TÉCNICO LISBOA

Patient Scheduling in an Urgency Room

IASD2020/21 Assignment #1

(Version 1.3, November 6, 2020)

Introduction

Efficient patient scheduling in an urgency room may lead to a life or death situation. The situation is even more critical in pandemic times, like the ones we leave today, in which the numbers of patients may increase significantly. Depending on a given number of patients in the waiting room and doctors in service, the goal is to minimize the patient waiting time for a consult. A critical component is how we should handle the different seriousness levels that each patient has. As standard in every urgency room, patients are labeled depending on their needs (maximum waiting time and the duration of the consults), usually with color bracelets. This project aims at addressing this problem of finding the patient/medical doctor association schedule, from now called the PMDA problem.

1 Problem Statement

Consider the following specifications:

- Let $\mathcal{M}$ be a set of medical doctors in the urgency room. For several reasons that may include the need to be focusing on other services, for each $m \in \mathcal{M}$ we have an efficient rate m_e, which indicates the ration of time the medical doctor will be focused on the patient. For a time interval of 10 minutes, a doctor with $m_e = 0.5$ will only contribute to 5 min of the overall time for the consult of a patient.
- Consider a set of bracelet colors (patient labels) denoted as $\mathcal{L}$, where each $l \in \mathcal{L}$ has a tuple with two time intervals (l_{mw}, l_{ct}): 1) l_{mw} indicates the maximum time a patient with label l should wait; and 2) l_{ct} indicates the respective amount of time needed in consult.

- $\mathcal{P}$ represents the set of patients. Each patient has associated a tuple (p_{wt}, p_{label}): 1) p_{wt} indicates the time the patient was waiting before the scheduling; and 2) p_{label} indicates the label associated to the patient p.
- t is the time interval in which the evaluation and new assignments are done. For simplicity, we will consider a fixed time interval equal to 5 min. Notice that patients can be in a consult in a specific time interval and leave without accomplish the total time needed for a consult (this means that he/she must return).

As indicated in the introduction, the problem consists of assigning patients to medical doctors. Two main concerns must be considered: 1) the patients must not wait more than the time indicated by their labels; and 2) we must minimize the long waiting times for all the patients in the waiting room. For the second concern, we consider the following cost associated to all the patients in the waiting room: $C(\mathcal{P}) = \sum_{(p\ in\ \mathcal{P})} p_{cw}^2$ where p_{cw} is the patient waiting time.

2 Objective

This mini-project aims to solve the previous section's problem using uninformed/informed search methods. This includes defining:

- A state representation;
- The operators;
- The goal condition;
- The heuristics (if needed); and
- The search strategy,

allowing an appropriate search method to find the optimal solution. The implementation should be done in Python version 3.x. No extra modules, besides the Python Standard Library, are allowed. The search algorithm implementations are the ones from the GitHub repository of the course textbook, namely the module `search.py` available from `https://github.com/aimacode/aima-python`. The problem should be implemented as a Python class with the name PMDAProblem, which derives from the abstract class search problem, and that defines (at least) the following methods:

`actions(s)` Returns a list (or a generator) of operators applicable to state s;

`result(s, a)` Returns the state resulting from applying action a to state s;

`goal_test(s)` Returns True if state s is a goal state, and False otherwise;

`path_cost(c, s1, a, s2)` Returns the path cost of state s2, reached from state s1 by applying action a, knowing that the path cost of s1 is c;

`load(f)` Loads a problem from a (opened) file object f (see below for format specification);

`save(f)` Saves a solution to a (opened) file object f (see below for format specification); and

`search()` Computes the solution to the problem. It should return `True` or `False`, indicating whether it was possible or not to find a solution.

If the students want to use informed search methods, an heuristic must be defined:

`heuristic(n)` returns the heuristic of node n.

The choice of state and action representations and the searching algorithm is entirely up to the students choice, with the following restrictions:

- The use of implemented algorithms in the `search.py` module is mandatory;
- Only algorithms taught in the theoretical classes are allowed; and
- No changes in the algorithms are allowed.

3 Input and Output formats

3.1 Input file

This file format specifies a PMDA problem. It is a text file, organized by lines. Empty lines should be ignored. Non-empty lines should have one of these forms:

`MD <code> <efficiency>`
to specify a medical doctor with code `<code>` and efficiency `<efficiency>`.

`PL <code> <max_waiting_time> <consult_time>`
to specify the labels. For the label `<code>`, patients must be attended in `<max_waiting_time>` and the consult will take `<consult_time>`.

`P <code> <current_waiting_time> <label>`
to specify the patients in the waiting room. The `<code>` specifies the patient's code, `<current_waiting_time>` the respective starting waiting time, and `<label>` the associated patient's label.

In the problem definition file, we will get all the lines specifying the available medical doctors before the patient labels; and all the patient labels set before the list of presenting the list of patients in the waiting room.

3.2 Output file

The solution to the PMDA problem should be returned using the following file format. Again, it is a text file, where blank lines are ignored, and each non-blank line should have one of these two formats:

`MD <code> <patient1_code> <patient2_code> <patient3_code> ...`
to specify the assignment of patients. The medical doctor with code `<code>` will first see patient with code `<patient1_code>`, followed by `<patient2_code>`, and so on.

When a medical doctor has associated an empty slot (no patient needs attention), the code should contain the sentence `empty`. See the example below.

Notice that, if the input problem is infeasible, the `search()` method should return `False`. The evaluation will not consider any output file.

4 Evaluation

The deliverable for this mini-project has two components:

- A single Python file, called solution.py, implementing the above mentioned PMDAProblem class, and
- A report in the form of a short questionnaire.

Both components are submitted to a Moodle platform. Instructions for this platform are available at the course webpage. The grade is computed in the following way:

- 30% from the public tests;
- 30% from the private tests;
- 30% from the questionnaire; and
- 10% from the code structure.

Deadline: **18-Nov-2019**. Projects submitted after the deadline will not be considered for evaluation.

5 Example files

For the PMDA Problem specified with:

```
MD 0001 1
MD 0002 0.5
```

```
PL 01 10 15
PL 02 20 10
PL 03 40 5

P 001 0 01
P 002 15 03
P 003 0 02
P 004 5 01
P 005 10 03
P 006 15 03
```

we get can get following results:

```
MD 0001 006 004 004 004 003 003 001 005
MD 0002 002 002 001 001 001 001 005 empty
```

which corresponds to a waiting time of 10 min for patient `001`, 15 min for `002`, 20 min for `003`, 10 min for `004`, 40 min for `005` , and 15 min for patient `006`, and a total cost of 2650. This cost corresponds to an optimal solution. (Notice there may be other results giving the same optimal cost).