chosen for component \( i \) \((\text{discrete})\)
- \( r_j = \) amount of resource \( j \) produced \((\text{continuous})\)

### Integrated Model

\[
\begin{align*}
\min & \sum_j c_j r_j \\
\text{s.t.} & \quad r_j = \sum_i a_{ijt_i} q_i, \quad \forall j \\
& \quad L_j \leq r_j \leq U_j, \quad \forall j \\
\end{align*}
\]

\( z_{ij} = a_{ijt_i} q_i \) and element \((t_i, (a_{ij1} q_i, \ldots, a_{ijn} q_i), z_{ij})\)

\( \bigvee_{k \in D_{ti}} (z_{ij} = a_{ijk} q_i) \rightarrow \text{automatic and dynamic convex hull relax.} \)
Example 2: Product Configuration: SIMPL Model
Example 2: Product Configuration: SIMPL Model

**OBJECTIVE**  \( \min \sum_{j} c[j] \cdot r[j] \)

**CONSTRAINTS**

- \( r[j] = \sum_{i} a[i][j][t[i]] \cdot q[i] \) for all \( j \)
- \( q[1] \geq 1 \Rightarrow q[2] = 0 \)
- \( t[1] = 1 \Rightarrow t[2] \in \{1, 2\} \)
- \( t[3] = 1 \Rightarrow (t[4] \in \{1, 3\} \text{ and } t[5] \in \{1, 3, 4, 6\} \text{ and } t[6] = 3) \)

**SEARCH**

- **type** = \{ bb:bestdive \}
- **branching** = \{ quant \}, t:most, q:least:triple, types:most
- **inference** = \{ q:redcost \}
Example 2: Product Configuration: SIMPL Model

OBJECTIVE  \( \min \ \text{sum } j \text{ of } c[j]*r[j] \)

CONSTRAINTS
Example 2: Product Configuration: SIMPL Model

OBJECTIVE  \( \min \sum_j c[j] \times r[j] \)

CONSTRAINTS

resource means { 
  \( r[j] = \sum_i a[i][j][t[i]] \times q[i] \) forall j

relaxation = \{ lp, cp \} }

inference = \{ q:redcost \}

Aron, Hooker and Yunes
An Integrated Solver for Optimization Problems 25
Example 2: Product Configuration: SIMPL Model

**OBJECTIVE**  \( \min \sum_j c[j] \cdot r[j] \)

**CONSTRAINTS**

resource means {
  \( r[j] = \sum_i a[i][j][t[i]] \cdot q[i] \) \text{ forall } j
  relaxation = \{ lp, cp \}
}

**SEARCH**

search type = { bb:bestdive }
broadcast = {
  quant, t:most, q:least:triple, types:most
}
inference = { q:redcost }
Example 2: Product Configuration: SIMPL Model

**OBJECTIVE**  \[ \text{min} \sum \text{j of } c[j]*r[j] \]

**CONSTRAINTS**
- resource means:
  \[ r[j] = \sum i \text{ of } a[i][j][t[i]]*q[i] \text{ for all } j \]
- relaxation = \{ lp, cp \}

**SEARCH**
- type = \{ bb:bestdive \}
Example 2: Product Configuration: SIMPL Model

**OBJECTIVE**  \( \text{min sum } j \text{ of } c[j] \cdot r[j] \)

**CONSTRAINTS**
- **resource means** \{ 
  - \( r[j] = \sum i \text{ of } a[i][j][t[i]] \cdot q[i] \text{ forall } j \)
- **relaxation** = \{ lp, cp \} 

**SEARCH**
- **type** = \{ bb:bestdive \}
- **branching** = \{ t:most, q:least:triple \}
Example 2: Product Configuration: SIMPL Model

OBJECTIVE  \[ \min \sum_j c[j]r[j] \]

CONSTRAINTS

resource  \[ \text{means } \{ \]
\[ r[j] = \sum_i a[i][j][t[i]]q[i] \text{ forall } j \]
\[ \text{relaxation } = \{ \text{lp, cp} \} \]

SEARCH

\[ \text{type } = \{ \text{bb:bstdive} \} \]
\[ \text{branching } = \{ \text{t:most, q:least:triple} \} \]
\[ \text{inference } = \{ \text{q:redcost} \} \]
Example 2: Product Configuration: SIMPL Model

**OBJECTIVE**
\[ \min \sum_j c[j] \cdot r[j] \]

**CONSTRAINTS**
- **resource** means
  \[ r[j] = \sum_i a[i][j][t[i]] \cdot q[i] \quad \text{forall} \ j \]
  relaxation = \{ lp, cp \}
- **quant** means
  \[ q[1] \geq 1 \implies q[2] = 0 \]
  relaxation = \{ lp, cp \}

**SEARCH**
- **type** = \{ bb:bestdive \}
- **branching** = \{ t:most, q:least:triple \}
- **inference** = \{ q:redcost \}
Example 2: Product Configuration: SIMPL Model

**OBJECTIVE**
\[ \min \sum_j c[j] \times r[j] \]

**CONSTRAINTS**

- **resource** means
  \[ r[j] = \sum_i a[i][j][t[i]] \times q[i] \quad \text{forall} \quad j \]
  relaxation = \{ lp, cp \}

- **quant** means
  \[ q[1] \geq 1 \Rightarrow q[2] = 0 \]
  relaxation = \{ lp, cp \}

- **types** means
  \[ t[1] = 1 \Rightarrow t[2] \text{ in } \{1, 2\} \]
  \[ t[3] = 1 \Rightarrow (t[4] \text{ in } \{1, 3\} \text{ and } t[5] \text{ in } \{1, 3, 4, 6\} \]
  \[ \text{and } t[6] = 3 \]
  relaxation = \{ lp, cp \}

**SEARCH**

- **type** = \{ bb:bestdive \}
- **branching** = \{ t:most, q:least:triple \}
- **inference** = \{ q:redcost \}
Example 2: Product Configuration: SIMPL Model

**OBJECTIVE** \( \min \sum_j c[j] \cdot r[j] \)

**CONSTRAINTS**

- **resource** means
  \[
  r[j] = \sum_i a[i][j][t[i]] \cdot q[i] \quad \text{forall } j
  \]
  relaxation = \{ lp, cp \}

- **quant** means
  \[
  q[1] \geq 1 \Rightarrow q[2] = 0
  \]
  relaxation = \{ lp, cp \}

- **types** means
  \[
  t[1] = 1 \Rightarrow t[2] \in \{1, 2\}
  t[3] = 1 \Rightarrow (t[4] \in \{1, 3\} \text{ and } t[5] \in \{1, 3, 4, 6\}
  \text{and } t[6] = 3)
  \]
  relaxation = \{ lp, cp \}

**SEARCH**

- **type** = \{ bb:bestdive \}
- **branching** = \{ quant, t:most, q:least:triple \}
- **inference** = \{ q:redcost \}

Aron, Hooker and Yunes
An Integrated Solver for Optimization Problems
Example 2: Product Configuration: SIMPL Model

**OBJECTIVE** \[ \min \sum_j c[j] \times r[j] \]

**CONSTRAINTS**

- **resource** \textit{means} \{ 
  \[ r[j] = \sum_i a[i][j][t[i]] \times q[i] \text{ forall } j \]
  \ \textit{relaxation} = \{ \text{lp, cp} \} 
\}

- **quant** \textit{means} \{ 
  \[ q[1] \geq 1 \Rightarrow q[2] = 0 \]
  \ \textit{relaxation} = \{ \text{lp, cp} \} 
\}

- **types** \textit{means} \{ 
  \[ t[1] = 1 \Rightarrow t[2] \text{ in } \{1, 2\} \]
  \[ t[3] = 1 \Rightarrow (t[4] \text{ in } \{1, 3\} \text{ and } t[5] \text{ in } \{1, 3, 4, 6\} \]
  \[ \text{ and } t[6] = 3) \]
  \ \textit{relaxation} = \{ \text{lp, cp} \} 
\}

**SEARCH**

- **type** = \{ \text{bb:bestdive} \}
- **branching** = \{ \text{quant, t:most, q:least:triple, types:most} \}
- **inference** = \{ \text{q:redcost} \}
Example 2: Product Configuration

Computational Results: Number of Search Nodes

Problem Instances (Thorsteinsson and Ottosson, 2001)

<table>
<thead>
<tr>
<th>MILP</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
</tr>
<tr>
<td>3</td>
</tr>
<tr>
<td>5</td>
</tr>
<tr>
<td>7</td>
</tr>
<tr>
<td>9</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Integrated</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
</tr>
<tr>
<td>4</td>
</tr>
<tr>
<td>6</td>
</tr>
<tr>
<td>8</td>
</tr>
<tr>
<td>10</td>
</tr>
</tbody>
</table>
Example 2: Product Configuration
Computational Results: Number of Search Nodes

<table>
<thead>
<tr>
<th>Problem Instances (Thorsteinsson and Ottosson, 2001)</th>
<th>Nodes</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>700</td>
</tr>
<tr>
<td>6</td>
<td></td>
</tr>
<tr>
<td>7</td>
<td></td>
</tr>
<tr>
<td>8</td>
<td></td>
</tr>
<tr>
<td>9</td>
<td></td>
</tr>
<tr>
<td>10</td>
<td></td>
</tr>
</tbody>
</table>

- MILP
- Integrated

Aron, Hooker and Yunes
An Integrated Solver for Optimization Problems
Example 2: Product Configuration

Computational Results: CPU Time (s)

<table>
<thead>
<tr>
<th>Problem Instances (Thorsteinsson and Ottosson, 2001)</th>
<th>MILP</th>
<th>Integrated</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
<td>3</td>
</tr>
<tr>
<td>5</td>
<td>3</td>
<td>5</td>
</tr>
<tr>
<td>10</td>
<td>5</td>
<td>7</td>
</tr>
<tr>
<td>15</td>
<td>7</td>
<td>9</td>
</tr>
<tr>
<td>20</td>
<td>9</td>
<td></td>
</tr>
</tbody>
</table>

Aron, Hooker and Yunes

An Integrated Solver for Optimization Problems
Example 2: Product Configuration

Computational Results: CPU Time (s)

Aron, Hooker and Yunes

An Integrated Solver for Optimization Problems
Example 3: Job Scheduling on Parallel Machines

Given $n$ jobs and $m$ parallel (disjunctive) machines $c_{ij}$ and $p_{ij} =$ processing cost and time of job $i$ on machine $j$. Job $i$ has release date $r_i$ and due date $d_i$.

Objective: schedule all jobs and minimize total cost.

Jain and Grossmann (2001) ▶ Hybrid MILP/CP Benders decomposition approach ▶ Required development of special purpose code ▶ Up to 1000 times faster than commercial solvers

In SIMPL we can get the same results with very little effort.

Aron, Hooker and Yunes
An Integrated Solver for Optimization Problems
Example 3: Job Scheduling on Parallel Machines

- Given \( n \) jobs and \( m \) parallel (disjunctive) machines
Example 3: Job Scheduling on Parallel Machines

- Given $n$ jobs and $m$ parallel (disjunctive) machines
- $c_{ij}$ and $p_{ij}$ = processing cost and time of job $i$ on machine $j$
Example 3: Job Scheduling on Parallel Machines

- Given $n$ jobs and $m$ parallel (disjunctive) machines
- $c_{ij}$ and $p_{ij}$ = processing cost and time of job $i$ on machine $j$
- Job $i$ has release date $r_i$ and due date $d_i$
Example 3: Job Scheduling on Parallel Machines

- Given \( n \) jobs and \( m \) parallel (disjunctive) machines
- \( c_{ij} \) and \( p_{ij} \) = processing cost and time of job \( i \) on machine \( j \)
- Job \( i \) has release date \( r_i \) and due date \( d_i \)
- **Objective:** schedule all jobs and minimize total cost
Example 3: Job Scheduling on Parallel Machines

- Given $n$ jobs and $m$ parallel (disjunctive) machines
- $c_{ij}$ and $p_{ij} =$ processing cost and time of job $i$ on machine $j$
- Job $i$ has release date $r_i$ and due date $d_i$
- **Objective:** schedule all jobs and minimize total cost

- Jain and Grossmann (2001)
Example 3: Job Scheduling on Parallel Machines

- Given $n$ jobs and $m$ parallel (disjunctive) machines
- $c_{ij}$ and $p_{ij} = \text{processing cost and time of job } i \text{ on machine } j$
- Job $i$ has release date $r_i$ and due date $d_i$
- **Objective:** schedule all jobs and minimize total cost

**Jain and Grossmann (2001)**
- Hybrid MILP/CP Benders decomposition approach
- Required development of special purpose code
- Up to 1000 times faster than commercial solvers
Example 3: Job Scheduling on Parallel Machines

- Given $n$ jobs and $m$ parallel (disjunctive) machines
- $c_{ij}$ and $p_{ij}$ = processing cost and time of job $i$ on machine $j$
- Job $i$ has release date $r_i$ and due date $d_i$
- **Objective:** schedule all jobs and minimize total cost

Jain and Grossmann (2001)
- Hybrid MILP/CP Benders decomposition approach
- Required development of special purpose code
- Up to 1000 times faster than commercial solvers

In SIMPL we can get the same results with very little effort
Example 3: Job Scheduling: MILP Model

\[
x_{ij} = \text{whether or not job } j \text{ is assigned to machine } i \quad (\text{binary})
\]

\[
y_{jk} = \text{whether or not } j \text{ precedes } k \text{ on some machine} \quad (\text{binary})
\]

\[
t_j = \text{start time of job } j \quad (\text{continuous})
\]

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

\[
r_j \leq t_j \leq d_j - \sum_{i} p_{ij} x_{ij}, \quad \forall j
\]

\[
\sum_{i} x_{ij} = 1, \quad \forall j
\]

\[
y_{jk} + y_{kj} \leq 1, \quad \forall k > j
\]

\[
y_{jk} + y_{kj} \geq x_{ij} + x_{ik} - 1, \quad \forall k > j, i
\]

\[
y_{jk} + y_{kj} + x_{ij} + x_{i'k} \leq 2, \quad \forall k > j, i'
\]

\[
t_k \geq t_j + \sum_{i} p_{ij} x_{ij} - M(1 - y_{jk}), \quad \forall k \neq j
\]

\[
\sum_{j} p_{ij} x_{ij} \leq \max_{j} \{d_j\} - \min_{j} \{r_j\}, \quad \forall i
\]
Example 3: Job Scheduling: MILP Model

- \( x_{ij} \) = whether or not job \( j \) is assigned to machine \( i \) (binary)
Example 3: Job Scheduling: MILP Model

- $x_{ij} = \text{whether or not job } j \text{ is assigned to machine } i$ (binary)
- $y_{jk} = \text{whether or not } j \text{ precedes } k \text{ on some machine}$ (binary)
Example 3: Job Scheduling: MILP Model

- $x_{ij} =$ whether or not job $j$ is assigned to machine $i$ (binary)
- $y_{jk} =$ whether or not $j$ precedes $k$ on some machine (binary)
- $t_j =$ start time of job $j$ (continuous)
Example 3: Job Scheduling: MILP Model

- $x_{ij} = \text{whether or not job } j \text{ is assigned to machine } i \text{ (binary)}$
- $y_{jk} = \text{whether or not } j \text{ precedes } k \text{ on some machine (binary)}$
- $t_j = \text{start time of job } j \text{ (continuous)}$

$$
\begin{align*}
\min & \sum_{ij} c_{ij} x_{ij} \\
\text{s.t.} & \quad r_j \leq t_j \leq d_j - \sum_i p_{ij} x_{ij}, \ \forall \ j \\
& \quad \sum_i x_{ij} = 1, \ \forall \ j \\
& \quad y_{jk} + y_{kj} \leq 1, \ \forall \ k > j \\
& \quad y_{jk} + y_{kj} \geq x_{ij} + x_{ik} - 1, \ \forall \ k > j, \ i \\
& \quad y_{jk} + y_{kj} + x_{ij} + x_{i'k} \leq 2, \ \forall \ k > j, \ i' \neq i \\
& \quad t_k \geq t_j + \sum_i p_{ij} x_{ij} - M(1 - y_{jk}), \ \forall \ k \neq j \\
& \quad \sum_j p_{ij} x_{ij} \leq \max_j \{d_j\} - \min_j \{r_j\}, \ \forall \ i
\end{align*}
$$
Example 3: Job Scheduling on Parallel Machines
Benders Decomposition Approach

Master Problem
▶ Assign jobs to machines at minimum cost
▶ “Ignore” release dates and due dates
▶ $x_{ij} = 1$ if job $i$ assigned to machine $j$

Subproblem for machine $j$
▶ Try to find feasible schedule with given set of jobs
$\sum_{i \in I_j} x_{ij} \leq |I_j| - 1$

Aron, Hooker and Yunes

An Integrated Solver for Optimization Problems
Example 3: Job Scheduling on Parallel Machines
Benders Decomposition Approach

- **Master Problem**
  - Assign jobs to machines at minimum cost
  - “Ignore” release dates and due dates
  - $x_{ij} = 1$ if job $i$ assigned to machine $j$
Example 3: Job Scheduling on Parallel Machines

Benders Decomposition Approach

- **Master Problem**
  - Assign jobs to machines at minimum cost
  - “Ignore” release dates and due dates
  - \( x_{ij} = 1 \) if job \( i \) assigned to machine \( j \)

- **Subproblem for machine \( j \)**
  - Try to find feasible schedule with given set of jobs \( I_j \)
  - If *infeasible*, generate Benders cut
Example 3: Job Scheduling on Parallel Machines
Benders Decomposition Approach

- **Master Problem**
  - Assign jobs to machines at minimum cost
  - “Ignore” release dates and due dates
  - $x_{ij} = 1$ if job $i$ assigned to machine $j$

- **Subproblem for machine $j$**
  - Try to find feasible schedule with given set of jobs $I_j$
  - If infeasible, generate Benders cut

$$\sum_{i \in I_j} x_{ij} \leq |I_j| - 1$$
Example 3: Job Scheduling: Integrated Benders

\[ \begin{align*}
\text{min} & \quad \sum_{ij} c_{ij} x_{ij} \\
\sum_i x_{ij} & = 1, \quad \forall j \\
(\forall x_{ij} = 1) & \iff (y_j = i), \quad \forall i, j \\
\begin{align*}
\tau_j & \leq t_j \leq d_j - p_j y_j, \\
\tau_j & \leq x_{ij} \leq 1, \quad \forall i
\end{align*}
\end{align*} \]
Example 3: Job Scheduling: Integrated Benders

- $x_{ij} = \text{whether or not job } j \text{ is assigned to machine } i \text{ (binary)}$
Example 3: Job Scheduling: Integrated Benders

- $x_{ij} =$ whether or not job $j$ is assigned to machine $i$ (binary)
- $y_j =$ machine assigned to job $j$ (integer)
Example 3: Job Scheduling: Integrated Benders

- $x_{ij} =$ whether or not job $j$ is assigned to machine $i$ (binary)
- $y_j =$ machine assigned to job $j$ (integer)
- $t_j =$ start time of job $j$ (continuous)
Example 3: Job Scheduling: Integrated Benders

- $x_{ij} =$ whether or not job $j$ is assigned to machine $i$ (binary)
- $y_j =$ machine assigned to job $j$ (integer)
- $t_j =$ start time of job $j$ (continuous)

Integrated Benders Model
Example 3: Job Scheduling: Integrated Benders

- $x_{ij} =$ whether or not job $j$ is assigned to machine $i$ (binary)
- $y_j =$ machine assigned to job $j$ (integer)
- $t_j =$ start time of job $j$ (continuous)

**Integrated Benders Model**

$$\text{min} \sum_{ij} c_{ij} x_{ij}$$
Example 3: Job Scheduling: Integrated Benders

- $x_{ij} =$ whether or not job $j$ is assigned to machine $i$ (binary)
- $y_j =$ machine assigned to job $j$ (integer)
- $t_j =$ start time of job $j$ (continuous)

### Integrated Benders Model

\[
\begin{align*}
\min & \quad \sum_{ij} c_{ij} x_{ij} \\
\text{s.t.} & \quad \sum_i x_{ij} = 1, \quad \forall \ j
\end{align*}
\]
Example 3: Job Scheduling: Integrated Benders

- $x_{ij} =$ whether or not job $j$ is assigned to machine $i$ (binary)
- $y_j =$ machine assigned to job $j$ (integer)
- $t_j =$ start time of job $j$ (continuous)

**Integrated Benders Model**

$$\min \sum_{ij} c_{ij} x_{ij}$$
$$\sum_i x_{ij} = 1, \ \forall \ j$$
$$(x_{ij} = 1) \Leftrightarrow (y_j = i), \ \forall \ i, j$$
Example 3: Job Scheduling: Integrated Benders

- \( x_{ij} = \) whether or not job \( j \) is assigned to machine \( i \) (binary)
- \( y_j = \) machine assigned to job \( j \) (integer)
- \( t_j = \) start time of job \( j \) (continuous)

**Integrated Benders Model**

\[
\begin{align*}
\text{min} & \quad \sum_{ij} c_{ij} x_{ij} \\
\sum_i x_{ij} &= 1, \quad \forall j \\
(x_{ij} = 1) &\iff (y_j = i), \quad \forall i, j \\
r_j &\leq t_j \leq d_j - p_{y_jj}, \quad \forall j
\end{align*}
\]
Example 3: Job Scheduling: Integrated Benders

- \( x_{ij} \) = whether or not job \( j \) is assigned to machine \( i \) (binary)
- \( y_j \) = machine assigned to job \( j \) (integer)
- \( t_j \) = start time of job \( j \) (continuous)

Integrated Benders Model

\[
\begin{align*}
\min & \quad \sum_{ij} c_{ij} x_{ij} \\
\text{subject to} & \quad \sum_i x_{ij} = 1, \quad \forall \ j \\
& \quad (x_{ij} = 1) \iff (y_j = i), \quad \forall \ i, j \\
& \quad r_j \leq t_j \leq d_j - p_{y_j}, \quad \forall \ j \\
& \quad \text{cumulative}((t_j, p_{ij}, 1 \mid x_{ij} = 1), 1), \quad \forall \ i
\end{align*}
\]
Example 3: Job Scheduling: Integrated Benders

- $x_{ij} =$ whether or not job $j$ is assigned to machine $i$ (binary)
- $y_j =$ machine assigned to job $j$ (integer)
- $t_j =$ start time of job $j$ (continuous)

**Integrated Benders Model**

$$
\begin{align*}
&\text{min } \sum_{ij} c_{ij} x_{ij} \\
&\sum_i x_{ij} = 1, \forall j \\
&(x_{ij} = 1) \iff (y_j = i), \forall i, j \\
&r_j \leq t_j \leq d_j - p_{y_j j}, \forall j \\
&\text{cumulative}((t_j, p_{ij}, 1 | x_{ij} = 1), 1), \forall i
\end{align*}
$$

Need to tell the solver how to decompose the model
Example 3: Job Scheduling: SIMPL Model

OBJECTIVE
\[ \text{min sum } i,j \text{ of } c_{ij} \times x_{ij} \]

CONSTRAINTS
- assign means \( \{ \sum i \text{ of } x_{ij} = 1 \text{ forall } j \} \)
- relaxation = \{ ip:master \}
- \( x_{ij} = 1 \iff y_{j} = i \text{ forall } i, j \)
  - relaxation = \{ cp:sub \}
- \( r_{j} \leq t_{j} \leq d_{j} - p_{y_{j}} \text{ forall } j \)
  - relaxation = \{ ip:master, cp:sub \}
- machinecap means \( \{ \text{cumulative} \{ \{t_{j}, p_{i}, 1\} \text{ forall } j \mid x_{ij} = 1, 1\} \text{ forall } i \} \)
  - relaxation = \{ ip:master, cp:sub:decomp \}
- inference = \{ feasibility \}

SEARCH
- type = \{ benders \}
Example 3: Job Scheduling: SIMPL Model

**OBJECTIVE**  \[ \min \sum_{i,j} c[i][j] \times x[i][j] \]
Example 3: Job Scheduling: SIMPL Model

**OBJECTIVE**

\[
\text{min } \sum_{i,j} c[i][j] \cdot x[i][j]
\]

**CONSTRAINTS**

- **assign**
  \[
  \sum_i x[i][j] = 1 \quad \text{forall } j
  \]

- **relaxation**
  \[
  \text{ip:master}
  \]

- **xy**
  \[
  x[i][j] = 1 \iff y[j] = i \quad \text{forall } i, j
  \]

- **relaxation**
  \[
  \text{cp:sub}
  \]

- **tbounds**
  \[
  r[j] \leq t[j] \leq d[j] - p[y[j]][j] \quad \text{forall } j
  \]

- **relaxation**
  \[
  \text{ip:master, cp:sub}
  \]

- **machinecap**
  \[
  \text{cumulative} (\{t[j], p[i][j], 1\} \quad \text{forall } j | x[i][j]=1, i)
  \quad \text{forall } i
  \]

- **relaxation**
  \[
  \text{ip:master, cp:sub, decomp}
  \]

- **inference**
  \[
  \text{feasibility}
  \]

- **SEARCH**
  \[
  \text{type}\quad \{ \text{benders} \}
  \]
Example 3: Job Scheduling: SIMPL Model

OBJECTIVE \( \min \sum_{i,j} c[i][j] \cdot x[i][j] \)

CONSTRAINTS

assign means { 
  \( \sum_{i} x[i][j] = 1 \) for all \( j \) 
  relaxation = { ip:master } }

relaxation = { ip:master } 


tbounds means { 
  \( r[j] \leq t[j] \leq d[j] - p[y[j]][j] \) for all \( j \) 
  relaxation = { ip:master, cp:sub } }

machinecap means { 
  \( \sum_{j} (t[j], p[i][j], 1) \) for all \( i \) 
  relaxation = { ip:master, cp:sub } 
  decomp } 

inference = { feasibility }
Example 3: Job Scheduling: SIMPL Model

**OBJECTIVE**  \( \min \sum_{i,j} c[i][j] \times x[i][j] \)

**CONSTRAINTS**

- **assign**  \( \text{means} \)  
  \[ \sum_{i} x[i][j] = 1 \text{ forall } j \]
  \( \text{relaxation} = \{ \text{ip:master} \} \)

- **xy**  \( \text{means} \)  
  \[ x[i][j] = 1 \iff y[j] = i \text{ forall } i, j \]
  \( \text{relaxation} = \{ \text{cp:sub} \} \)

- **tbounds**  \( \text{means} \)  
  \[ r[j] \leq t[j] \leq d[j] - p[y[j]][j] \text{ forall } j \]
  \( \text{relaxation} = \{ \text{ip:master, cp:sub} \} \)

- **machinecap**  \( \text{means} \)  
  \[ \text{cumulative} \{ t[j], p[i][j], 1 \forall j | x[i][j] = 1, 1 \} \forall i \]
  \( \text{relaxation} = \{ \text{ip:master, cp:sub, decomp} \} \)

**inference**  \( \{ \text{feasibility} \} \)

**SEARCH**  \( \text{type} = \{ \text{benders} \} \)
Example 3: Job Scheduling: SIMPL Model

**OBJECTIVE**  \( \text{min} \sum_{i,j} c_{ij}x_{ij} \)

**CONSTRAINTS**

- **assign** means \{ 
  \( \sum_i x_{ij} = 1 \) for all \( j \) 
  \( \text{relaxation} = \{ \text{ip:master} \} \) \}

- **xy** means \{ 
  \( x_{ij} = 1 \iff y_j = i \) for all \( i, j \) 
  \( \text{relaxation} = \{ \text{cp:sub} \} \) \}

- **tbounds** means \{ 
  \( r_j \leq t_j \leq d_j - p_{y_j}[j] \) for all \( j \) 
  \( \text{relaxation} = \{ \text{ip:master, cp:sub} \} \) \}
Example 3: Job Scheduling: SIMPL Model

**OBJECTIVE** \[ \text{min} \ \sum_{i,j} c[i][j] \cdot x[i][j] \]

**CONSTRAINTS**
- **assign** means \{ \[ \sum_{i} x[i][j] = 1 \ \forall j \]
  \[ \text{relaxation} = \{ \text{ip:master} \} \] \}
- **xy** means \{ \[ x[i][j] = 1 \iff y[j] = i \ \forall i, j \]
  \[ \text{relaxation} = \{ \text{cp:sub} \} \] \}
- **tbounds** means \{ \[ r[j] \leq t[j] \leq d[j] - p[y[j]][j] \ \forall j \]
  \[ \text{relaxation} = \{ \text{ip:master, cp:sub} \} \] \}
- **machinecap** means \{ \[ \text{cumulative}\{\{ t[j], p[i][j], 1 \} \ \forall j | x[i][j] = 1, 1 \} \ \forall i \]
  \[ \text{relaxation} = \{ \text{ip:master, cp:sub:decomp} \} \] \}
  \[ \text{inference} = \{ \text{feasibility} \} \] \}
Example 3: Job Scheduling: SIMPL Model

OBJECTIVE \[ \min \sum_{i,j} c[i][j] \cdot x[i][j] \]

CONSTRAINTS

assign means { 
    sum i of x[i][j] = 1 forall j 
relaxation = { ip:master } }

xy means {
    x[i][j] = 1 \iff y[j] = i forall i, j 
relaxation = { cp:sub } }

tbounds means {
relaxation = { ip:master, cp:sub } }

machinecap means {
    cumulative({t[j],p[i][j],1} forall j | x[i][j]=1, 1) forall i 
relaxation = { ip:master, cp:sub:decomp } }

inference = { feasibility }

SEARCH

type = { benders }
Example 3: Job Scheduling on Parallel Machines

Computational Results

<table>
<thead>
<tr>
<th>Machines</th>
<th>Jobs</th>
<th>Nodes</th>
<th>Time (s)</th>
<th>Iterations</th>
<th>Cuts</th>
</tr>
</thead>
<tbody>
<tr>
<td>3</td>
<td>2</td>
<td></td>
<td>0.00</td>
<td>2</td>
<td></td>
</tr>
<tr>
<td>7</td>
<td>3</td>
<td></td>
<td>0.02</td>
<td>12</td>
<td>0.09</td>
</tr>
<tr>
<td>12</td>
<td>3</td>
<td></td>
<td>16.50</td>
<td>26</td>
<td>0.58</td>
</tr>
<tr>
<td>15</td>
<td>5</td>
<td></td>
<td>14.30</td>
<td>22</td>
<td>0.96</td>
</tr>
<tr>
<td>20</td>
<td>5</td>
<td></td>
<td>3123.34</td>
<td>30</td>
<td>3.21</td>
</tr>
<tr>
<td>22</td>
<td>5</td>
<td></td>
<td>&gt;5.0M</td>
<td>&gt;48h</td>
<td>6.70</td>
</tr>
</tbody>
</table>

Aron, Hooker and Yunes
Example 3: Job Scheduling on Parallel Machines

Computational Results

Instances from Jain and Grossmann (2001)

<table>
<thead>
<tr>
<th>Jobs</th>
<th>Machines</th>
<th>MILP Nodes</th>
<th>Time (s)</th>
<th>Integrated Benders Iterations</th>
<th>Cuts</th>
<th>Time (s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>3</td>
<td>2</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>7</td>
<td>3</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>12</td>
<td>3</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>15</td>
<td>5</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>20</td>
<td>5</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>22</td>
<td>5</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Example 3: Job Scheduling on Parallel Machines

Computational Results

Instances from Jain and Grossmann (2001)

<table>
<thead>
<tr>
<th>Jobs</th>
<th>Machines</th>
<th>MILP</th>
<th>Integrated Benders</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>Nodes</td>
<td>Time (s)</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>0.02</td>
</tr>
<tr>
<td>12</td>
<td>3</td>
<td>11060</td>
<td>16.50</td>
</tr>
<tr>
<td>15</td>
<td>5</td>
<td>3674</td>
<td>14.30</td>
</tr>
<tr>
<td>20</td>
<td>5</td>
<td>159400</td>
<td>3123.34</td>
</tr>
<tr>
<td>22</td>
<td>5</td>
<td>&gt; 5.0M</td>
<td>&gt; 48h</td>
</tr>
</tbody>
</table>
Example 3: Job Scheduling on Parallel Machines

Computational Results

Instances from Jain and Grossmann (2001)

<table>
<thead>
<tr>
<th>Jobs</th>
<th>Machines</th>
<th>MILP</th>
<th>Integrated Benders</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>Nodes</td>
<td>Time (s)</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>0.02</td>
</tr>
<tr>
<td>12</td>
<td>3</td>
<td>11060</td>
<td>16.50</td>
</tr>
<tr>
<td>15</td>
<td>5</td>
<td>3674</td>
<td>14.30</td>
</tr>
<tr>
<td>20</td>
<td>5</td>
<td>159400</td>
<td>3123.34</td>
</tr>
<tr>
<td>22</td>
<td>5</td>
<td>&gt; 5.0M</td>
<td>&gt; 48h</td>
</tr>
</tbody>
</table>
Example 3: Job Scheduling on Parallel Machines
Computational Results (continued)

Instances from Jain and Grossmann (2001)
Shorter processing times make the problem easier to solve
Instances from Jain and Grossmann (2001)
Shorter processing times make the problem easier to solve
Instances from Jain and Grossmann (2001)  
Shorter processing times make the problem easier to solve

<table>
<thead>
<tr>
<th>Jobs</th>
<th>Machines</th>
<th>MILP</th>
<th>Integrated Benders</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>Nodes</td>
<td>Time (s)</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>0.01</td>
</tr>
<tr>
<td>12</td>
<td>3</td>
<td>4950</td>
<td>1.98</td>
</tr>
<tr>
<td>15</td>
<td>5</td>
<td>14000</td>
<td>19.80</td>
</tr>
<tr>
<td>20</td>
<td>5</td>
<td>140</td>
<td>5.73</td>
</tr>
<tr>
<td>22</td>
<td>5</td>
<td>&gt; 16.9M</td>
<td>&gt; 48h</td>
</tr>
<tr>
<td>25</td>
<td>5</td>
<td>&gt; 4.5M</td>
<td>&gt; 48h</td>
</tr>
</tbody>
</table>

Aron, Hooker and Yunes

An Integrated Solver for Optimization Problems
Example 3: Job Scheduling on Parallel Machines
Computational Results (continued)

Instances from Jain and Grossmann (2001)
Shorter processing times make the problem easier to solve

<table>
<thead>
<tr>
<th>Jobs</th>
<th>Machines</th>
<th>MILP Nodes</th>
<th>MILP Time (s)</th>
<th>Integrated Benders Iterations</th>
<th>Cuts</th>
<th>Time (s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>3</td>
<td>2</td>
<td>1</td>
<td>0.00</td>
<td>1</td>
<td>0</td>
<td>0.00</td>
</tr>
<tr>
<td>7</td>
<td>3</td>
<td>1</td>
<td>0.01</td>
<td>1</td>
<td>0</td>
<td>0.01</td>
</tr>
<tr>
<td>12</td>
<td>3</td>
<td>4950</td>
<td>1.98</td>
<td>1</td>
<td>0</td>
<td>0.01</td>
</tr>
<tr>
<td>15</td>
<td>5</td>
<td>14000</td>
<td>19.80</td>
<td>1</td>
<td>0</td>
<td>0.03</td>
</tr>
<tr>
<td>20</td>
<td>5</td>
<td>140</td>
<td>5.73</td>
<td>3</td>
<td>3</td>
<td>0.12</td>
</tr>
<tr>
<td>22</td>
<td>5</td>
<td>&gt; 16.9M</td>
<td>&gt; 48h</td>
<td>5</td>
<td>4</td>
<td>0.38</td>
</tr>
<tr>
<td>25</td>
<td>5</td>
<td>&gt; 4.5M</td>
<td>&gt; 48h</td>
<td>16</td>
<td>22</td>
<td>0.86</td>
</tr>
</tbody>
</table>
Future Work

Regarding SIMPL itself...

▶ Support other integrated approaches, e.g. local search, B&P
▶ More features: non-linear solver, cutting planes, more constraints
▶ More powerful language in SEARCH section (like OPL)
▶ More intelligent model compilation (e.g. detect special structures)
▶ Improve performance (code optimization)
▶ etc.
Future Work

- Regarding **SIMPL** itself...
Future Work

Regarding SIMPL itself...

- Support other integrated approaches, e.g. local search, B&P
Future Work

Regarding **SIMPL** itself...

- Support other integrated approaches, e.g. local search, B&P
- **More features:** non-linear solver, cutting planes, more constraints
Future Work

Regarding **SIMPL** itself...

- Support other integrated approaches, e.g. local search, B&P
- **More features**: non-linear solver, cutting planes, more constraints
- More powerful language in **SEARCH** section (like OPL)
Future Work

Regarding SIMPL itself...

- Support other integrated approaches, e.g. local search, B&P
- **More features:** non-linear solver, cutting planes, more constraints
- More powerful language in SEARCH section (like OPL)
- More intelligent model compilation (e.g. detect special structures)
Regarding SIMPL itself...

- Support other integrated approaches, e.g. local search, B&P
- More features: non-linear solver, cutting planes, more constraints
- More powerful language in SEARCH section (like OPL)
- More intelligent model compilation (e.g. detect special structures)
- Improve performance (code optimization)
Regarding **SIMPL** itself...

- Support *other* integrated approaches, e.g. local search, B&P
- **More features:** non-linear solver, cutting planes, more constraints
- More powerful language in **SEARCH** section (like OPL)
- More intelligent model compilation (e.g. detect special structures)
- Improve performance (code optimization)
- etc.
Future Work (continued)

Regarding SIMPL’s availability...

▶ Distribute source code?

▶ Distribute executable?

▶ Add it to NEOS? COIN-OR?

▶ We are still thinking about it...

▶ Whichever way we go, we still need to clean it up a bit...
Regarding SIMPL’s availability...
Regarding SIMPL’s availability...
  ▶ Distribute source code?
  ▶ Distribute executable?
  ▶ Add it to NEOS? COIN-OR?
  ▶ We are still thinking about it...
  ▶ Whichever way we go, we still need to clean it up a bit...
Future Work (continued)

- Regarding SIMPL’s availability...
  - Distribute source code?
  - Distribute executable?

Aron, Hooker and Yunes
An Integrated Solver for Optimization Problems
Future Work (continued)

Regarding SIMPL’s availability...
  ▶ Distribute source code?
  ▶ Distribute executable?
  ▶ Add it to NEOS? COIN-OR?
Future Work (continued)

- Regarding **SIMPL**’s availability...
  - Distribute source code?
  - Distribute executable?
  - Add it to NEOS? COIN-OR?
  - We are still thinking about it...
Regarding SIMPL’s availability...

- Distribute source code?
- Distribute executable?
- Add it to NEOS? COIN-OR?
- We are still thinking about it...
- Whichever way we go, we still need to clean it up a bit...
Many theoretical and technological breakthroughs over the last few decades have helped OR become more accessible and popular. Many important problems are still hard for traditional methods, but recent literature shows integrated methods can succeed when traditional methods fail. SIMPL is a step toward making integrated methods more accessible to a larger group of users. It is also a very useful research tool. We still have a long way to go, but important steps have been taken and initial results are encouraging.
Many theoretical and technological breakthroughs over the last few decades have helped OR become more accessible and popular.
Many theoretical and technological breakthroughs over the last few decades have helped OR become more accessible and popular.

Many important problems are still hard for traditional methods.
Conclusion

- Many theoretical and technological breakthroughs over the last few decades have helped OR become more accessible and popular.
- Many important problems are still hard for traditional methods.
- Recent literature shows integrated methods can succeed when traditional methods fail.
Conclusion

- Many theoretical and technological breakthroughs over the last few decades have helped OR become more accessible and popular.
- Many important problems are still hard for traditional methods.
- Recent literature shows integrated methods can succeed when traditional methods fail.
- SIMPL is...
Conclusion

- Many theoretical and technological breakthroughs over the last few decades have helped OR become more accessible and popular.
- Many important problems are still hard for traditional methods.
- Recent literature shows integrated methods can succeed when traditional methods fail.
- **SIMPL** is:
  - a step toward making integrated methods more accessible to a larger group of users.
Many theoretical and technological breakthroughs over the last few decades have helped OR become more accessible and popular.

Many important problems are still hard for traditional methods.

Recent literature shows integrated methods can succeed when traditional methods fail.

SIMPL is:
- a step toward making integrated methods more accessible to a larger group of users
- a very useful research tool
Many theoretical and technological breakthroughs over the last few decades have helped OR become more accessible and popular.

Many important problems are still hard for traditional methods.

Recent literature shows integrated methods can succeed when traditional methods fail.

SIMPL is

- a step toward making integrated methods more accessible to a larger group of users
- a very useful research tool

We still have a long way to go, but important steps have been taken and initial results are encouraging.
That’s All Folks!

Thank you!

Any Questions?
Constraint Programming Example

Constraint Programming Example

x ∈ {1, 3}
y ∈ {1, 2, 3}
z ∈ {1, 2, 3}
w ∈ {1, 2, 4}

element(z, [1, 3, 5], x)

alldifferent(x, y, z, w)

2z − w ≥ 0
Constraint Programming Example

\[x \in \{1, 3\}\]
\[y \in \{1, 2, 3\}\]
\[z \in \{1, 2, 3\}\]
\[w \in \{1, 2, 4\}\]
Constraint Programming Example

\begin{align*}
x & \in \{1, 3\} \\
y & \in \{1, 2, 3\} \\
z & \in \{1, 2, 3\} \\
w & \in \{1, 2, 4\}
\end{align*}

\text{element}(z, [1, 3, 5], x) \quad (x \text{ is the } z^{\text{th}} \text{ element of } [1, 3, 5])

\text{alldifferent}(x, y, z, w) \quad (\text{all variables take distinct values})
Constraint Programming Example

\[
x \in \{1, 3\}
\]
\[
y \in \{1, 2, 3\}
\]
\[
z \in \{1, 2, 3\}
\]
\[
w \in \{1, 2, 4\}
\]

\text{element}(z, [1, 3, 5], x) \quad (x \text{ is the } z^{\text{th}} \text{ element of } [1, 3, 5])

\text{alldifferent}(x, y, z, w) \quad (\text{all variables take distinct values})
Constraint Programming Example

\[ x \in \{1, 3\} \]
\[ y \in \{1, 2, 3\} \]
\[ z \in \{1, 2, 3\} \]
\[ w \in \{1, 2, 4\} \]

\textbf{element}(z, [1, 3, 5], x) \quad (x \text{ is the } z^{th} \text{ element of } [1, 3, 5])

\textbf{alldifferent}(x, y, z, w) \quad (\text{all variables take distinct values})

\[ 2z - w \geq 0 \]
Constraint Programming Example

$x \in \{1, 3\}$
$y \in \{1, 2, 3\}$
$z \in \{1, 2, 3\}$
$w \in \{1, 2, 4\}$

\text{element}(z, [1, 3, 5], x) \quad (x \text{ is the } z^{\text{th}} \text{ element of } [1, 3, 5])
\text{alldifferent}(x, y, z, w) \quad (\text{all variables take distinct values})

2z - w \geq 0
Constraint Programming Example

\[ x \in \{1, 3\} \]
\[ y \in \{1, 2, 3\} \]
\[ z \in \{1, 2, 3\} \]
\[ w \in \{1, 2, 4\} \]

\textbf{element}(z, [1, 3, 5], x) \quad (x \text{ is the } z^{th} \text{ element of } [1, 3, 5])

\textbf{alldifferent}(x, y, z, w) \quad (\text{all variables take distinct values})

\[ 2z - w \geq 0 \]
Constraint Programming Example

\[ x \in \{1, 3\} \]
\[ y \in \{1, 2, 3\} \]
\[ z \in \{1, 2, 3\} \]
\[ w \in \{1, 2, 4\} \]

\textbf{element}(z, [1, 3, 5], x) \quad (x \text{ is the } z^{\text{th}} \text{ element of } [1, 3, 5])

\textbf{alldifferent}(x, y, z, w) \quad (\text{all variables take distinct values})

\[ 2z - w \geq 0 \]
Constraint Programming Example

\[ x \in \{1, 3\} \]
\[ y \in \{1, 2, 3\} \]
\[ z \in \{1, 2, 3\} \]
\[ w \in \{1, 2, 4\} \]

\texttt{element}(z, [1, 3, 5], x) \quad (x \text{ is the } z^{th} \text{ element of } [1, 3, 5])
\texttt{alldifferent}(x, y, z, w) \quad (\text{all variables take distinct values})
2z - w \geq 0
Constraint Programming Example

\( x \in \{1, 3\} \)
\( y \in \{1, 2, 3\} \)
\( z \in \{1, 2, 3\} \)
\( w \in \{1, 2, 4\} \)

\[ \text{element}(z, [1, 3, 5], x) \]  
\( x \) is the \( z \)th element of \([1, 3, 5]\)

\[ \text{alldifferent}(x, y, z, w) \]  
(all variables take distinct values)

\[ 2z - w \geq 0 \]
Constraint Programming Example

\[ x \in \{1, 3\} \]
\[ y \in \{1, 2, 3\} \]
\[ z \in \{1, 2, 3\} \]
\[ w \in \{1, 2, 4\} \]

\text{element}(z, [1, 3, 5], x) \quad (x \text{ is the } z^{th} \text{ element of } [1, 3, 5])
\text{alldifferent}(x, y, z, w) \quad (\text{all variables take distinct values})
\[ 2z - w \geq 0 \]