1 Data Structures

To represent an employee on a given day, we constructed a Java class called `EmployeeState`. An `EmployeeState` object is defined by two `IloIntVar`s, `shift` and `length`. The domain of `shift` is $\{0, 1, 2, 3\}$, as defined in the handout. The domain of `length` is $\{0, 4, 5, 6, 7, 8\}$, where 0 is the shift length corresponding to the off shift, and the remainder of the domain is defined such that shift length is $\geq$ `minConsecutiveWork` and $\leq$ `maxDailyWork`. It directly follows that shift length constraints were imposed within the `EmployeeState` class itself. In addition, since we observed that `maxConsecutiveNightShift` $= 1$ for all scheduling instances, we imposed this constraint (within the `EmployeeState` class) by asserting that `shift` cannot be the night shift if yesterday's `shift` was the night shift. With this, an instance of the employee scheduling problem can be represented as a `numEmployees` $\times$ `numDays` two-dimensional array of `EmployeeState` objects.

Initially, we modeled an `EmployeeState` with three `IloIntVar`s, one corresponding to the shift, and two (with domains $\{-1, 0, 1, 2, \ldots, 23\}$) corresponding to the start and end times respectively. Realizing that we could simply require employees to start at the beginning of their shifts without violating any constraints, we replaced these start and end time variables with the `length` variable described above. As expected, replacing two variables with domains of length 25 with a single variable with a domain of length 6 vastly reduced our search space (and thus improved performance). It also added parsimony to the `EmployeeState` class, as we no longer had to impose constraints ensuring consistency between the shift and start and end times.

2 Optimization

2.1 Global versus Local Constraints

When attempting to improve the performance of our model prior to switching to DFS, we refactored all of our local constraints imposed using the `count` method (i.e. our `maxTotalNightShift` and `minDemandDayShift` constraints) to global constraints imposed using the `distribute` method. We added these constraints in their local and in their global form to the optimizer. Technically, adding the global form of the constraint did not further constrain the assignment space, although we noticed that optimizer performance improved with both constraints when the IMB intelligent search was used; a marginal increase in performance was noted by changing local constraints to global constraints after we switched to DFS.

2.2 Variable and Value Selection

We modified the default variable selector to choose the unassigned variable with the smallest domain size. If multiple variables had the same domain size at a decision point, the selector would break ties by choosing the variable with the largest impact (largest average reduction in search space by assignment). If multiple variables had the same impact, the selector would break ties by choosing one of the variables uniformly at random from the set of tiebreak variables.

We modified the default value selector to assign variables to the largest value in their domain. In regards to our `IloIntVar`s representing shift length for an `EmployeeState`, this would correspond to choosing a shift length of 8 hours by default. This behavior is beneficial because, all else being equal, choosing the maximum time that an employee can work for a given shift as compared to choosing a smaller amount of time will allow us to better satisfy the `minWeeklyWork` constraint. In regards to our `IloIntVar`s representing shift for an `EmployeeState`, this would correspond to choosing night shifts by default; this behavior is neither remarkably beneficial or pernicious.

2.3 Symmetry Breaking

We observed that there existed symmetry between individual employee schedules in this problem. That is, given a valid total scheduling, the schedule for employee i could be interchanged with the schedule for employee j, and the result would still be a valid total scheduling. We broke this symmetry by asserting lexicographic ordering over the employee schedules.

2.4 Search Phases

We partitioned our assignment space into two distinct search phases: all `IloIntVar`s corresponding to the training phase (first numShifts days of schedule), and all `IloIntVar`s corresponding to the remaining phase of the scheduling problem. Generally, we found that the subsystem corresponding to the training phase had fewer relative degrees of freedom as compared to the subsystem corresponding to the remaining phase, mainly because there had to be one of each shift type in this phase. In addition, creating a feasible assignment for the training phase of a particular employee did not add a significant amount of constraint to the satisfaction of the remaining assignment (some minor constraints if the final training shift for a particular employee was a night shift, which would prevent a night shift from being applied to that employee on their first non-training day). This partition increased our performance by an order of magnitude.

2.5 Business Discussion

Recall that our ultimate output is simply one feasible solution from the space of feasible assignments for the given scheduling problem. Since we didn't aim to maximize or minimize some objective function that could be defined with some element of practicality i.e limiting the number of part time (20 hours/week) and full time (40 hours/week) workers, we experimented with imposing additional constraints in order to improve the practicality of our solution (as defined by our own metrics). For example, we tried to minimize the total work hours of a given scheduling solution, which would theoretically minimize the cost of business operations, but ran into some time out errors. Next, we tried to encourage hiring of part time workers over full time workers, which would theoretically decrease business expenses related to providing benefits to full time employees, by starting value initialization of our shift `IloIntVar` at it's lowest domain value. This worsened our overall performance, thus we removed this from our final implementation. We also tried imposing the opposite: encouraging the hiring of full time workers over part time workers. Although this would probably increase business expenses related to providing benefits to full time employees, our rationale was that having a larger critical mass of full time employees would contribute to improving business culture.

3 Discussion

We implemented randomized restarts triggered after hitting a certain failure threshold, which we empirically determined to be 100,000 failures. Even after repeated tuning of our failure parameter, adding in randomized restarts did not decrease the number of given instances that timed our prior to finding a feasible solution. We were happy to see that there was an increase in performance between implementing constraints locally and globally. Intuitively speaking, there's a greater fidelity to propagate global constraints to a system as compared to aggregating local constraints. Thus, although local and global constraints may impose the same truth, we experienced some deviation in performance potentially due to the summation of the deviations in the way the assignment tree is pruned prior to starting DFS and the constraint-dependent heuristic calculation (impact) at each decision point. We are excited to talk about this during the post mortem. We spent 35 hours on this project.