TÉCNICO LISBOA

IASD 2023/24 Project Assignment #1: Autonomous shared vehicle transportation system

Rodrigo Ventura and Luis Custódio
Instituto Superior Técnico, University of Lisbon

(Version 1.0, September 11, 2023)

1 Introduction

Climate changes present a major challenge to our civilisation, thus the decarbonisation of our cities is a major goal to mitigate their effects. One way to approach decarbonisation is through the use of cleaner and more efficient transportation systems.

This project aims at contributing to this goal by exploring the idea of an intelligent transportation framework comprising a fleet of shared autonomous vehicles. The system receives as input a set of transportation requests from users, and schedules a fleet of autonomous vehicles to drive users from their pickup points to the desired destinations. Each vehicle has a pre-defined amount of seats. The goal is to maximize efficiency in passenger transportation. All of these concepts will be formalized below.

This year's IASD project consists in developing a program using Python to determine the best schedule for the vehicles, in particular, which and when each vehicle picks up and drops off users. The project will be divided in three parts/assignments, each one with a deliverable. This first assignment aims at reading a problem from an input file and determining the cost of a given schedule.

2 Problem Statement and Solution

The problem is formalised as follows. Let

- $\mathcal{P}$ be a set of geographical points, including all pickup/drop-off points and the vehicles initial point (i.e., a single location where all the vehicles are initially),
- $\mathcal{R}$ be a set of user requests, in the form of tuples (t, o, d, n), where $t \geq 0$ is the request time, $o \in \mathcal{P}$ and $d \in \mathcal{P}$ are the pickup (origin) and drop-off (destination) points ($o \neq d$), and n is the number of passengers to transport jointly,
- $\mathcal{V}$ be a set of autonomous vehicles, where s_v is the amount of seats in vehicle $v \in \mathcal{V}$, and all vehicles in $\mathcal{V}$ are initially placed at initial point $p_0 \in \mathcal{P}$, and
- for any given pair of points $p, q \in \mathcal{P}$, let real valued function $T : \mathcal{P} \times \mathcal{P} \to \mathbb{R}$ be the transportation time from p to q, negleting the pickup/drop-off time. These times are pairwise shortest, and thus they form a metric space, that is, for any $p, q, r \in \mathcal{P}$:

$$
\begin{aligned}
&T(p,p) = 0 \\
&T(p,q) > 0 \quad \text{if } p \neq q \\
&T(p,q) = T(q,p) \\
&T(p,q) \leq T(p,r) + T(r,q)
\end{aligned}
$$

The solution is a temporal plan comprising a set of actions, where each action is defined by a tuple (a, v, r, t) where $a \in \{\text{Pickup}, \text{Dropoff}\}$ is the action (the names are self-evident), $v \in \mathcal{V}$ is the vehicle, $r \in \mathcal{R}$ the request being served, and t is the time this action takes place. A solution to be valid must satisfy a set of consistency constraints and a few simplifications, namely:

- the plan starts at time zero;
- for each vehicle $v \in \mathcal{V}$, the action (a, v, r, t) with lowest time t in the plan must be a Pickup at time $t \geq T_{p_0,p}$ where p_0 is the initial position of all vehicles and p is the pickup point of the request r;
- for a request (t, o, d, n), the pickup time must be greater or equal than t;

- any two time consecutive actions of a vehicle $v \in \mathcal{V}$, corresponding to two points $p_1, p_2 \in \mathcal{P}$ at two time instants t_1 and t_2, take exactly the defined transportation time, that is $t_2 - t_1 = T_{p_1,p_2}$
- any request is satisfied exactly once with a unique Pickup and a unique Drop-off;
- the seating capacity of each vehicle cannot be exceded at any time.

Assuming that the main goal is user satisfaction, the solution cost will be based on the delay on arrival, with respect to the direct travel time. Thus, for each request $r = (t, o, d, n) \in \mathcal{R}$, the ideal time of arrival would be $t + T_{o,d}$, and thus the delay d_r of request r is defined by

$$d_r = t_d - t - T_{o,d}, \tag{1}$$

where t_d is the drop-off time of this request, corresponding to an action $(\text{Dropoff}, v, r, t_d)$. There are many ways to obtain a scalar cost from multiple ones, and in this project we will consider the sum of all delays

$$J = \sum_{r \in \mathcal{R}} d_r. \tag{2}$$

which, appart from a scale factor, is the average delay.

3 Objective

The goal of the whole project is to develop a program using Python (version 3) to search for the optimal solution to the above problem, with respect to the defined cost function (2). The project will be divided in three parts, each one with one deliverable. This document defines the first assignment and deliverable.

4 First Assignment

The first assignment is to develop a Python program capable of reading a problem, through an input file whose format will be defined in the next section, and determining the cost of a given solution according to (2).

5 Input file format

Points are indexed from 0 to $N_P - 1$, where p_0 corresponds to index 0 and N_P is the number of points in $\mathcal{P}$. Likewise, requests are indexed from 0 to $N_R - 1$, where N_R is the number of requests in $\mathcal{R}$, and vehicles are indexed from 0 to $N_V - 1$, where N_V is the number of vehicles in $\mathcal{V}$. The transportation times specified by the function T, defined in section 2, are put together into a matrix A, where each element a_{ij} is the transportation time between point indices[1] $i - 1$ and $j - 1$. Note that this matrix is symmetric and with zeros in the diagonal.

[1]Note that in mathematical notation indices start at 1, while in Python they start at 0.

The input file contains data to specify a problem, following the formalization introduced in section 2. It is a text file, composed of one or more lines, where each line contains one or more space separated fields. The data format is as follows:

- a line starting with a '#' is considered a comment and should be ignored;
- a line starting with a 'P' specifies the number of points N_P in its second field, followed by $N_P - 1$ lines containing the transportation times among points in $\mathcal{P}$:
  - the transportation times can be organized in a matrix, as described above; since the matrix is symmetric and with zeros in the diagonal, it suffices to specify the upper triangular above the diagonal;
  - the following lines contain the $N_P - 1$ rows of this matrix, where each line is a space separated sequence of number, which are the elements of the upper triangular part of the matrix;
  - therefore, these lines will have, sequentially, $N_P - 1, N_P - 2, \ldots, 1$ numbers each.
- a line starting with a 'R' specifies the number of requests N_R in its second field, and is followed by the N_R lines, one per request (t, o, d, n), each one with 4 space separated numbers, corresponding to:
  1. the time of the request (t),
  2. the pickup point index (o),
  3. the drop-off point index (d), and
  4. the amount of passengers (n);

This is an example of a problem input file, with 4 points, 5 requests, and 2 vehicles:

```
# this is a comment
P 4
20 30 40
   50 60
      70
R 5
10 1 2 1
15 1 3 1
16 2 3 1
20 2 1 1
25 1 2 2
V 2
4
5
```

6 Output plan representation

The solution to the scheduling problem is a temporal plan for the N_v vehicles. Each one of these vehicles has a timeline consisting of a sequence of actions, temporally ordered. The plan is a Python list, where each action is a tuple of 4 items:

1. a string, either '`Pickup`' or ´`Dropoff`', with self-evident meaning;
2. an integer corresponding to the vehicle index;
3. an integer corresponding to the request index; and
4. a float corresponding to the time of the action.

In the list, these actions may appear in any order.

Here is an example of a temporal plan for the problem exemplified in section 5.

```
[('Pickup', 0, 3, 30.0), ('Pickup', 1, 4, 25.0), ('Pickup', 1, 0, 25.0),
 ('Dropoff', 1, 4, 75.0), ('Dropoff', 1, 0, 75.0), ('Pickup', 0, 2, 30.0),
 ('Dropoff', 0, 3, 80.0), ('Pickup', 0, 1, 80.0), ('Dropoff', 0, 1, 140.0),
 ('Dropoff', 0, 2, 140.0)]
```

7 Deliverable

The Python program to be developed must be capable of (1) reading an input file, and (2) determine the cost of a given solution.

The Python program to be delivered should be called `solution.py` and include (at least) a class with name `FleetProblem` containing (at least) the following methods (see Annex A for the class template):

`load(fh)` loads a puzzle (initial configuration) from an file object[2] `fh`;

`cost(sol)` determines the cost of the solution given as argument `sol`, where `sol` is a list of actions, as defined in section 6.

For instance, for solution given in section 6 has cost $J = 144$.

8 Evaluation

The deliverable for this assignment is submitted and tested through *DEEC Moodle*[3], with the submission of a single python file, called `solution.py`, implementing the modules mentioned above. Instructions for this platform are available on the course webpage. Finally, the grade is computed in the following way:

[2] A file object corresponds to an opened file, e.g., the return of the *open()* function.

[3] URL: `https://moodle.deec.tecnico.ulisboa.pt/`

- 50% from the public tests;
- 50% from the private tests; and
- -10% from the code structure, quality and readability.

Deadline: **29-September-2023**.
Projects submitted after the deadline will not be considered for evaluation.

Closing Remarks on Ethics:

- Any kind of sharing code outside your group is considered plagiarism;
- Developing your code in any open software development tool is considered sharing code;
- You can use GitHub. Make sure you have private projects and remove them afterward;
- If you get caught in any plagiarism, either by copying the code/ideas or sharing them with others, you will not be graded; and
- The scripts and other supporting materials produced by the instructors cannot be made public.

A Deliverable Template

```
import search

class FleetProblem(search.Problem):

  def load(self, fh):
    """Loads a problem from the opened file object fh."""
    pass

  def cost(self, sol):
    """Compute cost of solution sol."""
    pass
```