As a team of two people, we spent approximately 25 person-hours. This was divided into about 6 hours for constraint programming, 12 hours for search, 4 hours for metrics, and 3 hours for the writing the report.

Constraint Model Description

Decision Variables

We split our decision variables into shift assignments x and hours h:

- **Shift Assignments x** - This was a 2d array with dimensions of `n_employees` by `n_days`. Each employee would be assigned either a shift (1,2,3) or an off shift (0) for each day.
- **Hours h** - This was a 2d array with the same dimensions as shift assignments. If an employee worked a shift on the given day, this array would contain how many hours they worked. If they had an off shift, this array must contain 0.

Constraint Implementations

We implemented constraints pretty directly, and have listed out each below:

- **Consistency for h,x**: A shift assignment is off-shift iff hours worked that day is 0
- **Min Employees per Shift**: Implemented using the count constraint for each shift
- **Hour Bounds (per day / per week)**: Implemented by summing across employee shifts
- **Training Requirement**: Used `alldiff` constraint on each employee for the first 4 days
- **Night Shift Limits**: Sliding window test for consecutive night shifts, and count constraint on total night shifts
- **Symmetry Constraint**: We made sure the employees were ordered using lexigraphic ordering on their assignments to avoid checking equivalent branches of the search tree.

We tried to figure out a way to include the distribute constraint for assigning employees to shifts. However, no matter how we formulated the problem, we couldn't figure out how to set a specific cardinality for number of workers on each shift since it was a $\geq$ constraint on count and not $=$.

Search Heuristics

We started by realizing that each week was essentially separate (besides the relatively minor consecutive night shifts constraint). As such, we decided to split our search into phases by week, so that our model could focus on one section of the problem at a time. Within each week, we first did search on hours decision variables, and then on shifts decision variables (this ordering worked better in our tests).
We also tested out different heuristics for selecting variables within hours and shifts. We ended up going with random variable selection for both. For value selection, we just chose to maximize value directly. For hours, this meant we preferred working more hours (to satisfy the minimum daily

demand constraint). For shifts, this meant we preferred day shifts over night shifts over off shifts, which also makes sense.
At the end of everything, we realized that certain instances acutally became unsolvable after adding search heuristics. As a result, we ended up using a portfolio of two solvers, one with search and one without search.

Analysis

Proposed Constraint Improvements

- **Sleep Schedule** - Employees should work consecutive night/day shifts to reduce "jet lag."
- **Shift Preferences** - Employees may prefer specific shifts; we can enforce a minimum satisfaction level.
- **Consecutive Off Shifts** - Some employees prefer blocks of off days for vacations.
- **Cost Efficiency** - Maintain a threshold on worker-to-demand ratios to minimize costs while ensuring coverage.

Metrics and Quality Assessment

We use key metrics to evaluate schedule quality:

- **Fairness: Total Hours Worked (Per Employee)** - Ensures balanced workloads. Example: `[80, 80, 80, 79, 80, 79]`.
- **Fairness: Avg Hours Worked (Per Shift) (STD)** - Standard deviation of shift hours, expected around 1. Example: `[0.0, 0.87, 0.72, 0.49]`.
- **Fairness: Total Night Shifts Worked (Per Employee)** - Ensures even night shift distribution. Example: `[2, 2, 2, 2, 2]`.
- **Fairness: Total Off Shifts (Per Employee)** - Ensures fair off-shift distribution. Example: `[3, 2, 3, 2, 3]`.
- **Efficiency: Shift Distribution (Per Employee)** - Ensures a reasonable mix of shift types. Example: `[0.22, 0.14, 0.30, 0.32]`.
- **Efficiency: Avg Hours Worked (Per Shift)** - Measures shift utilization. Example: `[0.0, 6.91, 7.50, 7.41]`.
- **Efficiency: Avg Extra Employees on Shift** - Tracks overstaffing. Example: `['N/A', 0.64, 0.07, 2.71]`.

Over all of the solutions that we solve our algorithm seems to score relatively high on fairness across employees and alright on efficiency over shifts (The algorithm greatly prefers shift 3 due to the effects of our valueselector in our search heuristics). More specifically, employees tend to work roughly the same amount of night shifts and off shifts and approximately (with a std of 1 hour) the same amount of hours per shift. They also tend to work about 7 - 8 hours per shift indicating high efficiency per shift (but oftentimes shift 3 would be overfilled i.e. has the most slack).