Constraint Solving - Global Constraints

1. Traveling Salesperson (TSP)

The distances between a set cities in Bavaria are specified in files **bavariaNN.txt** (where NN represents the number of cities considered), in **Bavaria_benchmarks.zip**[†].

The files start by the number **k** of cities, followed by the adjacency matrix that constains the distances between all pairs of cities. For example, the file "bavaria07.txt" contains the following text:

```
7
  0 107 241 190 124  80 316
107   0 148 137  88 127 336
241 148   0 374 171 259 509
190 137 374   0 202 234 222
124  88 171 202   0  61 392
 80 127 259 234  61   0 386
316 336 509 222 392 386   0
```

The TSP problem consists of finding the shortest tour required for a salesman to visit all cities, without visiting any city twice, and returning to the starting city. More formally, considering the graph G = (N,E) where N is the set of k nodes (corresponding to the cities) and E the set of edges between the nodes labelled with their costs (distances in this case), the TSP problem consists of finding the Hamiltonian cycle in the graph G with lowest cost.

Rank: Model (and solve) the problem with array **`rank[1..k]`** of decision variables, where **`rank[i]`** represents the i-th city to be visited in the tour. For example, tour $1 \rightarrow 5 \rightarrow 2 \rightarrow 6 \rightarrow 7 \rightarrow 4 \rightarrow 3 \rightarrow 1$ is represented by **`rank = [1,5,2,6,7,4,3]`**.

Next: Solve the problem with an alternative model using an array **`next[1..k]`** of decision variables, where **`next[i]`** represents the city that follows city **`i`** in the tour. The above solution is now represented by **`next = [5,6,1,3,2,7,4].`**

In both the above models adopt the symmetry breaking assumption that the tour starts in city 1, and make sure that your solution is not composed of sub-cycles. Which of the models is more efficient?

Global: Solve the TSP problem with the model Next, but now using the global constraints **circuit** and **circuitMin** available in Comet. Compare the efficiency of the execution for various graphs available in file "**bavaria.zip**".

[†] **Source: http://comopt.ifi.uni-heidelberg.de/software/TSPLIB95/**
benchmark: bayg29.tsp.gz

Sugestion for Reading Data Files:

To read a data file with integers adapt the following procedure that reads a file with data in the format of the data file " **bavariaNN.txt** ", with name **fname**, placed in the same directory of the code file.

```
function int [,] read_mat(Integer N, string fname){
   string [] dirs = System.getArgs();
   string directory = dirs[3].suffix(2);
   string ename = directory + "/" + fname;
   ifstream file(ename);
   N := file.getInt();      // the number of cities
   int d [1..N, 1..N];
   forall(i in 1..N, j in 1 .. N) d[i,j] = file.getInt();
   return d;
}
```

2. Job Shop

A job shop problem consists of scheduling J jobs, each consisting of T tasks, which have precedence constraints. The jobs are independent, except for the fact that the tasks are executed in machines of certain types and there are only a limited number of machines of each type. The goal is to finish all tasks within a certain makespan (Satisfaction) or to minimize the makespan.

i. Solve the job-shop problem for (small) instances obtained from the OR-library[1]:

For example, benchmark " la03.txt", (other benchmarks available in jobshop_benchmarks.zip, together with file read_jshp_mat.co with a function to read this type of files) with a wotherhich has the following data:

```
instance la03
Lawrence 10x5 instance (Table 3, instance 3); also called (setf3) or (F3)
10 5
1 23 2 45 0 82 4 84 3 38
2 21 1 29 0 18 4 41 3 50
2 38 3 54 4 16 0 52 1 52
4 37 0 54 2 74 1 62 3 57
4 57 0 81 1 61 3 68 2 30
4 81 0 79 1 89 2 89 3 11
3 33 2 20 0 91 4 20 1 66
4 24 1 84 0 32 2 55 3  8
4 56 0  7 3 54 2 64 1 39
4 40 1 83 0 19 2  8 3  7
```

specifies a problem with 10 jobs (rows) and 5 tasks each, where each row indicates for that job the types of the machines in which the tasks are executed and their duration. For example job 1 is composed of 5 tasks, to be executed, respectively, in machines of type 1,2,0,4 and 3, with duration 23, 45, 82, 84 and 38.

a) Consider the problem variants of satisfaction (finish all tasks before some time T) or minimisation (minimise this time T).

ii. Consider a variant of the problem where all tasks of all jobs can be executed in any of M machines of the same type (where N is a given number). Adapt the program you have used to solve the job shop problem, and specify a function

cumulate(int o,int h, int maxCap, var<CP>{int}[] s, var<CP>{int}[] d, int [] r)

to constrain a set of tasks starting in time slots **s** and durations **d,** each consuming **r** units of resources, to be executed between time slots **o** and **h**, so that in none of these time slots the resources used by the tasks in execution exceeds **maxCap**.

a) Reformulate your solution to the **jobshop** instances, using this user-defined function **cumulate**.

b) Replace your function (**cumulate**) by the system-defined global constraint **cumulative/6**. Check the efficiency of the specialised propagation algorithms that is used in the global constraint, compared with the propagation obtained by simple decomposition of the global constraint (as you did in the spec of function cumulate).

[1] ORlibrary URL: http://people.brunel.ac.uk/~mastjjb/jeb/info.html. Job shop benchmarks from library (available in the course page) obtained from http://people.brunel.ac.uk/~mastjjb/jeb/orlib/files/jobshop1.txt