Screen Name: KidButtowski • **Name:** Anderson Addo • **CS Login:** aaddo

Variables and Constraints Used

I'll be brief here to make space for the more interesting parts of the report! I used two different 2d arrays to hold the decision variables, one for shift codes and one for working durations. Each 2D array was of size [numEmployees][numDays]. When a constraint on a day (i.e. a column of these matrices) was needed, it was calculated on the fly; no transpose views were used.
I used the normal obvious constraints that were discussed in class. Nothing special there!

Different Search Algorithms

My fastest algorithm for search was arguably the least interesting. It solved the model in two phases, first on the flattened shiftCodes matrix, then on the flattened hoursWorked matrix. Variables in both matrices were chosen by looking for variables with the smallest domain (to find unsat variables faster). Upon a tie, variables with the largest domain max value were chosen. For shiftCodes, this was because evening shifts and day shifts, which have the two largest shift codes, are the least likely to cause conflicts with constraints. For the hoursWorked matrix, this criteria didn't matter much. I surmised that the value chooser that matters more for hoursWorked since hoursWorked domain changes are less complicated. Though, upon second thought, maybe choosing hoursWorked variables with smallest domain min might have led to faster conflicts discovery since the only issue hoursWorked values can cause would stem from someone cannot work long enough to full hork hour quotas.

shiftCodes values were chosen based on IBS (followed by random selection upon ties) since shift choices generally have large downstream effects. hoursWorked values were picked using their highest available domain value, since I assumed it's safer to test larger work durations first since work quotas are more likely to be met (though, on second thought, it's also possible that criteria makes it easier to violate the 40hrs/week limit for employees). There was also use of backtrack limits and automated restarts.

Symmetry breaking was also used. I constrained the "order" of the employees in the schedule such that the first 2 days of their onboarding period is lexicographically ordered. I initially tried using all 4 days; I think this made the symmetry breaking a bit too restrictive and slowed things down.

Making this search algorithm was strange because there were several intuitively obvious optimizations I tried that slowed things down. My initial designs were "smarter." One thing I had tried was to solve the model employee by employee (so there were 2 phases *per employee*) to get smaller search trees. The value picker for both matrices was also IBS at one point. Another thing I tried was to instead solve the entire model week by week using two phases on flattered matrices per week (see employeeBasedMatrixFlattenerByWeek function). Since weeks are *mostly* independent, I figured the smaller search trees would be beneficial. I also tried explicitly giving day shifts the highest priority (and night shifts the lowest priority) when choosing values for shiftCodes, since night shifts caused the most conflicts and day shifts satisfied the most constraints.
Regardless, the final simple search algorithm was able to solve most of the instances the fastest.

I actually made a number of search algorithms *before* my fastest one (partially because I misunderstood the assignment). These ones were not optimized for

speed, but for schedule quality (which I evaluated by how regular the schedule of each individual employee was). For me, regularity involved regularity in work hours and regularity in shift type over the entire work period.

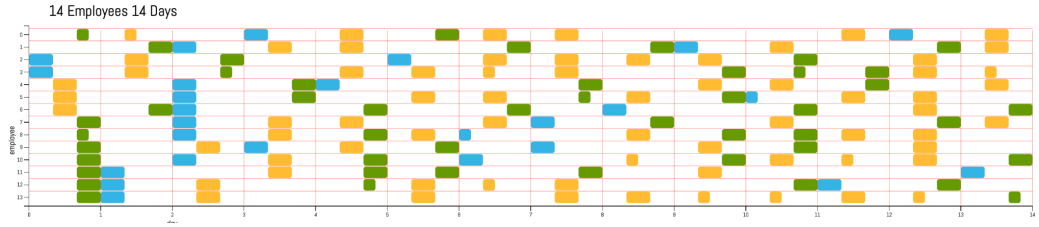
- **setSearchMethodDayEveningBiasFirstAttempt:** This was my first algorithm! The idea behind this was to see what happened if we delegated 50% of the workforce as day workers and 50% as evening workers by changing their intValueChoosers. So, 50% of the workforce (day workers) was given shiftCode intValueChoosers that prioritized smaller values, and the other half prioritized higher values. Note that for the day workers, I assumed it would be ok that technically, night shifts would be chosen *before* day shifts since too many night shifts cause quick conflicts (so intValueChoosers would move on to day shifts quickly). intVarChoosers were all impact based, and all the hours for all employees were solved in a single phase afterwards. The outputs resulted in regular schedules for employees, but the day workers took all the offshifts first, so it resulted in pretty unfair work distribution (example in appendix).

- **setSearchMethodDayEveningBiasRandomOff:** This had a similar motivation to setSearchMethodDayEveningBiasFirstAttempt, but I wanted to prevent people from hogging offshifts. In this algorithm, the day workers were given intValueChoosers that explicitly prioritized day shifts and deprioritized night and evening shifts, but gave off shifts a random evaluation *per employee*. Similarly, the intValueChoosers for evening workers gave night and evening shifts equal evaluations, and off shifts random evaluations. It worked; off shifts were better distributed, but the schedule is less regular. Example in appendix.

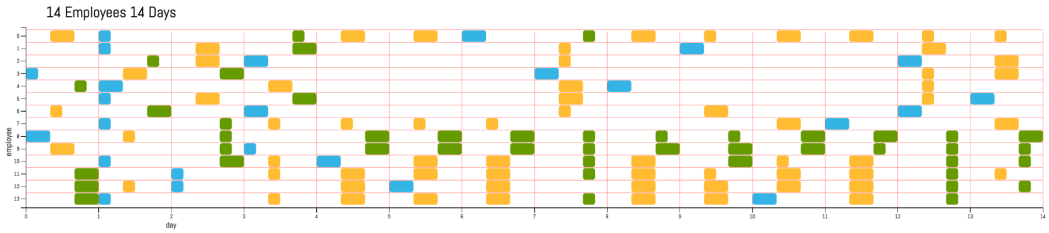
- **setSearchMethodCoreStaffSupportStaff:** This algorithm was an attempt to make certain employees more “important” than other employees. The motivation here is that if we solve the model day by day (instead of employee by employee) and use a fixed order for which employee’s variables get solved first, then an IBS value chooser could result in certain employees being more “important” to the satisfaction of constraints since, per day, they are always solved first in a way that causes the most impact to the model. I figured that more important employees would happen to have the most day shifts since due to the influence of minDemandDayShift. It ended up not making a huge difference from setSearchMethodDayEveningBiasRandomOff though (though in some instances it resulted in more contiguous periods of offshifts). Example in appendix.

- **setSearchMethodCoreStaffSupportStaffBadOffwork:** This algorithm was very similar to setSearchMethodCoreStaffSupportStaff; the only difference is that now the value chooser tries IBS for all the shifts *except* the off shift, then only chooses offshift if its the only remaining option. Didn’t really make a huge difference; looks just like a shuffled version of setSearchMethodCoreStaffSupportStaff.

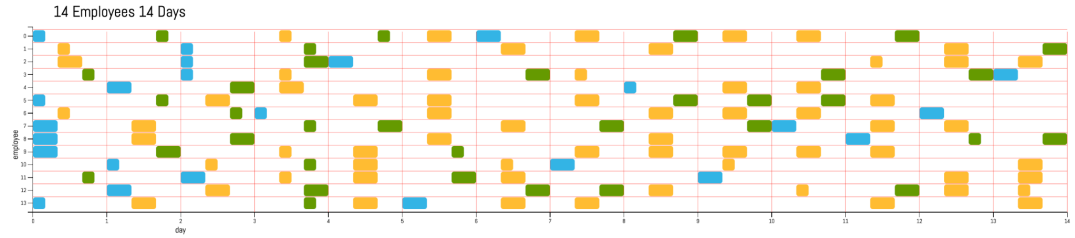
Appendix



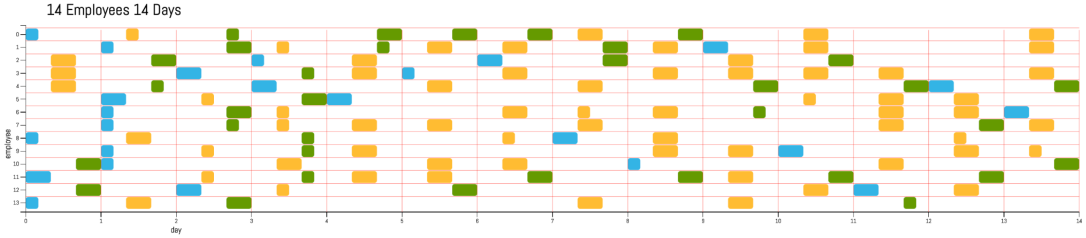
^Output from my fastest performance-based search algorithm, with DepthFirst on and symmetry breaking enabled. Irregularity on individual employee schedules remains even if DepthFirst is not used. Notice the ordering of the employees are based on the first few days of their schedule.



^This is output from setSearchMethodDayEveningBiasFirstAttempt. No symmetry breaking was used, and DepthFirst was not used (just for speed, but also because when I was initially designing all these quality-focused algorithms I didn't know we were supposed to use DepthFirst and didn't have time to redesign them with DepthFirst in mind. This is the case for all my quality-focused algorithms).



^This is output from setSearchMethodDayEveningBiasRandomOff. Offshifts are better distributed, but individual schedule regularity is a little worse.



^ Output from setSearchMethodCoreStaffSupportStaff. setSearchMethodCoreStaffSupportStaffBadOffwork is similar, just a little shuffled.