Combining Optimization and Constraint Programming

J. N. Hooker

Carnegie Mellon University

IBM T. J. Watson Lab

August 2000

Some work is joint with…

Ignacio Grossmann, *Carnegie Mellon University*
Hak-Jin Kim, *Carnegie Mellon University*
María Auxilio Osorio, *Universidad Autónoma de Puebla*
Greger Ottosson, *Uppsala Universitet*
Erlendur Þorsteinsson, *Carnegie Mellon University*

Optimization and constraint programming

- *Optimization* is an old field related to mathematics and engineering.
    - Uses such techniques as linear, integer and nonlinear programming.
- *Constraint programming* is a relatively new field that developed in the computer science and artificial intelligence communities.
    - Uses search and constraint propagation to solve problems formulated in a programming language.

Optimization

- Linear and mixed integer programming models use a restricted modeling vocabulary.
  - Equations, inequalities, numeric functions.
  - Models are declarative.
- Solution methods exploit the structure of problem classes
  - Job shop scheduling problems, traveling salesman problems, etc.

Constraint Programming

- Versatile modeling framework.
  - Logical conditions, all-different constraints, variable indices.
  - Models are written in a programming language.
- Solution method exploits the structure of particular constraints.
  - All-different, cumulative (for scheduling), element (for variable indices), etc.

A Modeling Example

Traveling salesman problem:

Let c_{ij} = distance from city i to city j.

Find the shortest route that visits each of n cities exactly once.

Integer Programming Model

Let x_{ij} = 1 if city i immediately precedes city j, 0 otherwise

$$
\begin{aligned}
\text{minimize} \quad & \sum_{ij} c_{ij} x_{ij} \\
\text{subject to} \quad & \sum_{i} x_{ij} = 1, \quad \text{all } j \\
& \sum_{j} x_{ij} = 1, \quad \text{all } i \\
& \sum_{i \in V} \sum_{j \in W} x_{ij} \geq 1, \quad \text{all disjoint } V, W \subset \{1, \ldots, n\} \\
& x_{ij} \in \{0,1\}
\end{aligned}
$$

Subtour elimination constraints

Constraint Programming Model

Let y_k = the kth city visited.

The model would be written in a specific constraint programming language but would essentially say:

Variable indices

$$\begin{aligned} \text{minimize} \quad & \sum_k c_{y_k y_{k+1}} \\ \text{subject to} \quad & \text{all-different}(y_1, \ldots, y_n) \\ & y_k \in \{1, \ldots, n\} \end{aligned}$$

"Global" constraint

Constraint Programming Model

The traveling salesman problem can also be written,

$$\begin{aligned} \min \quad & \sum_j c_{jy_j} \\ \text{s.t.} \quad & \text{circuit}\{y_1,\ldots y_n\} \\ & y_j \in \{1,\ldots,n\} \end{aligned}$$

Where y_j is the city that follows city j. The circuit constraint requires that $y_1, \ldots, y_n$ define a Hamiltonian cycle.

Integration of optimization and constraint programming

- Optimization and constraint programming have complementary strengths.
- There is much interest in combining them.
    - OPL combines mathematical programming from CPLEX with constraint programming from ILOG Solver, but in a limited way.
    - ECLiPSe will combine mathematical programming from XPRESS MP with constraint programming from CHIP.
    - Many problems are being addressed with hybrid methods.

Complementary strengths

- Optimization excels at problems in which the constraints (or objective function) may contain many variables, as in cost or profit functions.
    - Relaxation techniques are useful.
- Constraint satisfaction is more effective on problems in which the constraints contain few variables.
    - Domain reduction and constraint propagation are useful.

Complementary strengths

- *Relaxation* removes some constraints to make the problem easier.
    - For example, a continuous relaxation drops integrality constraints on variables to obtain a linear or nonlinear programming problem.
    - Solving the relaxation gives a bound on the optimal value, which is useful in a branch-and-bound search.
- Relaxation is especially useful when constraints contain many variables; for example, cost constraints.

Complementary strengths

- The *domain* of a variable is the set of values it can have.
    - *Domain reduction* deduces that a given variable in a constraint can take only certain values if that constraint is to be satisfied.
    - *Constraint propagation* passes the reduced domains to other constraints, where they can be reduced further.
- Domain reduction and constraint propagation are useful when the constraint contains only a few variables.
    - For example, in *binary* constraints, which contain two variables.

Complementary strengths

- Optimization relies on deep analysis of the mathematical structure of specific classes of problems.
    - Particularly polyhedral analysis, which yields strong cutting planes.
- Constraint satisfaction identifies subsets of problem constraints that have special structure.
    - Represents them with *global constraints* (for example, `all-different, cumulative`) and applies tailor-made domain-reduction algorithms.

Complementary strengths

- Optimization can be very fast when the problem has special structure.
- Constraint satisfaction becomes faster when more constraints are added, even if they are unstructured.

One scheme for integration

- Use *both constraint propagation and relaxation* during branch-and-bound search.
- Use both *relaxations* and *global constraints* to exploit *structure*.
    - Apply special-purpose optimization methods to *relaxations*.
    - Associate domain reduction techniques *and* special-purpose relaxations with *global constraints*.

A motivating example

$$
\begin{array}{ll}
\min & 5x_1 + 8x_2 + 5x_3 \\
\text{subject to} & 3x_1 + 5x_2 + 3x_3 \geq 30 \\
& \text{all-different}\{x_1, x_2, x_3\} \\
& x_j \in \{1, \ldots, 4\}
\end{array}
$$

Formulate and solve 3 ways:

- a constraint programming problem
- an integer programming problem
- a combined approach

Solve as a constraint programming problem

$$
\begin{aligned}
&5x_1 + 8x_2 + 4x_3 \le z \\
&3x_1 + 5x_2 + 2x_3 \ge 30 \\
&\text{all-different}\{x_1, x_2, x_3\} \\
&x_j \in \{1, \ldots, 4\}
\end{aligned}
$$

Start with z = ∞.
Will decrease as feasible solutions are found.

Use domain reduction

- Bounds propagation on $\begin{array}{l} 5x_1 + 8x_2 + 4x_3 \le z \\ 3x_1 + 5x_2 + 2x_3 \ge 30 \end{array}$

For example, $3x_1 + 5x_2 + 2x_3 \ge 30$ implies

$$x_2 \ge \frac{30 - 3x_1 - 2x_3}{5} \ge \frac{30 - 12 - 8}{5} = 2$$

So the domain of x_2 is reduced to $\{2,3,4\}$.

Use domain reduction

- Maintain hyperarc consistency on

$$\text{all-different}\{x_1, x_2, x_3\}$$

Suppose for example:

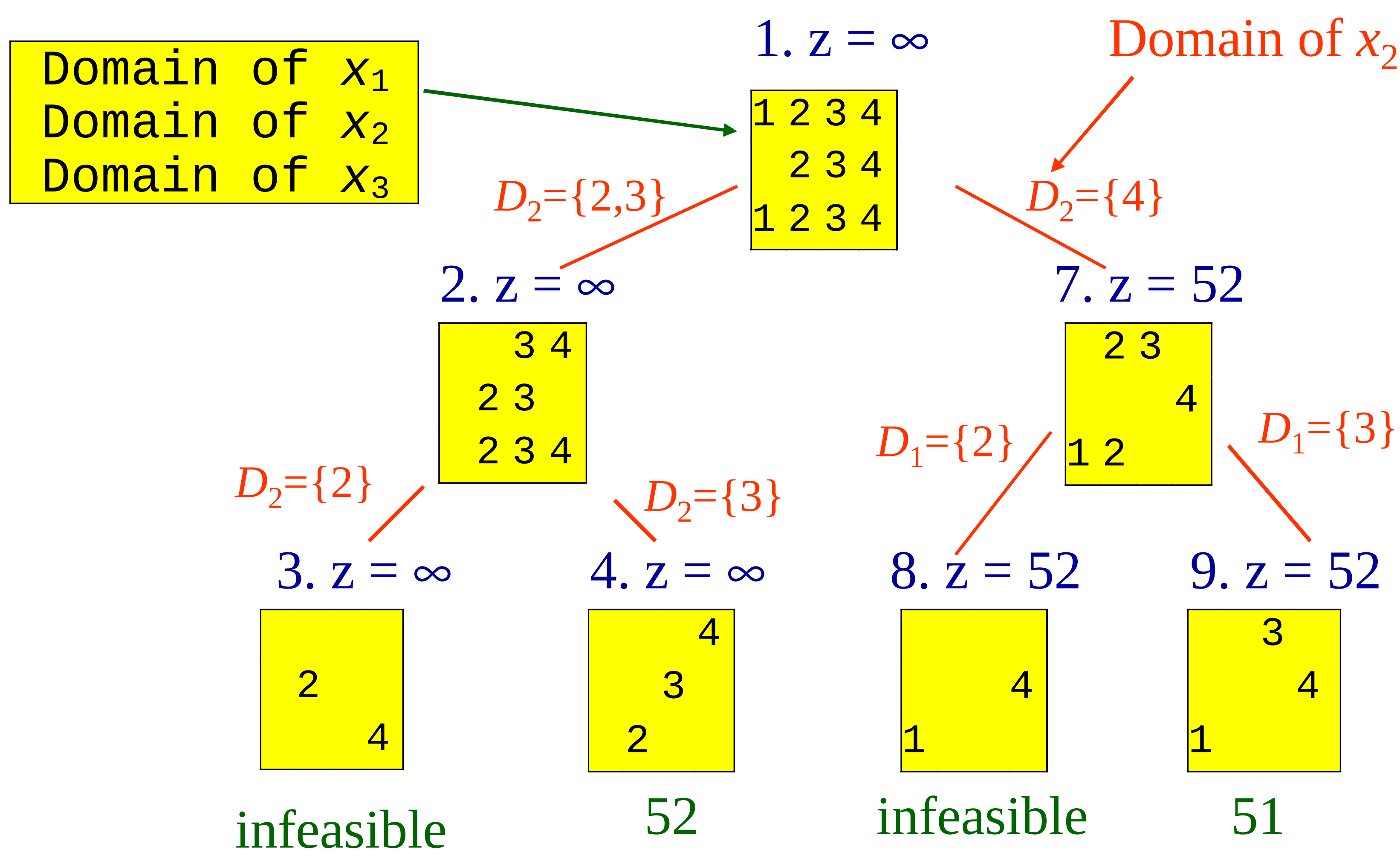
1	2			Domain of x_1
1	2			Domain of x_2
1	2	3	4	Domain of x_3

Then one can reduce the domains:

1	2		
1	2		
		3	4

Use domain reduction

- In general, use a theorem of Berge and maximum cardinality bipartite matching to reduce domains for all-different.
- Cycle through domain reductions and bounds propagation until a fixed point is obtained



Domain of x1
Domain of x2
Domain of x3
1. z = ∞
1 2 3 4
2 3 4
1 2 3 4
Domain of x2
D2={2,3}
D2={4}
2. z = ∞
3 4
2 3
2 3 4
7. z = 52
2 3
4
1 2
D2={2}
D2={3}
D1={2}
D1={3}
3. z = ∞
2
4
4. z = ∞
4
3
2
8. z = 52
4
1
9. z = 52
3
4
1
infeasible
52
infeasible
51

Solve as an integer programming problem

Let y_{ij} be 1 if $x_i = j$, 0 otherwise.

$$\begin{aligned}
\text{min} \quad & 4x_1 + 3x_2 + 5x_3 \\
\text{subject to} \quad & 4x_1 + 2x_2 + 4x_3 \geq 17 \\
& x_i = \sum_{j=1}^{5} j y_{ij}, \quad i = 1,2,3 \\
& \sum_{j=1}^{5} y_{ij} = 1, \quad i = 1,2,3 \\
& y_{jk} \in \{0,1\}, \quad \text{all } j,k
\end{aligned}$$

Continuous relaxation

Use a linear programming algorithm to solve a continuous relaxation of the problem at each node of the search tree to obtain a lower bound on the optimal value of the problem at that node.

$$
\begin{array}{ll}
\min & 4x_1 + 3x_2 + 5x_3 \\
\text{subject to} & 4x_1 + 2x_2 + 4x_3 \geq 17 \\
& x_i = \sum_{j=1}^{5} j y_{ij}, \quad i = 1,2,3 \\
& \sum_{j=1}^{5} y_{ij} = 1, \quad i = 1,2,3 \\
& 0 \leq y_{ij} \leq 1, \quad \text{all } i, j
\end{array}
$$

Relax integrality

Branch and bound

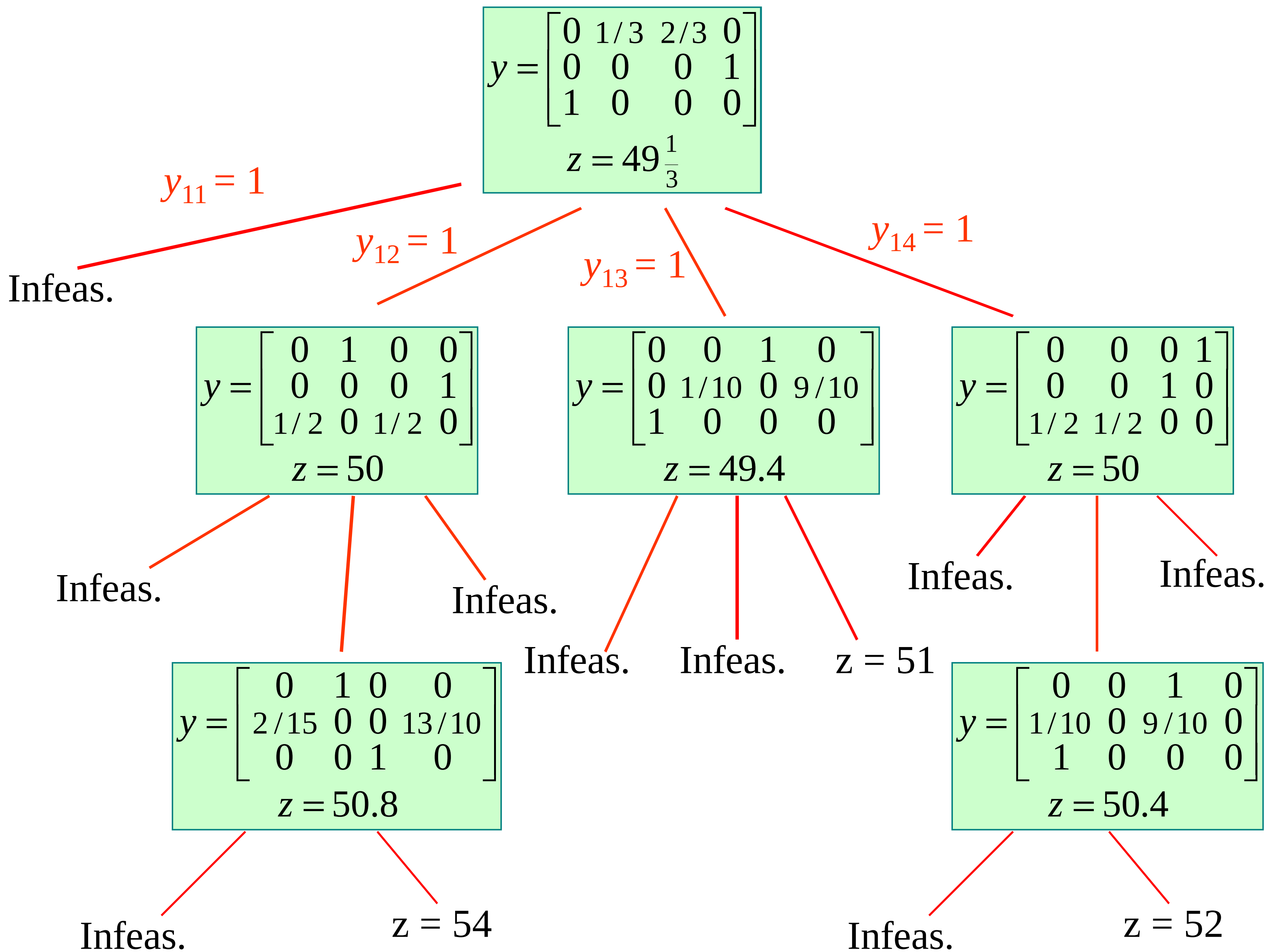
The *incumbent solution* is the best feasible solution found so far.

At each node of the branching tree:

- If Optimal value of relaxation $\geq$ Value of incumbent solution

There is no need to branch further.

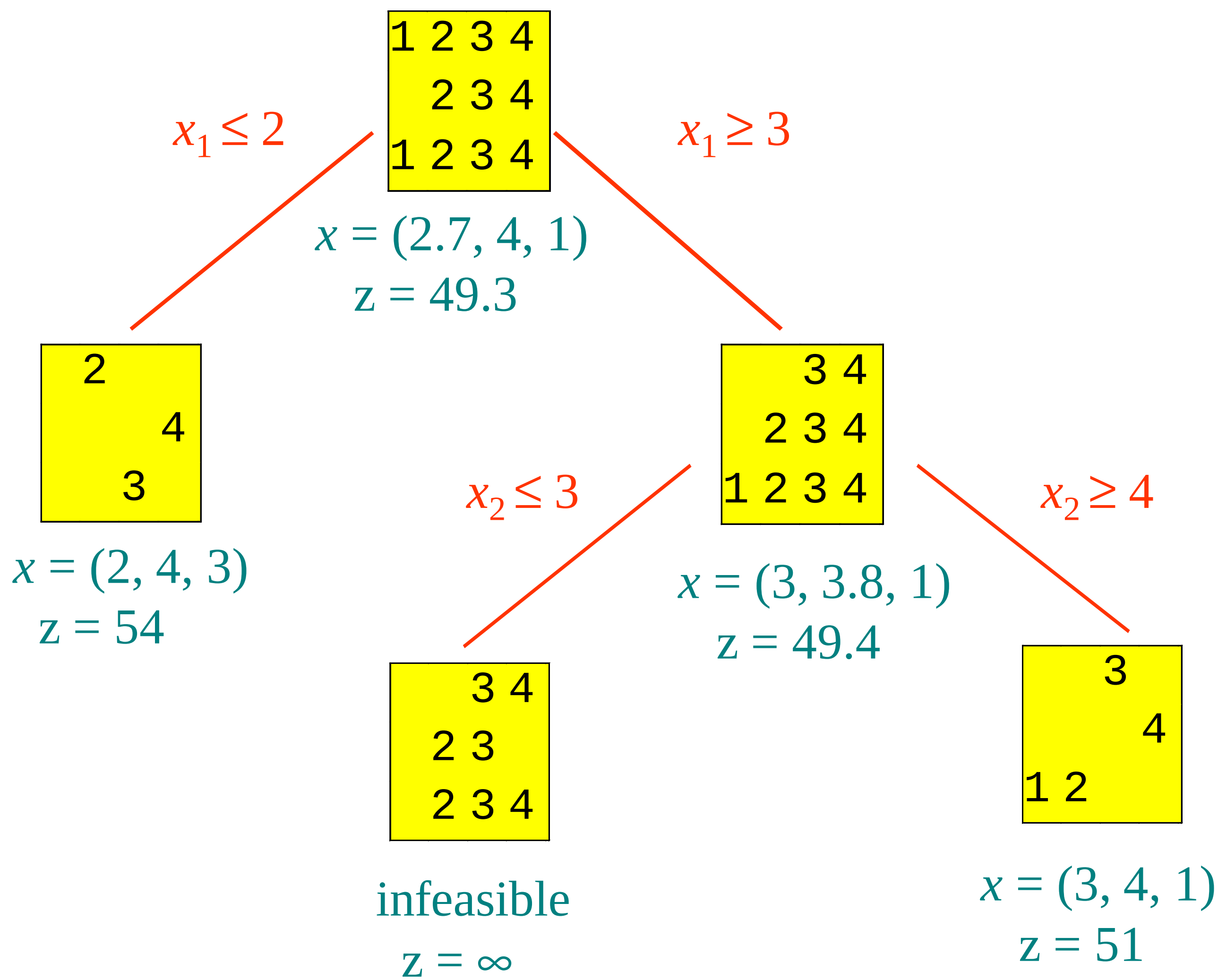
- No feasible solution in that subtree can be better than the incumbent solution.



$y = \begin{bmatrix} 0 & 1/3 & 2/3 & 0 \\ 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 \end{bmatrix}$
$z = 49\frac{1}{3}$
$y_{11} = 1$
Infeas.
$y_{12} = 1$
$y = \begin{bmatrix} 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 1/2 & 0 & 1/2 & 0 \end{bmatrix}$
$z = 50$
$y_{13} = 1$
$y = \begin{bmatrix} 0 & 0 & 1 & 0 \\ 0 & 1/10 & 0 & 9/10 \\ 1 & 0 & 0 & 0 \end{bmatrix}$
$z = 49.4$
$y_{14} = 1$
$y = \begin{bmatrix} 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \\ 1/2 & 1/2 & 0 & 0 \end{bmatrix}$
$z = 50$
Infeas.
Infeas.
Infeas.
Infeas.
z = 51
Infeas.
Infeas.
$y = \begin{bmatrix} 0 & 1 & 0 & 0 \\ 2/15 & 0 & 0 & 13/10 \\ 0 & 0 & 1 & 0 \end{bmatrix}$
$z = 50.8$
$y = \begin{bmatrix} 0 & 0 & 1 & 0 \\ 1/10 & 0 & 9/10 & 0 \\ 1 & 0 & 0 & 0 \end{bmatrix}$
$z = 50.4$
Infeas.
z = 54
Infeas.
z = 52

Combined approach

- Use continuous relaxation of original knapsack constraint $3x_1 + 5x_2 + 2x_3 \geq 30$ (do not use y_{ij}'s).
- Use bounds propagation.
- Maintain hyperarc consistency for all-different.
- Branch on nonintegral variable when possible; otherwise branch by splitting domain.



1 2 3 4
2 3 4
1 2 3 4
x = (2.7, 4, 1)
z = 49.3
x1 ≤ 2
x1 ≥ 3
2
4
3
x = (2, 4, 3)
z = 54
3 4
2 3 4
1 2 3 4
x = (3, 3.8, 1)
z = 49.4
x2 ≤ 3
x2 ≥ 4
3 4
2 3
2 3 4
infeasible
z = ∞
3
4
1 2
x = (3, 4, 1)
z = 51

Cooperation between optimization and constraint programming

- **Exploiting problem structure:** Global constraints provide a practical means to take advantage of specialized algorithms (optimization needs this).
- **Relaxation technology:** Optimization can supply continuous relaxations and “back propagation” for global constraints (constraint programming needs this).
- **Inference technology:** Constraint programming can supply inference methods to reduce the search, e.g. domain reduction (cutting planes strengthen the relaxation).

Cooperation between optimization and constraint programming

- **Exploiting problem structure:** Global constraints provide a practical means to take advantage of specialized algorithms (optimization needs this).
- **Relaxation technology:** Optimization can supply continuous relaxations and “back propagation” for global constraints (constraint programming needs this).
- **Inference technology:** Constraint programming can supply inference methods to reduce the search, e.g. domain reduction (cutting planes strengthen the relaxation).

Exploiting problem structure

- Constraint programming generally processes constraints *locally* or one at a time (results are “propagated” through the constraint store -- more on this later).
- A *global constraint* represents a specially-structured *set* of constraints (all-different, circuit, element, cumulative, etc.).
- By processing a global constraint, one exploits the *global* structure of the set of constraints it represents.
- Every global constraint “brings with it” a set of procedures (an idea suggested by the practice of modeling in a programming language).

Exploiting problem structure

- One can also associate *relaxations* and "*back propagation*" methods with global constraints. (More on this later.)
- This provides a principle for moving research output into commercial code.
    - Domain reduction methods are normally put to use right away, although as a result many are proprietary.
    - Cutting plane methods tend to appear in the open literature, but many are not used in commercial codes (they are designed for special problems rather than special constraints).

Exploiting problem structure

- One can associate specialized cutting planes, etc., with global constraints representing constraint sets for which the cutting planes are designed.
  - For example, the global constraint circuit($y_1,\ldots,y_n$) requires that $y_1,\ldots,y_n$ represent a hamiltonian cycle on a graph, where y_j = vertex that follows vertex j. It could invoke a continuous relaxation that contains some TSP (separating) cuts.
  - This can put to use the large body of results in polyhedral analysis that are now employed only in specialized codes.

Exploiting problem structure

This approach creates a growing vocabulary of global constraints. This can get out of hand, but consider:

- The modeling language can exploit the expertise of the user
    - A domain expert thoroughly understands the structure of the problem in the real world (as opposed to the structure of its mathematical representation), and global constraints can capture this structure.

Exploiting problem structure

Example: cumulative$((t_1,\ldots,t_n),(d_1,\ldots,d_n),(r_1,\ldots,r_n),L)$

t_i = start time of job i
d_i = duration
r_i = rate of resource consumption
L = limit on total rate of resource consumption at any time

Use cumulative$((t_1,\ldots,t_n),(d_1,\ldots,d_n),(1,\ldots,1),m)$ for m-machine scheduling.

Exploiting problem structure

- The user need only have advanced knowledge of the vocabulary that applies to his/her own application domain.
- A powerful feature of ordinary language is that we identify useful concepts that abbreviate clusters of more elementary concepts.
    - This requires learning more words but is inseparable from the task of learning how to do the task at hand.
    - Perhaps it is the same with modeling. The atomistic approach of optimization modeling misses this opportunity.

Cooperation between optimization and constraint programming

- **Exploiting problem structure:** Global constraints provide a practical means to take advantage of specialized algorithms (optimization needs this).

- **Relaxation technology:** Optimization can supply continuous relaxations and “back propagation” for global constraints (constraint programming needs this).

- **Inference technology:** Constraint programming can supply inference methods to reduce the search, e.g. domain reduction (cutting planes strengthen the relaxation).

Relaxations (IP → CP)

- It is useful to work with an example: relaxation of the *element* constraint, which is important because it implements variable indices.
- *Variable indices* are a key modeling device for CP. A few models will illustrate their use.

Variable indices

Traveling salesman problem

$$\begin{array}{ll} \min & \sum_j c_{y_j y_{j+1}} \\ \text{s.t.} & \text{all-different}\{y_1, \ldots y_n\} \\ & y_j \in \{1, \ldots, n\} \end{array}$$

or...

$$\begin{array}{ll} \min & \sum_j c_{j y_j} \\ \text{s.t.} & \text{circuit}\{y_1, \ldots y_n\} \\ & y_j \in \{1, \ldots, n\} \end{array}$$

Variable indices

Quadratic assignment problem

$$\begin{aligned} \min \quad & \sum_{i,j} v_{ij} c_{y_i y_j} \\ \text{s.t.} \quad & \text{all-different}\{y_1, \ldots y_n\} \\ & y_j \in \{1, \ldots, n\} \end{aligned}$$

y_i = site assigned to facility i

v_{ij} = traffic between facility i and j

c_{kl} = distance between location k and l

Variable Indices

Assignment problem with two linked formulations

$$\begin{array}{ll} \min & \text{some objective} \\ \text{s.t.} & \text{constraints on } x_i\text{'s} \\ & \text{constraints on } y_j\text{'s} \\ & x_{y_j} = j, \text{ all } j \end{array}$$

x_i = employee assigned time slot i
y_j = time slot assigned employee j

Variable Indices

- The linkage of two models improves constraint propagation.
- Here a variable (rather than a constant) has a variable subscript.

Element constraint

The constraint $c_y \le 5$ can be implemented:

$$
\begin{aligned}
& z \le 5 \\
& \text{element}(y, (c_1, \ldots, c_n), z)
\end{aligned}
$$

The constraint $x_y \le 5$ can be implemented:

$$
\begin{aligned}
& z \le 5 \\
& \text{element}(y, (x_1, \ldots, x_n), z)
\end{aligned}
$$

(this is a slightly different constraint)

Element constraint

element can be processed with a discrete domain reduction algorithm that maintains hvperarc consistency.

The more interesting case is $\text{element}(y,(x_1,\ldots,x_n),z)$

$$D_z \leftarrow D_z \cap \bigcup_{j\in D_y} D_{x_j}$$

$$D_y \leftarrow D_y \cap \{\, j \mid D_x \cap D_{x_j} \neq \emptyset \}$$

$$D_{x_j} \leftarrow \begin{cases} D_z & \text{if } D_y = \{j\} \\ D_{x_j} & \text{otherwise} \end{cases}$$

Example... $\text{element}(y, (x_1, x_2, x_3, x_4), z)$

The initial domains are:

$$D_z = \{20,30,60,80,90\}$$
$$D_y = \{1,3,4\}$$
$$D_{x_1} = \{10,50\}$$
$$D_{x_2} = \{10,20\}$$
$$D_{x_3} = \{40,50,80,90\}$$
$$D_{x_4} = \{40,50,70\}$$

The reduced domains are:

$$D_z = \{80,90\}$$
$$D_y = \{3\}$$
$$D_{x_1} = \{10,50\}$$
$$D_{x_2} = \{10,20\}$$
$$D_{x_3} = \{80,90\}$$
$$D_{x_4} = \{40,50,70\}$$

Continuous relaxation of element

$\text{element}(y,(c_1,\ldots,c_n),z)$ is trivial.

The convex hull relaxation is $\min_i\{c_i\} \le z \le \max_i\{c_i\}$

$\text{element}(y,(x_1,\ldots,x_n),z)$ has the following relaxation

$$\sum_{i\in D_y}\frac{x_i}{m_i} - \left(\sum_{i\in D_y}\frac{1}{m_i}\right)z \ge -k+1$$

$$-\sum_{i\in D_y}\frac{x_i}{m_i} + \left(\sum_{i\in D_y}\frac{1}{m_i}\right)z \ge -k+1$$

provided $0 \le x_i \le m_i$ (and where $k = |D_y|$).

If $0 \le x_i \le m$ for all i, then the *convex hull* relaxation of

element$(y, (x_1, \ldots, x_n), z)$ is

$$\sum_{j \in D_y} x_j - (k-1)m \le z \le \sum_{j \in D_y} x_j$$

plus bounds, where $k = |D_y|$.

Example...

$$\begin{aligned}&\text{element}(y,(x_1,x_2),z)\\&0 \le x_1 \le 5\\&0 \le x_2 \le 5\end{aligned}$$

The convex hull relaxation is:

$$\begin{aligned}&x_1 + x_2 - 5 \le z \le x_1 + x_2\\&0 \le x_1 \le 5\\&0 \le x_2 \le 5\end{aligned}$$

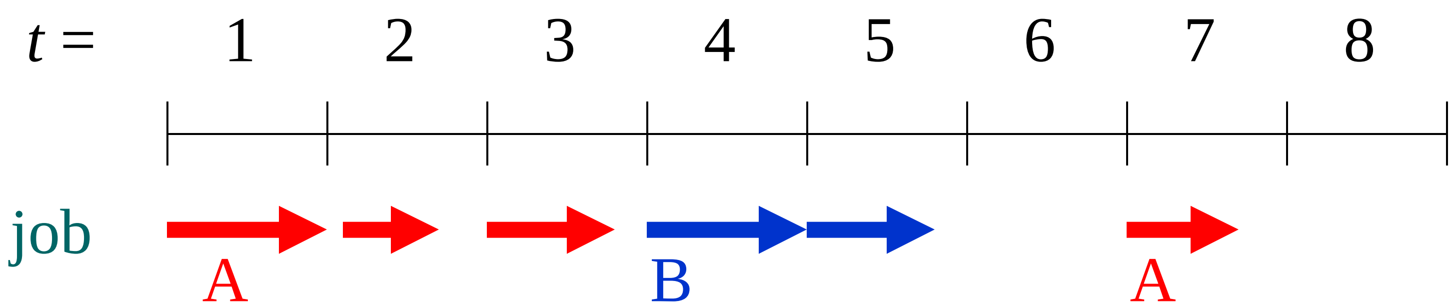
If $0 \le x_1 \le 4$ the above remains valid and we have

$$5x_1 + 4x_2 - 20 \le 9z \le 5x_1 + 4x_2 + 20$$

Discrete lot sizing example

- Manufacture at most one product each day.
- When manufacturing starts, it may continue several days. (R_i = minimum run length).
- Switching to another product incurs a cost.
- There is a certain demand for each product on each day.
- Products are stockpiled to meet demand between manufacturing runs.
- Minimize inventory cost + changeover cost.

Discrete lot sizing example



$y_t =$ A A A B B 0 A 0

0 = dummy job

IP model

(from L. Wolsey)

$$
\begin{aligned}
\min \quad & \sum_{t,i} \left(h_{it} s_{it} + \sum_{j \neq i} q_{ij} \delta_{ijt} \right) \\
\text{s.t.} \quad & s_{i,t-1} + x_{it} = d_{it} + s_{it}, \quad \text{all } i,t \\
& z_{it} \geq y_{it} - y_{i,t-1}, \quad \text{all } i,t \\
& z_{it} \leq y_{it}, \quad \text{all } i,t \\
& z_{it} \leq 1 - y_{i,t-1}, \quad \text{all } i,t \\
& \delta_{ijt} \geq y_{i,t-1} + y_{jt} - 1, \quad \text{all } i,t \\
& \delta_{ijt} \geq y_{i,t-1}, \quad \text{all } i,t \\
& \delta_{ijt} \leq y_{jt}, \quad \text{all } i,t \\
& z_{it} \leq y_{i,t+r-1}, \ r = 1, \ldots, R_i, \text{ all } i,t \\
& x_{it} \leq C y_{it}, \ \text{all } i,t \\
& \sum_i y_{it} = 1, \ \text{all } t \\
& y_{it}, z_{it}, \delta_{ijt} \in \{0,1\} \\
& x_{it}, s_{it} \geq 0
\end{aligned}
$$

The model

total inventory + changeover cost

stock level

changeover cost

daily production

$$
\begin{aligned}
\min \quad & \sum_t (u_t + v_t) \\
\text{s.t.} \quad & u_t \ge \sum_i h_i s_{it}, \ \text{all } t \\
& v_t \ge q_{y_{t-1} y_t}, \ \text{all } t \\
& s_{i,t-1} + x_{it} = d_{it} + s_{it}, \ \text{all } i,t \\
& 0 \le x_{it} \le C,\ s_{it} \ge 0, \ \text{all } i,t \\
& (y_t \ne i) \rightarrow (x_{it} = 0), \ \text{all } i,t \\
& (y_{t-1} \ne i = y_t) \rightarrow (y_{t+1} = \cdots = y_{t+R_i-1} = i), \ \text{all } i,t
\end{aligned}
$$

inventory balance

Relaxation

Put into relaxation

Generate inequalities to put into relaxation

$$
\begin{aligned}
\min \quad & \sum_t (u_t + v_t) \\
\text{s.t.} \quad & u_t \ge \sum_i h_i s_{it}, \ \text{all } t \\
& v_t \ge q_{y_{t-1} y_t}, \ \text{all } t \\
& s_{i,t-1} + x_{it} = d_{it} + s_{it}, \ \text{all } i,t \\
& 0 \le x_{it} \le C, \ s_{it} \ge 0, \ \text{all } i,t \\
& (y_t \ne i) \rightarrow (x_{it} = 0), \ \text{all } i,t \\
& (y_{t-1} \ne i = y_t) \rightarrow \left(y_{t+1} = \cdots = y_{t+R_i-1} = i\right), \ \text{all } i,t
\end{aligned}
$$

Apply constraint propagation to everything

To solve the example

- At each node of search tree:
  - Apply domain reduction and constraint propagation.
  - Generate and solve continuous relaxation to get bound.
- Characteristics:
  - Relaxation is somewhat weaker than in IP because logical constraints are not all relaxed.
  - But LP relaxations are much smaller--quadratic rather than cubic size.
  - Domain reduction helps prune tree.

Back propagation

- A global constraint can also be associated with a back propagation scheme, which reduces domains based on solution of the relaxation.
- A simple example is variable fixing using reduced costs, now used in both optimization and constraint programming.
    - For instance, one can give circuit($y_1,\ldots,y_n$) an assignment relaxation.
    - The value of the relaxation may be useless, but the reduced costs can allow one to exclude values of y_i.
    - Variables x_{ij} in assignment problem need not be defined. Just use appropriate data structure to solve problem.

A peek at a modeling framework for integration

Let every constraint i have *conditional form*

$$h_i(y) \rightarrow S_i(x)$$

Or be reducible to a set of conditionals.

$h_i(y)$ - *hard* constraint (belonging to NP); contains variables $y = (y_1, \ldots, y_m)$.

$S_i(x)$ - set of *easy* constraints (belonging to NP and co-NP?) that will go into relaxation; contains variables $x = (x_1, \ldots, x_n)$.

The objective function has the form $f(x)$.

The basic search algorithm

- *Branch* on domains of variables y_j.
- Use *inference* to deduce, when possible, whether partially specified variables y_j satisfy constraints $h_i(y)$.
- *Solve* the relaxed problem of minimizing $f(x)$ with respect to x, subject to the $S_i(x)$'s that are enforced by true $h_i(y)$'s.
    - This relaxation can be strengthened, or augmented by other relaxations, if desired.
- Back-propagate from the solution of the relaxation.
- Search for value of y consistent with this solution.
- Continue in a branch-and-relax fashion.

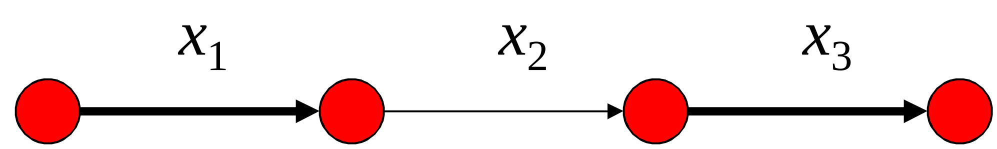
Cooperation between optimization and constraint programming

- **Exploiting problem structure:** Global constraints provide a practical means to take advantage of specialized algorithms (optimization needs this).
- **Relaxation technology:** Optimization can supply continuous relaxations and "back propagation" for global constraints (constraint programming needs this).
- **Inference technology:** Constraint programming can supply inference methods to reduce the search, e.g. domain reduction (cutting planes strengthen the relaxation).

Inference (CP $\rightarrow$ IP)

- Inference accelerates search by making the constraint set more nearly consistent--i.e., by making implications explicit.
    - The most popular approach is to reduce domains (aim for hyperarc consistency), which reduces branching.
- The research project of finding domain reduction algorithms is analogous to discovery of good cutting planes.
- The structural analysis associated with cutting plane theory (e.g., special subgraphs, etc.) may suggest constraints that are good in the sense that they move closer to consistency.
    - For example, alternating paths in matching correspond to derived constraints (which strictly dominate facet-defining cuts).

Inference (CP $\rightarrow$ IP)



Convex hull description of this matching problem:

$$\begin{aligned} x_1 + x_2 &\le 1 \\ x_2 + x_3 &\le 1 \\ x_1, x_2, x_3 &\ge 0 \end{aligned}$$

Alternating path shown corresponds to $x_1 + 2x_2 + x_3 \le 2$.

This strictly dominates both facets (in 0-1 sense).
It clearly dominates.
It also excludes (0,1,1), which satisfies 1st facet,
and (1,1,0), which satisfies 2nd facet.

Inference (CP $\rightarrow$ IP)

- Domain reduction can be regarded as constraint (cut) generation.
- To reduce the domain of y_j is to post an in-domain constraint $y_j \in D_j$.
- The generated in-domain constraints are normally regarded as forming a *constraint store*,which allows communication among constraints.
- But they can be regarded as forming a discrete *relaxation* (i.e., optimization problem solved to get a bound)
    - Each domain element belongs to some feasible solution, but simply picking an element from each domain may not yield a solution.

Inference (CP $\rightarrow$ IP)

- One might solve this relaxation to optimality (usually trivial) and generate "separating" in-domain constraints.
    - Solution of relaxation would guide domain reduction.
- One might also allow a broader class of easy constraints in the constraint store.
    - For example, constraints whose dependency graph has limited induced width, to be solved by nonserial dynamic programming.