DM841

Discrete Optimization

Part I

Lecture 7

Global constraints

Marco Chiarandini

Department of Mathematics & Computer Science
University of Southern Denmark

Outline

Outline

domain and member

Domain constraints

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

Membership

```
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);`	$x \div y = d \wedge x \bmod y = m$	✓		

Posting Constraints in Gecode

- Handling of exceptions to ensure consistency and variable unsharing in arguments passed to posted constraints
- 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 $O(d^n)$

Extensional constraints: `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: `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)`

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:

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

- 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.

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, ...

Model:

Parameters: n

Variables:

$$s_i \in \{1..n-1\} \text{ for } i = 0..n-1$$

Constraints:

$$s_i = j \iff b_{ij} = 1 \text{ for } i, j = 0..n-1$$

$$\sum_{i=0}^{n-1} b_{ij} = s_j \text{ for } j = 0..n-1$$

```
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 these constraints.

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])
```

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)

$\rightsquigarrow$ 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());
}
```

```
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());
}
```

Implied constraints:

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

$$\underbrace{\sum_{i=0}^{n-1} i x_i}_{n} = \sum_{i=0}^{n-1} x_i \implies \sum_{i=0}^{n-1} (i-1) x_i = 0$$