# COL333 Assignment2

Submitted by: Dhairya Gupta, 2019CS50428

Ishan Anchit, 2019CS50434

Implementation details:

The assignment requires us to solve constraint satisfaction problem of automated nurse rostering.

Part(a):

In this assignment, we are using backtracking search with optimization wherever possible to get an efficient method of rostering. The configuration is given as a numpy array to each node of the CSP search tree. The children of each configuration node are assigned an entire column. So, at a depth ‘d’ in our CSP tree(with d = 0 at root), the node configuration will have (d + 1) columns assigned. We already assign the root with a column because by the property of commutativity, any other initial assignment would simply be a reordering and has no structural changes in configuration. Commutativity is heavily used to optimize the search algorithm in part(a) as well as part(b).

We assign columns sequentially because the constraints, by default, make them follow the Minimum Remaining Values Heuristic. Our order of iterating over values follows ‘Least constraining values’ heuristic. For example, we always assign maximum ‘Rest’ days preceding ‘Mornings’ which have not yet satisfid C2 below.

We are given 3 constraints:

C1. Each column will have m “Morning” slots, e “evening” slots, …

C2. Each row must have at least one ‘Rest’ in day 0 to day 6, day 7 to day 13…

C3. Morning slots cannot be preceded by Morning/Evening slots

Every column assignment at CSP node always take care of constraint 1. Also, since we progress sequentially column-wise(from right to left in our case), constraint3 is always taken care of. For constraint2, we can apply some forward checking. Another thing to note is that depending on when the ‘rest’ day requirement is complete, the assignment of next column only becomes a function of the previous column(to check constraint 3) and the rows for which constraint2 is already satisfied. We use commutativity further here, and can group rows into 4 groups(C2 satisfied, C3 satisfied, C2 and C3 satisfied, Both C2 and C3 not satisfied). Reordering within a group does not matter. This significantly reduces the number of nodes to be checked.

Another important aspect we have used is that we are calculating rosters week by week instead of all days together, taking care of C3 between 2 weeks. This also hugely lowers the number of nodes to be checked. It is feasible to calculate weeks separately because the constraints only depend on the immediately preceding column and not the entire week(‘Rest’ days can be stores in separate column array to remove dependence on remaining columns.)

Finally, we use forward checking as well. At any configuration of columns, if the number of rows requiring ‘Rest’ days in the future exceeds maximum ‘Rest’ days that can be given, we backtrack. Also, if we are at the 2nd last column, we can check if ‘Rest’ days’ constraint cannot be satisfied without moving to last column.

Part(b)

Many of the implementation details are common with part(a). A few differences in implementation are as follows:

We fix the first S rows to be for special nurses. It is tedious in part(b) to treat all weeks as separate. However, solving for many weeks in a single CSP search seemed to be too computationally inefficient. Therefore, we have used a greedy approach to assign columns at the beginning of a new week instead of iterating over all possible column assignments. Also, commutativity cannot be applied as before. However, we can still make use of commutativity with some changes. In this case, we have the constraints C2 and C3 as well as a preference to give ‘Morning’ and ‘Evening’ days to special nurses. So, we can classify our rows into 8 groups(compared to 4 groups previously). Each group has a True/False value for satisfying C2, C3 and being a special nurse row. Thus, commutativity within these groups gives reduced number of nodes to check. For example, when iterating over possible node assignments, we assign maximum number of ’Rest’ nodes to non-special nodes preceding ‘Morning’ slots with C2 not yet satisfied. Here we are following the ‘Least Constraining value’ heuristic(ruling out fewer choices) for neighbour assignments.