Alex Ding , Anirudh Narsipur

Project 2 Report

CSCI2951-O – Spring 2022
March 11, 2022

This project was relatively straightforward compared to the previous one. Since we were using a preexisiting solver creating the actual model was not that complicated. We relied heavily on CPO documentation to guide us. Between the two of us we spent around 15-18 hours on this project over two weeks.

Overview Of Model

Given n employees and a time frame of d days our decision variables are:

1. shifts_{nxd}
   A n x d matrix representing the shift each employee n_i works on days d_j for $i \in [1, n], j \in [1, d]$

2. hours_{nxd}
   A n x d matrix representing the number of hours each employee n_i works on days d_j for $i \in [1, n], j \in [1, d]$

Given these decision variables we then wrote constraints on the basis of the imposed requirements,namely:

1. Daily Structure : Our model assigns an integer to each shift and then constrains over those integers. So all values in shift are in $[0, 1, 2, 3]$.An employee working a consecutive shift is implicity taken care of as each employee can only work once a day that is there is only a single value hours$[i][j]$ for employee i on day j.We further constrain that off shifts imply 0 hours worked i.e shift$[i][j] = 0$ <=> hours$[i][j] = 0$

2. Business Rules : We impose business constraints by for each day in the schedule enforcing that the sum of the hours worked is greater than or equal to the minimum demand. Similarly we impose a constraint that the count of employees working a shift is greater than or equal to the minimum demand.

3. Training Requirements : Training requirements are imposed by simply requiring each employee to work a different shift for the first four days using an all different constraint.

4. Contractual Rules : Imposing contractual rules was the trickiest and we found that these affected perfomance greatly. We impose contractual rules by the constraints : shifts$[i][j]$ = Night $\implies$ shifts$[i][j+1]!$ = Night and by limiting the total count of night shifts for an employee to the maximum.

We initially used a 2D list for our domain variables but switched to numpy arrays as they were more convenient for slicing operations.

Depth First Search

Changing the search strategy to depth first dramatically slowed down the solver. The model was only able to solve 3 instances within the time limit. We tried various strategies for picking variables / values smartly by exploiting the structure of the problem. Namely we tried:

1. Exploting the symmetry between employees
2. The temporal ordering of the days
3. Grouping weeks together in search phases
4. Searching for the training schedule first

We tried the above search phase strategies along with available heuristics namely: impact based search , success rate and domain size. While there was some limited success with these strategies the solver was still too slow. The expection was searching for the training schedule first which led to a significant improvement.

What ended up working was randomly picking variables and values and limiting the failure count. We start with a small allowed failure limit which exponentially increases if the solver is unable to find a solution. This ended up solving all the instances well within the time limit. We also conducted a grid search for hyperparameters (gridsearch.py) and found that while other hyperparameter combinations did perform well they did not beat randomness by any significant measure.

Analysis

1. One significant way to improve the model would be to put in additional constraints such as employee schedule preferences.
2. Given the current information about business needs we have at our disposal the best way to measure the quality of our schedules is to measure the overall wastage in hours worked Daily and number of employees working each shift. If we have larger number of employees than required than the business is taking on unneccesary expenses.

   Using this metric we analysed the generated schedules , namely on average how much larger is the number of hours schedules daily and the number of employees per shift than the minimum requirement. The results are summarized below (the analysis can be found in analysis.py):

Schedule	Shift Wastage	Hours Wastage
28_40.sched	338.3433	108.65072
21_50.sched	548.3068	115.2382
14_17.sched	79.33390	7.047619
7_17.sched	112.02380	33.33333
28_27.sched	172.8032	25.039688
14_15.sched	51.05158	20.476190
28_65.sched	918.3730	0.9873949
14_14.sched	86.66666	5.238095
7_14.sched	110.95238	4.285714
21_30.sched	290.9523	120.15878
28_50.sched	592.0634	0.9875541
14_18.sched	175.7142	0.565770
21_40.sched	292.8837	140.21163
28_30.sched	218.1746	4.761904
7_20.sched	223.650793	0.0

The above numbers are percentages that is average excess in the schedule. That is for 28_40.sched the average shift is at 338% more than the minimum requirement and daily number of hours worked is 108% greater than the minimum requirement.

3. The above numbers suggest that our current schedules are incredibly wasteful. Ways to improve this are:

   1. Change model constraints from >= minDemand to = minDemand. This resulted in many of the instances becoming infeasible or timing out.
   2. Running the model multiple times to generate a more optimal solution or have the model keep searching for a better solution. We did not explore this as this is not very practical.
   3. Changing our business constraints in terms of minimum number of hours an employee has to work and their minimum hour requirements weekly. This could also take the form of reducing the number of part time employees and increasing the number of full time employees.The generated solutions have many employees working only 20-30 hours a week. Having fewer employees who work more hours would be more efficient (and also better for employees).