2508ICT Principles of Intelligent Systems

Assignment 2: Nurse Rostering

James Dahms: s2760619

Philip Ward: s2847153

# Problem State Space Representation

The problem is defined using a set of objects that represent different levels of the rostering system that was described in the assignment brief.

Nurse:

Grade – This is a representation of the grade the nurse carries, it is stored as an Enum to make things more space efficient, but it can either be SRN (1) or RN (2).

Shift – This represents the shift pattern that the nurse is assigned, again, it is stored as an Enum, however the values are D (4) for Day shift, N (5) for Night shift and DN (6) for the Day and Night shift pattern.

lastOff – This is the number of days that the nurse has worked since her last day off, it has a domain of all positive integers including 0, however in theory in the problem given it should not exceed the value of 5, as this is the maximum number of shifts a nurse should work before having a day off.

lastShift – This represents the shift type that the nurse was last assigned, it is defined in the Enums from the Roster class and has a value of either: NOT\_SET (-1), Day off (0), day shift (1) or night shift (2).

These variables are all used in deciding if the given nurse is able to take a shift that the program tries to give to her, so for example if the nurse is on shift pattern D and the program attempts to assign her a shift type of N, it would return false and not allow for the shift to be set.

Roster:

Roster – This is a 2D array which represents the actual roster that the nurses use, with the first number being the number of the nurse in question, and the second number being the number of the day which is being examined.

Period – This represents the length of time the roster represents, it is defined as one of the two Enums of ROSTER\_7\_DAY (7) and ROSTER\_14\_DAY (14), it is used for some of the calculations such as the maximum number of days that a nurse has worked and thus if she is able to work any more days.

Problem:

The problem class is the main class that holds all the required information for the rostering program. It is the one that has the majority of the methods that are called for deciding if a shift can be assigned and if the overall roster is doable or not.

nurseList – this is an array that holds a list of Nurse Objects, this array is used in parallel with the roster object to represent the nurses and their shifts.

Roster – This is a roster object that is created from the given number of nurses and the roster length.

maxShifts – This represents the maximum number of shifts that a nurse is able to work during the roster, it is defined by the system when the problem object is created and set to either 5 or 10 depending on if the roster length is 7 or 14 days.

minShiftDay / minShiftNight – these are two arrays that work in parallel with the roster object to define the minimum staffing requirements for each day, they are set automatically by the system if no input was defined by the user, but can contain any value.

As specified above, the majority of the non-numerical constraints have been translated into numerical ones by the use of Enums, these are used to set the properties of everything in the program so that stray values are kept out. As they are all set to numbers and all kept within certain values, this means that comparisons are easier to do, and thus a lot of the constraints that were specified are reduced to simple number comparisons which can be used to return Boolean values.

# Program Explanation

The program uses the recursive backtracking method of solving the problem, this was selected simply because it is a very easy to understand algorithm and therefore it was easier to understand from the ground up and build from scratch. While the system uses very basic backtracking to do the search, it does have some checks at the beginning, before the searching even really begins, to make sure that some of the things that are easily avoidable and would cause failure of the search but after a lot of processing are weeded out.

For example before beginning the search the program ensures that there are enough SRN nurses, and that they have the shift patterns to cover both the day and the night shifts, this means that if there were only three SRN nurses the program would not have to search through every permutation of the roster before coming to the conclusion that it is not possible to fill it.

The program also contains the functionality to specify certain aspects of the roster, such as being able to set beforehand what shift type a nurse will be doing on a certain day, and because the backtracking algorithm is given only shifts that are currently empty, these preset values will not be overwritten by the program attempting to find a solution.

# Experimental Results

## Run 1

* 7 Nurses over a period of 7 Days.
* 4 SRN Nurses and 3 RN Nurses
* Shift Pattern
  + D
  + DN
  + DN
  + DN
  + D
  + D
  + N
* No preset Shifts
* No preset staffing requirements

Result:

Nurse 1: D D D D D O O

Nurse 2: D D D N N O O

Nurse 3: D D D O O D D

Nurse 4: N N N O O N N

Nurse 5: D O O D D D D

Nurse 6: D O O D D D D

Nurse 7: N N N N N O O

Run time – 4 min 6 seconds

## Run 2

* Same settings as before, but for 14 days

Results: No results.

Run time – Cancelled after 30 minutes. This run exceeded reasonable time.

## Run 3

* 12 Nurses over a period of 7 Days.
* 4 SRN Nurses and 8 RN Nurses
* Shift Pattern
  + D
  + DN
  + DN
  + DN
  + D
  + D
  + D
  + D
  + D
  + DN
  + N
  + N
* No Preset Shifts
* No preset staffing requirements

Results: No results.

Run time – Cancelled after 30 minutes. This run exceeded reasonable time.

# Phase Transition Behavior

The program is able to correctly and easily solve a roster with 7 nurses. If the ward has 4 nurses or less the algorithm will fail. Any more nurses then 7 in a ward will start to increase the run time of the roster solver. The run time is increased exponentially for each nurse over 7 in a ward.

When the ward is set to a fortnightly roster the time it takes to solve a roster increases drastically. We have been unable to correctly solve a fortnightly roster, even with the minimum amount of nurses.

# Special Included Features

The program includes a fully functioning GUI that sits on top of the solver functions, it can display multiple Wards all with their own set of nurses, all these settings can then be saved for running at another time.

The RosterSolver class file also contains a variable which can be used as a switch to determine the direction that the search moves in, these two options are to either attempt to fill in the data for a nurse first, or to attempt to fill in the data for a day first. This choice has a significant impact on the speed of completion of the program, it is set to Day first by default as the nurse first approach did not seem to return results in a timely manner.

The program was also all created from scratch using only the information in the lecture slides to build the algorithm from, this was done as it was seen as a better way to get to understand the way that the algorithm works, rather than using any of the pre made ones which would be potentially cumbersome, but also detract from some of the potential learning.

# Missing Requirements

Most requirements where included in this assignment. The requirements that were missing are:

* No way for the user to enter set roster days for specific nurses.
* The number of shits a nurse can have in a single roster is either set at 5 for weekly rosters or 10 for fortnightly rosters.
* No heuristics were included into the roster solver to increase the speed of resolving a roster problem.
* The program can solve a roster for certain ward configurations but when the roster is too complicated the program cannot solve a roster in a reasonable time.