MiniBrass

Soft Global Constraints

Overview

Institute for
Software & Systems
Engineering

These few slides show how to use soft global constraints in your model

To familiarize yourself with the basics, consider looking at:

- Step-by-Step enhancing a MiniZinc model (establishes the core elements)
- Language Features
- Case Studies (for some specific examples)
- Also check out Willem-Jan van Hoeve's tutorial on soft globals: `http://www.andrew.cmu.edu/user/vanhoeve/papers/soft_global_cp2009.pdf`

`http://isse-augsburg.github.io/constraint-relationships/`

Soft Global Constraints

Institute for
Software & Systems
Engineering

- Special kind of global constraints $\rightarrow$ recurring combinatorial substructure
- Allows for a "maximally allowed degree of violation" of an otherwise hard constraint

- Popular variants (from the gcc)
  - Soft-Alldifferent
  - Soft-GCC
  - Soft-Regular
- We'll deal with soft-alldifferent

Example

Institute for Software & Systems Engineering

Say you wanted to assign students to projects, and *in principle*, every student should work on a different project.

```
include "alldifferent.mzn";
int: n = 5; int: m = 6;
set of int: STUDENT = 1..n;
set of int: PROJECT = 1..m;
array[STUDENT] of var PROJECT: x;

constraint alldifferent(x);
solve satisfy;
```

```
x = array1d(1..5 ,[1, 2, 3, 4, 5]);
----------
```

Example

Institute for Software & Systems Engineering

Now assume, you lack creativity and only find 4 projects for 5 students, so some have to work on the same task! You obviously still want to bound the number of students working on a project (say to 2)! Easy for `soft_alldifferent`.

```
include "soft_constraints/soft_alldifferent.mzn";

int: n = 5; int: m = 4;
set of int: STUDENT = 1..n;
set of int: PROJECT = 1..m;
array[STUDENT] of var PROJECT: x;
var int: maxPerProj = 2;
constraint soft_alldifferent(x, maxPerProj, true);
solve satisfy;
```

```
x = array1d(1..5 ,[1, 1, 2, 3, 4]);
----------
```

Semantics of Soft-Alldifferent

Institute for
Software & Systems
Engineering

$$\texttt{soft} - \texttt{alldifferent}([x_1, \ldots, x_n], c) \leftrightarrow \mu([x_1, \ldots, x_n]) \leq c$$

where μ is a *violation measure*. Two variants have been proposed in the gcc:

- $\mu_{\text{var}}([x_1, \ldots, x_n]) = \sum_{d \in D(X)} \max(|\{i \mid x_i = d\}| - 1, 0)$, the variable-based violation measure that counts how many variables would need to change their values[1]
- $\mu_{\text{dec}}([x_1, \ldots, x_n]) = |\{(i, j) \mid x_i = x_j, i < j\}|$, the decomposition-based violation measure that counts how many decomposed constraints would be violated by X[2]

[1] http://sofdem.github.io/gccat/gccat/Csoft_alldifferent_var.html
[2] http://sofdem.github.io/gccat/gccat/Csoft_alldifferent_ctr.html

Example of Soft-Alldifferent:

Institute for
Software & Systems
Engineering

$[x_1, x_2, x_3, x_4]$	μ_{var}	μ_{dec}
$(1, 1, 2, 3)$	1	1
$(1, 1, 2, 2)$	2	2
$(1, 1, 1, 2)$	2	3
$(1, 1, 1, 1)$	3	6

- $\mu_{\text{var}}([x_1, \ldots, x_n]) = \sum_{d \in D(X)} \max(|\{i \mid x_i = d\}| - 1, 0)$
- $\mu_{\text{dec}}([x_1, \ldots, x_n]) = |\{(i, j) \mid x_i = x_j, i < j\}|$

Decomposition in MiniZinc

Institute for
Software & Systems
Engineering

Provides a *default implementation* of `soft_alldifferent`:

```
predicate soft_all_different_int(array[int] of var int: x,
                                 var int: cost, bool: useDec) =
let {
  var int: mu = if useDec then
                  soft_all_different_dec(x, cost)
                else
                  soft_all_different_var(x, cost)
                endif;
}
in
(
  mu <= cost
);
```

Decomposition in MiniZinc (cont.)

Institute for
Software & Systems
Engineering

$$\mu_{\text{var}}([x_1, \ldots, x_n]) = \sum_{d \in D(X)} \max(|\{i \mid x_i = d\}| - 1, 0)$$

```
function var int: soft_all_different_var(array[int] of var int: x,
                                         var int: cost) :: promise_total =
let {
  set of int: seenValues = dom_array (x);
}
in
(
  sum(s in seenValues) ( max( count(x, s) - 1, 0 ) )
);
```

Decomposition in MiniZinc (cont.)

Institute for
Software & Systems
Engineering

$$\mu_{\text{dec}}([x_1, \ldots, x_n]) = |\{(i,j) \mid x_i = x_j, i < j\}|$$

```
function var int: soft_all_different_dec(array[int] of var int: x,
                                         var int: cost) :: promise_total
= (
        sum(i, j in index_set(x) where i < j) ( bool2int(x[i] = x[j]) )
);
```

Native Soft-Alldifferent

Institute for
Software & Systems
Engineering

If a solver happens to support `soft_alldifferent` *natively* (such as, e.g., JaCoP), we should use it![3]

```
predicate soft_all_different_int(array[int] of var int: x,
                                 var int: d, bool: useDec) =
    jacop_softalldiff(x, d, useDec);


predicate jacop_softalldiff(array[int] of var int: x, var int: d,
                            bool: useDec);
```

[3] Experimentally implemented in the forked JaCoP repository `https://github.com/Alexander-Schiendorfer/jacop`

How to use it?

Institute for Software & Systems Engineering

- Install MiniBrass from `http://isse-augsburg.github.io/constraint-relationships/`
- Write models using `soft_alldifferent`
- Get JaCoP from `https://github.com/Alexander-Schiendorfer/jacop` (for now)
- Try it!

References I

Institute for
Software & Systems
Engineering