MIP Modeling with Metaconstraints and Semantic Typing

André Ciré, John Hooker
Carnegie Mellon University

Tallys Yunes
University of Miami

MIP Workshop, July 2013

Two Perspectives on Optimization

- Reduce problems to **standard form** using atomistic constraints
  - Lose structure, but use highly engineered solvers
  - Current orthodoxy for MIP, SAT communities
- Solver must rely on specific **problem structure**
  - Use special purpose solver, or…
  - Convey structure to general solver with **global constraints**.
  - Current practice for CP solvers

In reality…

- You can’t solve NP-hard problems without exploiting special structure.

In reality…

- You can't solve NP-hard problems without exploiting special structure.
- For SAT solvers:
  - Careful reduction of problem to SAT form
  - This has become a minor industry

In reality…

- You can't solve NP-hard problems without exploiting special structure.
- For SAT solvers:
    - Careful reduction of problem to SAT form
    - This has become a minor industry
- For MIP solvers:
    - Careful choice of variables for tight formulation
    - Addition of valid inequalities
    - SOS1, SOS2, symmetry-breaking constraints, etc.
    - Solver parameters (e.g., which cuts?)

How to take advantage of structure

- Use a general solver that exploits structure directly
  - Such as a CP or integrated solver
  - ILOG CP Optimizer, CHIP, Gecode, Google CP Solver, ECLiPSe, G12, SIMPL, Xpress-Mosel
- Convey structure to MIP (or SAT) solver
  - Formulate problem with global constraints or **metaconstraints** to reveal structure
  - Automatically convert these to optimal MIP formulation

Conveying structure to MIP

- Given the advanced state of MIP technology…
  - Perhaps a logical next step is to convey problem structure to the solver.

Conveying structure to MIP

- Given the advanced state of MIP technology…
  - Perhaps a logical next step is to convey problem structure to the solver.
- Advantages of using metaconstraints
  - Better MIP formulation, tighter LP relaxation
  - Opportunity to enhance solver with **domain filtering**, **constraint propagation**.

Conveying structure to MIP

- Given the advanced state of MIP solvers…
  - Perhaps a logical next step is to convey problem structure to the solver.
- Advantages of using metaconstraints
  - Better MIP formulation, tighter LP relaxation
  - Opportunity to enhance solver with **domain filtering**, **constraint propagation**.
- However, metaconstraints pose a fundamental problem of variable management…

Variable management problem

- MIP formulation typically introduces **new variables**
  - Different metaconstraints may introduce variables that are functionally **the same variable**
  - …or related in some other way.
  - Recognizing these relationships is essential to obtaining a good model (in particular, a tight relaxation)
  - How can the solver "understand" what is going on in the model?

Variable management problem

- MIP formulation typically introduces **new variables**
  - Different metaconstraints may introduce variables that are functionally **the same variable**
  - …or related in some other way.
  - Recognizing these relationships is essential to obtaining a good model (in particular, a tight relaxation)
  - How can the solver “understand” what is going on in the model?
- Proposal: Model with **semantic typing of variables**.

Metaconstraints and semantic typing

- **Metaconstraints** convey problem structure
  - Each represents a structured collection of more elementary constraints.
- **Semantic typing** assigns a meaning to each variable.
  - Allows solver to deduce relationships among variables.
  - Good modeling practice in general.

How variables are introduced

- The MIP solver may reformulate a constraint containing **general integer variable** x_i in terms of 0-1 variables y_{ij} , where

$$x_i = \sum_j j y_{ij}$$

  - y_{ij} s may be **equivalent to other variables that appear** in the model or MIP formulations of other constraints.

How variables are introduced

- The solver may reformulate a **disjunction of linear systems**

$$\bigcup_k A_k x \geq b^k$$

using a convex hull (or big-*M*) formulation:

$$\begin{aligned} &A_k x^k \geq b^k y_k, \quad \text{all } k \\ &x = \sum_k x^k, \quad \sum_k y_k = 1 \\ &y_k \in \{0,1\}, \quad \text{all } k \end{aligned}$$

- Other constraints may be based on **same set of alternatives**, and corresponding auxiliary variables (y_k etc.) should be equated.

How variables are introduced

- A nonlinear or global solver may use **McCormick factorization** to replace nonlinear subexpressions with auxiliary variables
  - … to obtain a linear relaxation.

How variables are introduced

- A nonlinear or global solver may use **McCormick factorization** to replace nonlinear subexpressions with auxiliary variables
  - … to obtain a linear relaxation.
  - For example, bilinear term *xy* can be linearized by replacing it with new variable *z* and constraints

$$L_y x + L_x y - L_x L_y \le z \le L_y x + U_x y - L_x U_y$$
$$U_y x + U_x y - U_x U_y \le z \le U_y x + L_x y - U_x L_y$$

where $x \in [L_x, U_x],\ \ y \in [L_y, U_y]$

  - Factorization of different constraints may create variables for identical subexpressions.
  - They should be identified to get a tight relaxation.

How variables are introduced

- The MIP solver may reformulate **global constraints** from CP by introducing variables that have the same meaning.

How variables are introduced

- The MIP solver may reformulate **global constraints** from CP by introducing variables that have the same meaning.
  - For example, **sequence** constraint limits how many jobs of a given type can occur in given time interval:

$$\text{sequence}(x), \;\; x_i = \text{ job in position } i$$

and **cardinality** constraint limits how many times a given job appears

$$\text{cardinality}(x), \;\; x_j = \text{ job in position } j$$

Both may introduce variables

$$y_{ij} = 1 \text{ when job } j \text{ occurs in position } i$$

that should be identified.

How variables are introduced

- Popular global constraints include:

all-different	lex greater
among	nvalues
cardinality	path
circuit	range
clique	regular
cumulative	roots
cutset	same
cycle	sort
diffn	spread
element	sum
flow	symmetric alldiff

How variables are introduced

- The solver may introduce equivalent variables while interpreting metaconstraints designed for **classical MIP modeling situations:**
  - Fixed-charge network flow
  - Facility location
  - Lot sizing
  - Job shop scheduling
  - Assignment (3-dim, quadratic, etc.)
  - Piecewise linear

How variables are introduced

- A model may include **two formulations** of the problem that use related variables.
    - Common in CP, because it strengthens **propagation**.

How variables are introduced

- A model may include **two formulations** of the problem that use related variables.
  - Common in CP, because it strengthens **propagation**.
  - For example,

$$x_i = \text{ job assigned to worker } i$$

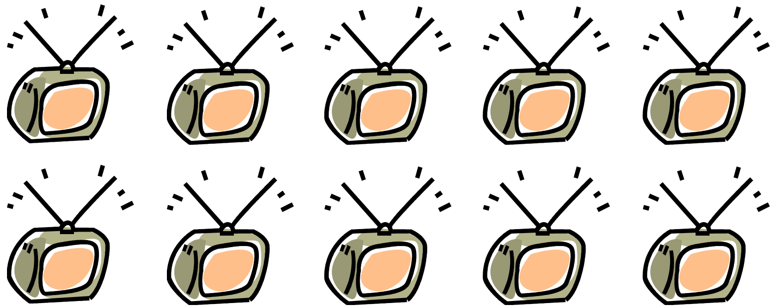
$$y_j = \text{ worker assigned to job } j$$

  - Solver should generate **channeling constraints** to relate the variables to each other:

$$j = x_{y_j}, \qquad i = y_{x_i}$$

Motivating example

- Allocate 10 advertising spots to 5 products



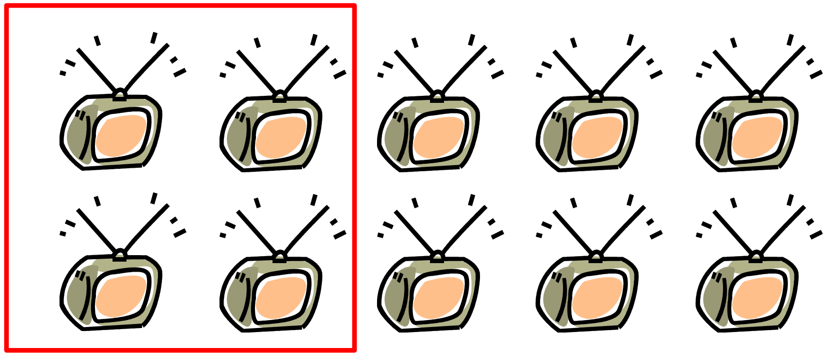
$x_i =$ how many spots allocated to product *i*

$y_{ij} =$ 1 if *j* spots allocated to product *i*

A B C D E

Motivating example

- Allocate 10 advertising spots to 5 products



$\leq$ 4 spots per product

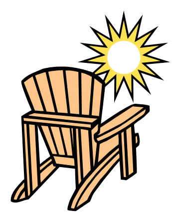
$x_i =$ how many spots allocated to product i

$y_{ij} =$ 1 if j spots allocated to product i

A

B

C

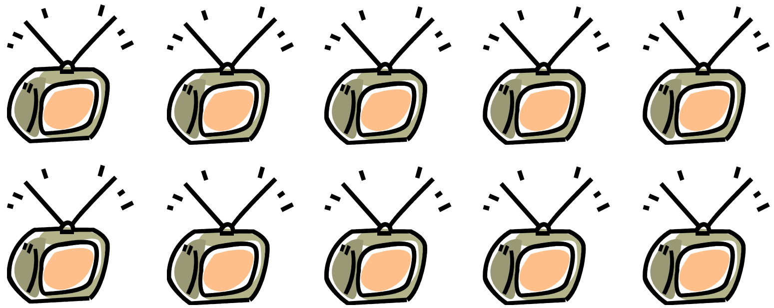


D

E

Motivating example

- Allocate 10 advertising spots to 5 products



$\leq$ 4 spots per product

Advertise $\leq$ 3 products

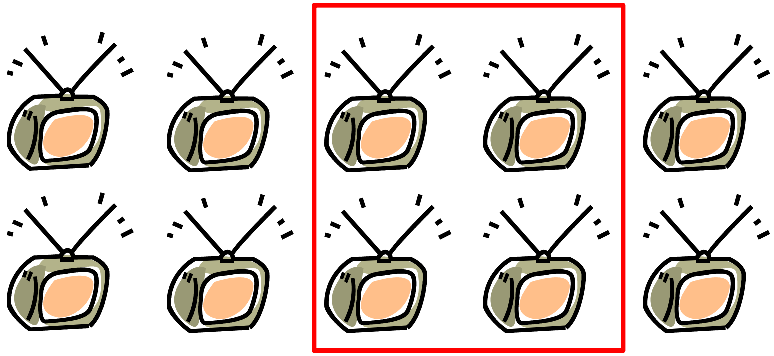
x_i = how many spots allocated to product i

y_{ij} = 1 if j spots allocated to product i

A B C D E

Motivating example

- Allocate 10 advertising spots to 5 products



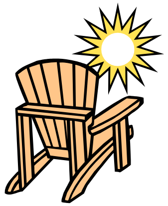
x_i = how many spots allocated to product i

y_{ij} = 1 if j spots allocated to product i

A

B

C



D

E

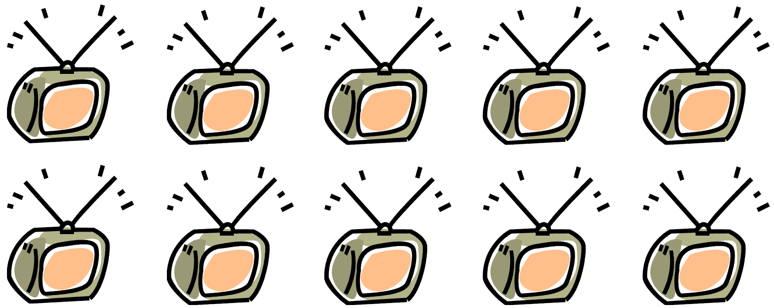
$\leq$ 4 spots per product

Advertise $\leq$ 3 products

$\geq$ 4 spots for at least one product

Motivating example

- Allocate 10 advertising spots to 5 products



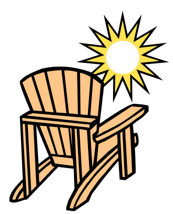
x_i = how many spots allocated to product i

y_{ij} = 1 if j spots allocated to product i

A

B

C



D

E

$\leq$ 4 spots per product

Advertise $\leq$ 3 products

$\geq$ 4 spots for at least one product

P_{ij} = profit from allocating j spots to product i

Objective: maximize profit

Motivating example

```
spots in {0..4}
product in {A,B,C,D,E})
```

Index sets

Motivating example

```
spots in {0..4}
product in {A,B,C,D,E}
data P{product,spots}
```

Data input

Motivating example

```
spots in {0..4}
product in {A,B,C,D,E}
data P{product,spots}
x[i] is howmany spots allocate(product i)
```

Declaration of variable x_i

Motivating example

```
spots in {0..4}
product in {A,B,C,D,E}
data P{product,spots}
x[i] is howmany spots allocate(product i)
```

Declaration of variable x_i

This makes it a variable declaration

$\wedge$

Motivating example

```
spots in {0..4}
product in {A,B,C,D,E}
data P{product,spots}
x[i] is howmany spots allocate(product i)
```

Declaration of variable x_i

This is the semantic type

Motivating example

```
spots in {0..4}
product in {A,B,C,D,E}
data P{product,spots}
x[i] is howmany spots allocate(product i)
```

Declaration of variable x_i

Indicates an integer quantity

Other keywords:
howmuch
whether

^

Motivating example

```
spots in {0..4}
product in {A,B,C,D,E}
data P{product,spots}
x[i] is howmany spots allocate(product i)
```

Declaration of variable x_i

How many of what?

^

Motivating example

```
spots in {0..4}
product in {A,B,C,D,E}
data P{product,spots}
x[i] is howmany spots allocate(product i)
```

Declaration of variable x_i

Predicate associated with variable *x*

Every variable is associated with a predicate that gives it meaning

Motivating example

```
spots in {0..4}
product in {A,B,C,D,E}
data P{product,spots}
x[i] is howmany spots allocate(product i)
```

Declaration of variable x_i

Other term of the predicate

Motivating example

```
spots in {0..4}
product in {A,B,C,D,E}
data P{product,spots}
x[i] is howmany spots allocate(product i)
```

Declaration of variable x_i

Associates index of **x[i]** with index set **product**

Motivating example

$$\max \sum_i P_{ix_i}$$

```
spots in {0..4}
product in {A,B,C,D,E}
data P{product,spots}
x[i] is howmany spots allocate(product i)
maximize sum{product i} P[i,x[i]]
```

Objective function

Motivating example

$$\max \sum_i P_{ix_i}$$

$$\sum_i x_i \leq 10$$

```
spots in {0..4}
product in {A,B,C,D,E}
data P{product,spots}
x[i] is howmany spots allocate(product i)
maximize sum{product i} P[i,x[i]]
sum{product i} x[i] <= 10
```

10 spots available

Motivating example

$$\max \sum_i P_{ix_i}$$

$$\sum_i x_i \leq 10$$

```
spots in {0..4}
product in {A,B,C,D,E}
data P{product,spots}
x[i] is howmany spots allocate(product i)
maximize sum{product i} P[I,x[i]]
sum{product i} x[i] <= 10
y[i,j] is whether allocate(product i, spots j)
```

Declare y_{ij}

Indicates 0-1 variable

Motivating example

$$\max \sum_i P_{ix_i}$$

$$\sum_i x_i \leq 10$$

```
spots in {0..4}
product in {A,B,C,D,E}
data P{product,spots}
x[i] is howmany spots allocate(product i)
maximize sum{product i} P[i,x[i]]
sum{product i} x[i] <= 10
y[i,j] is whether allocate(product i, spots j)
```

Declare y_{ij}

Associated with same predicate as **x[i]**

Motivating example

$$\max \sum_i P_{ix_i}$$

$$\sum_i x_i \leq 10, \ \sum_i y_{i0} \geq 2$$

```
spots in {0..4}
product in {A,B,C,D,E}
data P{product,spots}
x[i] is howmany spots allocate(product i)
maximize sum{product i} P[i,x[i]]
sum{product i} x[i] <= 10
y[i,j] is whether allocate(product i, spots j)
sum{product i} y[i,0] >= 2
```

At most 3 products advertised

Motivating example

$$\max \sum_i P_{ix_i}$$

$$\sum_i x_i \le 10, \; \sum_i y_{i0} \ge 2, \; \sum_i y_{i4} \ge 1$$

```
spots in {0..4}
product in {A,B,C,D,E}
data P{product,spots}
x[i] is howmany spots allocate(product i)
maximize sum{product i} P[i,x[i]]
sum{product i} x[i] <= 10
y[i,j] is whether allocate(product i, spots j)
sum{product i} y[i,0] >= 2
sum{product i} y[i,4] >= 1
```

At least 1 product gets ≥4 spots

Motivating example

$$\max \sum_i P_{ix_i}$$

$$\sum_i x_i \le 10,\ \sum_i y_{i0} \ge 2,\ \sum_i y_{i4} \ge 1$$

$$\sum_j y_{ij} = 1,\ x_i = \sum_j j y_{ij},\ \text{all } i$$

```
spots in {0..4}
product in {A,B,C,D,E}
data P{product,spots}
x[i] is howmany spots allocate(product i)
maximize sum{product i} P[i,x[i]]
sum{product i} x[i] <= 10
y[i,j] is whether allocate(product i, spots j)
sum{product i} y[i,0] >= 2
sum{product i} y[i,4] >= 1
{product i} sum{spots j} y[i,j] = 1
{product i} x[i] = sum{spots j} j*y[i,j]
```

Solver generates linking constraints because
x[i] and **y[i,j]** are associated with the same predicate.

Motivating example

$$\max \sum_i P_{ix_i}$$

$$\sum_i x_i \le 10, \ \sum_i y_{i0} \ge 2, \ \sum_i y_{i4} \ge 1$$

$$\sum_j y_{ij} = 1, \ x_i = \sum_j j y_{ij}, \text{ all } i$$

```
spots in {0..4}
product in {A,B,C,D,E}
data P{product,spots}
x[i] is howmany spots allocate (product i)
maximize sum{product i} P[i,x[i]]
```

This constraint must be linearized. Solver generates

$$z_i = \sum_{j=0}^{4} P_{ij} y'_{ij}, \ \sum_{j=0}^{4} y'_{ij} = 1, \ x_i = \sum_{j=0}^{4} j y'_{ij}, \text{ all } i$$

```
y'[i,j] is whether allocate(product i, spots j)
```

Motivating example

$$\max \sum_i P_{ix_i}$$

$$\sum_i x_i \le 10,\ \sum_i y_{i0} \ge 2,\ \sum_i y_{i4} \ge 1$$

$$\sum_j y_{ij} = 1,\ x_i = \sum_j j y_{ij},\ \text{all } i$$

```
spots in {0..4}
product in {A,B,C,D,E}
data P{product,spots}
x[i] is howmany spots allocate (product i)
maximize sum{product i} P[i,x[i]]
```

This constraint must be linearized. Solver generates

$$z_i = \sum_{j=0}^{4} P_{ij} y'_{ij},\ \sum_{j=0}^{4} y'_{ij} = 1,\ x_i = \sum_{j=0}^{4} j y'_{ij},\ \text{all } i$$

```
y'[i,j] is whether allocate(product i, spots j)
```

y and **y′** are identified because they have the same type:

```
y[i,j] is whether allocate(product i, spots j)
```

Predicates and relations

Predicate **`allocate`** denotes 2-place **relation** (set of tuples).
Schematically indicated by:

1	2
i	x_i
	`howmany`
`product`	**`spots`**

Predicates and relations

Predicate **`allocate`** denotes 2-place **relation** (set of tuples). Schematically indicated by:

1	2
i	x_i
	`howmany`
`product`	**`spots`**

Column corresponding to a variable must be a **function** of other columns.

Predicates and relations

Predicate **`allocate`** denotes 2-place **relation** (set of tuples).
Schematically indicated by:

1	2
i	x_i
	`howmany`
`product`	**`spots`**

Declaration of **`y[i,j]`** as

`whether` **`allocate (product i, spots j)`**

creates the 3-place relation

1	2	3
i	x_i	y_{ij}
	`howmany`	***`whether`***
`product`	**`spots`**	***`binary`***

Predicates and relations

Relation table reveals channeling constraints. For example,

```
x[i] is which job assign(worker i)
y[j] is which worker assign (job i)
```

1	2
j, x_i	i, y_j
which	***which***
job	**worker**

We can read off the channeling constraints

$$j = x_i = x_{y_i}$$

$$i = y_j = y_{x_i}$$

Previous work

- **Model management** uses semantic typing to help combine models and use inheritance.
  - Originally inspired by object-oriented programming
    Bradley & Clemence (1988)
  - *Quiddity:* a rigorous attempt to analyze conditions for variable identification
    Bhargava, Kimbrough & Krishnan (1991)
  - **SML** uses typing in a structured modeling framework
    Geoffrion (1992)
  - **Ascend** uses strongly-typed, object-oriented modeling
    Bhargava, Krishnan & Piela (1998)

Previous work

- Our semantic typing differs:
  - **Less ambitious** because it doesn't attempt model management.
    - There is only one model.
  - **More ambitious** because we recognize relationships other than equivalence.
  - We manage variables **introduced by solver**.

Previous work

- Modeling systems that convey some structure to solver:
  - All CP modelers (OPL, CHIP, etc.) use **global constraints**.
  - AIMMS uses **typed index sets**.
  - OPL, Xpress-Kalis, Comet, etc., use **interval variables**.
  - SAT solver SymChaff uses high-level **AI planning language** PDDL.
  - Lopes and Fourer (2009) use **UML** (Unified Modeling Language) to model multistage stochastic LPs with recourse.
  - SIMPL has **full metaconstraint capability**.

Previous work

- However, **none of these systems** deals systematically with the variable management problem.
    - We address it with semantic typing of variables.

Assignment problem

$$\min \sum_i c_{i x_i}$$

$$\text{alldiff}\left(x_1, \ldots, x_n\right)$$

```
worker in {1..m}
job in {1..n}
data C{worker,job}
x[i] is which job assign(worker i)
minimize sum{worker i} C[i,x[i]]
alldiff{x[*]}
```

Assignment problem

$$\min \sum_i c_{i x_i}$$

$$\text{alldiff}\left(x_1, \ldots, x_n\right)$$

```
worker in {1..m}
job in {1..n}
data C{worker,job}
x[i] is which job assign(worker i)
minimize sum{worker i} C[i,x[i]]
alldiff{x[*]}
```

Objective function is formulated

$$\max \sum_i c_{ij} y_{ij}, \;\; x_i = \sum_j y_{ij}, \text{ all } i$$

```
y[i,j] is whether assign(worker i, job j)
```

Assignment problem

$$\min \sum_i c_{ix_i}$$

$$\text{alldiff}\left(x_1,\ldots,x_n\right)$$

```
worker in {1..m}
job in {1..n}
data C{worker,job}
x[i] is which job assign(worker i)
minimize sum{worker i} C[i,x[i]]
alldiff{x[*]}
```

Objective function is formulated

$$\max \sum_i c_{ij} y_{ij}, \;\; x_i = \sum_j y_{ij}, \text{ all } i$$

```
y[i,j] is whether assign(worker i, job j)
```

Alldiff is formulated

$$\sum_j y'_{ij} = 1, \text{ all } i, \;\; \sum_i y'_{ij} = 1, \text{ all } j, \;\; x_i = \sum_j j y'_{ij}, \text{ all } i$$

```
y'[i,j] is whether assign(worker i, job j)
```

Solver identifies *y* and *y*′ to create classical AP.

Latin squares

j

i

2	3	1
3	1	2
1	2	3

Numbers in every row and column are distinct.
We will use **three** formulations to improve propagation.

Latin squares

j

i

2	3	1
3	1	2
1	2	3

$$\text{alldiff}\left(x_{i1},\ldots x_{in}\right), \text{ all } i$$

$$\text{alldiff}\left(x_{1j},\ldots x_{nj}\right), \text{ all } j$$

$$\text{alldiff}\left(y_{i1},\ldots y_{in}\right), \text{ all } i$$

$$\text{alldiff}\left(y_{1k}\ldots y_{nk}\right), \text{ all } k$$

$$\text{alldiff}\left(z_{j1},\ldots x_{jn}\right), \text{ all } j$$

$$\text{alldiff}\left(z_{1k},\ldots x_{nk}\right), \text{ all } k$$

Numbers in every row and column are distinct.
We will use **three** formulations to improve propagation.

```
row, col, num in {1..n}
x[i,j] is which num assign(row i, col j)
y[i,k] is which col assign(row i, num k)
z[j,k] is which row assign(col j, num k)
```

$$\text{alldiff}\left(x_{i1},\ldots x_{in}\right),\ \text{all } i$$

$$\text{alldiff}\left(x_{1j},\ldots x_{nj}\right),\ \text{all } j$$

$$\text{alldiff}\left(y_{i1},\ldots y_{in}\right),\ \text{all } i$$

$$\text{alldiff}\left(y_{1k}\ldots y_{nk}\right),\ \text{all } k$$

$$\text{alldiff}\left(z_{j1},\ldots x_{jn}\right),\ \text{all } j$$

$$\text{alldiff}\left(z_{1k},\ldots x_{nk}\right),\ \text{all } k$$

Latin squares

		j	
	2	3	1
i	3	1	2
	1	2	3

Numbers in every row and column are distinct.
We will use **three** formulations to improve propagation.

```
row, col, num in {1..n}
x[i,j] is which num assign(row i, col j)
y[i,k] is which col assign(row i, num k)
z[j,k] is which row assign(col j, num k)
{row i} alldiff{x[i,*]); {col j} alldiff{x[*,j])
{row i} alldiff{y[i,*]); {num k} alldiff{y[*,j])
{col j} alldiff{z[j,*]); {num k} alldiff{z[*,k])
```

Latin squares

The predicate **`assign`** denotes the 3-place relation

1	2	3
k, x_{ij}	j, y_{ik}	i, z_{jk}
which	***which***	***which***
num	**col**	**row**

```
row, col, num in {1..n}
x[i,j] is which num assign(row i, col j)
y[i,k] is which col assign(row i, num k)
z[j,k] is which row assign(col j, num k)
{row i} alldiff{x[i,*]); {col j} alldiff{x[*,j])
{row i} alldiff{y[i,*]); {num k} alldiff{y[*,j])
{col j} alldiff{z[j,*]); {num k} alldiff{z[*,k])
```

Latin squares

The predicate **`assign`** denotes the 3-place relation

1	2	3
k, x_{ij}	j, y_{ik}	i, z_{jk}
`which`	***`which`***	***`which`***
`num`	**`col`**	**`row`**

We can read off the channeling constraints:

$$k = x_{z_{jk} y_{ik}}, \quad j = y_{z_{jk} x_{ij}}, \quad i = z_{y_{ik} x_{ikj}}, \quad \text{all } i, j, k$$

which can be propagated.

Latin squares

```
{row i} alldiff{x[i,*]); {col j} alldiff{x[*,j])
{row i} alldiff{y[i,*]); {num k} alldiff{y[*,j])
{col j} alldiff{z[j,*]); {num k} alldiff{z[*,k])
```

The 3 formulations generate 3 identical MIP models:

$$x_{ij} = \sum_k k\delta_{ijk}^x;\ \ \sum_k \delta_{ijk}^x = 1,\ \text{all } i,j;\ \ \sum_j \delta_{ijk}^x = 1,\ \text{all } i,k;\ \ \sum_i \delta_{ijk}^x = 1,\ \text{all } j,k$$

$$y_{ik} = \sum_j j\delta_{ijk}^y,\ \ \sum_j \delta_{ijk}^y = 1,\ \text{all } i,k;\ \ \sum_k \delta_{ijk}^y = 1,\ \text{all } i,j;\ \ \sum_i \delta_{ijk}^y = 1,\ \text{all } j,k$$

$$z_{jk} = \sum_i i\delta_{ijk}^z,\ \ \sum_i \delta_{ijk}^z = 1,\ \text{all } j,k;\ \ \sum_k \delta_{ijk}^z = 1,\ \text{all } i,j;\ \ \sum_j \delta_{ijk}^z = 1,\ \text{all } i,k$$

Latin squares

```
{row i} alldiff{x[i,*]); {col j} alldiff{x[*,j])
{row i} alldiff{y[i,*]); {num k} alldiff{y[*,j])
{col j} alldiff{z[j,*]); {num k} alldiff{z[*,k])
```

The 3 formulations generate 3 identical MIP models:

$$x_{ij} = \sum_k k\delta^x_{ijk}; \ \sum_k \delta^x_{ijk} = 1, \text{ all } i,j; \ \sum_j \delta^x_{ijk} = 1, \text{ all } i,k; \ \sum_i \delta^x_{ijk} = 1, \text{ all } j,k$$

$$y_{ik} = \sum_j j\delta^y_{ijk}, \ \sum_j \delta^y_{ijk} = 1, \text{ all } i,k; \ \sum_k \delta^y_{ijk} = 1, \text{ all } i,j; \ \sum_i \delta^y_{ijk} = 1, \text{ all } j,k$$

$$z_{jk} = \sum_i i\delta^z_{ijk}, \ \sum_i \delta^z_{ijk} = 1, \text{ all } j,k; \ \sum_k \delta^z_{ijk} = 1, \text{ all } i,j; \ \sum_j \delta^z_{ijk} = 1, \text{ all } i,k$$

The solver declares $\delta^x_{ijk}, \ \delta^y_{ijk}, \ \delta^z_{ijk}$

```
whether assign(row i, col j, num k)
```

So it treats them as the same variable and generates only 1 MIP model.

Relating `which` variables

In general, an *n*-place predicate that denotes the relation

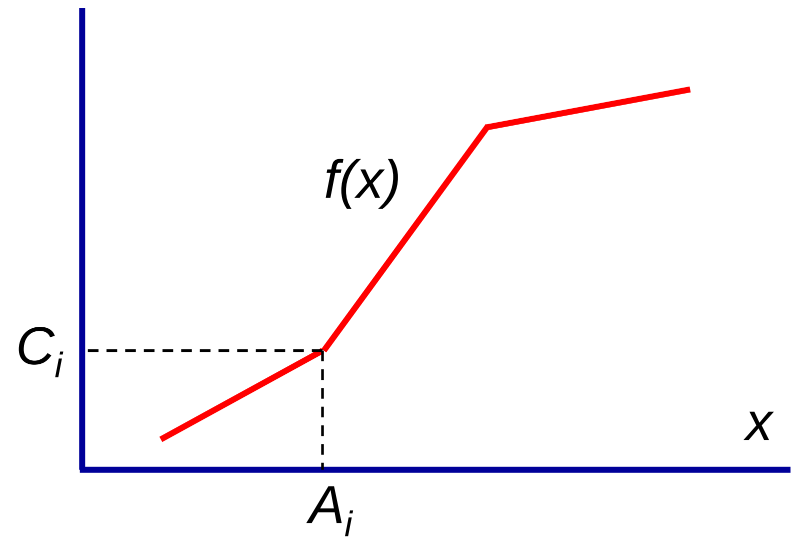
1	…	*k*	*k* + 1	…	*n*
$i_1,\ x^1_{i(1)}$	$\cdots$	$i_k,\ x^k_{i(k)}$	i_{k+1}	$\cdots$	i_n
`which`	…	**`which`**			
`entity`$_1$	…	**`entity`**$_k$	**`entity`**$_{k+1}$	…	**`entity`**$_n$

where $i(j) = i_1 \cdots i_{j-1} i_{j+1} \cdots i_n$

generates the channeling constraints

$$i_j = x^j_{x^1_{i(1)} \cdots\, x^{j-1}_{i(j-1)}\, x^{j+1}_{i(j+1)} \cdots\, x^k_{i(k)}\, i_{k+1} \cdots\, i_n},\ \text{all } i_1,\ldots,i_n,\ j = 1,\ldots,k$$

Piecewise linear

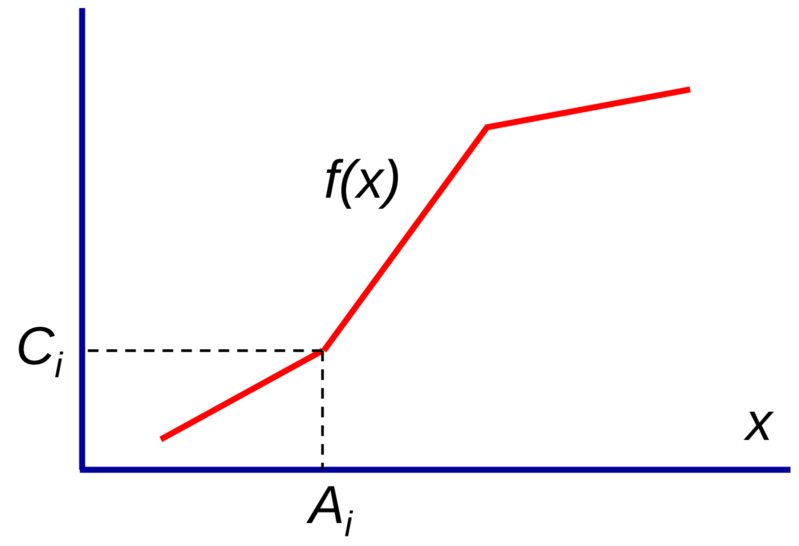


Piecewise linear function $z = f(x)$
Breakpoints in *A*, ordinates in *C*

```
x is howmuch output
index in {1..n}
data A,C{index i}
z is howmuch cost
piecewise(x,z,A,C)
```

this metaconstraint defines $z = f(x)$

Piecewise linear



Piecewise linear function $z = f(x)$
Breakpoints in *A*, ordinates in *C*

```
x is howmuch output
index in {1..n}
data A,C{index i}
z is howmuch cost
piecewise(x,z,A,C)
```

Solver generates the model

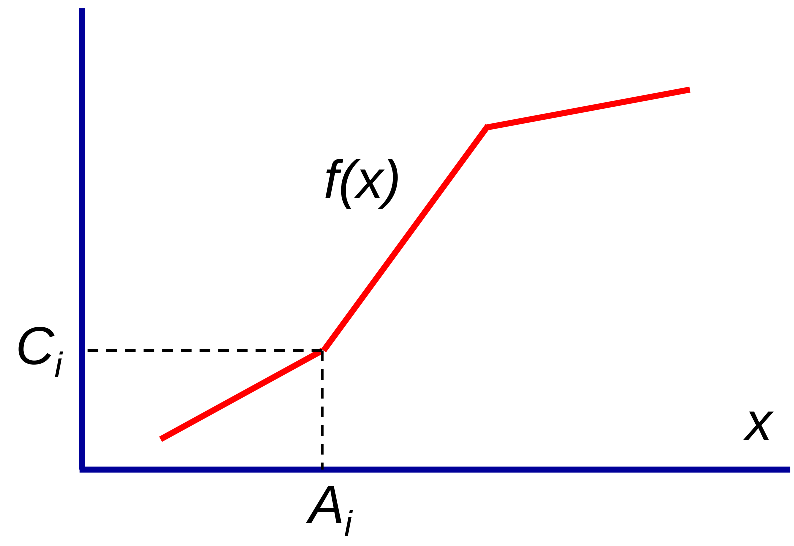
$$x = a_1 + \sum_{i=1}^{n-1} x_i, \quad z = c_1 + \sum_{i=1}^{n-1} \frac{c_{i+1} - c_i}{a_{i+1} - a_i} x_i$$

$$(a_{i+1} - a_i)\delta_{i+1} \le x_i \le (a_{i+1} - a_i)\delta_i, \quad \delta_i \in \{0,1\}, \; i = 1, \ldots, n-1$$

We need to declare auxiliary variables δ_i, x_i

Piecewise linear

Piecewise linear function $z = f(x)$
Breakpoints in A, ordinates in C



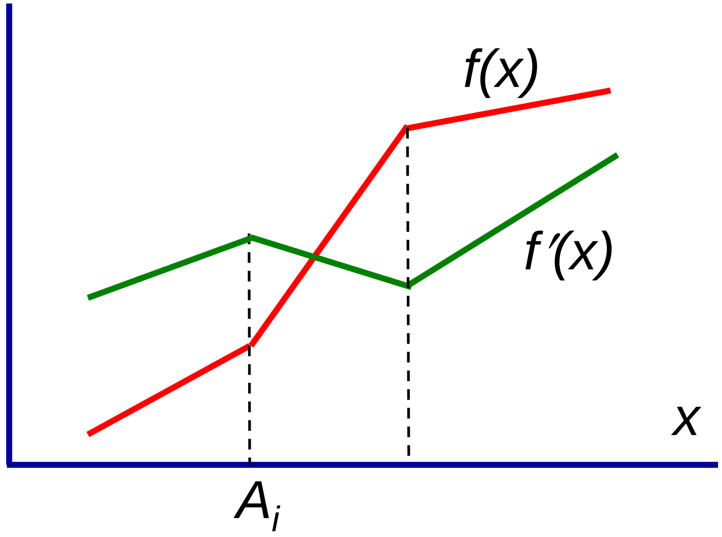

```
x is howmuch output
index in {1..n}
data A,C{index i}
z is howmuch cost
piecewise(x,z,A,C)
```

piecewise constraint induces solver to declare a new index set that associates **index** with **A,** and use it to declare δ_i, x_i

```
indexA in {1..n}
delta[i] is whether output(indexA i)
x[i] is howmuch output(indexA i)
```

Both declarations create predicates inherited from **output**

Piecewise linear



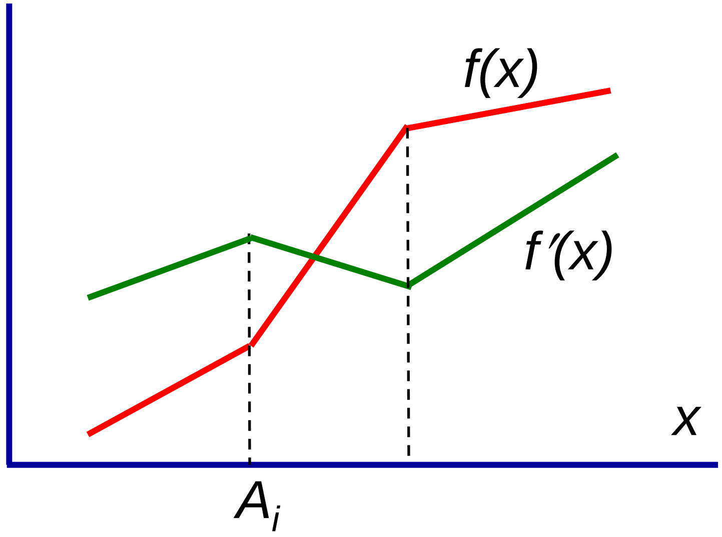
Suppose there is another piecewise function on the same break points

```
x is howmuch output
index in {1..n}
data A,C{index i}
z is howmuch cost
piecewise(x,z,A,C)
data C′{index i}
z′ is howmuch profit
piecewise(x,z′,A,C′)
indexA in {1..n}
delta[i] is whether output(indexA i)
x[i] is howmuch output(indexA i)
```

Piecewise linear

Suppose there is another piecewise function on the same break points

```
x is howmuch output
index in {1..n}
data A,C{index i}
z is howmuch cost
piecewise(x,z,A,C)
data C′{index i}
z′ is howmuch profit
piecewise(x,z′,A,C′)
indexA in {1..n}
delta[i] is whether output(indexA i)
x[i] is howmuch output(indexA i)
```



Because new piecewise constraint is associated with the same *x* and *A*, solver again creates **indexA**.

MIP model creates variables δ_i' and x_i' with same types as δ_i and x_i and so identifies them.

Interval variables

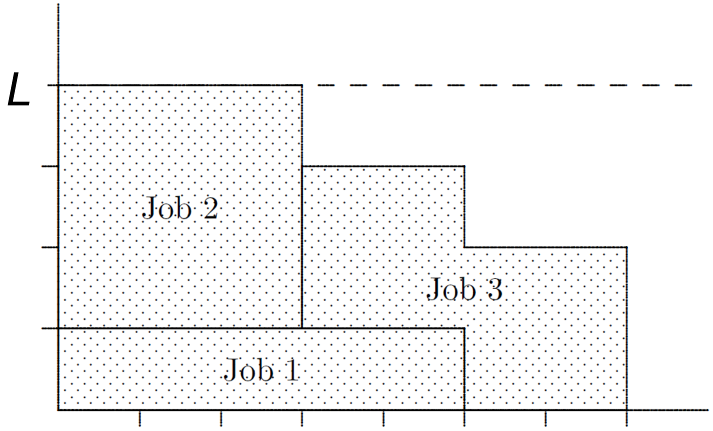
$$\text{cumulative}(x, D, R, L)$$

$$x_j \subseteq W_j, \text{ all } j$$

Each job *j* runs for a time interval x_j.
We wish to schedule jobs so that total resource consumption never exceeds *L*.

```
job in {1..n}
time in {t..T}
data W{job}, D{job}, R{job}
x[j] is interval running(job j) in W[j]
cumulative(x,D,R,L)
```

window, duration, resource



Interval variables

$$\text{cumulative}(x, D, R, L)$$

$$x_j \subseteq W_j, \text{ all } j$$

Each job *j* runs for a time interval x_j.
We wish to schedule jobs so that total resource consumption never exceeds *L*.

```
job in {1..n}
time in {t..T}
data W{job}, D{job}, R{job}
x[j] is interval running(job j) in W[j]
cumulative(x,D,R,L)
```

window, duration, resource

Solver generates the MIP model

$$\sum_t \delta_{jt} = 1, \text{ all } j; \quad \sum_j R_j \phi_{jt} \le L, \text{ all } t$$

$$\varphi_{jt} \ge \delta_{jt'}, \text{ all } t, t' \text{ with } 0 \le t - t' < D_j, \text{ all } j$$

```
delta[j,t] is whether running.start(job j, time t)
phi[j,t] is whether running(job j, time t)
```

Interval variables

$$\text{cumulative}(x, D, R, L)$$

$$x_j \subseteq W_j, \text{ all } j$$

$$\left| x_j^{\text{end}} - x_k^{\text{end}} \right| \geq T_0, \text{ all } j,k$$

Suppose we want finish times to be separated by at least T_0

```
job in {1..n}
time in {t..T}
data W{job}, D{job}, R{job}
x[j] is interval running(job j) in W[j]
cumulative(x,D,R,L)
{job j, job k} |x[j].end - x[k].end| >= T0
delta[j,t] is whether running.start(job j, time t)
phi[j,t] is whether running(job j, time t)
```

Interval variables

$$\text{cumulative}(x, D, R, L)$$

$$x_j \subseteq W_j, \text{ all } j$$

$$\left| x_j^{\text{end}} - x_k^{\text{end}} \right| \geq T_0, \text{ all } j, k$$

Suppose we want finish times to be separated by at least T_0

```
job in {1..n}
time in {t..T}
data W{job}, D{job}, R{job}
x[j] is interval running(job j) in W[j]
cumulative(x,D,R,L)
{job j, job k} |x[j].end - x[k].end| >= T0
delta[j,t] is whether running.start(job j, time t)
phi[j,t] is whether running(job j, time t)
```

Solver generates

$$\varepsilon_{jt} + \varepsilon_{kt'} \leq 1, \text{ all } t, t' \text{ with } 0 < t' - t < T_0, \text{ all } j, t \text{ with } j \neq k$$

```
epsilon[j,t] is whether running.end(job j, time t)
```

Interval variables

Variables δ_{jt} and ε_{jt} are related by an offset.
Solver associates **`running.`*`end`*** in declaration of ε_{jt} with **`running.`*`start`*** in declaration of δ_{jt} and deduces

$$e_{j,t+D_j} = \delta_{jt}, \text{ all } j,t$$

```
delta[j,t] is whether running.start(job j, time t)
phi[j,t] is whether running(job j, time t)
epsilon[j,t] is whether running.end(job j, time t)
```

Interval variables

Variables δ_{jt} and ε_{jt} are related by an offset.
Solver associates **`running.`*`end`*** in declaration of ε_{jt}
with **`running.`*`start`*** in declaration of δ_{jt} and deduces

$$e_{j,t+D_j} = \delta_{jt},\ \text{all } j,t$$

Solver also associates **`running.`*`end`*** in declaration of ε_{jt}
with **`running`** in declaration of ϕ_{jt} and deduces
the redundant constraints

$$\phi_{jt} \geq \varepsilon_{jt'},\ \text{all } t,t' \text{ with } 0 \leq t'-t < D_j,\ \text{all } j$$

```
delta[j,t] is whether running.start(job j, time t)
phi[j,t] is whether running(job j, time t)
epsilon[j,t] is whether running.end(job j, time t)
```

Pros and Cons of Semantic Typing

- Pros
  - Conveys problem structure to MIP solver
    - …by allowing use of metaconstaints
  - Incorporates state of the art in formulation, valid inequalities
  - Allows solver to expand repertory of techniques
    - Domain filtering, propagation
  - Good modeling practice
    - Self-documenting
    - Bug detection

Pros and Cons of Semantic Typing

- Cons
  - Modeler must be familiar with a large collection of metaconstraints
    - Rather than few primitive constraints

Pros and Cons of Semantic Typing

- Cons
  - Modeler must be familiar with a large collection of metaconstraints
    - Rather than few primitive constraints
  - *Response*
    - *Modeler must be familiar with the underlying **concepts** anyway*
    - *Modeling system can offer sophisticated help, improve modeling*

Pros and Cons of Semantic Typing

- Cons
  - Modeler must be familiar with a large collection of metaconstraints
    - Rather than few primitive constraints
  - *Response*
    - *Modeler must be familiar with the underlying **concepts** anyway*
    - *Modeling system can offer sophisticated help, improve modeling*
  - OR community is not accustomed to high-level modeling
    - Typed languages like Ascend never really caught on.

Pros and Cons of Semantic Typing

- Cons
  - Modeler must be familiar with a large collection of metaconstraints
    - Rather than few primitive constraints
  - *Response*
    - *Modeler must be familiar with the underlying **concepts** anyway*
    - *Modeling system can offer sophisticated help, improve modeling*
  - OR community is not accustomed to high-level modeling
    - Typed languages like Ascend never really caught on.
  - *Response*
    - *Train the next generation!*