Project – II
Employee Scheduling

CSCI 2951-O: Foundations of Prescriptive Analytics

Initial Due Date: Mar 4, **before class**
Final Due Date: March 11

1 Problem Statement

The manufacturers have now received customized product orders from the Electronic Car Consortium (ECC). The natural next step is to turn these specifications into actual ready-to-be-shipped products. Given the tight production deadline, the car sequencing lines must be up and running for 24 hours a day, 7 days a week. However, in the existence of complicated business rules and contracts, creating employee schedules that can carry out the production is not an easy task to accomplish.

Building good schedules that conform with labor rules is challenging. It takes several hours, if not days, for human experts to generate acceptable solutions. In order to reduce the time and effort, an automated approach has become imperative, and once again, CSCI 2951-O elites are called for action!

2 Employee Schedules

The schedule of the employees are subject to various business rules ranging from the daily patterns to meeting the demands as detailed below.

2.1 Daily Structure

The schedules are generated for a certain number of weeks/days (`numWeeks, numDays`) to be carried out by a number of employees (`numEmployees`).

There are 4 possible shifts (`numShifts`) in total. Three of these shifts are work shifts, namely; night, day, evening, and there is one off shift. The off shift is denoted by 0 while work shifts, night, day, and evening are denoted by 1, 2, and 3 respectively. The work day is divided into 24-hour intervals where the night shift covers [00:00..08:00), day shift covers [08:00..16:00) and the evening shift covers [16:00-24:00). The employees start and finish their continuous work at hourly intervals exactly.

For example, an employee can start working at 8AM, work for 8 consecutive hours and then finish at 4PM. This would correspond to working a regular day shift (i.e. 2nd shift). Notice that, if the employee finished at 2PM, it would still count as a day shift.

2.2 Business Rules over Shifts

First of all, at any given day, an employee can only be assigned to a single shift. That is, if one employee works the day shift, he/she cannot continue for the evening or any other shift. Obviously, if an employee is assigned to an off shift, he/she is not working other shifts.

Given the assignment of employees to shifts, there is a certain minimum demand that needs to be met on the number of employees needed every day for every shift. This is denoted by:

`minDemandDayShift[day][shift]`

For example, `minDemandDayShift[0][2]=4` means that there needs to be at least 4 employees working for the day shift on the first day.

In terms of the total number of actual hours of work carried out, 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.

2.3 Training Requirement

In order to get employees up to speed with the manufacturing process, the first 4 days of the schedule is treated specially where employees are assigned to unique shifts. This way, every employee can experience working for each shift in first 4 days including an off-shift as well, where they can complete mandatory on-boarding trainings.

For example if an employee works 8AM..10AM on Day 0, which would be a day shift, then that employee cannot work any hours corresponding to a day shift on Day 1, 2, and 3. However, working 5PM..8PM would be suitable for Day 1 and so on.

2.4 Contractual Rules

The labor rules dictate that employees cannot work more than 8 hours per day (`maxDailyWork=8`). On the other hand, the business contract requires employees to work at least 4 consecutive hours once they start a work shift (`minConsecutiveWork=4`).

As full-time employees, 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 (`minWeeklyWork, maxWeeklyWork`).

It is known that night shifts are stressful, therefore 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 (`maxTotalNigthShift`) across the scheduling horizon.

Notice that while these limitations are described here as constants (e.g., maximum 8 hours per day) based on the modern-day work standards, the input instances treat them as parameters. It is a good practice to build your model parameterized with those rather than hard-coding them. The exception to that rule is, when you can take advantage of those constants to exploit the structure they impose on the problem to build stronger models for specialized version of the problem at hand.

3 Your Task

Your main tasks is to employ IBM Optimization Studio to create a constraint model in CP Optimizer to generate employee schedules adhering to *all* the business rules stated above. In your report describe the decision variables of your model and list the constraints you utilized.

A follow up task is to analyze the schedule found and to make suggestions on how to improve the overall process. This is an open-ended task where you can consider different perspectives. For example, from the business side, one might try to increase the total production, and from the employer side one might try to improve work conditions.

In the project report, include details about: how do you assess the "*quality*" of generated schedules, what metrics did you use, ways to improve the schedule (e.g., add/drop new business rules/constraints, change conditions on the work patterns etc.). Feel free to modify the instances and share new schedules if you find any. Compare and contrast them with the schedule found for the original instance.

4 Input

The input files specify the parameters used for scheduling such as number of days, number of employees, minimum demand for shifts, maximum work duration and, so on. As mentioned in the Analysis part, you are free to modify these instances to generate different schedules.

5 Support Code

We provide you with simple CP instance object that parses the data files (`CPInstance.java`) so that you can immediately start modeling the problem. The compile and run scripts take care of exporting a license and linking to the constraint solver.

The CP instance includes the model to solve the Aussie graph coloring problem and Send More Money which were studied in class. The examples show how to invoke the solver. There is also a pretty print function that can be used to display employee schedules.

Notice that there are some solver parameters, `numWorkers` and `searchType`, which are already set in the support code. Leave them as-is for now. In the second part of this project, we will switch the `searchType` to depth-first search and assess its impact on our solutions.

The support code is written in Java. You are free to use the support code as is, tweak it as you see fit, or discard it completely. If your choice of programming language is different, the support code can serve as a reference. CP Optimizer has C++ and C# bindings as well. However at this point we cannot provide support on how to run the constraint solver in those languages.

Assuming you are in the `project` directory,

To compile the support code, type:
```
> ./compile.sh
```

To run the support code in a given instance, type:
```
> ./run.sh simple.sched
```

6 Output

The solution of employee scheduling is defined by the start and end time of each employee for every day in the time horizon. In case employees are assigned to off-shifts, -1 is used for start and end times.

Given an input file as its first command line argument via the `run` script, your solution should print out the result on its **last line**. You can print other statements before the result but they will be discarded. We suggest you to keep such debug information to a minimum since writing I/O is expensive. We use `stdout` for output and ignore other streams. Your submission will be tested on department's linux machines and it should work without any other software package installation. CP Optimizer is already installed on department machines.

The expected output is a JSON object with "Instance" mapped to the instance name, "Time" to the number of seconds it takes your program to solve this instance (in 2-digit precision), "Result" to the number of failures [1], and "Solution" to the start and end time of each employee for every day separated by a single whitespace.

```
./run.sh simple.sched
..running..bla..
..bla..
{"Instance": "simple.sched", "Time": 1.23, "Result": 123,
 "Solution": "8 16 8 16 8 12 -1 -1"}
```

In the above example, there are 2 employees scheduled over 2 days. The first employee works day shifts twice each from 8AM to 4PM, and the second employee works one half-day shift from 8AM to 12PM and is off the second day.

More precisely, given E employees and D days, for employee $e \in \{0..E-1\}$ and day $d \in \{0..D-1\}$, the "Solution:" prefix is followed by a list of numbers denoting the begin and end times of employees for every day. It is printed on a single line separated by a whitespace as follows:

[1] returned by the solver cp.getInfo(IloCP.IntInfo.NumberOfFails)

$$Begin[e_0][d_0] \quad End[e_0][d_0] \quad Begin[e_0][d_1] \quad End[e_0][d_1] \quad \ldots \quad Begin[e_0][d_{D-1}] \quad End[e_0][d_D - 1]$$
$$Begin[e_1][d_0] \quad End[e_1][d_0] \quad Begin[e_1][d_1] \quad End[e_1][d_1] \quad \ldots \quad Begin[e_1][d_{D-1}] \quad End[e_1][d_{D-1}]$$
$$\ldots$$
$$Begin[e_{E-1}][d_0] \; End[e_{E-1}][d_0] \; Begin[e_{E-1}][d_1] \; End[e_{E-1}][d_1] \; \ldots \; Begin[e_{E-1}][d_{D-1}] \; End[e_{E-1}][d_{D-1}]$$

Notice that, the solution presented in the output is for the *original* problem instances, not for the modified schedules you might come up with upon your analysis. You can share those new schedules for modified instances in your report and point out their differences.

Finally, to run a benchmark on all instances in a given input folder using a fixed time limit while collecting the result (i.e. the last line) to a log file, type:

```
> ./runAll.sh input/ 300 results.log
```

Please make sure to follow this convention *exactly*! Failing to do so will cause evaluating your submission incorrectly.

7 Submission

```
/solution
|_ src/
|_ compile.sh
|_ run.sh
|_ runAll.sh
|_ results.log
|_ report.pdf
|_ team.txt
```

- Submit all your source code under the (`/src`) folder. Remember that commenting your code is a good practice so that one can follow the flow of your program at a high-level.
- Submit your (possible updated) `compile` and `run` scripts which compiles your solution and runs it on a given instance respectively. Make sure that the *last line* of your solution output follows the specification above.
- Submit the last line found for all the *original* instances in the input directory in the `results.log`.
- Submit a short report in pdf format (1 or 2 pages max) that contains a brief discussion of your constraint model, decision variables and the constraints used. Include also your analysis as discussed in Section 3. Don't forget to write name(s) of the team members and the screenname(s) corresponding to this project. Please also include a note on how much time you spend on this project. This information is completely irrelevant to evaluation and kept solely for statistical purposes.
- Submit the cs login(s) of the team in `team.txt` listing one member per line in lowercase.

Have questions? and/or need clarifications? Please do not hesitate to contact us, your instructor and TAs are here to help. Good luck on your quest for helping managers and employees!