An Integrated Solver for Optimization Problems

Ionuț D. Aron[1] John N. Hooker[2] Tallys H. Yunes[3]

[1]Department of Computer Science, Brown University

[2]Tepper School of Business, Carnegie Mellon University

[3]Department of Management Science, University of Miami

June 6 2006

I'd Like To Thank the Program Committee for...

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

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?

Why Integrate?

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

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

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

Integration Continued

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

Integration Continued

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

Some benefits of integration:

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

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

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

Constraint Programming (CP)

Constraint Programming (CP)

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

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

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 (continued)

Constraint Programming (continued)

When compared to MILP models, CP models usually have

Constraint Programming (continued)

When compared to MILP models, CP models usually have

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

Constraint Programming (continued)

When compared to MILP models, CP models usually have

- More informative (larger) variable domains:
  - city $\in$ { 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.

Constraint Programming (continued)

When compared to MILP models, CP models usually have

- More informative (larger) variable domains:
  - city $\in$ { 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

Previous Work

Previous Work

Existing modeling languages and programming libraries support integration to a greater or lesser extent:

Previous Work

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

Previous Work (continued)

Previous Work (continued)

Some relevant concepts and techniques:

Previous Work (continued)

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

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
- Relaxation of global constraints as systems of linear inequalities: Hooker 00, Refalo 00, Williams & Yan 01, Yunes 02

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
- 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

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
- 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

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

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

Solution Method	Restriction	Inference	Relaxation

The Ubiquity of Search, Inference, Relaxation

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

Constraint-Based Control

Constraint-Based Control

- 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

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.)

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

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

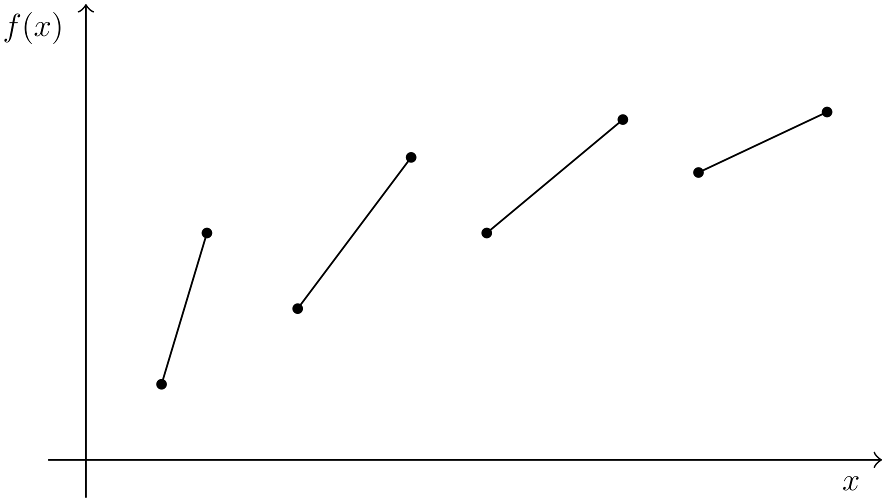
Example 1: Production Planning

- Manufacture several products at a plant of limited capacity
- Products made in one of several production modes (e.g. small scale, medium scale, etc.)
- $x =$ 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 = quantity of product i (continuous)
- y_{ik} = whether or not product i 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)
- λ_{ik}, μ_{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)
- λ_{ik}, μ_{ik} = weights for mode k (convex combination)

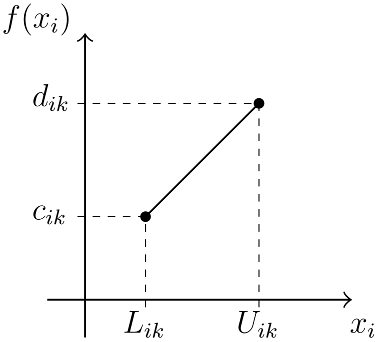
MILP Model

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)
- λ_{ik}, μ_{ik} = weights for mode k (convex combination)

MILP Model

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



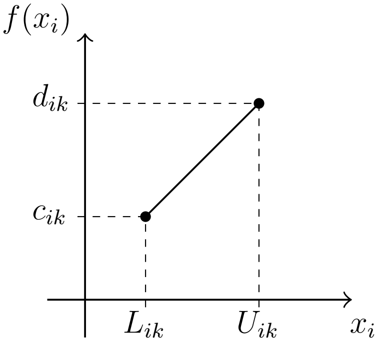
Example 1: Production Planning: MILP

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

MILP Model

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

$$\sum_i x_i \le C$$



Example 1: Production Planning: MILP

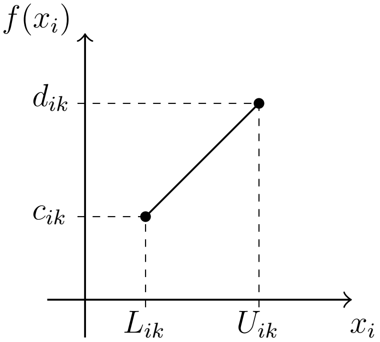
- x_i = quantity of product i (continuous)
- y_{ik} = whether or not product i is made in mode k (binary)
- λ_{ik}, μ_{ik} = weights for mode k (convex combination)

MILP Model

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

$$\sum_i x_i \leq C$$

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

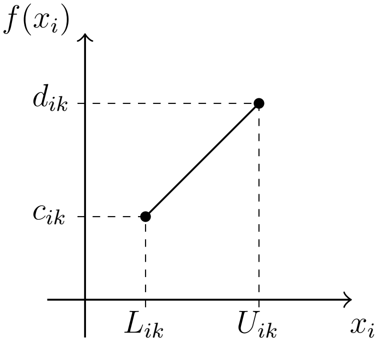


Example 1: Production Planning: MILP

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

MILP Model

$$\max \sum_{ik} \lambda_{ik} c_{ik} + \mu_{ik} d_{ik}$$
$$\sum_i x_i \leq C$$
$$x_i = \sum_k \lambda_{ik} L_{ik} + \mu_{ik} U_{ik}, \ \forall\, i$$
$$\sum_k \lambda_{ik} + \mu_{ik} = 1, \ \forall\, i$$

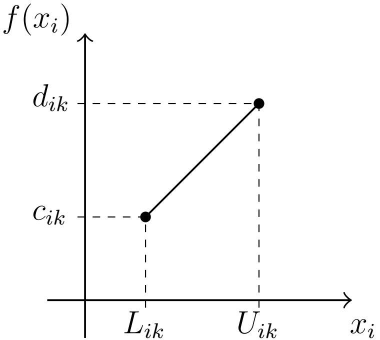


Example 1: Production Planning: MILP

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

MILP Model

$$\max \textstyle\sum_{ik} \lambda_{ik} c_{ik} + \mu_{ik} d_{ik}$$
$$\textstyle\sum_i x_i \leq C$$
$$x_i = \textstyle\sum_k \lambda_{ik} L_{ik} + \mu_{ik} U_{ik}, \ \forall\, i$$
$$\textstyle\sum_k \lambda_{ik} + \mu_{ik} = 1, \ \forall\, i$$
$$0 \leq \lambda_{ik} \leq y_{ik}, \ \forall\, i, k$$

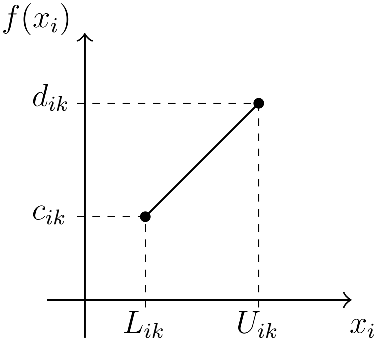


Example 1: Production Planning: MILP

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

MILP Model

$$\max \sum_{ik} \lambda_{ik} c_{ik} + \mu_{ik} d_{ik}$$
$$\sum_i x_i \leq C$$
$$x_i = \sum_k \lambda_{ik} L_{ik} + \mu_{ik} U_{ik}, \ \forall\, i$$
$$\sum_k \lambda_{ik} + \mu_{ik} = 1, \ \forall\, i$$
$$0 \leq \lambda_{ik} \leq y_{ik}, \ \forall\, i, k$$
$$0 \leq \mu_{ik} \leq y_{ik}, \ \forall\, i, k$$

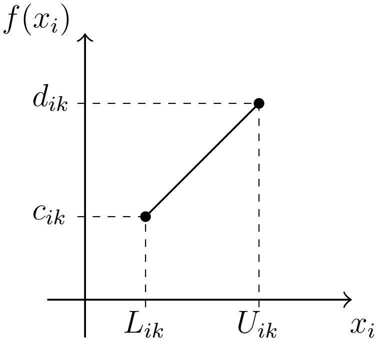


Example 1: Production Planning: MILP

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

MILP Model

$$\max \sum_{ik} \lambda_{ik} c_{ik} + \mu_{ik} d_{ik}$$
$$\sum_i x_i \leq C$$
$$x_i = \sum_k \lambda_{ik} L_{ik} + \mu_{ik} U_{ik}, \ \forall\, i$$
$$\sum_k \lambda_{ik} + \mu_{ik} = 1, \ \forall\, i$$
$$0 \leq \lambda_{ik} \leq y_{ik}, \ \forall\, i, k$$
$$0 \leq \mu_{ik} \leq y_{ik}, \ \forall\, i, k$$
$$\sum_k y_{ik} = 1, \ \forall\, i$$



Example 1: Production Planning: Integrated

Example 1: Production Planning: Integrated

- x_i = quantity of product i (continuous)

Example 1: Production Planning: Integrated

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

Example 1: Production Planning: Integrated

- x_i = quantity of product i (continuous)
- u_i = 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

$$\max \sum_i u_i$$

$$\sum_i x_i \leq C$$

Example 1: Production Planning: Integrated

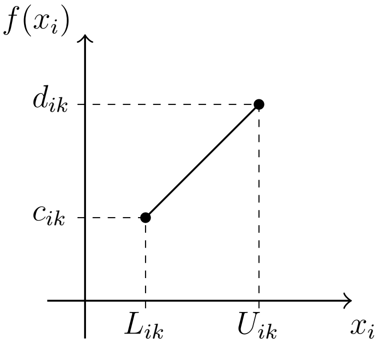
- x_i = quantity of product i (continuous)
- u_i = net income from product i (continuous)

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

Example 1: Production Planning: SIMPL Model

- $x_i =$ quantity of product i (continuous)

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

Example 1: Production Planning: SIMPL Model

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

SIMPL Model

```
OBJECTIVE
```

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]
```

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
```

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] <= 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
   max sum i of u[i]
CONSTRAINTS
   capacity means {
      sum i of x[i] <= C
      relaxation = { lp, cp } }
   income means {
      piecewise(x[i],u[i],L[i],U[i],c[i],d[i]) forall i
      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
   max sum i of u[i]
CONSTRAINTS
   capacity means {
      sum i of x[i] <= 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
```

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] <= 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

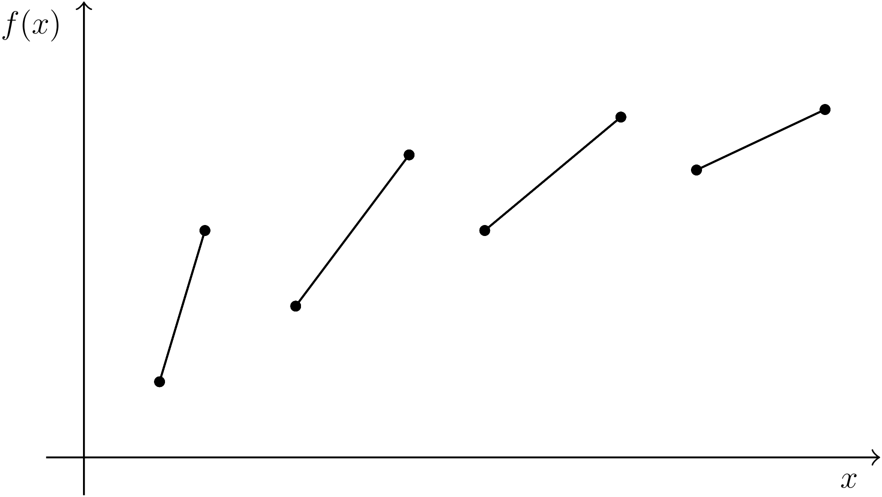
- x_i = quantity of product i (continuous)
- u_i = net income from product i (continuous)

SIMPL Model

```
OBJECTIVE
   max sum i of u[i]
CONSTRAINTS
   capacity means {
      sum i of x[i] <= 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 }
```

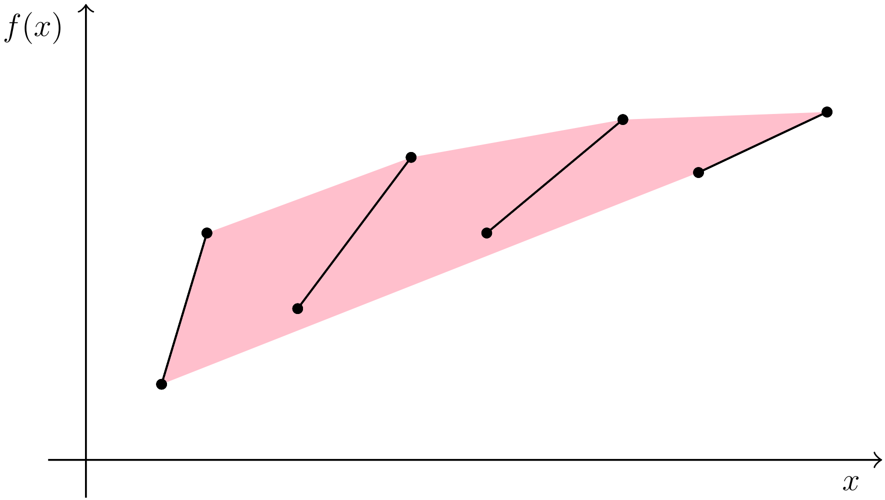
Example 1: Production Planning

Relaxation and Branching for `piecewise`



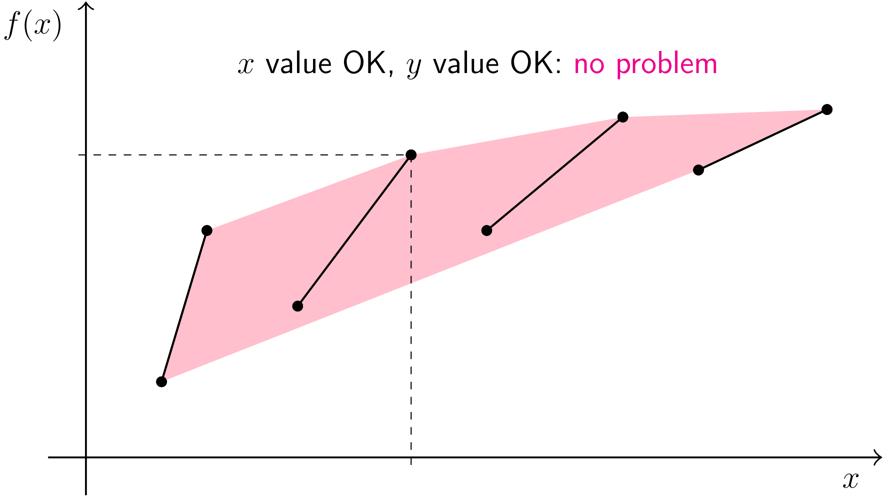
Example 1: Production Planning

Relaxation and Branching for `piecewise`



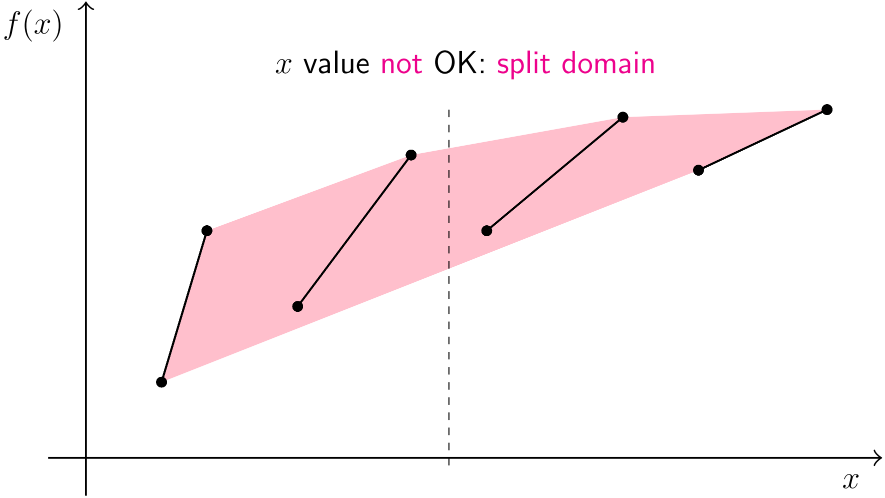
Example 1: Production Planning

Relaxation and Branching for `piecewise`



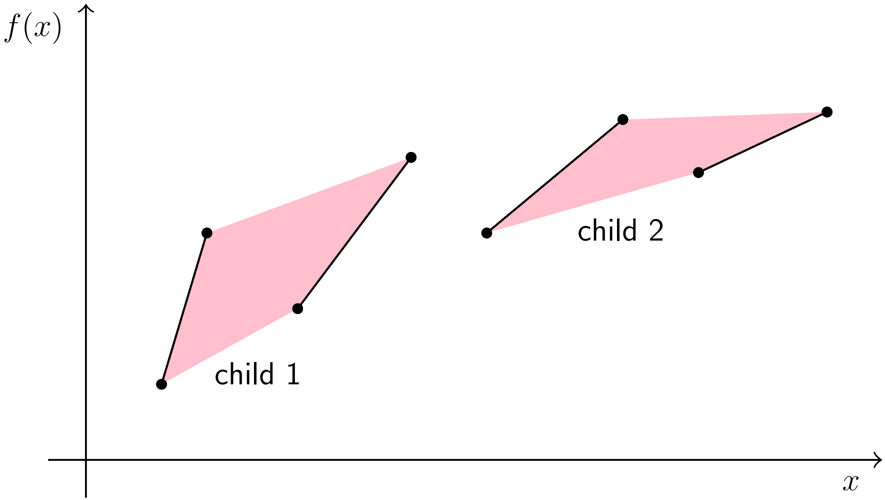
Example 1: Production Planning

Relaxation and Branching for `piecewise`



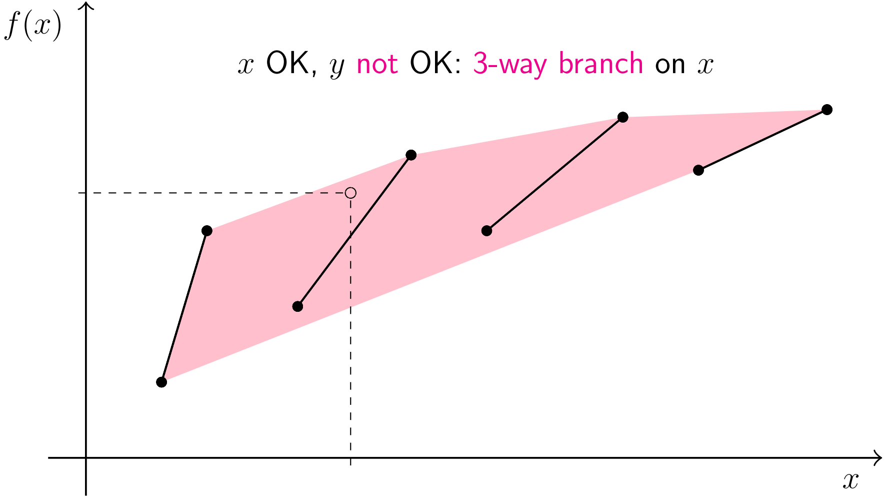
Example 1: Production Planning

Relaxation and Branching for `piecewise`



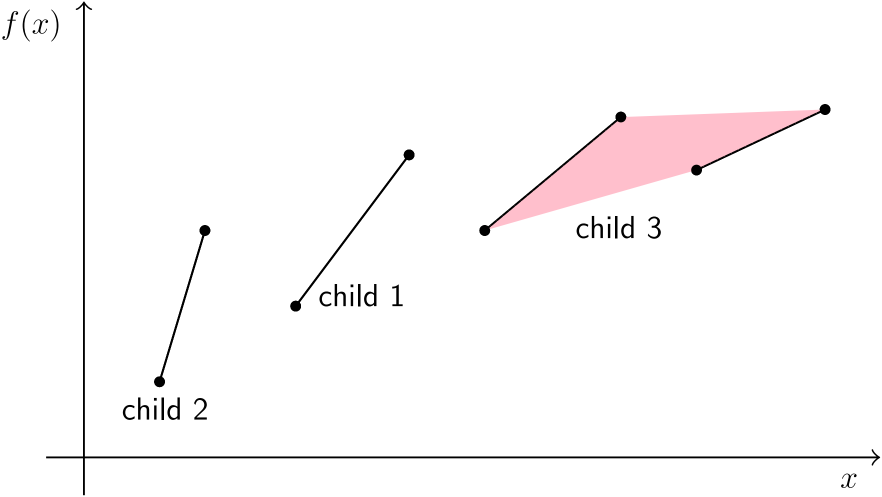
Example 1: Production Planning

Relaxation and Branching for `piecewise`



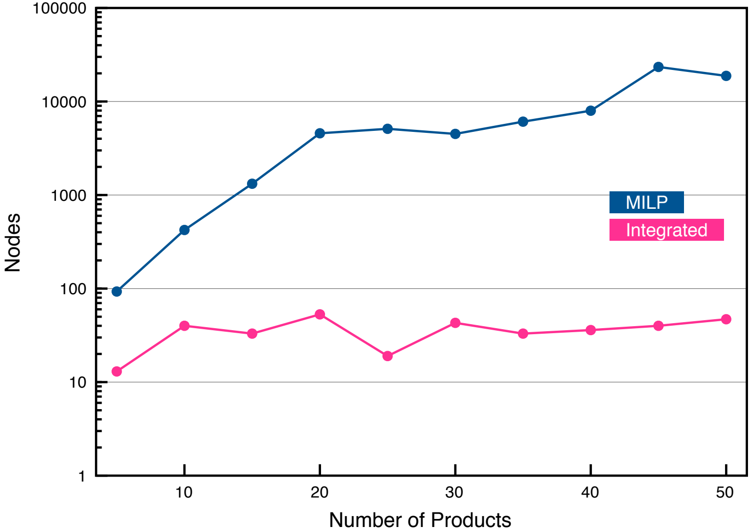
Example 1: Production Planning

Relaxation and Branching for `piecewise`



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

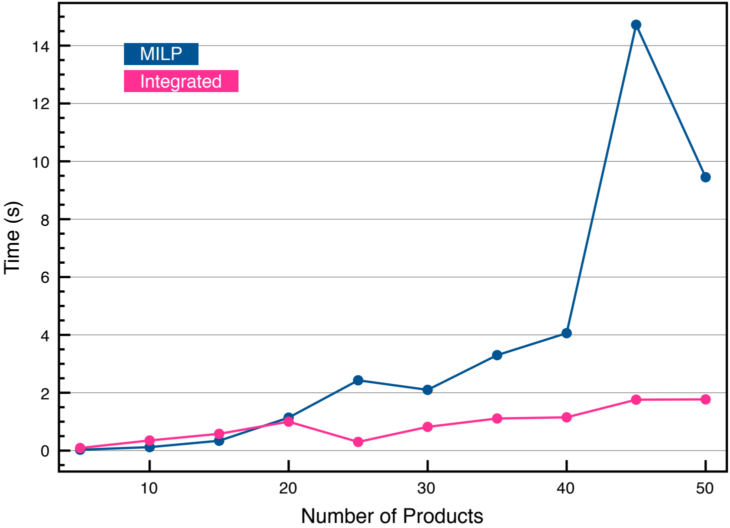
Example 1: Production Planning
Computational Results: Number of Search Nodes



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

Example 1: Production Planning

Computational Results: CPU Time (s)



Example 2: Product Configuration

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
- $c_j =$ unit cost of resource j
- Lower and upper bounds on resource usage/production

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

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

Example 2: Product Configuration: MILP

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

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)

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)

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} = 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

$$\min \sum_j c_j r_j$$

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

$$\min \sum_j c_j r_j$$

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

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

$$\min \sum_j c_j r_j$$

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

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

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

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

Example 2: Product Configuration: MILP

- x_{ik} = 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

$$\min \sum_j c_j r_j$$
$$r_j = \sum_{ik} a_{ijk} q_{ik}, \ \forall\ j$$
$$L_j \le r_j \le U_j, \ \forall\ j$$
$$q_{ik} \le M_i x_{ik}, \ \forall\ i, k$$
$$\sum_k x_{ik} = 1, \ \forall\ i$$

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)
- $t_i =$ type chosen for component i (discrete)
- $r_j =$ amount of resource j produced (continuous)

Integrated Model

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

$$\min \sum_j c_j r_j$$

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

$$\min \sum_j c_j r_j$$

$$r_j = \sum_i a_{ijt_i} q_i, \ \forall\ j$$

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

$$\min \sum_j c_j r_j$$
$$r_j = \sum_i a_{ijt_i} q_i, \ \forall\ j$$
$$L_j \le r_j \le U_j, \ \forall\ j$$

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

$$\min \sum_j c_j r_j$$
$$r_j = \sum_i a_{ijt_i} q_i, \ \forall\, j$$
$$L_j \le r_j \le U_j, \ \forall\, j$$

converted to

$$z_{ij} = a_{ijt_i} q_i$$

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)

Integrated Model

$$\min \sum_j c_j r_j$$

$$r_j = \sum_i a_{ijt_i} q_i, \ \forall\ j$$

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

converted to

$z_{ij} = a_{ijt_i} q_i$ 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

$$\min \textstyle\sum_j c_j r_j$$

$$r_j = \textstyle\sum_i a_{ijt_i} q_i, \ \forall\, j$$

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

converted to

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

equivalent to

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

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)

Integrated Model

$$\min \sum_j c_j r_j$$
$$r_j = \sum_i a_{ijt_i} q_i, \ \forall\ j$$
$$L_j \leq r_j \leq U_j, \ \forall\ j$$

converted to

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

equivalent to

$\bigvee_{k \in D_{t_i}} (z_{ij} = a_{ijk} q_i) \rightarrow$ automatic and dynamic convex hull relax.

Example 2: Product Configuration: SIMPL Model

Example 2: Product Configuration: SIMPL Model

```
OBJECTIVE  min sum j of c[j]*r[j]
```

Example 2: Product Configuration: SIMPL Model

```
OBJECTIVE  min sum j of c[j]*r[j]
CONSTRAINTS
```

Example 2: Product Configuration: SIMPL Model

```
OBJECTIVE  min sum j of c[j]*r[j]
CONSTRAINTS
   resource means {
      r[j] = sum i of a[i][j][t[i]]*q[i] forall j
      relaxation = { lp, cp } }
```

Example 2: Product Configuration: SIMPL Model

```
OBJECTIVE  min sum j of c[j]*r[j]
CONSTRAINTS
   resource means {
      r[j] = sum i of a[i][j][t[i]]*q[i] forall j
      relaxation = { lp, cp } }








SEARCH
```

Example 2: Product Configuration: SIMPL Model

```
OBJECTIVE  min sum j of c[j]*r[j]
CONSTRAINTS
   resource means {
      r[j] = sum i of a[i][j][t[i]]*q[i] forall j
      relaxation = { lp, cp } }









SEARCH
   type = { bb:bestdive }
```

Example 2: Product Configuration: SIMPL Model

```
OBJECTIVE  min sum j of c[j]*r[j]
CONSTRAINTS
  resource means {
    r[j] = sum i of a[i][j][t[i]]*q[i] 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 of c[j]*r[j]
CONSTRAINTS
   resource means {
      r[j] = sum i of a[i][j][t[i]]*q[i] forall j
      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 of c[j]*r[j]
CONSTRAINTS
   resource means {
      r[j] = sum i of a[i][j][t[i]]*q[i] forall j
      relaxation = { lp, cp } }
   quant means {
      q[1] >= 1 => 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 of c[j]*r[j]
CONSTRAINTS
   resource means {
      r[j] = sum i of a[i][j][t[i]]*q[i] forall j
      relaxation = { lp, cp } }
   quant means {
      q[1] >= 1 => q[2] = 0
      relaxation = { lp, cp } }
   types means {
      t[1] = 1 => t[2] in {1, 2}
      t[3] = 1 => (t[4] in {1, 3} and t[5] in {1, 3, 4, 6}
                   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

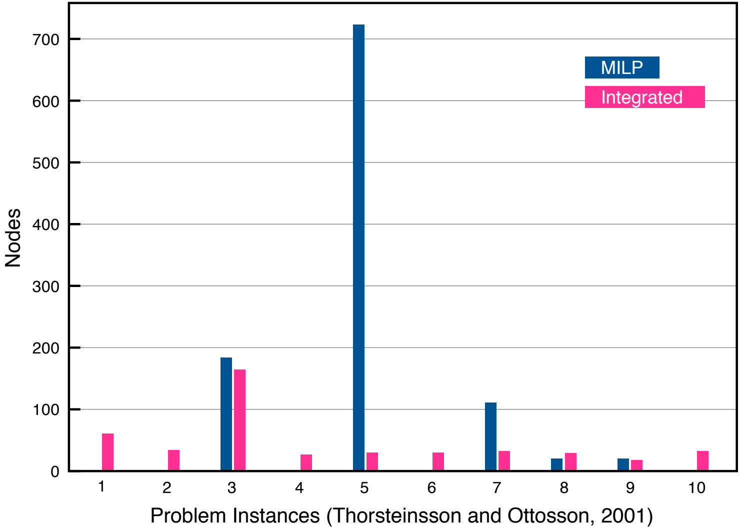
```
OBJECTIVE  min sum j of c[j]*r[j]
CONSTRAINTS
   resource means {
      r[j] = sum i of a[i][j][t[i]]*q[i] forall j
      relaxation = { lp, cp } }
   quant means {
      q[1] >= 1 => q[2] = 0
      relaxation = { lp, cp } }
   types means {
      t[1] = 1 => t[2] in {1, 2}
      t[3] = 1 => (t[4] in {1, 3} and t[5] in {1, 3, 4, 6}
                   and t[6] = 3)
      relaxation = { lp, cp } }
SEARCH
   type = { bb:bestdive }
   branching = { quant, t:most, q:least:triple             }
   inference = { q:redcost }
```

Example 2: Product Configuration: SIMPL Model

```
OBJECTIVE  min sum j of c[j]*r[j]
CONSTRAINTS
   resource means {
      r[j] = sum i of a[i][j][t[i]]*q[i] forall j
      relaxation = { lp, cp } }
   quant means {
      q[1] >= 1 => q[2] = 0
      relaxation = { lp, cp } }
   types means {
      t[1] = 1 => t[2] in {1, 2}
      t[3] = 1 => (t[4] in {1, 3} and t[5] in {1, 3, 4, 6}
                   and t[6] = 3)
      relaxation = { lp, cp } }
SEARCH
   type = { bb:bestdive }
   branching = { quant, t:most, q:least:triple, types:most }
   inference = { q:redcost }
```

Example 2: Product Configuration
Computational Results: Number of Search Nodes

Example 2: Product Configuration
Computational Results: Number of Search Nodes

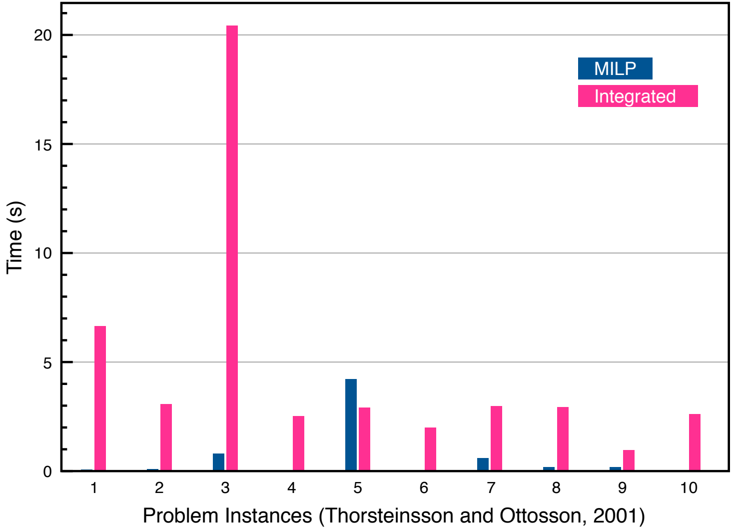


Example 2: Product Configuration

Computational Results: CPU Time (s)

Example 2: Product Configuration

Computational Results: CPU Time (s)



Example 3: Job Scheduling on Parallel Machines

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} = 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

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

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} = whether or not job j is assigned to machine i (binary)
- y_{jk} = whether or not j precedes k 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} = 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)

$$
\begin{aligned}
&\min \textstyle\sum_{ij} c_{ij} x_{ij} \\
&r_j \le t_j \le d_j - \textstyle\sum_i p_{ij} x_{ij},\ \forall\, j \\
&\textstyle\sum_i x_{ij} = 1,\ \forall\, j \\
&y_{jk} + y_{kj} \le 1,\ \forall\, k > j \\
&y_{jk} + y_{kj} \ge x_{ij} + x_{ik} - 1,\ \forall\, k > j,\ i \\
&y_{jk} + y_{kj} + x_{ij} + x_{i'k} \le 2,\ \forall\, k > j,\ i' \ne i \\
&t_k \ge t_j + \textstyle\sum_i p_{ij} x_{ij} - M(1 - y_{jk}),\ \forall\, k \ne j \\
&\textstyle\sum_j p_{ij} x_{ij} \le \max_j\{d_j\} - \min_j\{r_j\},\ \forall\, i
\end{aligned}
$$

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

Example 3: Job Scheduling: Integrated Benders

- x_{ij} = whether or not job j is assigned to machine i (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

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

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

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

$$\sum_i x_{ij} = 1, \ \forall\, j$$

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

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

$$\sum_i x_{ij} = 1, \ \forall \ j$$

$$(x_{ij} = 1) \Leftrightarrow (y_j = i), \ \forall \ i, j$$

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

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

$$\textstyle\sum_i x_{ij} = 1, \ \forall \ j$$

$$(x_{ij} = 1) \Leftrightarrow (y_j = i), \ \forall \ i, j$$

$$r_j \leq t_j \leq d_j - p_{y_j j}, \ \forall \ j$$

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

$$\min \textstyle\sum_{ij} c_{ij} x_{ij}$$
$$\textstyle\sum_i x_{ij} = 1,\ \forall\ j$$
$$(x_{ij} = 1) \Leftrightarrow (y_j = i),\ \forall\ i, j$$
$$r_j \le t_j \le d_j - p_{y_j j},\ \forall\ j$$
$$\text{cumulative}((t_j, p_{ij}, 1 \,|\, x_{ij} = 1), 1),\ \forall\ i$$

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

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

Need to tell the solver how to decompose the model

Example 3: Job Scheduling: SIMPL Model

Example 3: Job Scheduling: SIMPL Model

```
OBJECTIVE  min sum i,j of c[i][j]*x[i][j]
```

Example 3: Job Scheduling: SIMPL Model

```
OBJECTIVE  min sum i,j of c[i][j]*x[i][j]
CONSTRAINTS
```

Example 3: Job Scheduling: SIMPL Model

```
OBJECTIVE  min sum i,j of c[i][j]*x[i][j]
CONSTRAINTS
 assign means {
   sum i of x[i][j] = 1 forall j
   relaxation = { ip:master } }
```

Example 3: Job Scheduling: SIMPL Model

```
OBJECTIVE  min sum i,j of c[i][j]*x[i][j]
CONSTRAINTS
 assign means {
   sum i of x[i][j] = 1 forall j
   relaxation = { ip:master } }
 xy means {
   x[i][j] = 1 <=> y[j] = i forall i, j
   relaxation = { cp:sub } }
```

Example 3: Job Scheduling: SIMPL Model

```
OBJECTIVE  min sum i,j of c[i][j]*x[i][j]
CONSTRAINTS
 assign means {
   sum i of x[i][j] = 1 forall j
   relaxation = { ip:master } }
 xy means {
   x[i][j] = 1 <=> y[j] = i forall i, j
   relaxation = { cp:sub } }
 tbounds means {
   r[j] <= t[j] <= d[j] - p[y[j]][j] forall j
   relaxation = { ip:master, cp:sub } }
```

Example 3: Job Scheduling: SIMPL Model

```
OBJECTIVE  min sum i,j of c[i][j]*x[i][j]
CONSTRAINTS
 assign means {
   sum i of x[i][j] = 1 forall j
   relaxation = { ip:master } }
 xy means {
   x[i][j] = 1 <=> y[j] = i forall i, j
   relaxation = { cp:sub } }
 tbounds means {
   r[j] <= t[j] <= d[j] - p[y[j]][j] forall j
   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 } }
```

Example 3: Job Scheduling: SIMPL Model

```
OBJECTIVE  min sum i,j of c[i][j]*x[i][j]
CONSTRAINTS
 assign means {
   sum i of x[i][j] = 1 forall j
   relaxation = { ip:master } }
 xy means {
   x[i][j] = 1 <=> y[j] = i forall i, j
   relaxation = { cp:sub } }
 tbounds means {
   r[j] <= t[j] <= d[j] - p[y[j]][j] forall j
   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

Instances from Jain and Grossmann (2001)

Long Processing Times

		MILP		Integrated Benders		
Jobs	Machines	Nodes	Time (s)	Iterations	Cuts	Time (s)
3	2					
7	3					
12	3					
15	5					
20	5					
22	5					

Example 3: Job Scheduling on Parallel Machines
Computational Results

Instances from Jain and Grossmann (2001)

Long Processing Times

		MILP		Integrated Benders		
Jobs	Machines	Nodes	Time (s)	Iterations	Cuts	Time (s)
3	2	1	0.00			
7	3	1	0.02			
12	3	11060	16.50			
15	5	3674	14.30			
20	5	159400	3123.34			
22	5	$>$ 5.0M	$>$ 48h			

Example 3: Job Scheduling on Parallel Machines

Computational Results

Instances from Jain and Grossmann (2001)

Long Processing Times

		MILP		Integrated Benders		
Jobs	Machines	Nodes	Time (s)	Iterations	Cuts	Time (s)
3	2	1	0.00	2	1	0.00
7	3	1	0.02	12	14	0.09
12	3	11060	16.50	26	37	0.58
15	5	3674	14.30	22	31	0.96
20	5	159400	3123.34	30	52	3.21
22	5	$>$ 5.0M	$>$ 48h	38	59	6.70

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

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

Short Processing Times

		MILP		Integrated Benders		
Jobs	Machines	Nodes	Time (s)	Iterations	Cuts	Time (s)
3	2					
7	3					
12	3					
15	5					
20	5					
22	5					
25	5					

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

Short Processing Times

		MILP		Integrated Benders		
Jobs	Machines	Nodes	Time (s)	Iterations	Cuts	Time (s)
3	2	1	0.00			
7	3	1	0.01			
12	3	4950	1.98			
15	5	14000	19.80			
20	5	140	5.73			
22	5	$>$ 16.9M	$>$ 48h			
25	5	$>$ 4.5M	$>$ 48h			

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

Short Processing Times

		MILP		Integrated Benders		
Jobs	Machines	Nodes	Time (s)	Iterations	Cuts	Time (s)
3	2	1	0.00	1	0	0.00
7	3	1	0.01	1	0	0.01
12	3	4950	1.98	1	0	0.01
15	5	14000	19.80	1	0	0.03
20	5	140	5.73	3	3	0.12
22	5	$>$ 16.9M	$>$ 48h	5	4	0.38
25	5	$>$ 4.5M	$>$ 48h	16	22	0.86

Future Work

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)

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)

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 (continued)

Future Work (continued)

- Regarding SIMPL's availability...

Future Work (continued)

- Regarding SIMPL's availability...
  - Distribute source code?

Future Work (continued)

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

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...

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...

Conclusion

Conclusion

- Many theoretical and technological breakthroughs over the last few decades have helped OR become more accessible and popular

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

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

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
  - a very useful research tool

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
  - 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

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

Constraint Programming Example

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

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

Constraint Programming Example

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

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

Constraint Programming Example

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

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

Constraint Programming Example

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

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

Constraint Programming Example

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

$\text{element}(z, [1, 3, 5], x)$ (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

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

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

Constraint Programming Example

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

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

Constraint Programming Example

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

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

Constraint Programming Example

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

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