Constraint Programming

Element Constraints
(version of 7 November 2022)
Quadratic Assignment Problem (QAP)

Decision:
Where to place each facility?

Input: Distance between any two locations

Input: Weight between any two facilities (e.g., amount of traffic)
Quadratic Assignment Problem (QAP)

Input: Distance between any two locations

Input: Weight between any two facilities (e.g., amount of traffic)

Decision: Where to place each facility?
Problem:
Assign all facilities to different locations (let $x_i$ denote the location of facility $i$), minimizing

$$\sum_{i,j} D_{x_i,x_j} \cdot W_{i,j}$$
Problem: Assign all facilities to different locations (let $x_i$ denote the location of facility $i$), minimizing

$$\sum_{i,j} D_{x_i,x_j} \cdot W_{i,j}$$

2D element constraint: 2D array indexed by two variables.
Quadratic Assignment Model

Solver cp = makeSolver();
IntVar[] x = makeIntVarArray(cp, n, n);

cp.post(allDifferent(x));

// build the objective function
IntVar[] weightedDist = new IntVar[n * n];
for (int k = 0, i = 0; i < n; i++) {
    for (int j = 0; j < n; j++) {
        weightedDist[k] = mul(element(d, x[i], x[j]), w[i][j]);
        k++;
    }
}
IntVar totCost = sum(weightedDist);
Objective obj = cp.minimize(totCost);
Element2D(int[][] T, IntVar x, IntVar y, IntVar z)

- $T[x][y] = z$

<table>
<thead>
<tr>
<th></th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
<td>8</td>
<td>9</td>
<td>6</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>9</td>
<td>2</td>
<td>4</td>
</tr>
<tr>
<td>2</td>
<td>9</td>
<td>8</td>
<td>9</td>
<td>8</td>
</tr>
<tr>
<td>3</td>
<td>1</td>
<td>9</td>
<td>2</td>
<td>5</td>
</tr>
</tbody>
</table>

- How to create an efficient propagator for Element2D?
- We don’t want to create holes in $D(z)$, but holes are fine in $D(x)$ and $D(y)$. 
\( T[x][y] = z \)

<table>
<thead>
<tr>
<th>x</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>rSup</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
<td>8</td>
<td>9</td>
<td>6</td>
<td>4</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>9</td>
<td>2</td>
<td>4</td>
<td>4</td>
</tr>
<tr>
<td>2</td>
<td>9</td>
<td>8</td>
<td>9</td>
<td>8</td>
<td>4</td>
</tr>
<tr>
<td>3</td>
<td>1</td>
<td>9</td>
<td>2</td>
<td>5</td>
<td>4</td>
</tr>
<tr>
<td>cSup</td>
<td>4</td>
<td>4</td>
<td>4</td>
<td>4</td>
<td></td>
</tr>
</tbody>
</table>

- \( D(x) = \{0, 1, 2, 3\} \)
- \( D(y) = \{0, 1, 2, 3\} \)
- \( D(z) = [1..9] \) (interval domain)

<table>
<thead>
<tr>
<th>z</th>
<th>x</th>
<th>y</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>3</td>
<td>0</td>
</tr>
<tr>
<td>2</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>2</td>
<td>3</td>
<td>2</td>
</tr>
<tr>
<td>4</td>
<td>1</td>
<td>3</td>
</tr>
<tr>
<td>5</td>
<td>3</td>
<td>3</td>
</tr>
<tr>
<td>6</td>
<td>0</td>
<td>3</td>
</tr>
<tr>
<td>8</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>8</td>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td>8</td>
<td>2</td>
<td>3</td>
</tr>
<tr>
<td>9</td>
<td>0</td>
<td>2</td>
</tr>
<tr>
<td>9</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>9</td>
<td>2</td>
<td>0</td>
</tr>
<tr>
<td>9</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>9</td>
<td>3</td>
<td>1</td>
</tr>
</tbody>
</table>

sorted

low

up
\[ T[x][y] = z \]

- \( \mathcal{D}(x) = \{0,1,2,3\} \)
- \( \mathcal{D}(y) = \{0,1,2,3\} \)
- Assume \( \mathcal{D}(z) = [1..7] \) (interval domain)
\[ T[x][y] = z \]

\[ D(x) = \{0, 1, 2, 3\} \]

\[ D(y) = \{0, 1, 2, 3\} \]

\[ D(z) = [1..7] \text{ (interval domain)} \]
\[ T[x][y] = z \]

- \( D(x) = \{0,1,2,3\} \)
- \( D(y) = \{0,1,2,3\} \)
- \( D(z) = [1..7] \) (interval domain)

<table>
<thead>
<tr>
<th>x</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>rSup</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
<td>8</td>
<td>9</td>
<td>6</td>
<td>4</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>9</td>
<td>2</td>
<td>4</td>
<td>4</td>
</tr>
<tr>
<td>2</td>
<td>9</td>
<td>8</td>
<td>9</td>
<td>8</td>
<td>3</td>
</tr>
<tr>
<td>3</td>
<td>1</td>
<td>9</td>
<td>2</td>
<td>5</td>
<td>3</td>
</tr>
<tr>
<td>cSup</td>
<td>4</td>
<td>3</td>
<td>3</td>
<td>4</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>z</th>
<th>x</th>
<th>y</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>3</td>
<td>0</td>
</tr>
<tr>
<td>2</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>2</td>
<td>3</td>
<td>2</td>
</tr>
<tr>
<td>4</td>
<td>1</td>
<td>3</td>
</tr>
<tr>
<td>5</td>
<td>3</td>
<td>3</td>
</tr>
<tr>
<td>6</td>
<td>0</td>
<td>3</td>
</tr>
<tr>
<td>8</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>8</td>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td>8</td>
<td>2</td>
<td>3</td>
</tr>
<tr>
<td>9</td>
<td>0</td>
<td>2</td>
</tr>
<tr>
<td>9</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

\( \text{low} \)

\( \text{up} \)
\[ T[x][y] = z \]

- \( D(x) = \{0,1,2,3\} \)
- \( D(y) = \{0,1,2,3\} \)
- \( D(z) = [1..7] \) (interval domain)
\[ T[x][y] = z \]

- **D(x)** = \{0, 1, 2, 3\}
- **D(y)** = \{0, 1, 2, 3\}
- **D(z)** = [1..7] (interval domain)
$T[x][y] = z$

- $D(x) = \{0, 1, 2, 3\}$
- $D(y) = \{0, 1, 2, 3\}$
- $D(z) = [1..7]$ (interval domain)
\( T[x][y] = z \)

- **D(x)** = \{0, 1, 2, 3\}
- **D(y)** = \{0, 1, 2, 3\}
- **D(z)** = \([1..7]\) (interval domain)
$T[x][y] = z$

- $D(x) = \{0,1,2,3\}$
- $D(y) = \{0,1,2,3\}$
- $D(z) = [1..7]$ (interval domain)
$T[x][y] = z$

- $D(x) = \{0, 1, 2, 3\}$
- $D(y) = \{0, 1, 2, 3\}$
- $D(z) = [1..6, 7]$ (interval domain)
\[ T[x][y] = z \]

- \( D(x) = \{0, 1, 3\} \)
- \( D(y) = \{0, 2, 3\} \)
- Assume \( D(z) = [2..6] \) (interval domain)
$T[x][y] = z$

- $D(x) = \{0, 1, 3\}$
- $D(y) = \{0, 2, 3\}$
- $D(z) = [2..6]$ (interval domain)
\( T[x][y] = z \)

- **D(x) = \{0,1,3\}**
- **Assume D(y) = \{2,3\}**
- **D(z) = [2..6] (interval domain)**
\[ T[x][y] = z \]

- **D(x) = \{0,1,3\}**
- **D(y) = \{2,3\}**
- **D(z) = [2..6] (interval domain)**
$T[x][y] = z$

- $D(x) = \{0, 1, 3\}$
- $D(y) = \{2, 3\}$
- $D(z) = [2, 3, 4, 6]$ (interval domain)
Why do we only need to restore these values on backtrack?
public class Element2D extends AbstractConstraint {

    private final int[][] T;
    private final IntVar x, y, z;
    private final int n, m;
    private final StateInt[] rSup;
    private final StateInt[] cSup;

    private final StateInt low;
    private final StateInt up;
    private final ArrayList<Triple> zxy;

    @Override
    public void post() {
        ... // some init
        x.propagateOnDomainChange(this);
        y.propagateOnDomainChange(this);
        z.propagateOnBoundChange(this);
        propagate();
    }
}
private void updateSupports(int lostPos) {
  if (rSup[zxy.get(lostPos).x].decrement() == 0) {
    x.remove(zxy.get(lostPos).x);
  }
  if (cSup[zxy.get(lostPos).y].decrement() == 0) {
    y.remove(zxy.get(lostPos).y);
  }
}

public void propagate() {
  int l = low.value(), u = up.value();
  int zMin = z.min(), zMax = z.max();

  while (zxy.get(l).z <= zMin || !x.contains(zxy.get(l).x) || !y.contains(zxy.get(l).y)) {
    updateSupports(l++);
    if (l > u) throw new InconsistencyException();
  }
  z.removeBelow(zxy.get(l).z);
  low.setValue(l);
  // similarly for up
}
Element 1D Domain Consistency (Exercise)

- \( T[y] = z, D(z) = \{3, 4, 5\} \)

\[
\begin{array}{c|cccccc}
D(y) & 0 & 1 & 2 & 3 & 4 & 5 \\
\hline
T & 3 & 4 & 5 & 5 & 4 & 3 \\
\end{array}
\]

For each value \( v \) in \( D(z) \), set \( z_{\text{Sup}}(v) \leftarrow | \{ i \in D(y) : T[i] = z \} | \)

Filtering: \( z_{\text{Sup}}(v) = 0 \Rightarrow D(z) \leftarrow D(z) \setminus \{ v \} \)

- Filtering: \( T[i] \notin D(z) \Rightarrow D(y) \leftarrow D(y) \setminus \{ i \} \)
T[y]=z, assume $D(z) = \{3, 4, 5\}$

<table>
<thead>
<tr>
<th>D(y)</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
</tr>
</thead>
<tbody>
<tr>
<td>3</td>
<td>4</td>
<td>5</td>
<td>5</td>
<td>4</td>
<td>3</td>
<td></td>
</tr>
</tbody>
</table>

| T    | 3 | 4 | 5 | 5 | 4 | 3 |

Filtering: $T[i] \not\in D(z) \Rightarrow D(y) \leftarrow D(y) \setminus \{i\}$

$z_{Sup}$

<table>
<thead>
<tr>
<th>2</th>
<th>2</th>
<th>2</th>
</tr>
</thead>
</table>

For each value $v$ in $D(z)$, set

$z_{Sup}(v) \leftarrow |\{i \in D(y) : T[i] = z\}|$

Filtering: $z_{Sup}(v) = 0 \Rightarrow D(z) \leftarrow D(z) \setminus \{v\}$

$D(z)$

| 3 | 4 | 5 |
Element1D Domain Consistency

T[y]=z, D(z)={3,4,5}

For each value v in D(z), set
zSup(v) ← |{ i in D(y) : T[i] = z }|
Filtering: zSup(v) = 0 ⇒ D(z) ← D(z) \ { v }

Filtering: T[i] ∉ D(z) ⇒ D(y) ← D(y) \ { i }
Element1D Domain Consistency

T[y]=z, D(z)={3,4,5}

For each value v in D(z), set
\[ z_{Sup}(v) \leftarrow | \{ i \in D(y) : T[i] = z \} | \]

Filtering: \( z_{Sup}(v) = 0 \Rightarrow D(z) \leftarrow D(z) \setminus \{ v \} \)

Filtering: \( T[i] \not\in D(z) \Rightarrow D(y) \leftarrow D(y) \setminus \{ i \} \)
\[ T[y] = z, \ D(z) = \{4, 5\}, \text{ assume 1 and 4 are removed from } D(y): \]

\[
\begin{array}{cccccc}
D(y) & 0 & 1 & 2 & 3 & 4 & 5 \\
\hline
T & 3 & 4 & 5 & 5 & 4 & 3 \\
\end{array}
\]

For each value \( v \) in \( D(z) \), set
\[
z_{\text{Sup}}(v) \leftarrow | \{ i \in D(y) : T[i] = z \} |
\]
Filtering: \[ z_{\text{Sup}}(v) = 0 \Rightarrow D(z) \leftarrow D(z) \setminus \{ v \} \]
**Element1D Domain Consistency**

- $T[y]=z$, $D(z)=$\{4,5\}, assume 1 and 4 are removed from $D(y)$:

<table>
<thead>
<tr>
<th></th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
</tr>
</thead>
<tbody>
<tr>
<td>$D(y)$</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>$T$</td>
<td>3</td>
<td>4</td>
<td>5</td>
<td>5</td>
<td>4</td>
<td>3</td>
</tr>
</tbody>
</table>

**zSup**

<table>
<thead>
<tr>
<th></th>
<th>2</th>
<th>0</th>
<th>2</th>
</tr>
</thead>
<tbody>
<tr>
<td>$zSup$</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>$D(z)$</td>
<td>3</td>
<td>4</td>
<td>5</td>
</tr>
</tbody>
</table>

**Filtering:** $T[i] \not\in D(z) \Rightarrow D(y) \leftarrow D(y) \setminus \{i\}$

**For each value $v$ in $D(z)$, set**

$zSup(v) \leftarrow |\{i \in D(y) : T[i] = z\}|$

**Filtering:** $zSup(v) = 0 \Rightarrow D(z) \leftarrow D(z) \setminus \{v\}$
Stable Matching

George

Halle

Keira

Clive

Google

IBM

NASA

SAP
Stable Matching

Inputs, say for internships:
Inputs, say for internships:

– Every company provides a ranking of all the students.
Inputs, say for internships:

– Every company provides a ranking of all the students.
– Every student provides a ranking of all the companies.
A matching of student Halle with IBM is *stable* if:

- If IBM prefers another student, say George, over Halle, then George must prefer their matched company over IBM.
- If student Halle prefers another company, say NASA, over IBM, then NASA must prefer their matched student over Halle.
More precisely

- **Input:**
  - Given are $n$ students and $n$ companies, where each student (resp. company) has ranked each company (resp. student) with a unique number between 1 and $n$ in order of preference (the lower the number, the higher the preference), say for summer internships.

- **Problem:**
  - Match the students and companies such that there is no pair of a student and a company who would both prefer to be matched with each other than with their actually matched ones.
IBM prefers Keira over Clive (1 vs 2) (smaller number is higher preference). But Keira prefers their matched company (SAP) over IBM (2 vs 4).

**rankStudents[c,s]**

<table>
<thead>
<tr>
<th></th>
<th>George</th>
<th>Halle</th>
<th>Keira</th>
<th>Clive</th>
</tr>
</thead>
<tbody>
<tr>
<td>Google</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
</tr>
<tr>
<td>IBM</td>
<td>4</td>
<td>3</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>NASA</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
</tr>
<tr>
<td>SAP</td>
<td>3</td>
<td>4</td>
<td>1</td>
<td>2</td>
</tr>
</tbody>
</table>

**rankCompanies[s,c]**

<table>
<thead>
<tr>
<th></th>
<th>Google</th>
<th>IBM</th>
<th>NASA</th>
<th>SAP</th>
</tr>
</thead>
<tbody>
<tr>
<td>George</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
</tr>
<tr>
<td>Halle</td>
<td>2</td>
<td>1</td>
<td>3</td>
<td>4</td>
</tr>
<tr>
<td>Keira</td>
<td>1</td>
<td>4</td>
<td>3</td>
<td>2</td>
</tr>
<tr>
<td>Clive</td>
<td>4</td>
<td>3</td>
<td>1</td>
<td>2</td>
</tr>
</tbody>
</table>
Difficult Problem?

Not really:

```python
function stableMatching {
    Initialize all $m \in M$ and $w \in W$ to free
    while $\exists$ free man $m$ who still has a woman $w$ to propose to {
        $w =$ first woman on $m$’s list to whom $m$ has not yet proposed
        if $w$ is free
            $(m, w)$ become engaged
        else some pair $(m', w)$ already exists
            if $w$ prefers $m$ to $m'$
                $m'$ becomes free
                $(m, w)$ become engaged
            else
                $(m', w)$ remain engaged
    }
}
```

In 1962, David Gale and Lloyd Shapley proved that, for any equal number of men and women, it is always possible to solve the stable matching problem and make all matchings stable.

https://en.wikipedia.org/wiki/Gale-Shapley_algorithm#Algorithm
enum Students = {George, Halle, Keira, Clive};
enum Companies = {Google, IBM, NASA, SAP};

int rankCompanies[Students, Companies];
int rankStudents[Companies, Students];
...

var{Companies} company[Students];
var{Students} student[Companies];
Stable Matching

Data and variables

`enum Students = {George, Halle, Keira, Clive};
enum Companies = {Google, IBM, NASA, SAP};`

`int rankCompanies[Students, Companies];
int rankStudents[Companies, Students];
...

`var{Companies} company[Students];
var{Students} student[Companies];`

`rankCompanies[Halle, Google]` is the ranking of Google in Halle’s preferences.
enum Students = {George, Halle, Keira, Clive};
enum Companies = {Google, IBM, NASA, SAP};

int rankCompanies[Students, Companies];
int rankStudents[Companies, Students];
...
var{Companies} company[Students];
var{Students} student[Companies];

**rankCompanies[Halle, Google] is the ranking of Google in Halle’s preferences**

**rankStudents[Google, Halle] is the ranking of Halle in the preferences of Google**
Stable Matching

solve {
    forall(s in Students)
    student[company[s]] = s;
    forall(c in Companies)
    company[student[c]] = c;

    ...
}

Stable Matching

```plaintext
solve {
    forall(s in Students)
        student[company[s]] = s;
    forall(c in Companies)
        company[student[c]] = c;

    forall(s in Students, c in Companies)
        rankCompanies[s,c] < rankCompanies[s,company[s]]
        => rankStudents[c,student[c]] < rankStudents[c,s];
    forall(c in Companies, s in Students)
        rankStudents[c,s] < rankStudents[c,student[c]]
        => rankCompanies[s,company[s]] < rankCompanies[s,c];
}
```
Stable Matching

```
solve {
    forall(s in Students)
        student[company[s]] = s;
    forall(c in Companies)
        company[student[c]] = c;

    forall(s in Students, c in Companies)
        rankCompanies[s,c] < rankCompanies[s,company[s]]
        => rankStudents[c,student[c]] < rankStudents[c,s];
    forall(c in Companies, s in Students)
        rankStudents[c,s] < rankStudents[c,student[c]]
        => rankCompanies[s,company[s]] < rankCompanies[s,c];
}
```

`s prefers c over their company`
Stable Matching

```plaintext
solve {
  forall(s in Students)
    student[company[s]] = s;
  forall(c in Companies)
    company[student[c]] = c;

  forall(s in Students, c in Companies)
    rankCompanies[s,c] < rankCompanies[s,company[s]]
    => rankStudents[c,student[c]] < rankStudents[c,s];
  forall(c in Companies, s in Students)
    rankStudents[c,s] < rankStudents[c,student[c]]
    => rankCompanies[s,company[s]] < rankCompanies[s,c];
}
```
Stable Matching

solve {
    forall(s in Students)
        student[company[s]] = s;
    forall(c in Companies)
        company[student[c]] = c;
    forall(s in Students, c in Companies)
        rankCompanies[s,c] < rankCompanies[s,company[s]]
            => rankStudents[c,student[c]] < rankStudents[c,s];
    forall(c in Companies, s in Students)
        rankStudents[c,s] < rankStudents[c,student[c]]
            => rankCompanies[s,company[s]] < rankCompanies[s,c];
}
enum Students = {George,Halle,Keira,Clive};
enum Companies = {Google,IBM,NASA,SAP};
int rankCompanies[Students,Companies];
int rankStudents[Companies,Students];
...
var{CompanyManagers} company[Students];
var{Students} student[CompanyManagers];
solve {
  forall(s in Students)
    student[company[s]] = s;
  forall(c in Companies)
    company[student[c]] = c;

  forall(s in Students, c in Companies)
    rankCompanies[s,c] < rankCompanies[s,company[s]]
    => rankStudents[c,student[c]] < rankStudents[c,s];
  forall(c in Companies, s in Students)
    rankStudents[c,s] < rankStudents[c,student[c]]
    => rankCompanies[s,company[s]] < rankCompanies[s,c];
}
enum Students = {George, Halle, Keira, Clive};
enum Companies = {Google, IBM, NASA, SAP};
int rankCompanies[Students, Companies];
int rankStudents[Companies, Students];
...
var{CompanyManagers} company[Students];
var{Students} student[CompanyManagers];
solve {
    forall(s in Students)
        student[company[s]] = s;
    forall(c in Companies)
        company[student[c]] = c;

    forall(s in Students, c in Companies)
        rankCompanies[s, c] < rankCompanies[s, company[s]]
        => rankStudents[c, student[c]] < rankStudents[c, s];
    forall(c in Companies, s in Students)
        rankStudents[c, s] < rankStudents[c, student[c]]
        => rankCompanies[s, company[s]] < rankCompanies[s, c];
}
enum Students = {George, Halle, Keira, Clive};
enum Companies = {Google, IBM, NASA, SAP};
int rankCompanies[Students, Companies];
int rankStudents[Companies, Students];...

var{CompanyManagers} company[Students];
var{Students} student[CompanyManagers];
solve {
  forall(s in Students)
    student[company[s]] = s;
  forall(c in Companies)
    company[student[c]] = c;

  forall(s in Students, c in Companies)
    rankCompanies[s,c] < rankCompanies[s,company[s]]
    => rankStudents[c,student[c]] < rankStudents[c,s];
  forall(c in Companies, s in Students)
    rankStudents[c,s] < rankStudents[c,student[c]]
    => rankCompanies[s,company[s]] < rankCompanies[s,c];
}
Stable Matching

- Two interesting features:
  - Element constraint:
    - useful in many applications.
  - Logical combination of constraints.
Stable Matching

- Two interesting features:
  - Element constraint:
    - useful in many applications.
  - Logical combination of constraints.
- The Element constraint:
  - Ability to index an array/matrix with a variable or an expression containing variables.
Two interesting features:

- Element constraint:
  - Useful in many applications.
- Logical combination of constraints.

The Element constraint:

- Ability to index an array/matrix with a variable or an expression containing variables.

Logical combination of constraints:

- Can be handled by reification, for instance.
Element1DVar Constraint

- $T[y] = z$

- How to propagate efficiently?

- Two possibilities:
  - Domain consistency
  - Relaxed domain consistency
    - assume range domains for $D(z)$ and $D(T[i]) \ \forall i$
Element 1DVar: Relaxed Domain Consistency

- $T[y]=z$
  - $T = \{1,3\},[1..2],[1,9],[1,2,6]\}$
  - $y = \{0,1,3\}$
  - $z = \{4,7\}$

- Relaxed domains
  - $T = [[1..3],[1..2],[1..9],[1..6]]$
  - $y = \{0,1,3\}$
  - $z = [4..7]$
  - What can we remove and how?
Element1DVar: Relaxed Domain Consistency

- Relaxed domains
  - $T = [[1..3],[1..2],[1..9],[1..6]]$
  - $y = \{0,1,3\}$
  - $z = [4..7]$

- Filtered domains
  - $T = [[1..3],[1..2],[1..9],[4..6]]$
  - $y = \{3\}$
  - $z = [4..6]$
Relaxed Domain Consistency: Filtering Rules

\[ \forall i \in D(y) \]
\[ \quad \text{– } \min(D(T[i])) > \max(D(z)) \Rightarrow D(y) \leftarrow D(y) \setminus \{i\} \]
\[ \quad \text{– } \max(D(T[i])) < \min(D(z)) \Rightarrow D(y) \leftarrow D(y) \setminus \{i\} \]
\[ \min(D(z)) \leftarrow \max(\min(D(z), \min_{i \in D(y)} \min(T[i]))) \]
\[ \max(D(z)) \leftarrow \min(\max(D(z), \max_{i \in D(y)} \max(T[i]))) \]
\[ |D(y)| = 1 \Rightarrow T[y] = z \text{ (equality constraint)} \]

– Note that a T[i] variable domain can only be filtered under that condition
Element1DVar: Domain Consistency

- $T[y] = z$
  - $T = \{1,6\}, [1..2], \{1,9\}, \{1,2,6\}$
  - $y = \{0,1,2,3\}$
  - $z = \{4,6,7\}$

- After (non-relaxed) domain-consistent filtering
  - $T = \{1,6\}, [1..2], \{1,9\}, \{1,2,6\}$
  - $y = \{0,3\}$
  - $z = \{6\}$
Domain Consistency: Filtering Rules

- \( \forall i \in D(y) \)
  \(- D(T[i]) \cap D(z) = \emptyset \Rightarrow D(y) \leftarrow D(y) \setminus \{i\} \)

- \( \forall v \in D(z) \)
  \(- \exists i \in D(y) : v \in D(T[i]) \Rightarrow D(z) \leftarrow D(z) \setminus \{v\} \)

- \(|D(y)|=1 \Rightarrow T[y]=z \) (equality constraint)
  \(- a \ T[i] \) variable domain can only be filtered under that condition

Can be quite slow to compute: intersection with sparse-set domains?

How do you propose to implement this to run efficiently?
DC Filtering: residue = support caching

∀i ∈ D(y)  

\[ D(T[i]) \cap D(z) = \emptyset \Rightarrow D(y) \leftarrow D(y)\{i\} \]

If we find for a value i ∈ D(y) some value v such that v ∈ D(T[i]) and v ∈ D(z), then remember it. Let us call it supportT[i].

There is a high chance that, on the next propagate, this value is still preventing the removal D(y) ← D(y)\{i\}.

But if supportT[i] ∉ D(T[i]), then one needs to look for a new support, if not possibly perform D(y) ← D(y)\{i\}.

O(1) check if the support is still valid, else O(|D(T[i])|).
DC Filtering: residue = support caching

- $\forall v \in D(z)$
  - $\not\exists i \in D(y) : v \in D(T[i]) \Rightarrow D(z) \leftarrow D(z) \setminus \{v\}$

- Again, the same residue idea: assume $v$ cannot be removed. Then we can store an index $i$ such that $v \in D(T[i])$: call it $\text{support}_z[v]$.  

- There is a high chance that, on the next propagate, $v \in D(T[\text{support}_z[v]])$, and in this case we cannot remove $v$ from $D(z)$.  

- Otherwise, look for a new support.  

- $O(1)$ check if the support is still valid, else $O(|D(y)|)$. 

Remark about Residues

- Residues remain valid on backtrack (no need for reversibles) because a support for the constraint in a node of the search tree is also a support for the ancestors of that node (do you see why?)
How to implement \( \text{cp.post}(x > y \Rightarrow w < z) \)?

Easy: \((x > y)\) can be reified \( b_1: \text{BoolVar} \equiv x > y \)

Same for \((w < z)\): \( b_2: \text{BoolVar} \equiv w < z \)

\[ b_1 \Rightarrow b_2 \equiv (!b_1 \text{ or } b_2) \equiv (1 - b_1) + b_2 \geq 1 \text{ (views, sum, etc)} \equiv b_1 \leq b_2 \]

In order to make it even more general, we have

```java
public void post(BoolVar b) throws InconsistencyException {
    b.fix(true);
    fixPoint();
}
```

This way you can post arbitrary logical expressions in MiniCP