COMP-4691/8691: Assignment 4
Meta-heuristics

1 Goal of the Assignment

In this assignment, you will implement procedures to solve and optimise a *Firefighter Scheduling Problem* (cf. Section 2). To this end, you will have to implement some meta-heuristics.

You are asked to document every step and every decision you made throughout the completion of the assignment. Explain your decisions (such as your definitions of neighbours) in the file `decisions.txt`. Meta-heuristics is a domain where you can be extremely creative (as opposed to other parts of computer science where there is often one obvious solution). As a consequence, it is very important that you explain your work. Documentation and comments will be an important part of your final mark; good or efficient implementations with bad comments will result in low grades.

2 Firefighter Scheduling Problem

The *Firefighter Scheduling Problem* falls under the umbrella of the *Employee Scheduling Problem* in Operational Research. It is the transposition of a classic job-shop scheduling problem into the context of a local fire station made up of part-time firefighters. Like many combinatorial problems, it is often too hard to compute optimally. The problem is defined as follows.

You are supposed to schedule the work shifts of a group of firefighters at a fire station. There are a number of constraints that need to be satisfied—for instance, there should be enough firefighters during each shift to respond to emergencies as required. Additionally, each firefighter has their own preferences—some might prefer working during nighttime, while others might prefer to work during the morning.

The schedule covers a period of 21 days (3 weeks) numbered 0 to 20. It repeats at the end of this period. This implies that the constraints that apply over consecutive days also apply between Day 20 and Day 0. In other words, a sequence of k consecutive days is any list $[d \bmod 21, (d+1) \bmod 21, \ldots, (d+k-1) \bmod 21]$.

We now describe the 8 hard constraints that a solution needs to satisfy to be valid. If you find that the definition of a constraint is ambiguous, you can look up the method `is_feasible` from file `firefighter.py` (how the solutions are represented is explained in the next section).

C1 For each day d, each firefighter i should be allocated to one of the following shifts: morning (M), afternoon (A), night (N), or off-duty (F).

$$v_{i,d} \in \{\text{M}, \text{A}, \text{N}, \text{F}\}$$

C2 Each firefighter works 14 days out of 21 (7 off-duty days).

C3 For each firefighter, the number of consecutive days with the same work shift (M, A, N) is between 2 and 4 (inclusive). For instance, if a firefighter is working in the morning on Day 10 and in the night on Day 11, they should work in the night on Day 12 (minimum of 2 consecutive days) and they can work in the night on Days 13 and 14 but not on Day 15 (maximum of 4 consecutive days).

C4 The consecutive number of work shifts (M, A, N) for a firefighter is between 3 and 6. For instance, if the firefighter is off-duty on Day 1 and working on Day 2, they must work on Days 3 and 4, and may work during Days 5 to 7 (inclusive) but not Day 8.

C5 The consecutive number of off-duty shifts for a firefighter is between 1 and 3. In other words, a firefighter cannot have 4 off-duty days consecutively.

C6 Throughout the 21 days period, each firefighter gets at least one off-duty shift over an entire weekend (at least one of Days 5 and 6; 12 and 13; or 19 and 20).

C7 There is a minimum number of firefighters that should attend certain shifts: 3 firefighters every morning shift; 4 every afternoon shift, 2 every night shift. There is no maximum number of firefighters in any shift.

C8 The shifts must follow this order: M $\rightsquigarrow$ A $\rightsquigarrow$ N $\rightsquigarrow$ M. In other words, if a firefighter is working in the morning, they should work in the afternoon next. Off-duty shifts can take place at any time between consecutive shifts (so you can have: M followed by F followed by A, but neither M followed by F followed by M, nor M followed by F followed by N).

Each firefighter has a cost associated with each shift of each day of the week. For instance, Firefighter i may associate each Monday mornings (shift M of Days 0, 7, and 14) with a given cost $c_{i,0,\mathrm{M}}$. The cost of a solution is the sum of all shifts that the firefighters perform during the week (if a given firefighter performs the same shift during the same day n times during the week, the corresponding cost counts n times).

The goal is to find a solution with minimal cost.

3 The Code

We provide the following code to help with the implementation.

File `firefighter.py` contains a description of the `SchedulingProblem`. This file contains a number of constants for the problem (both at the beginning of the file and in the initialisation of `SchedulingProblem`). It assumes that the schedule is a list of strings, one for each firefighter and each character of the string represents one day. For instance, for a problem with 2 firefighters and 4 days, a schedule could be

```
[ "MMAA" , "FFMA" ]
```

which indicates that Firefighter 0 is taking a morning shift on Day 0, a morning shift on Day 1, etc.

The cost function is a table `costs` such that the cost for a Firefighter `i` of working in shift `s` on day `d` is `costs[i][d%7][s]` if `s` is not F (otherwise, the cost is zero). Notice that the preference is defined weekly, which is why we use the "modulo 7" operator.

The class `SchedulingProblem` offers the methods `is_feasible`, which verifies if the specified schedule is feasible and `cost`, which computes the cost of a specified schedule given the specified cost function.

This file also contains methods to save the schedule to a file (the name of the file is based on the current time, which will be useful for test purposes), to load it, and to read the cost function from a saved file.

`create_costs.py` is a main file that generates a random cost function based on a seed (here the two course codes). Your code will be tested against a different seed. Run this main and use the `read_costs` function from `firefighter.py` to access it.

`create_solution.py` will be used to generate the starting solution.

`neighbours.py` will be used to perform local search.

`model.py` contains an implementation of a `pulp` MILP model to compute solutions. You'll find that it is impossible to compute the optimal solution in reasonable time, but this will be used to perform Large Neighbourhood Search (LNS). Figure 1 shows how to use the methods in this file to compute a solution.

For this purpose, you will need to create a `ModelBuilder` object, use the method `build_model`, run `model.solve(PULP_CBC_CMD(msg=False))`, and call `extract_solution`.

The variables in the MILP model can be accessed by `mb._choices[i][d][s]` (where `mb` is the model builder) for `i` is the number of the firefighter, `d` is the number of the day, and `s` is the shift (represented by the char in `firefighter.py`). This variable evaluates to 1 iff the firefighter `i` performs Shift `s` on Day `d`. If you search for a solution where Firefighter `i` performs Shift `s` on Day `d`, you can simply add the constraint `mb._choices[i][d][s]==1` to the model.

```
prob = SchedulingProblem()
costs = read_costs(prob)
mb = ModelBuilder(prob)
model = mb.build_model(costs)
res = model.solve(PULP_CBC_CMD(msg=False))
if res != 1:
  print('No solution found')
sol = mb.extract_solution()
```

Figure 1: How to use the `pulp` solver and `ModelBuilder` to find the optimal solution.

4 Your Task

Your task will be to implement a number of algorithms. Your decisions need to be explained in file `decisions.txt` and your code needs to be commented so that it is easy to understand.

There is a solution of value 98.93 for the problem you are trying to solve (i.e., when using the course code ids as a seed for `create_costs.py`). Though, it may be possible to achieve more optimal solutions.

You are free to deviate from the implementation presented during the lectures, but make sure that you justify these changes. "It worked better on my test scenarios" is a reasonable justification.

4.1 Generate Feasible Solutions (10 pts)

Based on the description of the problem and its parameters, write the method in `create_solution.py` that computes a feasible schedule. Your method needs to use the seed so that different seeds lead (generally) to different solutions. A good implementation should be able to, for hundreds of different seeds, return hundreds of different solutions. Hint: the number of firefighters has been defined such that it should be easy to compute feasible solutions.

If you are unable to generate solutions, which is a challenging task which you may decide to come back to later, you can use the file `example.sched` for the next steps of your implementation.

4.2 Local Search and Variable Neighbourhood Search (40 pts)

1. Modify the file `neighbours.py` in order to introduce four new neighbourhoods. Explain clearly what each neighbourhood does. Run local search based on each neighbourhood, and test the strength of the neighbourhood. Remember that in the context of Variable Neighbourhood Search (VNS), each neighbourhood could be bad by itself (have many local optimals of low quality) but be strong when combined.

2. Use all the neighbourhoods you have access to (there should be five of them unless you came up with more or fewer than four neighbourhoods) and implement VNS so that it is called automatically when running `python vns.py`. Your implementation should use the methods `load_last_schedule` and `read_costs` to know where to start from. (No restart is expected in this implementation.) Also use `save_schedule` every time you find a better schedule so that we can evaluate the quality of your solution.

4.3 Large Neighbourhood Search (50 pts)

1. Find a way to represent schedules with some destroyed part.

2. Define three destroy methods.

3. Use the `pulp` model from `model.py` to repair a given schedule.

4. Implement Large Neighbourhood Search so that it is called automatically when running `python lns.py x` where `x` is a number between 1 and 3 that calls the corresponding destroy method. Once again, this should use the `load_last_schedule` and `read_costs` methods, as well as `save_schedule`.

4.4 Extra Marks

This part of the assignment is optional. It can give you extra marks which could compensate for any lost points from the previous questions. It will never make you get more than full mark.

If you find an error in the `pulp` model, send a report as a private post on EdStem explaining where the problem is. I believe the model is correct, but I may have made a mistake. You will also get extra points if you find ways to improve this model.