MiniBrass

Cost Function Networks

Universität
Augsburg
University

Overview

These few slides show how to use cost functions (also known as weighted constraints) in your model such that a dedicated solver (toulbar2) can access them

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)
- Soft Global Constraints (`soft-alldiff`, `soft-regular`, `soft-gcc`)

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

Cost Functions

- Different underlying model than pure constraint satisfaction and optimization (CSOP)
- Assumes a *decomposable* cost function that represents the violation of a constraint
- Ranges from 0 (no violation) to some upper bound k (maximally possible violation)
  - The objective of the CSOP is $F(x)$
  - i.e., for $x \in X$, $F(x) = \sum_{f \in F} f(x)$
  - Integer domains, integer objectives
- Algorithmic support (`toulbar2`[1] (Allouche et al., 2010))
  - Soft local consistency (specialized propagation)
  - Limited discrepancy search
  - Russian Doll Search, Branch-and-Bound etc

[1] `https://mulcyber.toulouse.inra.fr/projects/toulbar2/`

Example Cost Function

Idea: Variables $X = \{x, y\}$, assign cost to every assignment (i.e., the cartesian product of the domains)

x	y	c
0	0	4
0	1	3
1	0	2
1	1	4

Example Usage

Assume you choose dinner and wine and let your friends rate dinner/wine combinations, as well as the wine individually and you want to find the solution minimizing the overall *dissatisfaction*.

```
int: steak = 1; int: fish = 2; int: pizza = 3;
int: red = 1; int: white = 2;
set of int: MEAL = {steak, fish, pizza};
set of int: WINE = {red, white};
var MEAL: meal; var WINE: wine;

var 0..4: mealA; var 0..4: mealB; var 0..4: mealC;
var 0..4: wineA; var 0..4: wineB; var 0..5: wineC;

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

Example Usage

```
% Albert
constraint cost_function_binary(meal, wine,
 [
   /* steak, red */     2,
   /* steak, white */   10,
   /* fish, red */      20,
   /* fish, white */    3,
   /* pizza, red */     5,
   /* pizza, white */   5
 ]
 , mealA);

constraint cost_function_unary(wine, [2, 1], wineA);
% analogously for Berta and Carl (see dinner.mzn)
```

```
----------
meal = 2; % fish
wine = 2; % white
```

Native Support

- Cost functions are first-class citizen in `toulbar2`, accessed by `wcsp` files.
- Classical constraints are cost functions that map to a value higher than some k – to denote unacceptable violation.
- Example:

```
wcsp 2 2 1 100000000
2 2
2 0 1 0 4
0 0 4
0 1 3
1 0 2
1 1 4
```

A simple WCSP file with 2 variables, at most 2 domain values, precisely 1 cost function and an upper bound $k = 100000000$.[2]

[2] Read more about the WCSP format at `http://costfunction.org/mobyle/htdocs/portal/help/wcsp.html`

Access via Numberjack

- Numberjack[3](Hebrard et al., 2010) is a modeling package for multiple solvers, written in Python
- Interfaces to toulbar2 using the objective function!
  - If the objective function (`Minimize`, `Maximize` objects) is a *sum*, every sub-expression is translated to a cost function.
  - Very useful for *weighted constraints*, in the sense that satisfaction maps to 0 and violation to a specific weight for all assignments of the variables in the scope of the cost function

```
from Numberjack import *
x,y,z = VarArray(3,0,2) # three variables, each with domain {0,1,2}
# state desirable constraints, and minimize their violation
# --> their negation being true
model = Model(
   Minimise( 2 * (x + 1 != y) + # x + 1 = y should hold with weight 2
            (z != y + 2)  )) # z = y + 2 should hold with weight 1
solver.solve()
```

[3] http://numberjack.ucc.ie/

Generated WCSP

```
from Numberjack import *
x,y,z = VarArray(3,0,2) # three variables, each with domain {0,1,2}
model = Model(
    Minimise( 2 * (x + 1 != y) + # x + 1 = y should hold with weight 2
             (z != y + 2)  )) # z = y + 2 should hold with weight 1
solver.solve()
```

```
wcsp 3 3 5 100000000
3 3 3 # order is z, y, x
2 0 1 0 9 # z = y + 2
0 0 1
0 1 1
0 2 1
1 0 1
1 1 1
1 2 1
2 0 0
2 1 1
2 2 1
```

```
2 1 2 0 9 # y = x + 1
0 0 2
0 1 2
0 2 2
1 0 0
1 1 2
1 2 2
2 0 2
2 1 0
```

Accessing Toulbar2 via MiniZinc

```
from Numberjack import *
x,y,z = VarArray(3,0,2) # three variables, each with domain {0,1,2}
model = Model(
    Minimise( 2 * (x + 1 != y) + # x + 1 = y should hold with weight 2
             (z != y + 2)  )) # z = y + 2 should hold with weight 1
solver.solve()
```

The same model, written in MiniZinc[4]

```
var 0..2: x; var 0..2: y; var 0..2: z;

var bool: sc1 = (x + 1 = y);
var bool: sc2 = (z = y + 2);

solve minimize 2 * (not sc1) + (not sc2);
```

[4] Compile using `mzn_numberjack` with other solvers than `toulbar2` commented out in the portfolio solver `fzn/njportfolio.py` of the Numberjack source.

Explicit Cost Function Modeling

- Internally, the Cartesian product of the domains of the involved variables is built and an expression evaluated (e.g., $x + 1 = y$).
- However, it is not possible to inject arbitrary cost functions from MiniZinc
  - Enumerate support and cost function *explicitly*
  - Use, e.g., MiniZinc functions and comprehensions to calculate cost vector

Proposal

Add cost functions as special global constraints to MiniZinc

Decomposition in MiniZinc

Provides a *default implementation* of `cost_function` that *encodes* cost functions using table constraints for any solver.

```
% e.g. x \in 0..2 ->, costs = [4, 1, 3] leads to
% f(x -> 0) = 4, f(x -> 1) = 1, f(x -> 2) = 3
% costVariable takes the value of f(x)
predicate cost_function_unary(var int: x,
                              array[int] of int: costs,
                              var int: costVariable ) =
let {
  array[int] of int: folded = [ x_ | x_ in dom(x)];
}
in ( cost_function_unary_safe(folded, costs, x, costVariable));
```

Safe Cost Function Encoding

```
predicate cost_function_unary_safe(array[int] of int: folded,
                                array[int] of int: costs,
                                var int: x, var int: costVar) =
assert(max(index_set(folded)) == max(index_set(costs)),
"Dimensions of cost vector and flattened domains must agree",
let {
   array[int] of int: tableDecomp = [ if(j == 1) then folded[i]
                                  else  costs[i] endif
                                  | i in index_set(costs), j in 1..2 ];
} in table([x,costVar], array2d(index_set(costs), 1..2, tableDecomp)));
```

Analogously (as of now) for `cost_function_binary` and `cost_function_ternary`. All definitions are found in `soft_constraints/cost_functions.mzn`.

Example Usage

bincost-functions.mzn

```
array[1..2] of var 0..1: x;
% default definitions from MiniBrass lib
include "soft_constraints/cost_functions.mzn";

var 0..10: cVar;
% 0, 0 -> 4; 0, 1 -> 3; 1, 0 -> 2; 1, 1 -> 4
constraint cost_function_binary(x[1], x[2], [4, 3, 2, 4], cVar);

solve minimize cVar;
```

```
x = array1d(1..2 ,[0, 0]);
cVar = 4;
----------
x = array1d(1..2 ,[1, 0]);
cVar = 2;
----------
==========
```

Usage with MiniZinc Functions

comprehensions.mzn

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

% shows an exemplary comprehension based cost function
var 0..1: x; var 0..1: y; var 0..10: costVar;
% x y | 4 - (x + y) : (0,0) -> 4, (0,1) -> 3, (1, 0) -> 3, (1, 1) -> 2

function int: f(int: x, int: y) = ( 4 - (x + y) );

constraint cost_function_binary(x, y,
   [f(x_,y_) | x_ in dom(x), y_ in dom(y)], costVar);
solve minimize costVar;
```

```
x = 1;
y = 1;
costVar = 2;
----------
==========
```

Native Cost Function

Toulbar2 supports `cost_function` *natively*. We should use this![5]

Define `soft_constraints/cost_functions.mzn` in Numberjack's `mzn-lib` dir:

```
predicate cost_function_binary(var int: x, var int: y,
                               array[int] of int: costs,
                               var int: costVariable );
```

such that `cost_function_binary` does not get decomposed but sent to Numberjack.

On Numberjack's side, we need to treat `cost_function_binary` in `fzn2py`:

```
def cost_function_binary(var1, var2, costs, costVar):
    return PostBinary(var1, var2, costs)
```

`costVar` is ignored since it is directly posted to the cost function.

[5] Experimentally implemented in the forked Numberjack repository `https://github.com/Alexander-Schiendorfer/Numberjack` on the `feature/wcsp-encoding` branch.

How to use it?

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

```
git clone -b feature/wcsp-encoding
  https://github.com/Alexander-Schiendorfer/Numberjack.git
```

References I

Allouche, D., de Givry, S., and Schiex, T. (2010).
Toulbar2, an open source exact cost function network solver.
Technical report, Technical report, INRIA.

Hebrard, E., O'Mahony, E., and O'Sullivan, B. (2010).
Constraint programming and combinatorial optimisation in numberjack.
In *Integration of AI and OR Techniques in Constraint Programming for Combinatorial Optimization Problems*, pages 181–185. Springer.