Caixa Mágica
software

Summer Internships 2017

Schedule Optimiser

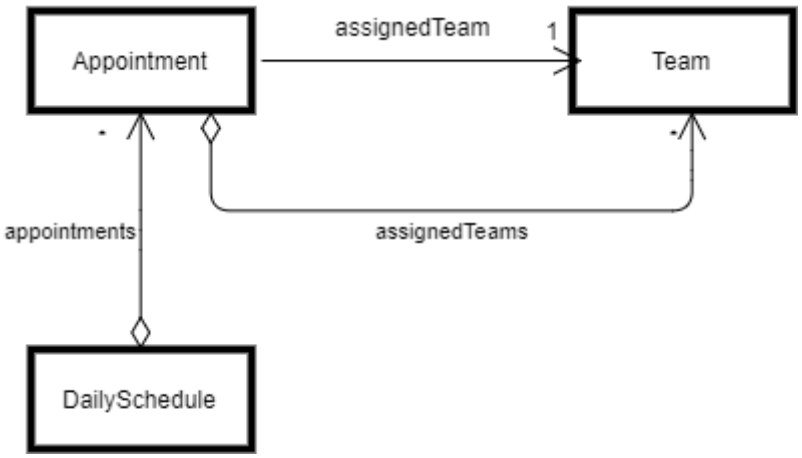
Weekly Report - 31/07/2017-04/08/2017 (Week 4)

Trainee: Daniel Ramos

Mentor: Afonso Ribeiro

Co-Mentor: Vítor Martins

Summary:



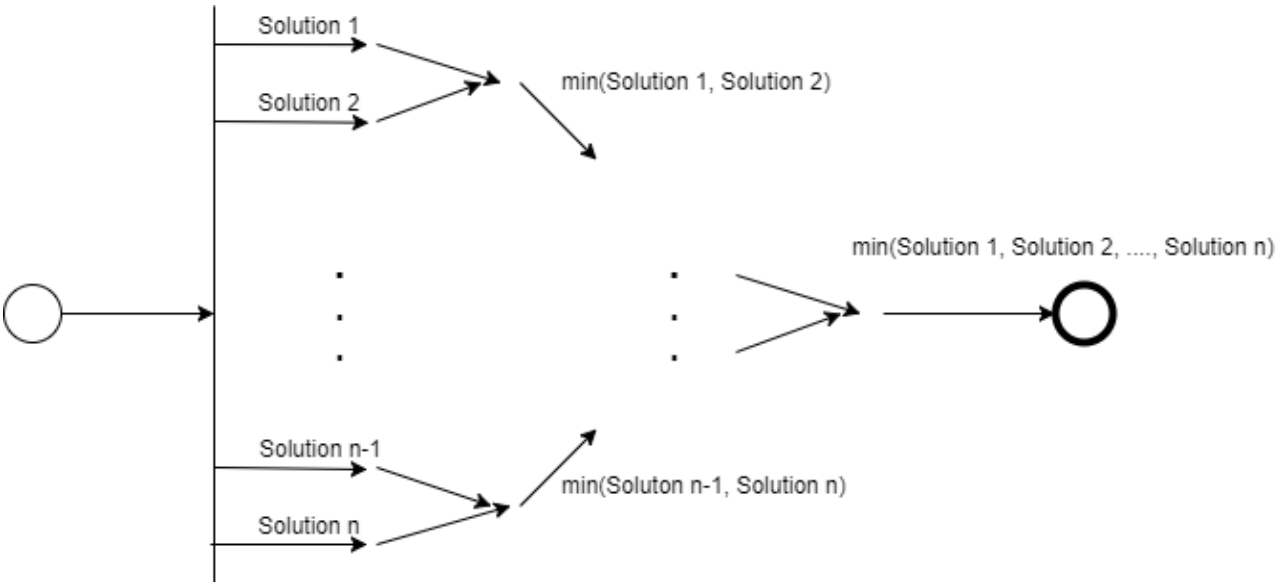

```
Minimize(DailySchedule schedule, CostFunction fn, TemperatureSchedule temp) {
        double temperature = temp.getTemperature(0);
        for (int i = 0; i < MAX_TRIES; i++)
         {
                Appointment app = schedule.randomAppointment();
                Team oldTeam = appointment.assignedTeam();
                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);


                temperature = temp.getTemperature(i);

        }
```

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

$$f : DailySchedule \rightarrow Cost$$

Depending on the selected cost 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.

The algorithm's execution time can range from 10 seconds to 5 minutes, depending on the number of available teams for each appointment. The reason for such wide interval is because it's necessary to make requests to the google map's API to obtain travel times to different locations, the more teams the more combinations of trips are possible, hence more requests must be done.
A cache is used to store the most recent travel times, we can take advantage of that by running multiple instances of the algorithm in parallel (without losing performance), selecting at the end the best solution found. An example of execution with 100 execution flows is as follows:



Completed Tasks:

- Graphical interface to visualise results.
- Cost Function Paradigm implemented, it is now easy to change the way the algorithm behaves.
- Find the best optimising parameters.

To-do:

- Change the algorithm to support appointments with an uncertain start time (start times within specified intervals).