Topic 9: Modelling for CBLS
(Version of 22nd February 2018)

Jean-Noël Monette and Pierre Flener

Optimisation Group
Department of Information Technology
Uppsala University
Sweden

Course 1DL441:
Combinatorial Optimisation and Constraint Programming,
whose part 1 is Course 1DL451:
Modelling for Combinatorial Optimisation

Outline

Outline

Local Search

Revisit the slides on local search (LS) and constraint-based local search (CBLS) of Topic 7: Solving Technologies.

Historically:

- No general-purpose LS solvers.
- No separation between model and search.

Recently:

- There are general-purpose CBLS solvers, supporting some degree of separation between model and search.
  Ex: Comet, EasyLocal++, LocalSolver, OscaR/CBLS.
- But there is still not much good-modelling practice.

Local Search from a MiniZinc Model

Our `fzn-oscar-cbls` backend for OscaR/CBLS:

- It is one of the few LS backends; it does tabu search.
- Currently, the following predicates have a specific neighbourhood: `all_different`, `circuit`, `subcircuit`, `inverse`, `global_cardinality`, `global_cardinality_closed`, `int_lin_eq`, `bool_lin_eq`, `global_cardinality_low_up`, and `global_cardinality_low_up_closed`. Note that just because a constraint-specific neighbourhood exists does not mean that it will be used.
- Currently, it ignores all the constraints flagged using `symmetry_breaking_constraint` and keeps those flagged using `implied_constraint` (see slide 8).
- Currently, there is no way to suggest a search heuristic by annotations, but we are working on it.

Outline

Rules of Thumb

Make explicit as much problem structure as possible:

- Use higher-level variables whenever possible.
  Examples: Use a single integer variable with a domain of k values instead of an array of k Boolean variables.
  Use a single set variable of cardinality k instead of an array of k integer variables.
- Define redundant variables and the objective function by constraints expressing total functions, as those are candidate one-way constraints.
- Use predicates that have specific neighbourhoods.
- Avoid disjunction whenever possible.

Symmetry-Breaking & Implied Constraints

- There is some evidence that symmetry-breaking constraints hinder local search.
- One reason might be that symmetry-breaking constraints forbid some solutions, but do not change the search space and hence do not prevent the search from moving in their direction.
- It might be beneficial rather to increase the number of symmetries in a model, but this may be hard to do.
- The effect of implied constraints on local search is not really known. You need to make experiments.

Outline

Outline

Example (The Warehouse Location Problem, WLP)

A company considers opening warehouses at some candidate locations in order to supply its existing shops:

- Each candidate warehouse has the same maintenance cost.
- Each candidate warehouse has a supply capacity, which is the maximum number of shops it can supply.
- The supply cost to a shop depends on the warehouse.

Determine which warehouses to open, and which of them should supply the various shops, so that:

1. Each shop must be supplied by exactly one actually opened warehouse.
2. Each actually opened warehouse supplies at most a number of shops equal to its capacity.
3. The sum of the actually incurred maintenance costs and supply costs is minimised.

UPPSALA UNIVERSITET

WLP: Sample Instance Data

$$\texttt{Shops} = \{\text{Shop}_1, \text{Shop}_2, \ldots, \text{Shop}_{10}\}$$

$$\texttt{Whs} = \{\text{Berlin, London, Ankara, Paris, Rome}\}$$

$$\texttt{maintCost} = 30$$

Capacity =

Berlin	London	Ankara	Paris	Rome
1	4	2	1	3

SupplyCost =

	Berlin	London	Ankara	Paris	Rome
Shop_1	20	24	11	25	30
Shop_2	28	27	82	83	74
Shop_3	74	97	71	96	70
Shop_4	2	55	73	69	61
$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$
Shop_{10}	47	65	55	71	95

WLP Model 1: Variables (reminder)

Automatic enforcement of the total-function constraint (1):

`Supplier` =

Shop_1	Shop_2	$\cdots$	Shop_{10}
$\in$ `Whs`	$\in$ `Whs`	$\cdots$	$\in$ `Whs`

`Supplier[s]` denotes the supplier warehouse for shop `s`.

Redundant decision variables:

`Open` =

Berlin	London	Ankara	Paris	Rome
$\in$ `0..1`	$\in$ `0..1`	$\in$ `0..1`	$\in$ `0..1`	$\in$ `0..1`

`Open[w]=1` if and only if (iff) warehouse `w` is opened.

WLP Model 1: Objective & Cons. (reminder)

Objective (3):

```
minimize
 maintCost * sum(Open)
 +
 sum(s in Shops)(SupplyCost[s,Supplier[s]])
```

One-way channelling constraint:

```
forall(s in Shops)(Open[Supplier[s]] = 1)
```

Capacity constraint (2):

```
global_cardinality_low_up_closed
(Supplier, Whs, [0 | i in Whs], Capacity)
```

WLP Model 1: Objective & Cons. (reminder)

Objective (3): a total function

```
minimize
 maintCost * sum(Open)
 +
 sum(s in Shops)(SupplyCost[s,Supplier[s]])
```

One-way channelling constraint: not a total function

```
forall(s in Shops)(Open[Supplier[s]] = 1)
```

Capacity constraint (2): a specific neighbourhood exists

```
global_cardinality_low_up_closed
(Supplier, Whs, [0 | i in Whs], Capacity)
```

Replace the one-way channelling by the two-way one:

```
forall(w in Whs)
 (Open[w] = bool2int(exists(s in Shops)(Supplier[s]=w)))
```

The `Open[w]` variables are now defined by a total function.

Search may now be performed only on the `Supplier[s]` decision variables, using the specific neighbourhood for `global_cardinality_low_up_closed`.

Finding a known-to-be optimal solution to the instance of slide 12, using `fzn-oscar-cbls`, averaged over 5 runs:

Model	Seconds
Model 1 with one-way channelling	3.88
Model 1 with two-way channelling	0.97

Outline

The Graph Colouring Problem

Given a graph `(V,E)`, colour each vertex so that adjacent vertices have different colours, using at most `n` colours.

Model 1, MIP-style

Variable `C[v,k]` in `0..1` is `1` iff vertex `v` has colour `k`.

1. One colour per vertex:

```
forall(v in V)(sum(C[v,..]) = 1)
```

2. Different colours on edge ends:

```
forall((u,v) in E, k in 1..n)
    (C[u,k] + C[v,k] <= 1)
```

Depending on the used moves, such as changing the value of one variable, the constraint (1) might be violated.

Given a graph `(V,E)`, colour each vertex so that adjacent vertices have different colours, using at most `n` colours.

Model 2, MiniZinc / CP / LCG-style

Variable `C[v]` in `1..n` is `k` iff vertex `v` has colour `k`.

1. One colour per vertex: enforced by choice of variables.
2. Different colours on edge ends:

```
forall((u,v) in E)(C[u] != C[v])
```

The search only needs to satisfy the constraint (2).

The Graph Colouring *Optimisation* Problem

Given a graph `(V,E)`, colour each vertex so that adjacent vertices have different colours, using a minimum of colours.

Model 2.1

Variable `C[v]` in `1..card(V)` is the colour of vertex `v`.
Minimise variable `N`, the number of possibly used colours:

1. One colour per vertex: enforced by choice of variables.
2. Different colours on edge ends:

```
forall((u,v) in E)(C[u] != C[v])
```

3. Objective var: `N = max(C)`, a total-function constraint

The variable `N` in `1..card(V)` represents the highest used colour: in a minimal solution, the `N` first colours are used and `N` is the number of actually used colours.

Despite `N` being total-function defined, Model 2.1 poorly drives the search as there is no way to distinguish between a candidate solution with only one variable equal to `N` and a candidate solution with several variables equal to `N`:

Example (`N = max(C)`)

Assume currently `C[1..8] = [1,2,3,4,2,3,1,4]`, with `N=4` if constraint (3) becomes a one-way constraint:

1. The move `C[8]:=3` to `[1,2,3,4,2,3,1,3]` keeps `N=4` and one move, namely `C[4]:=1`, suffices to reach the assumed minimal solution `[1,2,3,1,2,3,1,3]` with `N=3`.
2. The move `C[6]:=4` to `[1,2,3,4,2,4,1,4]` keeps `N=4` but three moves, namely `C[4]:=1` and `C[6]:=3` and `C[8]:=3`, are required to reach the assumed minimal solution `[1,2,3,1,2,3,1,3]` with `N=3`.

These two moves are not distinguished by the search, although the first move is better than the second one.

A better model bounds the number of colours by `N`:

Model 2.2

Model 2.1, except:

3 Objective variable: `forall(v in V)(C[v] <= N)`

Now the search is better driven, as the violation of the non-total-function constraint (3) depends on the number of variables with a value larger than `N`: see the next slide.

This is a counter-example to a rule of thumb on slide 7: the objective variable is not functionally defined anymore.

This slide reflects the current version of `fzn-oscar-cbls`; in future versions or in other (CB)LS backends, Model 2.1 might be better handled than Model 2.2.

Example (`forall(v in V)(C[v] <= N))`)

Assume currently `C[1..8] = [1,2,3,4,2,3,1,4]`, but `N=3`, as `N` is not fixed by the constraint (3), hence two conjuncts of (3) are violated, namely `C[4]<=N` and `C[8]<=N`, with the violation $(4-3)+(4-3)=2$ of (3):

1. The move `C[8]:=3` to `[1,2,3,4,2,3,1,3]` keeps `N=3` but only one conjunct of (3) is now violated, namely `C[4]<=N`, with the violation $4-3=1$ of (3).
2. The move `C[6]:=4` to `[1,2,3,4,2,4,1,4]` keeps `N=3` but three conjuncts of (3) are now violated, namely `C[4]<=N` and `C[6]<=N` and `C[8]<=N`, with the violation $(4-3)+(4-3)+(4-3)=3$ of (3).

In order to reach the assumed minimal solution `[1,2,3,1,2,3,1,3]` with `N=3`, probing the first move reveals a decrease by 1 of the violation of (3), while probing the second move reveals an increase by 1 of that violation, hence the first move is now preferred over the second one.

We implicitly broke some value symmetries in the previous models: if there is a solution with `N` colours, then it does not matter whether those are the values `1..N` or not.

Model 2.3

Model 2.1, except:

3. Objective var: `N = nvalue(C)`, a total-fct constraint

The local search may again be poorly driven, in the same way as with Model 2.1: see the next slide.

Example (`N = nvalue(C)`)

Assume currently `C[1..8] = [1,2,3,4,2,3,1,4]`, with `N=4` if constraint (3) becomes a one-way constraint:

1. The move `C[8]:=3` to `[1,2,3,4,2,3,1,3]` keeps `N=4` and one move, namely `C[4]:=1`, suffices to reach the assumed minimal solution `[1,2,3,1,2,3,1,3]` with `N=3`.
2. The move `C[6]:=4` to `[1,2,3,4,2,4,1,4]` keeps `N=4` but two moves, namely `C[3]:=4` and `C[4]:=1`, are required to reach the value-symmetric variant `[1,2,4,1,2,4,1,4]` of the assumed minimal solution `[1,2,3,1,2,3,1,3]` with `N=3`.

These two moves are not distinguished by the search, although the first move is better than the second one.

Outline

UPPSALA UNIVERSITET

Bibliography

Björdal, Gustav; Monette, Jean-Noël; Flener, Pierre; and Pearson, Justin.
A constraint-based local search backend for MiniZinc.
Constraints, 20(3):325-345, 2015.

Hoos, Holger H. and Stützle, Thomas.
Stochastic Local Search: Foundations & Applications.
Elsevier / Morgan Kaufmann, 2004.

Van Hentenryck, Pascal and Michel, Laurent.
Constraint-Based Local Search.
The MIT Press, 2005.

Prestwich, Steven.
Supersymmetric modeling for local search.
SymCon 2002.