Constraint Programming

Constraint Programming

- **Basic idea:** Programming with constraints, i.e. constraint solving embedded in a programming language
- **Constraints:** linear, non-linear, finite domain, Boolean, . . .
- **Programming:** logic, functional, object-oriented, imperative, concurrent, . . .
- **Systems:** Prolog III/IV, CHIP, ECLIPSE, ILOG, OCRE, NCL, . . .

Finite Domain Constraints

Constraint satisfaction problem (CSP)

- $n$ variables $x_1, \ldots, x_n$
- For each variable $x_j$ a *finite domain* $D_j$ of possible values, often $D_j \subseteq \mathbb{N}$.
- $m$ constraints $C_1, \ldots, C_m$, where $C_j \subseteq D_{i_1} \times \cdots \times D_{i_k}$ is a relation between $k_i$ variables $x_{i_1}, \ldots, x_{i_k}$. Write also $C_{i_1,\ldots,i_k}$.
- A *solution* is an assignment of a value $D_j$ to $x_j$, for each $j = 1, \ldots, n$, such that all relations $C_j$ are satisfied.

Coloring Problem

- Decide whether a map can be colored by 3 colors such that neighboring regions get different colors.
- For each region a variable $x_j$ with domain $D_j = \{\text{red, green, blue}\}$.
- For each pair of variables $x_i, x_j$ corresponding to two neighboring regions, a constraint $x_i \neq x_j$.
- NP-complete problem.

Resolution by Backtracking

- Instantiate the variables in some order.
- As soon as all variables in a constraint are instantiated, determine its truth value.
- If the constraint is not satisfied, backtrack to the last variable whose domain contains unassigned values, otherwise continue instantiation.

Efficiency Problems

*Mackworth 77*

1. If the domain $D_j$ of a variable $x_j$ contains a value $v$ that does not satisfy $C_j$, this will be the cause of repeated instantiation followed by immediate failure.
2. If we instantiate the variables in the order \( x_1, x_2, \ldots, x_n \), and for \( x_1 = v \) there is no value \( w \in D_j \), for \( j > i \), such that \( C_{ij}(v, w) \) is satisfied, then backtracking will try all values for \( x_j \), fail and try all values for \( x_{j-1} \) (and for each value of \( x_{j-1} \) again all values for \( x_j \)), and so on until it tries all combinations of values for \( x_{j+1}, \ldots, x_j \) before finally discovering that \( v \) is not a possible value for \( x_j \).

The identical failure process may be repeated for all other sets of values for \( x_1, \ldots, x_{i-1} \) with \( x_i = v \).

**Local Consistency**

- Consider CSP with unary and binary constraints only.
- **Constraint graph \( G \)**
  - For each variable \( x_i \) a node \( i \).
  - For each pair of variables \( x_i, x_j \) occurring in the same binary constraint, two arcs \((i, j)\) and \((j, i)\).
- The node \( i \) is consistent if \( C_i(v) \), for all \( v \in D_i \).
- The arc \((i, j)\) is consistent, if for all \( v \in D_i \) with \( C_i(v) \) there exists \( w \in D_j \) with \( C_j(w) \) such that \( C_{ij}(v, w) \).
- The graph is node consistent resp. arc consistent if all its nodes (resp. arcs) are consistent.

**Arc Consistency**

**Algorithm AC-3** (Mackworth 77):

```plaintext
begin
  for \( i \leftarrow 1 \) until \( n \) do \( D_i \leftarrow \{ v \in D_i \mid C_i(v) \} \);
  \( Q \leftarrow \{ (i, j) \mid (i, j) \in \text{arcs}(G), i \neq j \} \)
  while \( Q \) not empty do
    begin
      select and delete an arc \((i, j)\) from \( Q \);
      if \( \text{REVISE}(i, j) \) then
        \( Q \leftarrow Q \cup \{ (k, i) \mid (k, i) \in \text{arcs}(G), k \neq i, k \neq j \} \)
      end
  end
end```

**Arc Consistency** (2)

```plaintext
procedure \( \text{REVISE}(i, j) \):
begin
  \( \text{DELETE} \leftarrow \text{false} \)
  for each \( v \in D_i \) do
    if there is no \( w \in D_j \) such that \( C_{ij}(v, w) \) then
      begin
        delete \( v \) from \( D_i \);
        \( \text{DELETE} \leftarrow \text{true} \)
      end;
  return \( \text{DELETE} \)
end```

**Crossword Puzzle**
Dechter 92

Word List
Aft Laser
Ale Lee
Eel Line
Heel Sails
Hike Sheet
Hoses Steer
Keel Tie
Knot

Solution

1 Across 4 Across 7 Across 8 Across
Hoses 6 Laser 10 Heel 17 Aft 27
Sails 1 Hike 11 Keel 11 Laser 22
Sheet 9 Knot 6 Sails 22
Steer 3 Line 18 Tie 19

2 Down 3 Down 5 Down 6 Down
Hoses 4 Laser 7 Heel 14 Aft 29
Sails 25 Laser 8 Sails 12
Sheet 28 Knot 23
Steer 26 Line 16

Lookahead

Apply local consistency dynamically during search

- **Forward Checking**: After assigning to x the value v, eliminate for all uninstantiated variables y the values from \(D_y\) that are incompatible with v.

- **Partial Lookahead**: Establish arc consistency for all \((y, y')\), where y, y' have not been instantiated yet and y will be instantiated before y'.

- **Full Lookahead**: Establish arc consistency for all uninstantiated variables.

n-Queens Problem

Place n queens in an \(n \times n\) chessboard such that no two queens threaten each other.

- **Variables** \(x_i, i = 1, \ldots, n\) with domain \(D_i = \{1, \ldots, n\}\) indicating the column of the queen in line i.

- **Constraints**
  - \(x_i \neq x_j\), for \(1 \leq i < j \leq n\) (vertical)
- $x_i \neq x_j + (j - i)$, for $1 \leq i < j \leq n$ (diagonal 1)
- $x_i \neq x_j - (j - i)$, for $1 \leq i < j \leq n$ (diagonal 2)

**Forward Checking**

**Partial Lookahead**

**Full Lookahead**
Typical structure of a constraint program

- Declare the variables and their domains
- State the constraints
- Enumeration (labeling)

The constraint solver achieves only local consistency. In order to get global consistency, the domains have to be enumerated.

Labeling

- Assigning to the variables their possible values and constructing the corresponding search tree.

Important questions

1. In which order should the variables be instantiated (variable selection)?
2. In which order should the values be assigned to a selected variable (value selection)?

Static vs. dynamic orderings

Heuristics

Dynamic variable/value orderings

- Variable orderings
  - Choose the variable with the smallest domain “first fail”
  - Choose the variable with the smallest domain that occurs in most of the constraints “most constrained”
  - Choose the variable which has the smallest/largest lower/upper bound on its domain.

- Value orderings
  - Try first the minimal value in the current domain.
– Try first the maximal value in the current domain.
– Try first some value in the middle of the current domain.

### Constraint programming systems

<table>
<thead>
<tr>
<th>System</th>
<th>Avail.</th>
<th>Constraints</th>
<th>Language</th>
<th>Web site</th>
</tr>
</thead>
<tbody>
<tr>
<td>B-prolog</td>
<td>comm.</td>
<td>FinDom</td>
<td>Prolog</td>
<td><a href="http://www.probp.com">www.probp.com</a></td>
</tr>
<tr>
<td>CHIP</td>
<td>comm.</td>
<td>FinDom, Boolean, Linear $\mathbb{Q}$ Hybrid</td>
<td>Prolog, C, C++</td>
<td><a href="http://www.cosytec.com">www.cosytec.com</a></td>
</tr>
<tr>
<td>Choco</td>
<td>free</td>
<td>FinDom</td>
<td>Claire</td>
<td>choco-constraints.net</td>
</tr>
<tr>
<td>Eclipse</td>
<td>free non-profit</td>
<td>FinDom, Hybrid</td>
<td>Prolog</td>
<td><a href="http://www.icparc.ic.ac.uk/eclipse/">www.icparc.ic.ac.uk/eclipse/</a></td>
</tr>
<tr>
<td>GNU Prolog</td>
<td>free</td>
<td>FinDom</td>
<td>Prolog</td>
<td>gnu-prolog.inria.fr</td>
</tr>
<tr>
<td>IF/Prolog</td>
<td>comm.</td>
<td>FinDom, Boolean, Linear $\mathbb{R}$</td>
<td>Prolog</td>
<td><a href="http://www.ifcomputer.co.jp">www.ifcomputer.co.jp</a></td>
</tr>
<tr>
<td>ILOG</td>
<td>comm.</td>
<td>FinDom, Hybrid</td>
<td>C++, Java</td>
<td><a href="http://www.ilog.com">www.ilog.com</a></td>
</tr>
<tr>
<td>NCL</td>
<td>comm.</td>
<td>FinDom</td>
<td></td>
<td><a href="http://www.engine.st.com">www.engine.st.com</a></td>
</tr>
<tr>
<td>Mozart</td>
<td>free</td>
<td>FinDom</td>
<td>Oz</td>
<td><a href="http://www.mozart-oz.org">www.mozart-oz.org</a></td>
</tr>
<tr>
<td>Prolog IV</td>
<td>comm.</td>
<td>FinDom, nonlinear intervals</td>
<td>Prolog</td>
<td><a href="http://www.prologia.fr">www.prologia.fr</a></td>
</tr>
<tr>
<td>Sicstus</td>
<td>comm.</td>
<td>FinDom, Boolean, linear $\mathbb{R}/\mathbb{Q}$</td>
<td>Prolog</td>
<td><a href="http://www.sics.se/sicstus/">www.sics.se/sicstus/</a></td>
</tr>
</tbody>
</table>

### Integer vs. constraint programming

**Practical Problem Solving**

- Model building : Language
- Model solving : Algorithms

#### IP vs. CP : Language

<table>
<thead>
<tr>
<th></th>
<th>IP</th>
<th>CP</th>
</tr>
</thead>
<tbody>
<tr>
<td>Variables</td>
<td>0-1</td>
<td>Finite domain</td>
</tr>
<tr>
<td>Constraints</td>
<td>Linear equations and inequalities</td>
<td>Arithmetic constraints Symbolic/global constraints</td>
</tr>
</tbody>
</table>

**Example**

- Variables : $x_1, \ldots, x_n \in \{0, \ldots, m - 1\}$
- Constraint : Pairwise different values
Example (2)

- Integer programming: Only linear equations and inequalities
  \[ x_i \neq x_j \iff x_i < x_j \lor x_i > x_j \]
  \[ x_i \leq x_j - 1 \lor x_i \geq x_j + 1 \]

- Eliminating disjunction
  \[ x_i - x_j + 1 \leq my_i, \quad x_j - x_i + 1 \leq my_j, \quad y_i + y_j = 1, \]
  \[ y_i, y_j \in \{0, 1\}, \quad 0 \leq x_i, x_j \leq m - 1, \]

- New variables: \( z_{ik} = 1 \) iff \( x_i = k \), \( i = 1, \ldots, n \), \( k = 0, \ldots, m - 1 \)
  \[ z_{i0} + \cdots + z_{im-1} = 1, \quad z_{1k} + \cdots + z_{nk} \leq 1, \]

- Constraint programming \( \rightarrow \) symbolic constraint
  \[ \text{alldifferent}(x_1, \ldots, x_n) \]

Symbolic/global constraints

- \( \text{alldifferent}([x_1, \ldots, x_n]) \)
- \( \text{cumulative}([s_1, \ldots, s_n], [d_1, \ldots, d_n], [r_1, \ldots, r_n], c, e). \)
  - \( n \) tasks: starting time \( s_i \), duration \( d_i \), resource demand \( r_i \)
  - resource capacity \( c \), completion time \( e \).

Diffn Constraint

- Nonoverlapping of \( n \)-dimensional rectangles \([O_1, \ldots, O_n, L_1, \ldots, L_n]\), where \( O_i \) resp. \( L_i \) denotes the origin resp. length in dimension \( i \)
  - \( \text{diffn}([O_{11}, \ldots, O_{1n}, L_{11}, \ldots, L_{1n}], \ldots, [O_{mn}, L_{mn}, \ldots, L_{mn}]) \)
• General form: `diffn(Rectangles, Min_Vol, Max_Vol, End, Distances, Regions)`

**IP vs. CP : Algorithms**

<table>
<thead>
<tr>
<th></th>
<th>IP</th>
<th>CP</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Inference</strong></td>
<td>Linear programming</td>
<td>Domain filtering</td>
</tr>
<tr>
<td></td>
<td>Cutting planes</td>
<td>Constraint propagation</td>
</tr>
<tr>
<td><strong>Search</strong></td>
<td>Branch-and-relax</td>
<td>Branch-and-bound</td>
</tr>
<tr>
<td></td>
<td>Branch-and-cut</td>
<td></td>
</tr>
<tr>
<td><strong>Bounds on</strong></td>
<td>Two-sided</td>
<td>One-sided</td>
</tr>
<tr>
<td>the objective</td>
<td></td>
<td></td>
</tr>
<tr>
<td>function</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Local vs. global reasoning**

Linear arithmetic constraints

\[
\begin{align*}
3x + y & \leq 7, \\
3y + x & \leq 7, \\
x + y & = z, \\
x, y & \in \{0, \ldots, 3\}
\end{align*}
\]

<table>
<thead>
<tr>
<th></th>
<th>CP</th>
<th>LP</th>
<th>IP</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td><code>x, y \leq 2, z \leq 4</code></td>
<td><code>x, y \leq 2, z \leq 3.5</code></td>
<td><code>x, y \leq 2, z \leq 3</code></td>
</tr>
</tbody>
</table>

Global reasoning in CP ? ⇞ global constraints !

**Global reasoning in CP**

**Example**

- \(x_1, x_2, x_3 \in \{0, 1\}\)
- pairwise different values
- **Local** consistency: 3 disequalities: \(x_1 \neq x_2, x_1 \neq x_3, x_2 \neq x_3\)
  \(\leadsto x_1, x_2, x_3 \in \{0, 1\}\), i.e., no domain reduction is possible
- **Global** constraint: `alldifferent(x_1, x_2, x_3)`
  \(\leadsto\) detects infeasibility (uses bipartite matching)

Global reasoning in CP : inside global constraints

**Summary**
ILP | CP(FD)
---|---
**Language** | Linear arithmetic | Arithmetic constraints
 | — | Symbolic constraints
**Algorithms** | Global consistency (LP) | Local consistency
 | Cutting planes | Domain reduction
 | Branch-and-bound | User-defined enumeration
 | Branch-and-cut |

- Symbolic constraints $\rightsquigarrow$ more expressivity + more efficiency

- Unifying framework for CP and IP: *Branch-and-infer* (Bockmayr/Kasper 98)

### Discrete Tomography

- Binary matrix with $m$ rows and $n$ columns
  - Horizontal projection numbers ($h_1, \ldots, h_m$)
  - Vertical projection numbers ($v_1, \ldots, v_n$)

**Properties**

- Horizontal convexity (h)
- Vertical convexity (v)
- Connectivity (polyomino) (p)

**Complexity** (Woeginger’01)

- polynomial: (h), (p,v,h)
- NP-complete: (p,v), (p,h), (v,h), (v), (h), (p)

#### IP Model

- **Variables** $x_{ij} = \begin{cases} 0 & \text{cell}(i,j) \text{ is labeled white} \\ 1 & \text{cell}(i,j) \text{ is labeled black} \end{cases}$

- **Constraints I:** Projections
  $$\sum_{j=1}^{n} x_{ij} = h_i, \quad \sum_{i=1}^{m} x_{ij} = v_j$$

- **Constraints II:** Convexity
  $$h_i x_{ik} + \sum_{l=k+1}^{n} x_{il} \leq h_i, \quad v_j x_{kj} + \sum_{l=k+1}^{m} x_{lj} \leq v_j,$$

**IP Model (contd)**
• **Constraints III: Connectivity**

\[
j + h - 1 - j + h - 1 \sum_{k=j}^{h-1} x_{ik} + h - 1 \sum_{k=j}^{h-1} x_{i(k+1)k} \leq h - 1
\]

- Various linear arithmetic models possible, e.g. convexity
- Enormous differences in size and running time, e.g. 1 day vs. < 1 sec
- Large number of constraints (∼ 3mn in the above model)

### Finite Domain Model

#### Variables

- \(x_i\) start of horizontal convex block in row \(i\), for \(1 \leq i \leq m\)
- \(y_j\) start of vertical convex block in column \(j\), for \(1 \leq j \leq n\)

#### Domain

- \(x_i \in [1, \ldots, n - h + 1]\), for \(1 \leq i \leq m\)
- \(y_j \in [1, \ldots, m - v + 1]\), for \(1 \leq j \leq n\)

### Conditional Propagation

- **Compatibility of** \(x_i\) and \(y_j\)

\[x_i \leq j < x_i + h_i \iff y_j \leq i < y_j + v_j\]

for \(1 \leq i \leq m\) and \(1 \leq j \leq n\)

### Conditional propagation

\[\text{if } x_i \leq j \text{ then } (\text{if } j < x_i + h_i \text{ then } (y_j \leq i, i < y_j + v_j))\]
• Connectivity

\[
\begin{align*}
row \ i & \quad 1 & x_i & x_i + h_i - 1 & n \\
row \ i+1 & \quad 1 & x_{i+1} & x_{i+1} + h_{i+1} - 1 & n
\end{align*}
\]

• Block \( i \) must start before the end of block \( i+1 \)

\[
x_i \leq x_{i+1} + h_{i+1} - 1, \text{ for } 1 \leq i \leq m - 1
\]

• Block \( i + 1 \) must start before the end of block \( i \)

\[
x_{i+1} \leq x_i + h_i - 1, \text{ for } 1 \leq i \leq m - 1
\]

**Cumulative**

2d and 3d Diffn Model