Constraint Relationships

Step-by-Step Enhancing a MiniZinc Model

Universität Augsburg
University

Synopsis

Institute for
Software & Systems
Engineering

We will ...

- start with an n-queens model in MiniZinc
- add some additional soft constraints (e.g. one queen should be placed in the center)
- prioritize these soft constraints using constraint relationships
- switch dominance properties (SPD, TPD)
- and solve with: Branch-and-Bound (BaB), Large Neighborhood Search (LNS)

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

Satisfaction Model

```
include "globals.mzn";
int: n; array[1..n] of var 1..n: queens; n = 8;

solve :: int_search(queens, first_fail, indomain_median, complete)
satisfy;

constraint all_different(queens);
constraint all_different([queens[i]+i | i in 1..n]);
constraint all_different([queens[i]-i | i in 1..n]);
```

```
queens = array1d(1..8 ,[4, 6, 1, 5, 2, 8, 3, 7]);
----------
```

Step 1: Add Soft Constraint Support

Institute for Software & Systems Engineering

```
include "globals.mzn";
include "soft_constraints/soft_constraints.mzn";
int: n; array[1..n] of var 1..n: queens; n = 8;

solve :: int_search(queens, first_fail, indomain_median, complete)
satisfy;

constraint all_different(queens);
constraint all_different([queens[i]+i | i in 1..n]);
constraint all_different([queens[i]-i | i in 1..n]);
```

- Adds `nScs` (a new parameter) boolean variables, one for each soft constraint
- Adds parameter `penalties` (use `[1 | i in 1..nScs]` if not relevant)
- Intuition: The cost of violating soft constraint `i` should be `penalties[i]`

Step 1: Behind the Scenes

```
include "link_set_to_booleans.mzn";
% MODEL-SPECIFIC parameters: nScs and penalties
int: nScs; set of int: SOFTCONSTRAINTS = 1..nScs;

% reified variables for each soft constraint
array[SOFTCONSTRAINTS] of var bool: satisfied;
array[SOFTCONSTRAINTS] of var bool: violated =
   [not satisfied[sc] | sc in SOFTCONSTRAINTS] ;

array[SOFTCONSTRAINTS] of int: penalties;
var int: penSum = sum(sc in SOFTCONSTRAINTS)
   (bool2int(not satisfied[sc]) * penalties[sc]);

var set of SOFTCONSTRAINTS: violatedScs;
constraint link_set_to_booleans(violatedScs,violated);
```

- Using `satisfied` *and* `violated` is done for convenience in expressing soft constraints
- There is also a variant without `violatedScs` for solvers that do not support set variables (e.g. MIP): `soft_constraints_noset.mzn`

Step 1: Behind the Scenes

```
% find the sorted permutation of soft constraint instances
array[SOFTCONSTRAINTS] of SOFTCONSTRAINTS: sortPermScs =
  arg_sort(penalties);
% invert, since arg_sort use <= and we need decreasing order
array[SOFTCONSTRAINTS] of SOFTCONSTRAINTS: mostImpFirst =
  [ sortPermScs[nScs-s+1] | s in SOFTCONSTRAINTS];
```

- `mostImpFirst` gives an array of soft constraint indices in decreasing order (start with most important ones)
- Used later as a (generic) labeling strategy

Step 2: Add Soft Constraints

Institute for Software & Systems

```
include "globals.mzn";
include "soft_constraints/soft_constraints.mzn";
int: n; array[1..n] of var 1..n: queens; n = 8;

solve :: int_search(queens, first_fail, indomain_median, complete)
minimize penSum;

constraint all_different(queens);
constraint all_different([queens[i]+i | i in 1..n]);
constraint all_different([queens[i]-i | i in 1..n]);
% soft constraint business
nScs = 3; penalties = [1 | i in SOFTCONSTRAINTS];
constraint queens[n div 2] = n div 2 <-> satisfied[1] ;
constraint queens[2] = queens[1] + 2 <-> satisfied[2];
constraint queens[3] = queens[2] + 2 <-> satisfied[3];
```

- Just tie reified soft constraint variables to actual soft constraints
- `[1 | i in 1..nScs]` makes it a Max-CSP when minimizing `penSum`

Step 2: What about Globals?

What if want to use global constraints as soft constraints? Consider (`reified.mzn`):

```
include "globals.mzn";
include "soft_constraints/soft_constraints.mzn";
array[1..5] of var 0..5: x;
constraint alldifferent(x); % hard constraint

% but it would be nice to have x increase ...
nScs = 1; penalties = [1];
constraint increasing(x) <-> satisfied[1];

solve maximize bool2int(satisfied[1]);
output["satisfied[1] = \(satisfied[1]), x = \(x)"];
```

Solved with Gecode:

```
MiniZinc: flattening error: 'increasing_int' is used in a reified
context but no reified version is available
```

Step 2: What about Globals?

a) Use a solver that has the reified global, e.g., here:
   - G12-FD
   - JaCoP

b) Use a decomposition from the MiniZinc standard library (reified_fix.mzn)

```
include "globals.mzn";
include "soft_constraints/soft_constraints.mzn";
array[1..5] of var 0..5: x;
constraint alldifferent(x); % hard constraint
nScs = 1; penalties = [1];
% copied from share/minizinc/std/increasing_int.mzn
predicate my_increasing_int(array[int] of var int: x) =
    forall(i in index_set(x) diff { min(index_set(x)) })
      (x[i-1] <= x[i]);

constraint my_increasing_int(x) <-> satisfied[1];
solve maximize bool2int(satisfied[1]);
output["satisfied[1] = \(satisfied[1]), x = \(x)"];
```

Step 2: What about Globals?

Institute for Software & Systems Engineering

Solved `reified_fix.mzn` with Gecode (same as `reified.mzn` solved with JaCoP or G12-FD):

```
satisfied[1] = false, x = [4, 1, 3, 0, 2]
----------
satisfied[1] = true, x = [0, 1, 2, 3, 4]
----------
==========
```

We do not get the performance benefits of globals (in a decomposition) but at least their conciseness.

Step 3: Prioritize Soft Constraints (Weights)

Institute for Software & Systems

```
include "globals.mzn";
include "soft_constraints/soft_constraints.mzn";
int: n; array[1..n] of var 1..n: queens; n = 8;

solve :: int_search(queens, first_fail, indomain_median, complete)
minimize penSum; output["penSum = \(penSum), queens = \(queens)"];

constraint all_different(queens);
constraint all_different([queens[i]+i | i in 1..n]);
constraint all_different([queens[i]-i | i in 1..n]);
% soft constraint business
nScs = 3; penalties = [2, 1, 1];
constraint queens[n div 2] = n div 2 <-> satisfied[1] ;
constraint queens[2] = queens[1] + 2 <-> satisfied[2];
constraint queens[3] = queens[2] + 2 <-> satisfied[3];
```

- If you manually set weights, just add them
- Switch from `[1 | i in 1..nScs]` to `[2, 1, 1]`

Step 3: Prioritize Soft Constraints (Relations)

Institute for Software & Systems

```
include "globals.mzn";
include "soft_constraints/soft_constraints.mzn";
include "soft_constraints/cr_types.mzn"; % graph types
include "soft_constraints/cr_weighting.mzn"; % weighting functions
int: n; array[1..n] of var 1..n: queens; n = 8;

solve :: int_search(queens, first_fail, indomain_median, complete)
minimize penSum; output["penSum = \(penSum), queens = \(queens)"];
constraint all_different(queens);
constraint all_different([queens[i]+i | i in 1..n]);
constraint all_different([queens[i]-i | i in 1..n]);
% soft constraint business
nScs = 3;
penalties = [weighting(s, SOFTCONSTRAINTS, crEdges, true)
             | s in SOFTCONSTRAINTS];
constraint queens[n div 2] = n div 2 <-> satisfied[1] ;
constraint queens[2] = queens[1] + 2 <-> satisfied[2];
constraint queens[3] = queens[2] + 2 <-> satisfied[3];
```

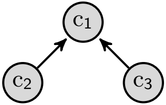
Step 3: Prioritize Soft Constraints (Relations)

Institute for Software & Systems Engineering

```
include "soft_constraints/cr_types.mzn"; % graph types
[... inside cr_types.mzn]
% constraint-relationship-types
int: nCrEdges;
array[1..nCrEdges, 1..2] of SOFTCONSTRAINTS: crEdges;
```

Concrete instantiation

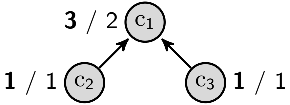
```
% data file for soft edges,
% constraint c1 is more important than c2 and c3
nCrEdges = 2;
crEdges = [| 2, 1 | 3, 1 |];
```



Step 3: Prioritize Soft Constraints (Relations)

```
include "soft_constraints/cr_weighting.mzn"; % graph types
[... inside cr_weighting.mzn]
function int: weighting(int: s, set of int: softConstraints,
                     array[int, 1..2] of int: edges,
                     bool: useSPD);
% weigh each constraint higher than the largest child
function int: weightingSPD(int: s, set of int: softConstraints,
                           array[int] of set of int: dominees) = (
1 + max(s_ in dominees[s])(weightingSPD(s_, softConstraints, dominees));
```

```
% weighting(s, SOFTCONSTRAINTS, crEdges, true)
[2, 1, 1]
% weighting(s, SOFTCONSTRAINTS, crEdges, false) more than chi. together
[3, 1, 1]
```



TPD / SPD

Step 4: Switch to PVS Architecture

Institute for

```
include "globals.mzn";
include "soft_constraints/soft_constraints.mzn";
include "soft_constraints/spd_worse.mzn"; % solution ordering
include "soft_constraints/pvs_spd.mzn"; % concrete PVS
int: n; array[1..n] of var 1..n: queens; n = 8; nScs = 3;

solve :: int_search(queens, first_fail, indomain_median, complete)
minimize penSum; output["penSum = \(penSum), queens = \(queens)"];
constraint all_different(queens);
constraint all_different([queens[i]+i | i in 1..n]);
constraint all_different([queens[i]-i | i in 1..n]);

constraint queens[n div 2] = n div 2 <-> satisfied[1] ;
constraint queens[2] = queens[1] + 2 <-> satisfied[2];
constraint queens[3] = queens[2] + 2 <-> satisfied[3];
```

- Defines *single-predecessor-dominance* as solution ordering with predicate `spd_worse`
- Hides the weight calculation in `pvs_spd.mzn`

Step 4: Switch to PVS Architecture

Institute for Software & Systems Engineering

Idea Many soft constraint formalisms are generalized by partial valuation structures that give the codomain of the objective function an *algebraic structure*.

- Defines *single-predecessor-dominance* as solution ordering with predicate `spd_worse`
- Hides the weight calculation in `pvs_spd.mzn`

Quellen I