DM841

Discrete Optimization

Part II

Lecture 5

Global Constraints

Marco Chiarandini

Department of Mathematics & Computer Science
University of Southern Denmark

Outline

In Gecode: http://www.gecode.org/doc-latest/reference/group__TaskModelInt.html

In Minizinc: from the root of the minizinc installation:

```
lib/minizinc/std/globals.mzn
gnome-open doc/index.html
```

Integer Variables


```
IntVar x(home, 1, 4);
IntVar y(x);
```

```
IntVar x(home, 1, 4);
IntVar y;
y=x;
```

In both cases y is not allocating a new data structure but it is a reference to the data structure of x

Overloaded operator:

```
std::cout << x <<std::endl;
```

Access the domain of the variables via iterator:

```
for (IntVarValues i(x); i(); ++i)
std::cout << i.val() << ' ';
```

Access the ranges via iterator:

```
for (IntVarRanges i(x); i(); ++i)
std::cout << i.min() << ".." << i.max() << ' ';
```

Variable Interface

assigned(), update()

```
IntVar x(home, 0, 0);
rel(home, x, IRT_NQ, 0);
home.status();
```

Variables never reach empty domains, not either when the status is failed.

status() can be good for debugging purposes: check at the root node

```
int main(int argc, char* argv[]) {
   Options opt("SEND+MORE=MONEY");
   opt.parse(argc, argv);
   Money* m = new Money(opt);
   SpaceStatus status = m->status();
   if (status == SS_FAILED)
    cout << "Status: " << m->status() << " the space is failed at root."<< endl;
   else if (status == SS_SOLVED)
    cout << "Status: " << m->status()
    << " the space is not failed but the space has no brancher left."<< endl;
   else if (status == SS_BRANCH)
    cout << "Status: " << m->status()
    << " the space is not failed and we need to start branching."<< endl;
   m->print(cout);
   DFS<Money> e(m);
   while (Money* s = e.next()) {
    s->print(cout);
    delete s;
   }
   delete m;
   return 0;}
```

Arrays of Variables


```
IntVarArray x(home, 4, -10, 10);
```

```
IntVarArray x(home, 4); // does not create the array
for (int i=0; i<4; i++)
x[i] = IntVar(home, -10, 10);
```

Variables are only deleted when the space is deleted.

Matrix Interface


```
IntVarArgs x(n*m);

Matrix<IntVarArgs> mat(x, n, m);

IntVar mij = mat(i,j);
```

Argument Arrays

For:

- dynamically builded arrays
- temporary variables
- arguments for post functions

They allocate memory from the heap and the memory is freed when their desctructor is executed. (They cannot be updated.)

```
IntVarArgs x;
IntVarArgs x(5);
IntVarArgs x(home,5,0,10);
```

```
IntVarArgs x;
x << IntVar(home,0,10);
IntVarArgs y;
y << IntVar(home,10,20);
y << x;
linear(home, IntVarArgs()<<x[0]<<x[1], IRT_EQ, 0);
```

Concatenation:

```
IntVarArgs z = x+y;
```

Slices

```
IntVarArgs x(home, 10, 0, 10);
```

```
x.slice(5) // returns an array with elements x[5],x[6], . . . ,x[9]
x.slice(5,1,3) // returns x[5],x[6],x[7]
x.slice(5,-1) // returns x[5],x[4], . . . ,x[0]
x.slice(3,3) // returns x[3],x[6],x[9] .
x.slice(8,-2) // returns x[8],x[6],x[4],x[2],x[0]
x.slice(8,-2,3) // returns x[8],x[6],x[4]
```

```
IntArgs::create(n,start,inc)
```

Outline

domain and member

```
IntArgs a(4, 1,-3,5,-7)
IntSet d(a);
dom(home, x, d);
```

```
member(home, x, y)
```

$$y \in \{x_1, \ldots, x_n\}$$

Arithmetic Constraints

```
linear(home, a, x, IRT_EQ, c);
```

```
rel(home, x+2*sum(z) < 4*y);
```

```
SendMoreMoney(void) : l(*this, 8, 0, 9) {
  IntVar s(l[0]), e(l[1]), n(l[2]), d(l[3]),
         m(l[4]), o(l[5]), r(l[6]), y(l[7]);
  rel(*this, s != 0);
  rel(*this, m != 0);
  distinct(*this, l);
  rel(*this,             1000*s + 100*e + 10*n + d
                       + 1000*m + 100*o + 10*r + e
            == 10000*m + 1000*o + 100*n + 10*e + y);
  branch(*this, l, INT_VAR_SIZE_MIN(), INT_VAL_MIN());
}
```

Watch CP-2 of Van Hentenryck

Arithmetic Constraints

post function	constraint posted	bnd	dom	GCCAT
`min(home, x, y, z);`	$\min(x,y) = z$	✓	✓	minimum
`max(home, x, y, z);`	$\max(x,y) = z$	✓	✓	maximum
`abs(home, x, y);`	$\lvert x\rvert = y$	✓	✓	abs_value
`mult(home, x, y, z);`	$x \cdot y = z$	✓	✓	
`sqr(home, x, y);`	$x^2 = y$	✓	✓	
`sqrt(home, x, y);`	$\lfloor\sqrt{x}\rfloor = y$	✓	✓	
`pow(home, x, n, y);`	$x^n = y$	✓	✓	
`nroot(home, x, n, y);`	$\lfloor\sqrt[n]{x}\rfloor = y$	✓	✓	
`div(home, x, y, z);`	$x \div y = z$	✓		
`mod(home, x, y, z);`	$x \bmod y = z$	✓		
`divmod(home, x, y, d, m);`	$x \div y = d \land x \bmod y = m$	✓		

Global Constraint: alldifferent

Global constraint:

set of more elementary constraints that exhibit a special structure when considered together.

alldifferent constraint

Let $x_1, x_2, \ldots, x_n$ be variables. Then:

$$\texttt{alldifferent}(x_1, ..., x_n) = \{(d_1, ..., d_n) \mid \forall i, d_i \in D(x_i), \quad \forall i \neq j,\ d_i \neq d_j\}.$$

Note: different notation and names used in the literature
In Gecode `distinct`
In Minizinc `all_different_int(array[int] of var int: x)`

Global Constraint: `table`

Extensioanl Constraints:
In Gecode: `TupleSet` + `extensional`

```
TupleSet t;
t.add(IntArgs(3, 0,0,0));
t.add(IntArgs(3, 0,1,0));
t.add(IntArgs(3, 1,0,0));
t.finalize();
```

```
BoolVarArray x(home, 3, 0, 1);
extensionl(home, x, t);
```

Later regular

Global Constraint: Sum

Sum constraint

Let $x_1, x_2, \ldots, x_n$ be variables. To each variable x_i, we associate a scalar $c_i \in \mathbb{Q}$. Furthermore, let z be a variable with domain $D(z) \subseteq \mathbb{Q}$. The `sum` constraint is defined as

$$\texttt{sum}([x_1, \ldots, x_n], z, c) = \left\{ (d_1, \ldots, d_n, d) \mid \forall i, d_i \in D(x_i), d \in D(z), d = \sum_{i=1,\ldots,n} c_i d_i \right\}.$$

In Gecode: `linear(home, x, IRT_GR, c)`

```
linear(Home home, const IntArgs &a, const IntVarArgs &x,
IntRelType irt, IntVar y, IntConLevel icl=ICL_DEF)
```

In Minizinc: `sum_pred`:

```
s = sum(i in index_set(x)) (coeffs[i]*x[i])
```

Reified constraints

- Constraints are in a big conjunction
- How about disjunctive constraints?

$$A + B = C \quad \vee \quad C = 0$$

or soft constraints?

- Solution: reify the constraints:

$$\begin{array}{l}(A + B = C \quad \Leftrightarrow \quad b_0) \quad \wedge \\ (C = 0 \quad \Leftrightarrow \quad b_1) \quad \wedge \\ (b_0 \quad \vee \quad b_1 \quad \Leftrightarrow \quad \mathit{true})\end{array}$$

- These kind of constraints are dealt with in efficient way by the systems
- Then if optimization problem (soft constraints) $\Rightarrow \min \sum_i b_i$

In Gecode:

- almost all constraints have a reified version.
- Full and half reification.

```
rel(home, x, IRT_EQ, y, eqv(b));
rel(home, x, IRT_EQ, y, imp(b));
rel(home, x, IRT_EQ, y, pmi(b));
```

Half reification:
One way implication instead of double way.

Posting Constraints in Gecode

- All post functions for constraints and branchers only accept variable argument arrays. A variable array is automatically casted to a variable argument array.
- All data structures passed are copied.
- Selecting the consistency level
  - `ICL_VAL`: value propagation
  - `ICL_BND`: bound consistency
  - `ICL_DOM`: domain consistency
  - `ICL_DEF`:default (constraint dependent)
    Eg: `linear`: achieves `ICL_BND` in $O(n)$ and `ICL_DOM` in $O(d^n)$

Example: Magic Sequence

A magic sequence of length n is a sequence of integers $x_0, \ldots, x_{n-1}$ between 0 and $n-1$, such that for all i in 0 to $n-1$, the number i occurs exactly x_i times in the sequence.

Example: $6, 2, 1, 0, 0, 0, 1, 0, 0, 0$ is a magic sequence since 0 occurs 6 times in it, 1 occurs twice, ...

```
IntVarArray s(home,n,0,n-1);
for (int k=0; k<=n-1; k++) {
  BoolVarArgs b(home, n, 0, 1);
  for (int i=0; i<=n-1; i++)
    rel(home, s[i], IRT_EQ, k, b[i]);
  linear(home, b, IRT_EQ, s[k]);
}
```

```
series[0] = (series[0]=0)+(series[1]=0)+(series[2]=0)+(series[3]=0)+(series[4]=0);
series[1] = (series[0]=1)+(series[1]=1)+(series[2]=1)+(series[3]=1)+(series[4]=1);
series[2] = (series[0]=2)+(series[1]=2)+(series[2]=2)+(series[3]=2)+(series[4]=2);
series[3] = (series[0]=3)+(series[1]=3)+(series[2]=3)+(series[3]=3)+(series[4]=3);
series[4] = (series[0]=4)+(series[1]=4)+(series[2]=4)+(series[3]=4)+(series[4]=4);
```

See video cp-3 for a development of the propagation arising from thes econstraints.

Global Constraint: Knapsack

Knapsack constraint

Rather than constraining the sum to be a specific value, the knapsack constraint states the sum to be within a lower bound l and an upper bound u, i.e., such that $D(z) = [l, u]$. The knapsack constraint is defined as

$$\text{knapsack}([x_1, \ldots, x_n], z, c) = \left\{(d_1, \ldots, d_n, d) \mid d_i \in D(x_i)\, \forall i, d \in D(z), d \leq \sum_{i=1,\ldots,n} c_i d_i\right\} \cap \left\{(d_1, \ldots, d_n, d) \mid d_i \in D(x_i)\, \forall i, d \in D(z), d \geq \sum_{i=1,\ldots,n} c_i d_i\right\}.$$

$$\min D(z) \leq \sum_{i=1,\ldots,n} c_i x_i \leq \max D(z)$$

In Gecode:
`linear(Home home, const IntArgs &a, const IntVarArgs &x, IntRelType irt, IntVar y, IntConLevel icl=ICL_DEF)`
In Minizinc: `s = sum(i in index_set(x)) (coeffs[i]*x[i])`

Global Constraint: cardinality

cardinality or gcc (global cardinality constraint)

Let $x_1, \ldots, x_n$ be assignment variables whose domains are contained in $\{v_1, \ldots, v_{n'}\}$ and let $\{c_{v_1}, \ldots, c_{v_{n'}}\}$ be count variables whose domains are sets of integers. Then

$$\texttt{cardinality}([x_1, ..., x_n],[c_{v_1}, ..., c_{v_{n'}}]) = \{(w_1, ..., w_n, o_1, ..., o_{n'}) \mid w_j \in D(x_j)\, \forall j, \texttt{occ}(v_i, (w_1, ..., w_n)) = o_i \in D(c_{v_i})\, \forall i\}.$$

(occ number of occurrences)

$\leadsto$ generalization of alldifferent

In Gecode: count

Magic Sequence Revised

```
MagicSequence(const SizeOptions& opt)
   : n(opt.size()), s(*this,n,0,n-1) {
   for (int i=n; i--; )
       count(*this, s, i, IRT_EQ, s[i]);
   linear(*this, s, IRT_EQ, n);
   linear(*this, IntArgs::create(n,-1,1), s, IRT_EQ, 0);
   branch(*this, s, INT_VAR_NONE(), INT_VAL_MAX());
}
```

$$\sum_{i=0}^{n-1} x_i = 0 \qquad \sum_{i=0}^{n-1} (i-1)x_i = 0$$

```
MagicSequence(const SizeOptions& opt)
   : n(opt.size()), s(*this,n,0,n-1) {
   count(*this, s, s, opt.icl());
   linear(*this, IntArgs::create(n,-1,1), s, IRT_EQ, 0);
   branch(*this, s, INT_VAR_NONE(), INT_VAL_MAX());
}
```

Global Constraint: among and sequence

among

Let $x_1, \ldots, x_n$ be a tuple of variables, S a set of variables, and l and u two nonnegative integers

$$\texttt{among}([x_1, ..., x_n], S, l, u)$$

At least l and at most u of variables take values in S.
In Gecode: `count`

sequence

Let $x_1, \ldots, x_n$ be a tuple of variables, S a set of variables, and l and u two nonnegative integers, s a positive integer.

$$\texttt{sequence}([x_1, ..., x_n], S, l, u, s)$$

At least l and at most u of variables take values from S in s consecutive variables

Car Sequencing Problem

Car Sequencing Problem

- an assembly line makes 50 cars a day
- 4 types of cars
- each car type is defined by options: {air conditioning, sun roof}

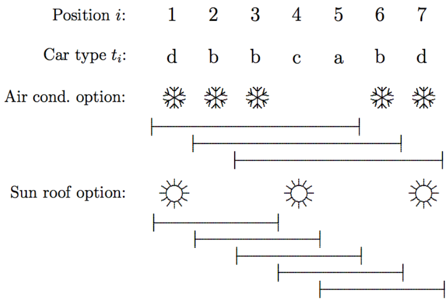
type	air cond.	sun roof	demand
a	no	no	20
b	yes	no	15
c	no	yes	8
d	yes	yes	7

- at most 3 cars in any sequence of 5 can be given air conditioning
- at most 1 in any sequence of 3 can be given a sun roof

Task: sequence the car types so as to meet demands while observing capacity constraints of the assembly line.

Car Sequencing Problem

Sequence constraints



Car Sequencing Problem: CP model

Car Sequencing Problem

Let t_i be the decision variable that indicates the type of car to assign to each position i in the sequence.

$$\begin{array}{ll}
\text{cardinality}([t_1, \ldots, t_{50}], (a, b, c, d), (20, 15, 8, 7), (20, 15, 8, 7)) & \\
\text{among}([t_i, \ldots, t_{i+4}], \{b, d\}, 0, 3), & \forall i = 1..46 \\
\text{among}([t_i, \ldots, t_{i+2}], \{c, d\}, 0, 1), & \forall i = 1..48 \\
t_i \in \{a, b, c, d\}, i = 1, \ldots, 50. &
\end{array}$$

Note: in Gecode `among` is `count`.
However, we can use `sequence` for the two `among` constraints above:

$$\text{sequence}([t_1, \ldots, t_{50}], \{b, d\}, 0, 3, 5),$$
$$\text{sequence}([t_1 1, \ldots, t_{50}], \{c, d\}, 0, 1, 3),$$

Car Sequencing Problem: MIP model

$$\begin{pmatrix} AC_i = 0 \\ SR_i = 0 \end{pmatrix} \vee \begin{pmatrix} AC_i = 1 \\ SR_i = 0 \end{pmatrix} \vee \begin{pmatrix} AC_i = 0 \\ SR_i = 1 \end{pmatrix} \vee \begin{pmatrix} AC_i = 1 \\ SR_i = 1 \end{pmatrix}$$

$$AC_i = AC_i^a + AC_i^b + AC_i^c + AC_i^d$$

$$SR_i = SR_i^a + SR_i^b + SR_i^c + SR_i^d$$

$$AC_i^a = 0, \quad AC_i^b = \delta_{ib}, \quad AC_i^c = 0, \quad AC_i^d = \delta_{id}$$

$$SR_i^a = 0, \quad SR_i^b = 0, \quad SR_i^c = \delta_{ic}, \quad SR_i^d = \delta_{id}$$

$$\delta_{ia} + \delta_{ib} + \delta_{ic} + \delta_{id} = 1$$

$$\delta_{ij} \in \{0, 1\}, \quad j = a, b, c, d$$

$$AC_i = \delta_{ib} + \delta_{id}, \quad SR_i = \delta_{ic} + \delta_{id}, \quad i = 1, \ldots, 50$$

$$\delta_{ib} + \delta_{ic} + \delta_{id} \le 1, \quad i = 1, \ldots, 50$$

$$\delta_{ij} \in \{0, 1\}, \quad j = b, c, d, \quad i = 1, \ldots, 50$$

$$\sum_{i=1}^{50} \delta_{ia} = 20, \quad \sum_{i=1}^{50} \delta_{ib} = 15, \quad \sum_{i=1}^{50} \delta_{ic} = 8, \quad \sum_{i=1}^{50} \delta_{id} = 7, \quad i = 1, \ldots, 50$$

$$\sum_{j=i}^{i+4} AC_j \le 3, \quad i = 1, \ldots, 46$$

$$\sum_{j=j}^{i+2} SR_j \le 1, \quad j = 1, \ldots, 48$$

Global Constraint: nvalues

nvalues

Let $x_1, \ldots, x_n$ be a tuple of variables, and l and u two nonnegative integers

$$\texttt{nvalues}([x_1, ..., x_n], l, u)$$

At least l and at most u different values among the variables

$\rightsquigarrow$ generalization of `alldifferent`

In Gecode: `nvalues`

Global Constraint: stretch

stretch (In Gecode: via regular and extensional)

Let $x_1, \ldots, x_n$ be a tuple of variables with finite domains,
v an m-tuple of possible values of the variables,
l an m-tuple of lower bounds and u an m-tuple of upper bounds.
A stretch is a maximal sequence of consecutive variables that take the same value, i.e., $x_j, \ldots, x_k$ for v if $x_j = \ldots = x_k = v$ and $x_{j-1} \neq v$ (or $j = 1$) and $x_{k+1} \neq v$ (or $k = n$).

stretch($[x_1, ..., x_n], \mathbf{v}, \mathbf{l}, \mathbf{u}$)   stretch-cycle($[x_1, ..., x_n], \mathbf{v}, \mathbf{l}, \mathbf{u}$)

for each $j \in \{1, \ldots, m\}$ any stretch of value v_j in x have length at least l_j and at most u_j.

In addition:

stretch($[x_1, ..., x_n], \mathbf{v}, \mathbf{l}, \mathbf{u}, P$)

with P set of patterns, i.e., pairs $(v_j, v_{j'})$. It imposes that a stretch of values v_j must be followed by a stretch of value $v_{j'}$

Global Constraint: `element`

"`element`" constraint

Let y be an integer variable,
z a variable with finite domain,
and c an array of constants, i.e., $c = [c_1, c_2, \dots, c_n]$.
The element constraint states that z is equal to the y-th variable in c, or $z = c_y$.
More formally:

$$\texttt{element}(y, z, [c_1, \dots, c_n]) = \{(e, f) \mid e \in D(y), f \in D(z), f = c_e\}.$$

Global Constraint: channel

"channel" constraint

Let y be array of integer variables, and x be an array of integer variables:

$$\texttt{channel}([y_1, \dots, y_n], [x_1, \dots, x_n]) =$$
$$\{([e_1, \dots, e_n], [d_1, \dots, d_n]) \mid e_i \in D(y_i), \forall i, d_j \in D(x_j), \forall j, e_i = j \wedge d_j = i\}.$$

Employee Scheduling problem

Four nurses are to be assigned to eight-hour shifts.
Shift 1 is the daytime shift, while shifts 2 and 3 occur at night.
The schedule repeats itself every week. In addition,

1. Every shift is assigned exactly one nurse.
2. Each nurse works at most one shift a day.
3. Each nurse works at least five days a week.
4. To ensure a certain amount of continuity, no shift can be staffed by more than two different nurses in a week.
5. To avoid excessive disruption of sleep patterns, a nurse cannot work different shifts on two consecutive days.
6. Also, a nurse who works shift 2 or 3 must do so at least two days in a row.

Employee Scheduling problem

Feasible Solutions

Solution viewed as assigning workers to shifts.

	Sun	Mon	Tue	Wed	Thu	Fri	Sat
Shift1	A	B	A	A	A	A	A
Shift2	C	C	C	B	B	B	B
Shift3	D	D	D	D	C	C	D

Solution viewed as assigning shifts to workers.

	Sun	Mon	Tue	Wed	Thu	Fri	Sat
Worker A	1	0	1	1	1	1	1
Worker B	0	1	0	2	2	2	2
Worker C	2	2	2	0	3	3	0
Worker D	3	3	3	3	0	0	3

Employee Scheduling problem

Feasible Solutions

Let w_{sd} be the nurse assigned to shift s on day d, where the domain of w_{sd} is the set of nurses $\{A, B, C, D\}$.
Let t_{id} be the shift assigned to nurse i on day d, and where shift 0 denotes a day off.

1. $\texttt{alldiff}(w_{1d}, w_{2d}, w_{3d}), d = 1, \ldots, 7$
2. $\texttt{cardinality}(W, (A, B, C, D), (5, 5, 5, 5), (6, 6, 6, 6))$
3. $\texttt{nvalues}(\{w_{s1}, \ldots, w_{s7}\}, 1, 2), s = 1, 2, 3$
4. $\texttt{alldiff}(t_{Ad}, t_{Bd}, t_{Cd}, t_{Dd}), d = 1, ..., 7$
5. $\texttt{cardinality}(\{t_{i1}, \ldots, t_{i7}\}, 0, 1, 2), i = A, B, C, D$
6. $\texttt{stretch-cycle}((t_{i1}, \ldots, t_{i7}), (2, 3), (2, 2), (6, 6), P), i = A, B, C, D$
7. $w_{t_{id} d} = i, \forall i, d, \quad t_{w_{sd} s} = s, \forall s, d$

CP Modeling Guidelines [Hooker, 2011]

1. A specially-structured subset of constraints should be replaced by a single global constraint that **captures the structure**, when a suitable one exists. This produces a more succinct model and can allow more effective filtering and propagation.

2. A global constraint should be replaced by a more specific one when possible, to exploit more effectively the special structure of the constraints.

3. The addition of redundant constraints (i..e, constraints that are implied by the other constraints) can improve propagation.

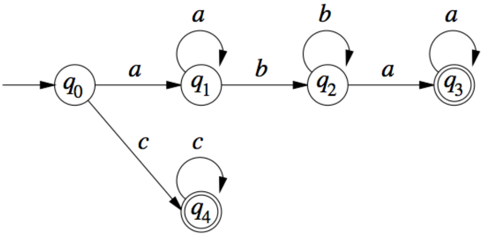
4. When two alternate formulations of a problem are available, including both (or parts of both) in the model may improve propagation. Different variables are linked through the use of channeling constraints.

Global Constraint: `regular`

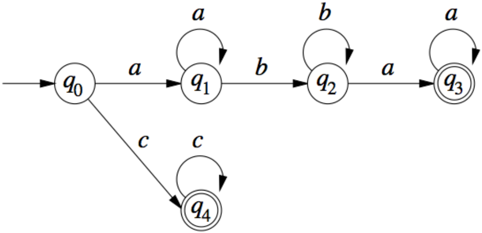
"`regular`" constraint

Let $M = (Q, \Sigma, \delta, q_0, F)$ be a DFA and let $X = \{x_1, x_2, \ldots, x_n\}$ be a set of variables with $D(x_i) \subseteq \Sigma$ for $1 \leq i \leq n$. Then

$$\texttt{regular}(X, M) = \{(d_1, ..., d_n) \mid \forall i, d_i \in D(x_i), [d_1, d_2, \ldots, d_n] \in L(M)\}.$$



Global Constraint: regular



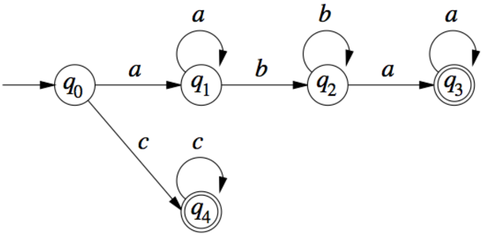
Example

Given the problem

$$x_1 \in \{a, b, c\}, \quad x_2 \in \{a, b, c\}, \quad x_3 \in \{a, b, c\}, \quad x_4 \in \{a, b, c\},$$

$$\texttt{regular}([x_1, x_2, x_3, x_4], M).$$

One solution to this CSP is $x_1 = a, x_2 = b, x_3 = a, x_4 = a$.



In Gecode:

```
DFA::Transition t[] = {{0, 0(a), 1}, {1, 0(a), 1}, {1, 1(b), 2}, {2, 1(b), 2},
                       {2, 0(a), 3}, {3, 0(a), 3}, {3, 0, -1},
                       {0,2(c),4}, {4, 2(c), 4}, {4, 0, -1}};
int f[] = {3,4,-1}; // vector of final states
DFA d(0, t, f);
BoolVarArray x(home, 4, 0(a), 3(d));
extensional(home, x, d);
```

```
REG r = ( REG(0) + (*REG(0) + REG(1) + *REG(1) + REG(0) + *REG(0)) ) | (*REG(3)));
DFA d(r);
extensional(home, x, d);
```

Example

Nonogram

Assignment problems

The assignment problem is to find a minimum cost assignment of m tasks to n workers ($m \leq n$).
Each task is assigned to a different worker, and no two workers are assigned the same task.
If assigning worker i to task j incurs cost c_{ij}, the problem is simply stated:

$$
\begin{aligned}
\min \quad & \sum_{i=1,\dots,n} c_{ix_i} \\
& \texttt{alldiff}([x_1, \dots, x_n]), \\
& x_i \in D_i, \forall i = 1, \dots, n.
\end{aligned}
$$

Note: cost depends on position. Recall: with $n = m$ min weighted bipartite matching (Hungarian method)
with supplies/demands transshipment problem

Circuit problems

Given a directed weighted graph $G = (N, A)$, find a circuit of min cost:

$$\min \sum_{i=1,\dots,n} c_{x_i x_{i+1}}$$

$$\texttt{alldiff}([x_1, \dots, x_n]),$$

$$x_i \in D_i, \forall i = 1, \dots, n.$$

Note: cost depends on sequence.

An alternative formulation is

$$\min \sum_{i=1,\dots,n} c_{i y_i}$$

$$\texttt{circuit}([y_1, \dots, y_n]),$$

$$y_i \in D_i = \{j \mid (i, j) \in A\}, \forall i = 1, \dots, n.$$

Global Constraint: circuit

"circuit" constraint

Let $X = \{x_1, x_2, \ldots, x_n\}$ be a set of variables with respective domains $D(x_i) \subseteq \{1, 2, ..., n\}$ for $i = 1, 2, ..., n$. Then

$$\texttt{circuit}(x_1, \ldots, x_n) = \{(d_1, ..., d_n) \mid \forall i, d_i \in D(x_i), d_1, ..., d_n \text{ is cyclic } \}.$$

Circuit problems

A model with redundant constraints is as follows:

$$
\begin{aligned}
\min \quad & z \\
& z \geq \sum_{i=1,\ldots,n} c_{x_i x_{i+1}} \\
& z \geq \sum_{i=1,\ldots,n} c_{i y_i} \\
& \texttt{alldiff}([x_1, \ldots, x_n]), \\
& \texttt{circuit}([y_1, \ldots, y_n]), \\
& x_1 = y_{x_n} = 1, \quad x_{i+1} = y_{x_i}, i = 1, \ldots, n-1 \\
& x_i \in \{1, \ldots, n\}, \forall i = 1, \ldots, n, \\
& y_i \in D_i = \{j \mid (i,j) \in A\}, \forall i = 1, \ldots, n.
\end{aligned}
$$

Scheduling Constraints

"disjunctive" scheduling

Let $(s_1, \ldots, s_n)$ be a tuple of (integer/real)-valued variables indicating the starting time of a job j. Let $(p_1, \ldots, p_n)$ be the processing times of each job.

$$\texttt{disjunctive}([s_1, \ldots, s_n], [p_1, \ldots, p_n]) = \{[d_1, \ldots, d_n] \mid \forall i, j, i \neq j\ (d_i + p_i \leq d_j) \vee (d_j + p_j \leq d_i)\}$$

Scheduling Constraints

cumulative for RCPSP [Aggoun and Beldiceanu, 1993]

- r_j release time of job j
- p_j processing time
- d_j deadline
- c_j resource consumption
- C limit not to be exceeded at any point in time

Let s be an n-tuple of (integer/real) values denoting the starting time of each job

$$\texttt{cumulative}([s_j],[p_j],[c_j],C) :=$$
$$\{([d_j],[p_j],[c_j],C) \mid \forall t \sum_{i \mid d_i \leq t \leq d_i + p_i} c_i \leq C\}$$

With $c_j = 1$ forall j and $C = 1 \rightsquigarrow$ disjunctive

Others

- Sorted constraints (sorted(x, y))
- Bin-packing constraints (binpacking(l, b, s))
- Geometrical packing constraints (nooverlap)
  diffn$((x^1, \Delta x^1), \ldots, (x^m, \Delta x^m))$ arranges a given set of multidimensional boxes in n-space such that they do not overlap (aka, nooverlap)
- Value precedence constraints (precede(x, s, t))
- Logical implication: conditional$(\mathcal{D}, \mathcal{C})$ between sets of constrains $\mathcal{D} \Rightarrow \mathcal{C}$ (ite)

More (not in gecode)

- $\texttt{clique}(x|G,k)$ requires that a given graph contain a clique of size k
- $\texttt{cycle}(x|y)$ select edges such that they form exactly y directed cycles in a graph.
- $\texttt{cutset}(x|G,k)$ requires that for the set of selected vertices V', the set $V \setminus V'$ induces a subgraph of G that contains no cycles.

Global Constraint Catalog

Global Constraint Catalog

Corresponding author: Nicolas Beldiceanu nicolas.beldiceanu@emn.fr

Online version: Sophie Demassey sophie.demassey@emn.fr

Google Search ○ Web ⦿ Catalog
○ all formats ⦿ html ○ pdf

Global Constraint Catalog
html / 2009-12-16

Search by:

NAME	Keyword	Meta-keyword	Argument pattern	Graph description
		Bibliography	Index	

Keywords (ex: *Assignment, Bound consistency, Soft constraint,...*) can be searched by **Meta-keywords** (ex: *Application area, Filtering, Constraint type,...*)

About the catalogue

The catalogue presents a list of 348 global constraints issued from the literature in constraint programming and from popular constraint systems. The semantic of each constraint is given together with a description in terms of graph properties and/or automata.

References

Hooker J.N. (2011). **Hybrid modeling**. In *Hybrid Optimization*, edited by P.M. Pardalos, P. van Hentenryck, and M. Milano, vol. 45 of **Optimization and Its Applications**, pp. 11–62. Springer New York.

van Hoeve W. and Katriel I. (2006). **Global constraints**. In *Handbook of Constraint Programming*, chap. 6. Elsevier.