CS4093 Special Topics in Computing: Constraint Programming and Optimisation
2nd Continuous Assessment 2015/16

The manager of an Engineering Maintenance company has asked you for help in designing work rotas for the maintenance team. There are *n* maintenance workers and the rota must schedule all workers over the next *m* days. In addition to the general maintenance work, there are some special skills needed for some tasks – *k* different skills in total. Each worker may or may not be trained in each skill. Depending on the expected work items, each day of the rota has a requirement for a minimum number of workers with that skill. A single worker may supply more than one skill at a time. Every worker must work at least *d* days over the period of the rota. Any days in addition to this number are considered overtime, and each worker has their own overtime rate. For health and safety reasons, there is a maximum number of days in a row, *p*, that any individual can work.

There are two stages in producing a solution for the company:
(i) find a schedule which ensures every worker is scheduled for at least the minimum number of days, that all skill requirements are satisfied, and that the total overtime cost is minimised;
(ii) find a schedule obeys all the requirements of part (i), and which also ensures no worker works more than the maximum allowed days in a row.

Problem instances are described in a standard format in text files:

```
n m k d p
o[0] o[1] ... o[n-1]
s[0][0] s[0][1] ... s[0][n-1]
...
s[k-1][0] s[k-1][1] ... s[k-1][n-1]
r[0][0] r[0][1] ... r[0][m-1]
...
r[k-1][0] r[k-1][1] ... r[k-1][m-1]
```

where *n* is the number of workers, *m* is the number of days,*k* is the number of skills, *d* is the minimum number of days each worker must work, and *p* is the maximum number of days in a row that can be worked. The values `s[i][j]` represent the skills for each worker, with s[i][j] == 1 if and only if skill i is possessed by worker *j*. The values r[i][j] represent the skill requirements for each day, with r[i][j] = q if and only if at least *q* workers with skill *i* are required on day *j*.

For example, in the file "rota1.txt":

```
3 3 2 1 2
5 9 7
1 0 1
0 1 1
1 2 1
2 1 1
```

there are 3 workers to be scheduled over 3 days, with 2 special skills. Each worker must work at least 1 day, and no worker should work for more than 2 days in a row. Worker 0 has overtime rate 5, worker 1 has rate 9, and worker 2 has rate 7. Worker 0 has skill 0, worker 1 has skill 1, and worker 2 has both skills. Day 0 requires 1 worker with skill 0 and two with skill 1, day 1 requires 2 workers with skill 0 and 1 with skill 1, and day 2 requires one worker with skill 0 and one with skill 1.

For the sample problem in rota 1, the optimal solution to part (i) has worker 0 only on day 1, worker 1 only on day 0, and worker 2 on all three days, with a total overtime cost of 14. The optimal solution to part (ii) for rota 1 has worker 0 on days 1 & 2, worker 1 on days 0 and 2, and worker 2 on days 0 and 1, with a total overtime cost of 21.

A class `RotaProblemReader.java` is provided, which will read the problem instance from a data file, and supports the following public getter methods:

```
public int getNumberOfWorkers();  //i.e. n
public int getNumberOfDays();     // m
public int getNumberOfSkills();   // k
public int getMinDaysWorked();    // d
public int getMaxSequence();      // p
public int[] getOvertimeRate();   // the o[] array
public int[][] getSkillWorkers(); // the s[][] array
public int[][] getMinSkills();    // the r[][] array
```

The assignment

1. Create a Choco class for modeling and solving problems as described above. Your class must be called "Rota.java", it must have an executable main method, and it must read in data from text files in the above format. Include in your submission any other Java classes that you wrote that your main class needs to compile and run. Do not include the Choco jar files. If you do not use RotaProblemReader.java, then you must include your own method for reading the files.
   Remember that the idea is to describe the problem to Choco and then ask Choco to solve it. You can help Choco by telling it what search strategy to use.
2. Run your Choco models on the three text files linked on the Assessment page. Write a brief report describing your models, explaining any different ways of representing the problem that you considered and why you chose the one you did, and showing the results you get when you run you Java/Choco code on the data files. The report should be formatted as a plain text file (.txt).

Marking scheme

Marks will be awarded for submitting a working Java/Choco program, for generating correct solutions, for the quality of model, for good use of Choco facilities including global constraints if needed, and for the explanation in the report. Marks will not be awarded for the quality of the Java code that supports the model, so don't waste time implementing a GUI to show the results (although if your code is inadequately commented and is difficult to understand, you may lose marks). If you do not manage to get a complete solution to part(i) or part(ii), you will get credit for how far you got, as long as you submit a working version and you explain in your report which constraints you have successfully implemented.

Submission Deadline

All files and reports must be contained in a single zipped tar file, and emailed to Ken Brown (k.brown@cs.ucc.ie) by 5pm, Wednesday 2nd March.

Plagiarism

This submission is part of the formal assessment for the module, and so must be your own work. You should not submit anyone else's work (or part of their work) as if it were your own, and you should not be collaborating with each other on how to solve the problem. However, you are free to re-use, without acknowledgement, any piece of Java/Choco code that has been posted on the CS4093 web pages or included in CS4093 Lecture slides, or included in the Choco manual. If you use code you obtained from anywhere else, you must give a citation for it in your report.