

FREQUENCY ALLOCATION PROBLEM

1. PROBLEM DESCRIPTION

This problem arises in cellular phone networks where the network is subdivided into cell areas. Each cell contains a number of transmitters whose locations are known. The problem is to devise an allocation scheme that minimises the intra- and inter-cell interferences of transmitters. For this purpose, the non-interference constraints are given as follows.

- frequencies allocated to any two transmitters of the same cell must be at least 16 units apart.
- the distances between any two transmitters belonging to different cells vary according to the individual geographical situation and are given as two-dimensional instance data matrix.

The objective is to assign a frequency to each transmitter such that these constraints are satisfied. An additional interest lies in using a minimal number of frequencies, but is not part of the problem formulation.

```
int nbCells = ...;
int nbFreqs = ...;
range Cells 1..nbCells;
range Freqs 1..nbFreqs;
int nbTrans[Cells] = ...;
int distance[Cells,Cells] = ...;

struct TransmitterType { Cells c; int t; };
{TransmitterType} Transmitters = { <c,t> | c in Cells & t in 1..nbTrans[c] };
var Freqs freq[Transmitters];

solve {
  forall(c in Cells & ordered t1,t2 in 1..nbTrans[c])
    abs(freq[<c,t1>] - freq[<c,t2>]) >= 16;

  forall(ordered c1,c2 in Cells: distance[c1,c2] > 0)
    forall(t1 in 1..nbTrans[c1] & t2 in 1..nbTrans[c2])
      abs(freq[<c,t1>] - freq[<c,t2>]) >= distance[c1,c2];
};
```

FIGURE 1. Frequency allocation programming problem in OPL

2. OPL MODEL

Our specification is a translation of the model in figure 1 (taken from [1, p. 195]), where we have omitted the OPL-specific search part. The instance data is characterised by the number of cells (`nbCells`) and the number of frequencies (`nbFreqs`), which serve to define the Integer ranges `Cells` and `Freqs`, respectively. The array `nbTrans` specifies the number of transmitters per cell, the distance matrix is called `distance`. A

compound data-type `TransmitterType` is used to record the tuples consisting of respective cell and transmitter numbers. The generation of these tuples is done in the subsequent statement, resulting in the set of tuples `Transmitters`. There is only one decision variable, namely the array `freq`. The first statement of the `solve` block asserts the intra-cell non-interference constraints, the second one caters for the inter-cell constraints and uses the `distance` matrix. We further also include the instance data for the model (copied from [2]) in figure 2 on the next page.

3. Z MODEL

3.1. Data modelling. Building the specification systematically, we start with those parts of the instance data which are referred to by other parts later on. These are the `nbCells` and `nbFreq` constants, as well as

$$\left| \begin{array}{l} nbCells, nbFreqs : \mathbb{N} \\ Cells, Freqs : \mathbb{F} \mathbb{N} \end{array} \right|$$

the associated Integer ranges. We further found it useful to separate all non-decision variables of the instance data into the following block.

$$\left| \begin{array}{l} nbTrans : Cells \rightarrow \mathbb{N} \\ distance : (Cells \times Cells) \rightarrow \mathbb{N} \\ Transmitters : Cells \leftrightarrow \mathbb{N} \\ \hline Cells = 1 .. nbCells \\ Freqs = 1 .. nbFreqs \\ Transmitters = \{c : Cells; t : \mathbb{N} \mid t \in 1 .. nbTrans(c)\} \end{array} \right|$$

Notice that Z allows a much more succinct presentation of `Transmitters` in form of a relation (compare with figure 1), we do not need an extra `TransmitterType` here. The resulting set comprehension is comparable to the OPL model.

3.2. Representing the constraints. As in the original model, the only decision variable in the schema below is `freq` which now appears as a function. Both quantified constraints are directly translated.

$$\begin{array}{l} \text{---} \textit{Frequency_Allocation_Problem} \text{---} \\ \hline freq : Transmitters \rightarrow Freqs \\ \hline \forall c : Cells \bullet \forall t_1, t_2 : 1 .. nbTrans(c) \mid t_1 < t_2 \bullet \\ \quad \text{abs}(freq(c, t_1) - freq(c, t_2)) \geq 16 \\ \\ \forall c_1, c_2 : Cells \mid c_1 < c_2 \wedge distance(c_1, c_2) > 0 \bullet \\ \quad \forall t_1 : 1 .. nbTrans(c_1); t_2 : 1 .. nbTrans(c_2) \bullet \\ \quad \text{abs}(freq(c_1, t_1) - freq(c_2, t_2)) \geq distance(c_1, c_2) \\ \hline \end{array}$$

In particular, the ease with which the quite complicated and nested constraints of figure 1 can be transformed into Z is remarkable. The nested first constraint is broken up into a doubly quantified expression, the `ordered` predicate replaced by the `<` relation (according to [1, p. 113]), and the constraint within the second `forall` statement is adjoined to the filter expression in the Z analogue. As a result, the reading based on Z is highly similar to figure 1. The `abs` function is one of the standard predicates we provide in the Appendix.

4. LITERATURE REFERENCES

The problem first appeared in the Ilog Solver manual [3, chap. 25] and was adapted for OPL in [1, sec. 10.4] and further also published in [2].

REFERENCES

- [1] Pascal Van Hentenryck. *The OPL Optimization Programming Language*. MIT Press, January 1999.
- [2] Pascal Van Hentenryck, Laurent Michel, Laurent Perron, and Jean-Charles Régin. Constraint Programming in OPL. In Gopalan Nadathur, editor, *Proceedings of the International Conference on Principles and Practice of Declarative Programming (PPDP'99)*, volume 1702 of *LNCS*, pages 98–116. Springer, 1999.
- [3] ILOG, France. *Ilog Solver 4.4, User's Manual*, May 1999.

```

nbCells = 25;
nbFreqs = 256;
nbTrans = [8 6 6 1 4 4 8 8 8 8 4 9 8 4 4 10 8 9 8 4 5 4 8 1 1];
distance = [
  [1 16 2 0 0 0 0 0 2 2 1 1 1 2 2 1 1 0 0 0 0 0 0 0 0]
  [1 2 16 0 0 0 0 0 2 2 1 1 1 2 2 1 1 0 0 0 0 0 0 0 0]
  [0 0 0 16 2 2 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 0 0 0 1 1]
  [0 0 0 2 16 2 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 0 0 0 1 1]
  [0 0 0 2 2 16 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 0 0 0 1 1]
  [0 0 0 0 0 0 16 2 0 0 1 1 1 0 0 1 1 1 1 2 0 0 0 1 1]
  [0 0 0 0 0 0 2 16 0 0 1 1 1 0 0 1 1 1 1 2 0 0 0 1 1]
  [1 2 2 0 0 0 0 0 16 2 2 2 2 2 2 1 1 1 1 1 1 1 0 1 1]
  [1 2 2 0 0 0 0 0 2 16 2 2 2 2 2 1 1 1 1 1 1 1 0 1 1]
  [1 1 1 0 0 0 0 1 1 2 2 16 2 2 2 2 2 2 1 1 2 1 1 0 1 1]
  [1 1 1 0 0 0 0 1 1 2 2 2 16 2 2 2 2 2 1 1 2 1 1 0 1 1]
  [1 1 1 0 0 0 0 1 1 2 2 2 2 16 2 1 1 1 1 1 1 1 1 1 1]
  [2 2 2 0 0 0 0 0 2 2 2 2 2 2 16 2 1 1 1 1 1 1 1 1 1]
  [2 2 2 0 0 0 0 0 2 2 2 2 2 2 16 1 1 1 1 1 1 1 1 1 1]
  [1 1 1 0 0 0 0 1 1 1 2 2 2 1 1 16 2 2 2 1 2 2 1 2 2]
  [1 1 1 0 0 0 0 1 1 1 2 2 2 1 1 2 16 2 2 1 2 2 1 2 2]
  [0 0 0 1 1 1 1 1 1 1 1 1 1 1 1 1 2 2 16 2 2 1 1 0 2 2]
  [0 0 0 1 1 1 1 1 1 1 1 1 1 1 1 1 2 2 2 16 2 1 1 0 2 2]
  [0 0 0 1 1 1 2 2 1 1 2 2 2 1 1 1 1 2 2 16 1 1 0 1 1]
  [2 0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 2 2 1 1 1 16 2 1 2 2]
  [2 0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 2 2 1 1 1 2 16 1 2 2]
  [1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 1 0 0 0 1 1 16 1 1]
  [1 0 0 1 1 1 1 1 1 1 1 1 1 1 1 1 2 2 2 2 1 2 2 1 16 2]
  [1 0 0 1 1 1 1 1 1 1 1 1 1 1 1 1 2 2 2 2 1 2 2 1 2 16]];
};

```

FIGURE 2. Instance data for the OPL model of figure 1
