IS703: Decision Support and Optimization

Week 6: Constraint Programming

Lau Hoong Chuin
School of Information Systems
Constraint programming
(adapted from Michael Trick’s INFORMS talk)

• Began in 1980s from AI world
• “Dream” of declarative programming
  – State the constraints
  – Solver finds a solution
• Application areas
  – Scheduling, sequencing, resource and personnel allocation, control etc.
• Two main contributions
  – A new language for combinatorial optimization
    • Rich language for constraints
    • Language for search procedures
  – A new approach to optimization
    • Orthogonal and complementary to standard OR methods like math programming and dynamic programming
    • Symbolic versus numerical
Modeling in Constraint Programming

• A rich constraint language
  – Arithmetic, higher-order, logical constraints
  – Global constraints for natural substructures

• Specification of a search procedure
  – Definition of search tree to explore
  – Specification of search strategy
Classic Example

\[\text{SEND} + \text{MORE} = \text{MONEY}\]
Send + More = Money

Assign distinct digits to the letters S, E, N, D, M, O, R, Y such that
SEND
+ MORE

= MONEY

holds.
Assign distinct digits to the letters $S, E, N, D, M, O, R, Y$ such that

\[
\begin{align*}
SEND & \quad + \quad MORE \\
\hline
=\quad MONEY
\end{align*}
\]

holds.

Solution

\[
\begin{align*}
9 & \quad 5 & \quad 6 & \quad 7 \\
+ & \quad 1 & \quad 0 & \quad 8 & \quad 5 \\
\hline
1 & \quad 0 & \quad 6 & \quad 5 & \quad 2
\end{align*}
\]
Constraint Modeling

Formalize the problem as a constraint problem:

- $n$ variables: $x_1, \ldots, x_n$
- $n$ domains: $D_1, \ldots, D_n$
- constraints: $c_1, \ldots, c_m$
- Find an assignment of variables to domain values such that all $m$ constraints are satisfied
Constraint Model for **MONEY**

- **8 variables**: \{S, E, N, D, M, O, R, Y\}
- **Domains**: \(0 \leq S, \ldots, Y \leq 9\)
- **Constraints**:
  \[
  c_1 = 1000 \cdot S + 100 \cdot E + 10 \cdot N + D \\
  + 1000 \cdot M + 100 \cdot O + 10 \cdot R + E \\
  = 10000 \cdot M + 1000 \cdot O + 100 \cdot N + 10 \cdot E + Y
  \]
more constraints

\[ c_2 : S \neq 0 \]
\[ c_3 : M \neq 0 \]
\[ c_4 : \{(S,E,N,D,M,O,R,Y) \in \mathbb{Z}^8 \mid S...Y \text{ all different}\} \]

Solution: \( S=9, E=5, N=6, D=7, M=1, O=0, R=8, Y=2 \)
Elements of Constraint Programming
(How does CP solver get the solution?)

Exploiting constraints during tree search

• propagation algorithms for constraints
  • A.k.a. domain pruning or filtering (consistency)
• branching strategy
• exploration strategy
\[
\begin{align*}
S &\in \{0..9\} \\
E &\in \{0..9\} \\
N &\in \{0..9\} \\
D &\in \{0..9\} \\
M &\in \{0..9\} \\
O &\in \{0..9\} \\
R &\in \{0..9\} \\
Y &\in \{0..9\}
\end{align*}
\]

\[
\begin{align*}
1000*S + 100*E + 10*N + D &+ 1000*M + 100*O + 10*R + E \\
&= 10000*M + 1000*O + 100*N + 10*E + Y
\end{align*}
\]
Propagate

\[
\begin{align*}
S & \in \{1 \ldots 9\} \\
E & \in \{0 \ldots 9\} \\
N & \in \{0 \ldots 9\} \\
D & \in \{0 \ldots 9\} \\
M & \in \{1 \ldots 9\} \\
O & \in \{0 \ldots 9\} \\
R & \in \{0 \ldots 9\} \\
Y & \in \{0 \ldots 9\} \\
\end{align*}
\]

0 \leq S, \ldots, Y \leq 9

\[
\begin{align*}
S \neq 0 \\
M \neq 0 \\
S, \ldots, Y \text{ all different}
\end{align*}
\]

\[
1000\times S + 100\times E + 10\times N + D \\
+ 1000\times M + 100\times O + 10\times R + E \\
= 10000\times M + 1000\times O + 100\times N + 10\times E + Y
\]
Propagate

\[
\begin{align*}
S & \in \{9\} \\
E & \in \{4, \ldots, 7\} \\
N & \in \{5, \ldots, 8\} \\
D & \in \{2, \ldots, 8\} \\
M & \in \{1\} \\
O & \in \{0\} \\
R & \in \{2, \ldots, 8\} \\
Y & \in \{2, \ldots, 8\}
\end{align*}
\]

\[
0 \leq S, \ldots, Y \leq 9
\]

\[
S \neq 0, \quad M \neq 0
\]

\[S \ldots Y \text{ all different}\]

\[
1000 \times S + 100 \times E + 10 \times N + D \\
+ 1000 \times M + 100 \times O + 10 \times R + E
\]

\[= 10000 \times M + 1000 \times O + 100 \times N + 10 \times E + Y\]
Propagate

\[ \begin{align*}
S &\in \{9\} \\
E &\in \{4..7\} \\
N &\in \{5..8\} \\
D &\in \{2..8\} \\
M &\in \{1\} \\
O &\in \{0\} \\
R &\in \{2..8\} \\
Y &\in \{2..8\}
\end{align*} \]

\[ S \neq 0 \quad M \neq 0 \]

\[ 1000*S + 100*E + 10*N + D \\
+ 1000*M + 100*O + 10*R + E \\
= 10000*M + 1000*O + 100*N + 10*E + Y \]

\[ S \in \{9\} \\
E \in \{5..7\} \\
N \in \{6..8\} \\
D \in \{2..8\} \\
M \in \{1\} \\
O \in \{0\} \\
R \in \{2..8\} \\
Y \in \{2..8\} \]

\[ 0 \leq S, \ldots, Y \leq 9 \]

\[ S \neq 0 \quad M \neq 0 \]

\[ S \ldots Y \text{ all different} \]
Branch

SEND + MORE = MONEY

S ∈ {9}
E ∈ {4..7}
N ∈ {5..8}
D ∈ {2..8}
M ∈ {1}
O ∈ {0}
R ∈ {2..8}
Y ∈ {2..8}

E = 4

S ∈ {9}
E ∈ {5..7}
N ∈ {6..8}
D ∈ {2..8}
M ∈ {1}
O ∈ {0}
R ∈ {2..8}
Y ∈ {2..8}

E ≠ 4

S ∈ {9}
E ∈ {5..7}
N ∈ {6..8}
D ∈ {2..8}
M ∈ {1}
O ∈ {0}
R ∈ {2..8}
Y ∈ {2..8}

E = 5

S ∈ {9}
E ∈ {5..7}
N ∈ {6..8}
D ∈ {2..8}
M ∈ {1}
O ∈ {0}
R ∈ {2..8}
Y ∈ {2..8}

E ≠ 5

S ∈ {9}
E ∈ {5..7}
N ∈ {6..8}
D ∈ {2..8}
M ∈ {1}
O ∈ {0}
R ∈ {2..8}
Y ∈ {2..8}
Propagate

SEND
+MORE
= MONEY

S ∈ \{9\}
E ∈ \{4..7\}
N ∈ \{5..8\}
D ∈ \{2..8\}
M ∈ \{1\}
O ∈ \{0\}
R ∈ \{2..8\}
Y ∈ \{2..8\}

E = 4

E = 4

E = 5

E = 5

E ≠ 4

E ≠ 4

E = 5

E ≠ 5

E ≠ 5

S ∈ \{9\}
E ∈ \{5\}
N ∈ \{6\}
D ∈ \{7\}
M ∈ \{1\}
O ∈ \{0\}
R ∈ \{8\}
Y ∈ \{2\}

S ∈ \{9\}
E ∈ \{6..7\}
N ∈ \{7..8\}
D ∈ \{2..8\}
M ∈ \{1\}
O ∈ \{0\}
R ∈ \{2..8\}
Y ∈ \{2..8\}
Complete Search Tree

SEND + MORE = MONEY

S ∈ {9}
E ∈ {4..7}
N ∈ {5..8}
D ∈ {2..8}
M ∈ {1}
O ∈ {0}
R ∈ {2..8}
Y ∈ {2..8}

E = 4

E ≠ 4

S ∈ {9}
E ∈ {5..7}
N ∈ {6..8}
D ∈ {2..8}
M ∈ {1}
O ∈ {0}
R ∈ {2..8}
Y ∈ {2..8}

E = 5

E ≠ 5

S ∈ {9}
E ∈ {5}
N ∈ {6}
D ∈ {7}
M ∈ {1}
O ∈ {0}
R ∈ {8}
Y ∈ {2}

E = 6

E ≠ 6

S ∈ {9}
E ∈ {6..7}
N ∈ {7..8}
D ∈ {2..8}
M ∈ {1}
O ∈ {0}
R ∈ {2..8}
Y ∈ {2..8}
Another example: Graph Coloring

• Color a map of (part of) Europe: Belgium, Denmark, France, Germany, Netherlands, Luxembourg

• No two adjacent countries same color

• Is four colors enough?
OPL example

enum Country
    {Belgium, Denmark, France, Germany, Netherlands, Luxembourg};
enum Colors {blue, red, yellow, gray};
var Colors color[Country];

solve {
    color[France] <> color[Belgium];
    color[France] <> color[Luxembourg];
    color[France] <> color[Germany];
    color[Luxembourg] <> color[Germany];
    color[Luxembourg] <> color[Belgium];
    color[Belgium] <> color[Netherlands];
    color[Belgium] <> color[Germany];
    color[Germany] <> color[Netherlands];
    color[Germany] <> color[Denmark];
};

• Note:
  – Variables non-numeric
  – Constraints are non-linear
Constraint Programming: Summary

• Domains:
  – For each variable: what is the set of possible values?
  – If empty for any variable, then infeasible
  – If singleton for any variable, then solution

• Constraints:
  – Capture interesting and well studied substructures
  – Need to
    • Determine if constraint is feasible wrt the domains
    • Prune inconsistent values from the domains
Comparing CP and IP

<table>
<thead>
<tr>
<th>Branch and Prune</th>
<th>Branch and Bound</th>
</tr>
</thead>
<tbody>
<tr>
<td>Prune</td>
<td>Bound</td>
</tr>
<tr>
<td>- Prune: eliminate infeasible configurations</td>
<td>- Bound: eliminate suboptimal solutions</td>
</tr>
<tr>
<td>- Branch: decompose into subproblems</td>
<td>- Branch: decompose into subproblems</td>
</tr>
<tr>
<td>Prune</td>
<td>Bound</td>
</tr>
<tr>
<td>- Carefully examine constraints to reduce possible variable values</td>
<td>- Use (linear) relaxation of problem (+ cuts)</td>
</tr>
<tr>
<td>Branch</td>
<td>Branch</td>
</tr>
<tr>
<td>- Use heuristics based on feasibility info</td>
<td>- Use information from relaxation</td>
</tr>
<tr>
<td>Main focus: constraints and feasibility</td>
<td>Main focus: objective function and optimality</td>
</tr>
</tbody>
</table>
Comparing CP and IP

• Complementary technologies
  – Integer programming
    • Objective function: relaxations
  – Constraint programming
    • Feasibility: domain reductions

• Might need to experiment with both

• CP particularly useful when IP formulation is hard or relaxation does not give much information
Formulation strengths

• Logical requirements, e.g.:
  if A=1 and B<=2 then either C>=3 or D=1.

• Really painful in IP, but straightforward in CP:
  \(( (A=1) \land (B\leq 2) ) \Rightarrow ((C\geq 3) \lor (D=1))\)
Global Constraints

• Recognize that some types of constraints come up often
• Create specialized routines to handle
  • Compact, declarative specifications
    – Strengthen and expand the language
    – Make modeling easier and more natural
    – Details hidden to user
  • Computational Efficiency: Strong pruning
    – Arc consistency, Global arc-consistency (not within course scope)
Global constraint: \texttt{alldifferent}

- Most well known and studied constraint.

\[ \texttt{alldifferent}(x,y,z) \]

states that \( x, y, \) and \( z \) take on different values. So \( x=2, y=1, z=3 \) would be ok, but not \( x=1, y=3, z=1 \).

- Clear uses in routing problems:
  \( x[i] \) is \( ith \) customer visited, \( \texttt{alldifferent}[x] \) says each customer visited at most once

- Very useful in many other situations
Alldifferent feasibility and pruning

- Feasibility? Given domains, create domain/variable bipartite graph
Alldifferent feasibility and pruning

- Pruning? Which edges are in no matching?

Domain is sharply reduced
Finding optimal solutions

• Constraint programs can find optimal solutions
• Typically works by finding a feasible solution and adding a constraint that future solutions must be better than it.
• Repeat until infeasible: the last solution found is optimal
Example Problem

- Painting cars
- Sequence cars to minimize paint changeover
- Cars cannot be sequenced too far out of order
Small example

10 cars in sequence. The order for assembly is 1, 2, ..., 10. A car must be painted within 3 positions of its assembly order. For instance, car 5 can be painted in positions 2 through 8 inclusive. Cars 1, 5, and 9 are red; 2, 6, and 10 are blue; 3 and 7 green; and 4 and 8 are yellow. Initial sequence 1, 2, ... 10 corresponds to color pattern RBGYRBGYRB and has 9 purgings. The sequence 2, 1, 5, 3, 7, 4, 8, 6, 10, 9 corresponds to color pattern BRRGGYYBBR and has 5 purgings.
OPL Program

int n=...;
int rnge=...;
int ncolor=...;
range Slots 1..n;
var Slots slot[1..n];
var Slots revslot[1..n];
int color[1..n]= ...;
minimize
    sum (j in 1..n-1) (color[revslot[j]]<>color[revslot[j+1]])
subject to {
    forall (i in Slots)
        i-rnge<=slot[i] <= i+rnge; /*Must be in range */
        alldifferent(slot); /*must choose different slots */
        forall (i in Slots) revslot[slot[i]] = i;
};
Global constraints

• Many different types of constraints have specialized routines, including:
  – Order constraints
  – Partitioning constraints (e.g. alldifferent)
  – Timetabling constraints
  – Graph constraints
  – Scheduling constraints
  – Bin-packing constraints

• Many others, and new ones being created all the time!
# A Catalog of Global Constraints

<table>
<thead>
<tr>
<th>Constraint</th>
<th>Constraint</th>
<th>Constraint</th>
<th>Constraint</th>
<th>Constraint</th>
<th>Constraint</th>
</tr>
</thead>
<tbody>
<tr>
<td>Alldifferent</td>
<td>Change_partition</td>
<td>Cycle_resource</td>
<td>Golomb</td>
<td>Minimum_pair</td>
<td>Nequivalence</td>
</tr>
<tr>
<td>Alldifferent_except_0</td>
<td>Circuit</td>
<td>Cyclic_change</td>
<td>Graph_crossing</td>
<td>Nclass</td>
<td>Ninterval</td>
</tr>
<tr>
<td>Alldifferent_interval</td>
<td>Circuit_cluster</td>
<td>Cyclic_change_joker</td>
<td>Group</td>
<td>Nequality</td>
<td>Npair</td>
</tr>
<tr>
<td>Alldifferent_modulo</td>
<td>Circular_change</td>
<td>Cyclic_cumulative</td>
<td>Group_skip_isolated_item</td>
<td>Number_of_rest</td>
<td>Number_of_values</td>
</tr>
<tr>
<td>Alldifferent_partition</td>
<td>Coloured_cumulative</td>
<td>Derangement</td>
<td>Inflexion</td>
<td>Orchard</td>
<td>Orchard</td>
</tr>
<tr>
<td>Alldifferent_same_value</td>
<td>Coloured_cumulatives</td>
<td>Diffn</td>
<td>Interval_and_count</td>
<td>Place_in_pyramid</td>
<td>Polyomino</td>
</tr>
<tr>
<td>Among</td>
<td>Common</td>
<td>Diff_2</td>
<td>Interval_and_sum</td>
<td>Relaxed_sliding_sum</td>
<td>Polyomino</td>
</tr>
<tr>
<td>Among_interval</td>
<td>Common_interval</td>
<td>Diff_2_cyclic</td>
<td>Inverse</td>
<td>Sliding_card_skip0</td>
<td>Reversal</td>
</tr>
<tr>
<td>Among_modulo</td>
<td>Common_modulo</td>
<td>Diff_2_min_dist</td>
<td>Longest_change</td>
<td>Sliding_sum</td>
<td>Reversal</td>
</tr>
<tr>
<td>Among_seq</td>
<td>Common_partition</td>
<td>Disjoint</td>
<td>Map</td>
<td>Smooth</td>
<td>Reversal</td>
</tr>
<tr>
<td>Assign_and_count</td>
<td>Connect_points</td>
<td>Disjoint_tasks</td>
<td>Max_index</td>
<td>Soft_alldifferent_ctr</td>
<td>Reversal</td>
</tr>
<tr>
<td>Assign_and_nvalue</td>
<td>Connected</td>
<td>Distance_change</td>
<td>Max_n</td>
<td>Soft_alldifferent_var</td>
<td>Reversal</td>
</tr>
<tr>
<td>Balance</td>
<td>Count</td>
<td>Distance_less</td>
<td>Max_nvalue</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Balance_modulo</td>
<td>Crossing</td>
<td>Distribute</td>
<td>Minimum</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Balance_partition</td>
<td>Cumulative</td>
<td>Domain_constraint</td>
<td>Minimum_modulo</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Bin_packing</td>
<td>Cumulative_2d</td>
<td>Element</td>
<td>Maximum_modulo</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Binary_tree</td>
<td>Cumulative_product</td>
<td>Element_greatereq</td>
<td>Maximum_pair</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Cardinality atleast</td>
<td>Cumulatives</td>
<td>Element_lesseq</td>
<td>Min_index</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Cardinality_atmost</td>
<td>Cycle</td>
<td>Elements</td>
<td>Min_n</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Change</td>
<td>Cycle_card_on_path</td>
<td>Elements_alldifferent</td>
<td>Min_nvalue</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Change_continuity</td>
<td>Cycle_cover</td>
<td>Global_cardinality</td>
<td>Minimum</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Change_pair</td>
<td>Cycle_or_accessibility</td>
<td></td>
<td>Minimum_except_0</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Minimum_modulo</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

33
Order constraints

- $\text{min}_n(X, n, [Y_1, \ldots, Y_m])$ and $\text{max}_n(X, n, [Y_1, \ldots, Y_m])$

$X$ is $n$th smallest value in $Y_1, \ldots, Y_m$

$X$ is $n$th largest value in $Y_1, \ldots, Y_m$
Value constraints

• \( \text{among}(N, [Y_1, \ldots, Y_n], [\text{val}_1, \ldots, \text{val}_m]) \)
  \( N \) vars in \([Y_1, \ldots, Y_n]\) take values \(\text{val}_1, \ldots, \text{val}_m\)
  e.g. \( \text{among}(2, [1, 2, 1, 3, 1, 5], [3, 4, 5]) \)

• \( \text{count}(N, [Y_1, \ldots, Y_n], \text{op}, X) \)
  where \(\text{op}\) is \(=,\lt,\gt,\leq,\geq\) or \(=/\)
  relation “\(Y_i \text{ op } X\)” holds \(N\) times
  note: \( \text{among}(n, [Y_1, \ldots, Y_m], [k]) = \text{count}(n, [Y_1, \ldots, Y_m], =, k) \)

• \( \text{distribute}([C_1, \ldots, C_n], [Y_1, \ldots, Y_n], [\text{val}_1, \ldots, \text{val}_m]) \)
  the number of times \(Y_i\) appears in \(\text{val}\) is \(C_i\)
  Useful for rostering problems (e.g. nurses on shifts)
Graph constraints

- \texttt{cycle(N, [X_1, \ldots, X_n])}
  
  there are \(N\) cycles in \(X_i\)

  e.g. \texttt{cycle(2, [2,1,5,3,4])} as we have the 2 cycles
  \((1)\rightarrow(2)\rightarrow(1)\) and \((3)\rightarrow(5)\rightarrow(4)\rightarrow(3)\)

  Useful for routing problems (e.g. sending engineers out to repair phones)

- \texttt{circuit(succ)}: the values in \(\text{succ}\) form a
  hamiltonian cycle (i.e. the sequence \(1, \text{succ}[1], \text{succ}[\text{succ}[1]]\) etc form a loop through \(1..n\))
CP in Scheduling

• Concepts of jobs, resources and time
• Temporal constraints: “before”, “after”
• Resource requirement constraints: jobs requiring machines
• Capacity constraints

• Examples of model:

```plaintext
forall(j in Jobs)
    forall(t in 1..nbTasks-1)
        task[j,t] precedes task[j,t+1];

forall(j in Jobs)
    forall(t in Tasks)
        task[j,t] requires tool[resource[j,t]];`
CP in Scheduling

- Disjunctive constraints
- *Cumulative* global constraint
- Edge-finding (Baptiste and Le Pape 1996)
Scheduling constraints

- \text{cummulative}([S_1, \ldots, S_n], [D_1, \ldots, D_n], [E_1, \ldots, E_n], [H_1, \ldots, H_n], L)
- schedules \( n \) jobs, each with a height \( H_i \)
- \( i \)-th job starts at \( S_i \), runs for \( D_i \) and ends at \( E_i \)
  i.e. \( E_i = S_i + D_i \)
- at any time, accumulated height of running jobs is less than \( L \)
Perspectives

- Many solution techniques
  - Integer programming
  - Constraint programming
  - Local search
  - Combinations

- Which to use?
Hybrid Methods

• Local and Global Search
  – Use CP/IP for very large neighborhood search (take a solution, remove large subset, find optimal completion)

• Combining CP and IP
  – Use LP as constraint handler
  – Use CP as subproblem solver in branch and price
  – …..
Constraint Solvers

• Academic Solvers
  – ECLIPSE, SICStus, Minion, etc

• Commercial: ILOG
  – www.ilog.com
  – largest family of optimisation products as C++(Java) libraries
  – ILOG Solver provides basic constraint solving functionality
  – ILOG Scheduler is an add-on to the Solver with classes for scheduling objects
    • activities
    • resources;
    • scheduling constraints