Topic 4: Modelling (for CP & LCG)
(Version of 2nd October 2020)

Pierre Flener and Gustav Björdal

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

1 Many thanks to Guido Tack for feedback
Outline

1. Viewpoints
2. Implied Constraints
3. Redundant Variables & Channelling Constraints
4. Pre-Computation
Outline

1. Viewpoints

2. Implied Constraints

3. Redundant Variables & Channelling Constraints

4. Pre-Computation
Recap

1. **Modelling**: express problem in terms of

   - parameters,

   - decision variables,

   - constraints, and

   - objective.

2. **Solving**: solve using a state-of-the-art solver.
Example (Student Seating Problem)

Given:

- $S$ students of $P$ programmes, and
- $C$ chairs positioned around tables, find a seating arrangement such that:

1. Each table has students of distinct study programmes.
2. Each table has either at least half its chairs occupied, or none.
3. A maximum number of student preferences on being seated at the same table are satisfied.

$S = 15$ students
$P = 5$ study programmes
$C = 20$ chairs, with $C \geq S$
Tables = [1..4, ..., 17..20]

What are suitable decision variables for this problem?
A viewpoint is a choice of decision variables.

**Example (Student Seating Problem)**

**Viewpoint 1:**
For each student: which chair is the student assigned to?

```plaintext
1 % Chair[s] = the chair of student s:
2 array[1..S] of var 1..C: Chair;
3 constraint alldifferent(Chair);
```

**Viewpoint 2:**
For each chair: which student, if any, is seated on it?

```plaintext
1 % Student[c] = the student, if any, on chair c:
2 array[1..C] of var 0..S: Student; % dummy student 0
3 constraint alldifferent_except_0(Student);
```

Let us now look at a generic problem in order to see how viewpoints differ when we start formulating constraints. We get back to this problem at slide 19.
Example (Objects, Shapes, and Colours)

There are $n$ objects, $s$ shapes, and $c$ colours, with $s \geq n$. Assign a shape and a colour to each object such that:

1. the objects have distinct shapes;
2. the numbers of objects of the used colours are distinct;
3. other constraints, yielding NP-hardness and distinguishing objects and shapes, are satisfied.

This problem can be modelled from different viewpoints:

1. Which colour, if any, does each shape have?
2. Which shapes, if any, does each colour have?
3. Which shape and colour does each object have?
4. . .

Each viewpoint comes with benefits and drawbacks.
Example (Objects, Shapes, and Colours)

Viewpoint 1: Which colour, if any, does each shape have?

```plaintext
1 int: n; % number of objects
2 int: s; % number of shapes
3 int: c; % number of colours
4 constraint assert(s >= n, "Not enough shapes");
5 % Colour[i] = the colour of the object of shape i:
6 array[1..s] of var 0..c: Colour; % 0 is a dummy colour
7 % There are n objects:
8 constraint count(Colour,0) = s - n;
9 % The numbers of objects of the used colours are distinct:
10 constraint
   alldifferent_except_0(global_cardinality(Colour,1..c));
11 % The objects have distinct shapes:
12 % implied by lines 6 and 8!
13 % ... add here the other constraints ...
14 solve satisfy;
```

Colour 0 is used when there is no object of the given shape. So what are the shape and colour of a particular object?!

Map the objects onto the shapes with a non-0 colour!
Example (Objects, Shapes, and Colours)

Viewpoint 2: Which shapes, if any, does each colour have?

1. int: n; % number of objects
2. int: s; % number of shapes
3. int: c; % number of colours
4. constraint assert(s >= n, "Not enough shapes");
5. % Shapes[i] = the set of shapes of colour i:
6. array[1..c] of var set of 1..s: Shapes;
7. % There are n objects:
8. constraint n = sum(colour in 1..c)(card(Shapes[colour]));
9. % The numbers of objects of the used colours are distinct:
10. constraint alldifferent_except_0(colour in 1..c) (card(Shapes[colour]));
11. % The objects have distinct shapes:
12. constraint n = card(array_union(Shapes));
13. % ... add here the other constraints ...
14. solve satisfy;

Post-process: map the objects onto actually used shapes.
Can we also model this viewpoint without set variables?

♫ Yes, see the next slide!
Example (Objects, Shapes, and Colours)

Viewpoint 2: Which shapes, if any, does each colour have?

1 int: n; % number of objects
2 int: s; % number of shapes
3 int: c; % number of colours
4 constraint assert(s >= n, "Not enough shapes");
5 % NbrObj[i,j] = the number of objects of colour i & shape j:
6 array[1..c,1..s] of var 0..1: NbrObj;
7 % There are n objects:
8 constraint n = sum(NbrObj);
9 % The numbers of objects of the used colours are distinct:
10 constraint alldifferent_except_0(colour in 1..c)
   (sum(NbrObj[colour,..]));
11 % The objects have distinct shapes:
12 constraint forall(shape in 1..s)(sum(NbrObj[..,shape])<=1);
13 % ... add here the other constraints ...
14 solve satisfy;

Which model for viewpoint 2 is clearer or better? ✜ Ask and try!
Example (Objects, Shapes, and Colours)

Viewpoint 3: Which shape & colour does each object have?

```plaintext
1 int: n; % number of objects
2 int: s; % number of shapes
3 int: c; % number of colours
4 constraint assert(s >= n, "Not enough shapes");
5 array[1..n] of var 1..s: Shape; % Shape[i] = shape of obj. i
6 array[1..n] of var 1..c: Colour; % Colour[i] = colour of i
7 % There are n objects:
8 % implied by lines 5 and 6!
9 % The numbers of objects of the used colours are distinct:
10 constraint alldifferent_except_0
   (global_cardinality_closed(Colour,1..c));
11 % The objects have distinct shapes:
12 constraint alldifferent(Shape);
13 % ... add here the other constraints ...
14 solve satisfy;
```

We have used two parallel arrays with the same index set but different domains in order to represent pair variables.
Which viewpoint is better in terms of:

■ Size of the search space:
  • Viewpoint 1: $O((c + 1)^s)$, which is independent of $n$
  • Viewpoint 2: $O(2^{s\cdot c})$, which is independent of $n$
  • Viewpoint 3: $O(s^n \cdot c^n)$

Does this actually matter?

■ Ease of formulating the constraints and the objective:
  • It depends on the unstated other constraints.
  • Ideally, we want a viewpoint that allows global-constraint predicates to be used.

■ Performance:
  • Hard to tell: we have to run experiments!

■ Readability:
  • Who is going to read the model?
  • What is their background?

There are no correct answers here: we actually need to think about this and run experiments.
Outline

1. Viewpoints

2. Implied Constraints

3. Redundant Variables & Channelling Constraints

4. Pre-Computation
Example (The Magic Series Problem)

The element at index $i$ in $I = 0..(n-1)$ is the number of occurrences of $i$. Solution: $\text{Magic} = [1,2,1,0]$ for $n=4$.

**Variables:** $\text{Magic} = \begin{array}{cccc} 0 & 1 & \cdots & n-1 \\ \in I & \in I & \cdots & \in I \end{array}$

**Constraint:**

```plaintext
forall(i in I) (\text{Magic}[i] = \text{sum}(j in I) (\text{Magic}[j]=i))
```

or, logically equivalently but better:

```plaintext
forall(i in I) (\text{count}(\text{Magic},i,\text{Magic}[i]))
```

or, logically equivalently and even better:

```plaintext
\text{global_cardinality_closed}(\text{Magic},I,\text{Magic})
```

**Implied Constraint:**

```plaintext
\text{sum}(\text{Magic}) = n \land \text{sum}(i in I) (\text{Magic}[i] \times i) = n
```

For $n=80$, using a CP solver: only 7 search nodes are explored instead of 302; the solving is 1,000 times faster.
**Definition**

An **implied constraint**, also called a **redundant constraint**, is a constraint that logically follows from other constraints.

**Benefit:**
Solving may be faster, without losing any solutions. However, not all implied constraints accelerate the solving.

**Good practice in MiniZinc:**
Flag implied constraints using the `implied_constraint` predicate. This allows backends to handle them differently, if wanted (see Topic 9: Modelling for CBLS):

```plaintext
predicate implied_constraint(var bool: c) = c;  vs
predicate implied_constraint(var bool: c) = true;
```

**Example**

```plaintext
constraint implied_constraint(sum(Magic) = n);
```

In Topic 5: Symmetry, we see the equally recommended `symmetry_breaking_constraint` predicate.
Outline

1. Viewpoints

2. Implied Constraints

3. Redundant Variables & Channelling Constraints

4. Pre-Computation
Redundant Decision Variables

Example (n-queens)

Use both the $n^2$ decision variables $\text{Queen}[r, c]$ in 0..1 and the $n$ decision variables $\text{Row}[c]$ in 1..n.

Definition

A redundant decision variable represents information that is already available via some other decision variables. We distinguish mutual and non-mutual redundancy.

Benefit: Easier modelling, or faster solving, or both.

Examples (see Topic 6: Case Studies)

- Each $\text{Queen}[., c]$ is mutually redundant with $\text{Row}[c]$.
- Best model of Black-Hole Patience: mutual redundancy
- Models 1 & 3 of Warehouse Location: non-mutual red.
Channelling Constraints

Example (n-queens)

One-way channelling from the \( n \) decision variables \( \text{Row}[c] \) in \( 1..n \) to the \( n^2 \) decision variables \( \text{Queen}[r,c] \) in \( 0..1 \):

\[
\text{constraint } \forall (c \in 1..n) (\text{Queen}[\text{Row}[c], c] = 1)
\]

Definition

A channelling constraint helps establish the coherence of the value of a variable that is redundant with other variables.

Examples (see Topic 6: Case Studies)

- Best model of Black-Hole Patience: 2-way channelling.
- Models 1 & 3 of Warehouse Location: 1-way vs 2-way.
- Experiment with channelling between the viewpoints for the \( \text{Objects, Shapes, and Colours} \) problem (slide 7).
Example (Student Seating, viewpoint 2 revisited)

1 % Student[c] = the student, if any, on chair c:
2 array[1..C] of var 0..S: Student; % dummy student 0
3 constraint alldifferent_except_0(Student);
4 % Pgm[s] = given study programme of student s:
5 array[0..S] of 0..P: Pgm; % dummy prog. 0 for stud. 0
6 % Programme[c] = the programme of the student, if any, on chair c (non-mut. redundant with Student):
7 array[1..C] of var 0..P: Programme;
8 % One-way channelling from Student to Programme:
9 constraint forall(c in 1..C)
10     (Programme[c] = Pgm[Student[c]]);
11 % Each table has students of distinct programmes:
12 constraint forall(T in Tables)
13     (alldifferent([Programme[c] | c in T])); % cons. (1)
14 ... % constraint (2) and objective (3): see slide 5

Note that Student uniquely determines Programme, but not vice-versa: one can also formulate (1) with Student.
Outline

1. Viewpoints
2. Implied Constraints
3. Redundant Variables & Channelling Constraints
4. Pre-Computation
Example (Prize-Pool Division)

Consider a maximisation problem where the objective function is the division of an unknown prize pool by an unknown number of winners:

```plaintext
1 ... 
2 array[1..5] of int: Pools = [1000,5000,15000,20000,25000];
3 var 1..5: x; % index of the actual prize pool within Pools
4 var 1..500: nbrWinners; % the number of winners
5 ... 
6 solve maximize Pools[x] div nbrWinners; % implicit: element!
```

**Observation:** We should beware of using the `div` function on decision variables, because:

- It yields weak inference, at least in CP & LCG solvers.
- Its inference takes unnecessary time and memory.

**Idea:** We can pre-compute all possible objective values.
Idea: We can pre-compute all possible objective values.

Example (Prize-Pool Division, revisited)

Pre-compute a 2d array, indexed by 1..5 and 1..500, for each possible value pair of \( x \) and \( \text{nbrWinners} \):

```
array[1..5] of int: Pools = [1000, 5000, 15000, 20000, 25000];
var 1..5: x; % index of the actual prize pool within Pools
var 1..500: nbrWinners; % the number of winners

array[1..5, 1..500] of int: objVal = array2d(1..5, 1..500,
    [Pools[p] div n | p in 1..5, n in 1..500]);

solve maximize objVal[x, nbrWinners]; % implicit: 2d-element!
```