Constraint Solving - Global Constraints (1)

Traveling Salesperson (TSP)

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[0..k-1]`** of decision variables, where **`rank[i]`** represents the i[th] city to be visited in the tour. For example,

`rank = [0,4,1,5,6,3,2]` represents the tour $0 \rightarrow 4 \rightarrow 1 \rightarrow 5 \rightarrow 6 \rightarrow 3 \rightarrow 2 \rightarrow 0$

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 = [4,5,0,2,1,6,3].`**

In both the above models use when convenient **global constraints**, namely **alldifferent**, **circuit** and element as available in Choco.

Also impose, if necessary, symmetry breaking constraints to guarantee that the tour starts in city 0. Which of the models is more efficient?

Testing / Benchmarks:

For both models, extend the file **tsp_aula.java**, available in the solutions, that includes an adjacency matrix of a graph with 15 nodes.

You may consider other graphs, e.g. from files **bavariaNN.txt**, where the above graph was extracted (**bavaria15.txt**, below).

```
15
   0 107 241 190 124  80 316  76 152 157 283 133 113 297 228
 107   0 148 137  88 127 336 183 134  95 254 180 101 234 175
 241 148   0 374 171 259 509 317 217 232 491 312 280 391 412
 190 137 374   0 202 234 222 192 248  42 117 287  79 107  38
 124  88 171 202   0  61 392 202  46 160 319 112 163 322 240
  80 127 259 234  61   0 386 141  72 167 351  55 157 331 272
 316 336 509 222 392 386   0 233 438 254 202 439 235 254 210
  76 183 317 192 202 141 233   0 213 188 272 193 131 302 233
 152 134 217 248  46  72 438 213   0 206 365  89 209 368 286
 157  95 232  42 160 167 254 188 206   0 159 220  57 149  80
 283 254 491 117 319 351 202 272 365 159   0 404 176 106  79
 133 180 312 287 112  55 439 193  89 220 404   0 210 384 325
 113 101 280  79 163 157 235 131 209  57 176 210   0 186 117
 297 234 391 107 322 331 254 302 368 149 106 384 186   0  69
 228 175 412  38 240 272 210 233 286  80  79 325 117  69   0
```

To read a data file with integers with the format above, use the class **graph.java** that is available in the web page (note: adapt, if necessary, the path for the file to be read).

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