FIT2004 S1/2016: Lab questions for week 6

THIS PRAC IS ASSESSED!
(Weight: 6 Marks)

CLASS: This programming exercise has to be prepared **in advance**. Your lab in which you will be interviewed is mainly to iron out bugs, clarify any doubts you have with your demonstrator, and finalize your implementation. Once past the **first hour** of your assigned lab (or earlier if someone volunteers), your demonstrator will come to you based on a random draw and assess your work. This lab assignment should be implemented in Python programming language. Practical work is marked on the efficiency and performance of your program **and also on your understanding of the program**. A *perfect* program with zero understanding implies you will get **zero** marks! "Forgetting" is not an acceptable explanation for lack of understanding. You must write all the code yourself, and may not use any external library routines that will invalidate the assessment of your learning. The usual input/output and other basic routines are exempted. Your attendance to your assigned lab is **compulsory**. If you are not interviewed in your lab, you will get zero marks.
DEMONSTRATORS are not obliged to mark programs that do not compile or that crash. Time allowing, they will try to help in tracking down errors, but they are not required to mark programs in such a state, particularly those that do not compile. Therefore keep backup copies of working partial solutions.
OBJECTIVE: This exercise will test your understanding of *dynamic programming* and problem-solving using this important technique. You will be handling a simplified version of a useful, real-world problem. This problem instance has several learning elements that will enhance your ability to understand and implement a dynamic programming algorithm. Constant practice of thinking about algorithmic problems and writing programs will help you evolve into a proficient programmer.

Supporting Material

For this exercise use the supporting material uploaded on the Unit's Moodle site under Week 6

Background

Consider a series of n **three dimensional** points $\{P_1, P_2, \ldots, P_n\}$. The task on hand is to summarise these points using a connected series of straight **line segments** (referred to here as **polylines**). Any possible polyline summary of n points starts at P_1 and ends at P_n, and potentially passes through some (or all) of the remaining $n-2$ points.

More formally, a polyline with m (where $m < n$) connected line segments defines *another* **ordered set** of $m+1$ (end) points, $\{ep_1, ep_2, \ldots ep_{m+1}\}$. This set of endpoints is a subset of

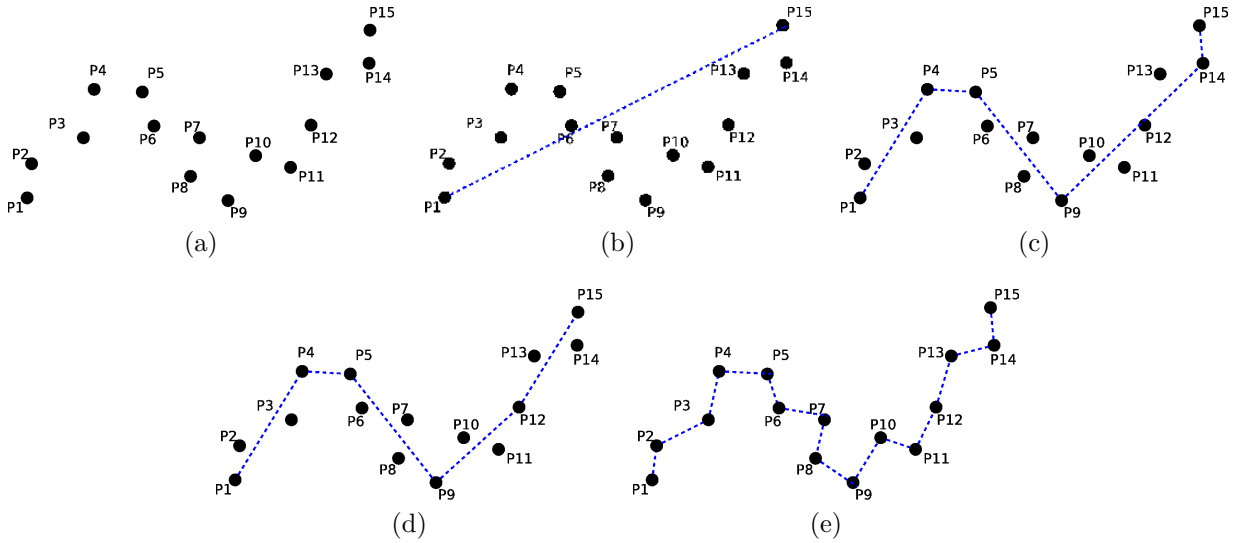

Figure 1: Two-dimensional example of polyline summarises of some 15 points.

the original n points with ep_1 always assigned to P_1 and ep_{m+1} always assigned to P_n; there is a line segment that connects any two **successive** endpoints ep_i ($= P_j$, say) and ep_{i+1} ($= P_k$, say) where $1 \leq i \leq m$, $1 \leq j < k \leq n$, and this line segment summarises all the points between P_j and P_k in the original set. In short, any polyline summary can be represented by the series of the end points of its line segments.

To get a clearer understanding of the polyline summaries, consider the two dimensional example in Fig. 1. In Fig. 1(a), a series of points $\{P_1, \ldots, P_{15}\}$ is shown. Four (of the many more possible) polyline summaries of these fifteen points are shown in Fig. 1(b-e).

In Fig. 1(b) the polyline contains only one line segment with the endpoints set containing the points $\{P_1, P_{15}\}$. This line segment summarizes all the points between P_1 and P_{15}. In Fig. 1(c) the polyline contains five line segments with $\{P_1, P_4, P_5, P_9, P_{14}, P_{15}\}$ as their endpoints. In other words, there are five connected line segments that can be defined, connecting each successive pair of endpoints:

(i) (P_1, P_4)

(ii) (P_4, P_5)

(iii) (P_5, P_9)

(iv) (P_9, P_{14})

(v) (P_{14}, P_{15})

Similarly, Fig. 1(d) is another polyline with five line segments but with (slightly) different set of endpoints. Finally, Fig 1(e) shows the extreme case where there is a line segment between every successive pair of points in the original set, leading to 14 line segments for that polyline summary.

Back to thinking about this problem more generally, it is not hard to see that, potentially, any pair of points (P_j, P_k) (where $1 \leq j < k \leq n$) can be the endpoints of a line segment in some candidate polyline summary. Indeed, the number of candidate polylines grows sharply as the number of points n in the original point set increases.[‡] To discriminate between the possible polyline summaries and choose an optimal one, a **cost function** is necessary.

[‡]Reason about this growth, while thinking about this assignment.

Therefore, given n points $\{P_1, P_2, \ldots, P_n\}$ in that order, we define the **cost** for any given polyline summary containing m (where $1 \leq m < n$) line segments with endpoints $\{ep_1, ep_2, \ldots, ep_{m+1}\}$ as the **sum of individual costs of each line segment** in that polyline.

$$\zeta(\{ep_1, ep_2, \ldots, ep_{m+1}\}) = \sum_{i=1}^{m} \text{line\_segment\_cost}(ep_i, ep_{i+1})$$

The individual cost of each line segment between two endpoints ep_i (= some P_j in the original point set) and ep_{i+1} (= some P_k in the original point set, where $1 \leq j < k \leq n$) is computed as

$$\text{line\_segment\_cost}(ep_i, ep_{i+1}) = 2 \cdot d_{jk} + \sum_{x=j+1}^{k-1} r_x,$$

where d_{jk} is the (Euclidean) distance between the endpoints P_j and P_k, and each r_x is the perpendicular (or shortest) distance of each point P_x $(j+1 \leq x \leq k-1)$ to the line segment going through P_j and P_k (see Fig. 2).

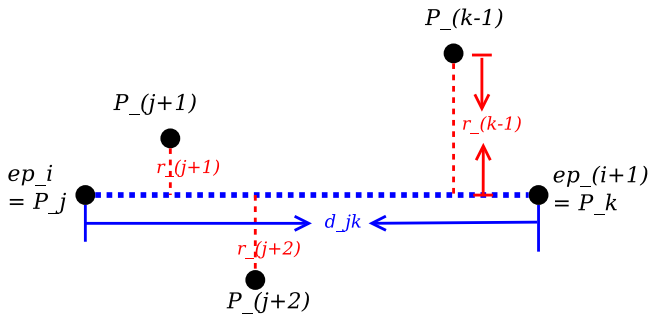

Figure 2: The line_segment_cost of a segment (from a larger polyline summary) between the endpoints (P_j, P_k) is composed of the distance d_{jk} between the endpoints and the sum of all perpendicular distances of the intermediate points to the line segment, $r_{j+1}, r_{j+2}, \ldots, r_{k-1}$. See line_segment_cost formula given above the main text.

Once this cost function is defined, any possible polyline summary of n points has its own cost. This sets up the optimisation problem of finding an optimal (**minimum cost**) polyline summary of a given set of n points. Interestingly, despite the fact that the number of possible polylines is very large[§], this problem can be addressed efficiently using a dynamic programming algorithm (DPA).

You will recall from the lectures, for DPA to be applicable for any problem, the problem should exhibit two properties:

- Optimal substructure
- Overlapping subproblem

These two properties for the current task of finding the optimal polyline summary can be **reasoned** as follows. Given some n points $\{P_1, \ldots, P_n\}$, let us define the optimal polyline summary of a **subproblem**, specifically, the subproblem that arises when we consider only the first k of the given n points, $\{P_1, \ldots, P_k\}$ $(1 \leq k \leq n)$. The solution of this subproblem depends further on its subproblems, that of finding the optimal polyline summary of first j points $\{P_1, \ldots, P_j\}$ (for some $1 \leq j < k$), and so on. Clearly, these subproblems overlap.

[§]Have you reasoned the growth?

Furthermore, the optimal substructure can be appreciated when we realize that the optimal cost of any subproblem solving for the points $\{P_1, \ldots, P_k\}$ is equal to the optimal cost of one of its subproblems, say $\{P_1, \ldots, P_j\}$ (for some $j < k$) **PLUS** the cost of stating an additional line segment from that P_j to P_k. So if the optimal cost of each subproblem is remembered (*memoized*) then the task of finding the optimal solutions for various subproblems (of incrementally increasing sizes) can be computed rather efficiently. The dynamic programming recurrence should be implicit in this reasoning.

What else do you need to solve this problem? You will need to write functions to compute the things required for calculating any line_segment_cost (see Fig. 2): the distance (d_{jk}) between any two three-dimensional points P_j and P_k, and the perpendicular distance (r_x) of any point P_x to the line segment defined by P_j and P_k. Both these tasks are something that will rely on your math prerequisites from previous semesters and high school.

Lab Assignment Questions

Based on this background, address the following questions:

1. Access the supporting material folder in the Week 6 section of moodle. You will find several text files, each containing a list of three-dimensional (x, y, z) points of different sizes: `points-file1.txt`, `points-file2.txt` and so on.
   Write a python program that:
   - takes as input, one of the above text files as an argument,
   - reads the list of (x, y, z) points in that file,
   - and outputs the **optimal** polyline summary for those input points using the cost function introduced in the background.**
2. Determine the time complexity of your program. You do not have to prove this formally. Just reason and justify your reasoning to your tutor when he interviews you in the lab.
3. How many polylines summaries are possible for n input points? (This characterizes the size of the search space.)

```
-=o0o=-
  END
-=o0o=-
```

Nonexaminal but recommended exercise
For your learning, try to familiarise with plotting using python. For instance, take a look at this link and give it a go: `https://pythonprogramming.net/matplotlib-3d-scatterplot-tutorial/`. There are other options too, if you search for 3D plots using python online. Using a 3D plotter, you can visualize the data you are reading as input and your optimal polyline summary. Enjoy!

**Output format: Simply print out the indexes of the endpoints defining the optimal polyline. If we assumed that the polyline summary illustrated in Fig. 1(c) were optimal, then your program's output would be the indexes: `(1, 4, 5, 9, 14, 15)`.