Caixa Mágica

software

Summer Internships 2017

Schedule Optimiser

Final report

Trainee: Daniel Ramos

Mentor: Afonso Ribeiro

Co-Mentor: Vítor Martins

Resume:

The purpose of this internship was to develop an optimiser for workplace scheduling - to get the best possible schedule for a given criteria.
This report contains both an analysis of the state-of-art and a solution to the problem.

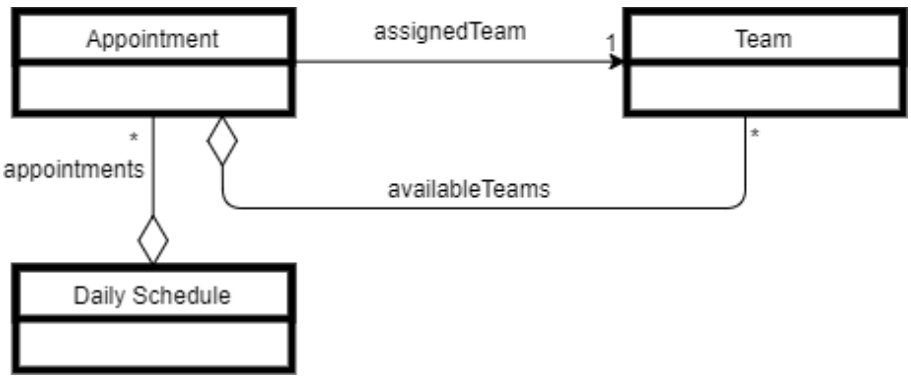
State-of-art:

There are 2 common approaches for workspace scheduling:

- Constraint programming: An approach that addresses the problem as generalised assignment problem. This means it is NP-Hard for any non-trivial objective function (can't be solved in a timely manner).
- Local Search: Meta-heuristic approach to get approximations of the optimal solution. Fast but not ideal, works very well for big problems.

Modelling the problem:

The problem can be easily modelled as a class diagram. We define an appointment as a general unit of work that must be performed by a team. A schedule is a collection of appointments for a given day.



The field availableTeams is of interest for the algorithm's execution. Each appointment can be performed by a set of teams, we define these as availableTeams.

Solution (Local Search - Algorithm**):**

```
Minimize(DailySchedule schedule, CostFunction fn, TemperatureSchedule temp) {
        double temperature = temp.getTemperature(0);
        for (int i = 0; i < MAX_TRIES; i++) {

                Appointment app = schedule.randomAppointment();

                Time oldTime = app.getStartTime();
                Time newTime = app.getRandomStartTime();
                app.changeStartTime(newTime);

                Team oldTeam = appointment.assignedTeam();
                Team newTeam = appointment.randomAvailableTeam();

                int oldCost = fn.cost(schedule);
                app.replaceAssignedTeam(newTeam);
                int newCost = fn.cost(schedule);

                if (newCost > oldCost)
                        with probability e^(oldCost-newCost)/temperature {
                                app.replaceAssignedTeam(oldTeam);
                                app.changeStartTime(oldTime);
                        }

                temperature = temp.getTemperature(i);
        }
}
```

Solution (Explanation**)**

• Cost / Objective Function:

The algorithm works by changing the team assigned to each appointment to minimise the solution cost, which is obtained through an objective function:

$$f : DailySchedule \rightarrow Cost$$

Depending on the selected objective function solutions can widely vary since the algorithm always searches for solutions that minimise the function.
The main idea is the algorithm's preferences over possible schedules can be captured by a function that maps these schedules to a real number (quality of the schedule), providing the algorithm a mathematical way to compare different types of schedules (as the function approaches 0 the better the solution).
This gives us the chance to easily change the kind of solution we want, for instance, we can choose to give solutions with a high number of teams a higher value, hence forcing the algorithm to find solutions with least number of teams.

• Temperature:

Searching for the best schedule is like searching for the top of the Alps. You can't always be climbing, you might have to go through a series of mountains, up and down, until you reach the mountain with highest altitude.
The temperature variable allows the algorithm to take inherent bad decisions (going downhill) in hope hope of a finding a 'higher mountain' that is, a better solution.
The object TemperatureSchedule decides the temperature, every iteration it returns a lower temperature to be used. It can be implemented as a linear, exponential, logarithmic, (…) functions.

• Multi-threading:

Running multiple instances of the algorithm gives us the chance to explore different paths, this is used to overcome one of the temperature approach limitations - temperature only allows for bad decisions in early stages of execution. This is a

necessary measure to guarantee that the user gets the best solution he cans, as it would not make sense for the last iterations of the algorithm to degrade the solution's quality. Making use of the last analogy about the Alps, essentially, running multiple instances of the algorithm is as deploying multiple robots to search for the highest mountain, whereas their decisions are nondeterministic. The one that finds the highest altitude wins.