1.5em 0pt

Employee Scheduling

Authors

Name(s)	cslogin(s)	screenname(s)
Naomi Lee	nlee16	beta_data
Aaron Wang	awang167	e

Constraint Model

Our solution strategy followed the exact directions, implementing the constraints outlined in the handout as well as in class.

Our thinking for the project is as follows:

Variables: Let there be n employees and d days. Then our variables will be x_ij, which denotes the (shift, num_hours_worked) for employee i on day j.

Note that these are in two separate matrices.

Domain: [0, 1, 2, 3]

Constraints:

1. Any given day, an employee can only be assigned to a single shift Baked into our domain already.

```
For day in days:
  For shift in shifts:
    Num_in_shift = schedule[:][day].count(shift)
    cp.add(num_in_shift >= minDemandDayShift[day][shift])
```

2. minDemandDayShift[0][2]=4 means that there need to be at least 4 employees working for the day shift on the first day.

3. there is a minimum demand that needs to be met to ensure the daily operation (minDailyOperation) for every day when considering all employees and shifts.

4. the first 4 days of the schedule are treated especially where employees are assigned to unique shifts. (if exception)

5. employees cannot work more than 8 hours per day (maxDailyWork=8). Baked into our duration schedule array

6. the contract requires employees to work at least 4 consecutive hours once they start a work shift (minConsecutiveWork=4).

7. The total number of hours an employee works cannot exceed the standard 40-hours per week and it should not be less than 20-hours.

8. night shifts cannot follow each other (maxConsecutiveNightShift=1) and also, there is a limit on the total number of night shifts that an employee can perform (maxTotalNightShift=2). (check on last shift)

9. limit on the total number of night shifts that an employee can perform (maxTotalNightShift) across the scheduling horizon.

Analysis & Improvements

As we created our constraints, we realized we were creating the same for loops over the number of employees, days, shifts, etc., and these constraints could be combined to make the algorithm more efficient.

However, this was not enough to make it run under 300 seconds. Therefore, we attempted a few methods to improve our generated schedules. A few techniques for optimizing

Time Spent

We probably spent around the 50 hours on the project (i.e. way too much time). Our greatest time sink was deciding to switch from Python to Java a bit too late in the game and getting the software up and running. We did a great job pseudocoding and laying out our task with the constraints; however, we ended up switching to Java only four days before the assignment was due and had difficulties running CPLEX locally until we decided to run on department machines.

After we were able to get our program up and running, our work picked up much speed. Our original constraint model ran nearly smoothly, but it was difficult to determine whether we implemented the constraints sufficiently or whether our algorithm was outputting the correct solution. Furthermore, finding and implementing the correct syntax for CPLEX required documentation reading and patience with experimentation. Despite the struggle, this was a very fun project with distinctly many interesting applications!