COMPSCI 330 Spring 2023
Case Study: Trajectory Data Analysis, Part I

Due Date: March 9, 2023

Directions

- ***Group work***: You work on the case study in a group of at most four. Ideally, two assignment groups work together, but it is not required.
- ***Submission:*** Each group should submit:
    1. Source code for all implementations in an approved programming language and a `README` file as a `.zip` archive.
    2. All analyses, figures, and discussions as presentation slides in `.ppt` or `.pptx` format; see Appendix B for details.
    3. List contributions of each group member in the slides.

Each group should make only ***ONE*** submission. Add every group member on gradescope, see here how to do this. Also, write the name of every group member in the `README` of your zip archive and the title slide of your slide deck.

1 Introduction

This case study asks you to design, implement, and analyze algorithms to perform various tasks for trajectory data analysis. A trajectory is a curve modeling the change of a set of variables over time. For most practical applications, trajectories are represented by a set of discrete observations. Trajectory data analysis arises in a range of applications, including traffic data analysis, analyzing migration and commute patterns, computational finance, computational molecular biology, computer vision, robotics, and machine learning. Furthermore, large trajectory data sets are being collected at an unprecedented rate and are widely available.

Part I of the case study consists of three tasks, each of which requires modeling a problem, designing and implementing an algorithm, and running some experiments [1]. Note that there might be multiple solutions for each task and you have to justify your design choices. For this case study, we shall use the GeoLife trajectory dataset. These are real-world, observed trajectories centered around Beijing. See Appendix A for details.

2 Definitions

Trajectories: A trajectory T is represented as a sequence of points sampled on the curve, i.e., $T = \langle p_1, p_2, \ldots, p_m \rangle$, where $p_i = (a_i, b_i) \in \mathbb{R}^2$ is a point in the plane. For simplicity, assume the actual curve is reconstructed by performing linear interpolation between two consecutive sample points, i.e., the polygonal curve with vertices $p_1, p_2, \ldots, p_m$. See Figure 1. We denote the number of vertices in T as $|T|$.

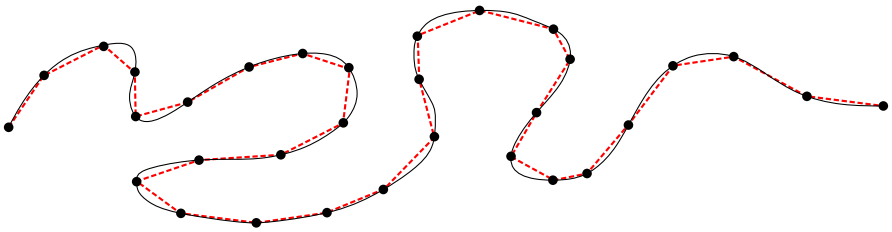


Figure 1: An original trajectory and its sampled points. The dashed trajectory is obtained by linearly interpolating between consecutive points.

3 Task 1: Identifying the Hubs

The first task is to identify *hubs* in trajectory data, the locations that have a high traffic density. These can be shopping complexes, tourist attractions, traffic bottlenecks, etc. Density at a point p can be defined as the number of observed points in a small fixed-size neighborhood (i.e., the number of observed points in a disk (or square) of a small fixed radius (or side length) centered at p) of p.

Recall that a trajectory is represented as a sequence of sampled points. Let $\mathscr{P}$ denote the set of all observation points on the input trajectories. We shall use $\mathscr{P}$ to identify hubs. To identify k hubs, for a parameter $k > 0$, we compute a set H of k high-density points, not necessarily points of $\mathscr{P}$,

[1] Part II will focus on clustering trajectory data and finding patterns in it

such that no two points in H lie within a specified distance of r from each other, i.e., they are not very close to each other. See Figure 2.

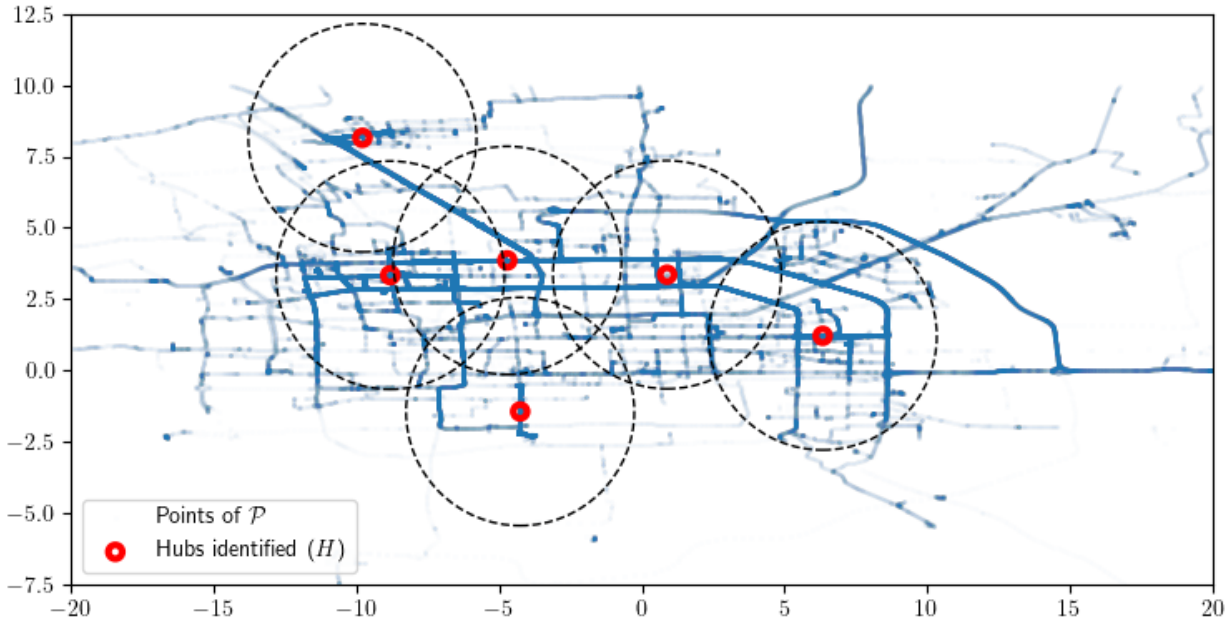


Figure 2: An example of $k = 6$ hubs identified from $\mathscr{P}$. The dotted circles show the separating distance r (4 kilometers in this case)

3.1 What to design

- Define a density function based on $\mathscr{P}$ that you will use and describe the choice of parameters.
- Design an algorithm density(p) that calculates the density for a point p in the plane, not necessarily a point of $\mathscr{P}$ using the set of points $\mathscr{P}$. Point p may or may not be in $\mathscr{P}$. You should do some pre-processing on $\mathscr{P}$ in $O(n \log n)$ time and build a data structure so that for a query point p, density(p) can be computed quickly simply by visiting the points of $\mathscr{P}$ in a small neighborhood of p.
- Design the algorithm hubs($\mathscr{P}; k, r$) that identifies $k > 0$ hubs of high density such that any two hubs are separated by at least distance r. For the sake of efficiency, the algorithm should call density as few times as possible.

3.2 What to code

- Implement the procedures density(p) and hubs($\mathscr{P}; k, r$).
- Make a scatter plot of (i) the set of hubs identified using $k = 10$ and $r = 8km$ and (ii) points of $\mathscr{P}$ in the background of the same figure using different markers and colors.

- Run the hubs algorithm with $k = 5, 10, 20, 40$ and $r = 2km$ at least three times and report the algorithm's runtime in milliseconds for these values of k.
- Run the hubs for the provided 10%, 30%, and 60% subsamples and the full dataset with $k = 10, r = 8km$ and report the algorithm's runtime in milliseconds for each dataset.

4 Task 2: Trajectory Simplification

Trajectory data is often high resolution, while many trajectory-analysis algorithms are computationally expensive, so one often compresses the trajectory data — by subsampling a subset of trajectory points carefully — to obtain a speedup at a small loss of accuracy. Trajectory simplification, defined below, is a popular way of compressing trajectory data.

Formally, a *simplification* of a trajectory $T = \langle p_1, \ldots, p_n \rangle$ is a subsequence $T^* = \langle p_{i_1}, p_{i_2} \ldots, p_{i_k} \rangle$ of T where $1 = i_1 < i_2 \ldots < i_k = n$. For example, $\langle p_1, p_4, p_7, p_{10} \rangle$ is a simplification of $\langle p_1, \ldots, p_{10} \rangle$ in Figure 3b. We say that the sequence $p_{i_j} p_{i_{j+1}}$ covers the points $p_{i_j}, p_{i_j+1}, \ldots p_{i_{j+1}}$ of T which lie between p_{i_j} and $p_{i_{j+1}}$. For a point $q \in \mathbb{R}^2$ and a line segment $e = ab$, the distance from q to e, denoted by $d(q, e)$, is the distance from q to its closest point on e.

Note that $d(q, e)$ is either the distance between q and its projection q' on the line supporting e (if q' lies on e) or the distance between q and one of the endpoints of e. See Figure 3a. The error of a simplification $T^* = \langle p_{i_1}, p_{i_2}, \ldots, p_{i_k} \rangle$ is defined as the maximum distance between a point $p_j \in T$ and the line segment of T^* that covers p_j. Namely,

$$\mathsf{err}(T^*, T) = \max_{1 \le j < k} \max_{i_j < r < i_{j+1}} d(p_r, p_{i_j} p_{i_{j+1}}). \tag{1}$$

See Figure 3b. There is a trade-off between $|T^*|$ and $\mathsf{err}(T^*, T)$. Given an error parameter $\varepsilon > 0$, a simplification T^* of T is called an *ε-simplification* if $\mathsf{err}(T^*, T) \le \varepsilon$. The size of T^* is the number of points in T^*

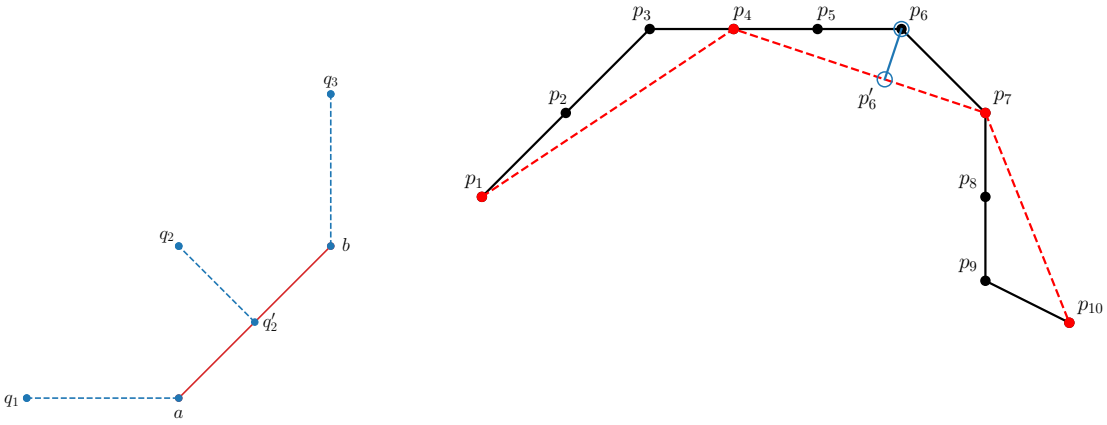
4.1 What to design

- Given T and ε, a fast heuristic TS-greedy that computes an ε-simplification of T of small size. Note that TS-greedy should always return an ε-simplification, though it may not be the smallest size. Ideally, the algorithm's average running time on a trajectory T should be $O(|T|)$ or $O(|T| \log |T|)$, e.g., on a real-world trajectory.

4.2 What to code

- Implement the procedure to compute $d(q, e)$, the distance between a point q and a segment e.
- Implement the greedy algorithm TS-greedy(T, ε) to compute an ε-simplification of T.
- Plot the trajectory ID `128-20080503104400` and its simplification using the above function for $\varepsilon = 0.03, 0.1, 0.3$ (kilometers). Each figure should contain two line plots: trajectory and its simplification, with markers of different colors.

- Evaluate and report the compression ratio $|T|/|T'|$ for trajectories `128-20080503104400`, `010-20081016113953`, `115-20080520225850`, and `115-20080615225707` using TS-greedy for $\varepsilon = 0.03km$.



(a) Projection of points on a line segment

(b) Trajectory simplification example

Figure 3: (3a) Distance between a point and segment $e = ab$: $d(q_2,e) = ||q_2 - q_2'||$, $d(q_1,e) = ||q_1 - a||$, and $d(q_3,e) = ||q_3 - b||$; q_2' is the projection of q_2 on e and $q_2q_2' \perp e$. (3b) Original trajectory $T = \langle p_1,\ldots,p_{10}\rangle$ denoted by the solid black line and its simplification $T' = \langle p_1,p_4,p_7,p_{10}\rangle$ denoted by the dashed red line.

5 Task 3: Comparing Trajectories

Task 3 involves designing and implementing an algorithm for computing similarity between a pair of trajectories. We define the distance between two trajectories with respect to the notion of *assignment*, as described below. Roughly speaking, each point on one trajectory is mapped to a point on the other, and we define the distance between the two trajectories as the average or the maximum of distances between matched points. We now define the distance function more formally.

Monotone Assignment: Let $P = \langle p_1,\ldots,p_r\rangle$ and $Q = \langle q_1,\ldots,q_s\rangle$ be two trajectories represented by sequences of points in $\mathbb{R}^2$. An *assignment* for P and Q is a sequence of pairs $E \subseteq P \times Q$ such that each point of $P \cup Q$ appears in at least one pair of E. If $(p_i,q_j) \in E$ then, we say that p_i is matched to q_j in E. A point may be matched to more than one point. For example, p_2 is matched to q_2 and q_3 in Figure 4a. E is called a monotone assignment if the following condition holds: if p_i is matched to q_j, then p_{i+1} cannot be matched to any point before q_j. That is, if $(p_{i+1},q_{j'}') \in E$, then, $j' \geq j$. Figure 4b gives an example where this condition does not hold. Note that a point p_i is matched to a contiguous set of points of Q and vice-versa. See Figure 4. We define the score of E, denoted by $\sigma_{\text{avg}}(P,Q;E)$ as:

$$\sigma_{\text{avg}}(P,Q;E) = \frac{1}{|E|} \sum_{(p_i,q_j)\in E} d^2(p_i,q_j). \tag{2}$$

Here, $d(\cdot,\cdot)$ is the Euclidean distance. We define the distance between P and Q, called dynamic time warping, as:

$$\mathrm{dtw}(P,Q) = \min_E \sigma_{\mathrm{avg}}(P,Q;E), \tag{3}$$

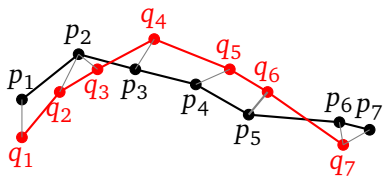
where the minimum is taken over all monotone assignments E. We call the assignment minimizing dtw as E_{avg}. Alternately, we can define the score of E to be:

$$\sigma_{\max}(P,Q;E) = \max_{(p_i,q_j)\in E} d(p_i,q_j) \tag{4}$$

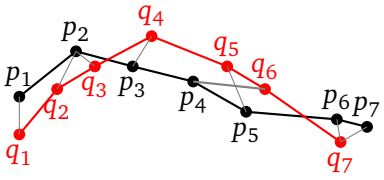
and define the Frechet distance as

$$\mathrm{fd}(P,Q) = \min_E \sigma_{\max}(P,Q;E). \tag{5}$$

The minimizing assignment for fd is called $E_{\max}$.



(a) An example of monotone assignment E



(b) An example of a non-monotone assignment E that includes (p_4,q_6) and (p_5,q_5)

Figure 4: Example of monotone and non-monotone assignments between black trajectory $P = \langle p_1,\ldots,p_6\rangle$ and red trajectory $Q = \langle q_1,\ldots q_8\rangle$. Thin gray lines indicate pairs in assignments.

5.1 What to design

- Design algorithms for computing $\mathrm{dtw}(P,Q)$ and $\mathrm{fd}(P,Q)$ as well as assignments E_{avg} and $E_{\max}$ that realize the minimum values. (See Assignment 4.)

5.2 What to code

- Implement the proposed algorithms to compute $\mathrm{dtw}(P,Q)$ and $\mathrm{fd}(P,Q)$
- Run both algorithms for trajectory pairs (`128-20080503104400`, `128-20080509135846`), (`010-20081016113953`, `010-20080923124453`), and (`115-20080520225850`, `115-20080615225707`). For each pair, plot the histograms of lengths of edges in E_{avg} and $E_{\max}$.
- Compute a simplification of each of $T_1 =$ `115-20080520225850`, $T_2 =$ `115-20080615225707` for $\varepsilon = 0.03, 0.1, 0.3$ (kilometers). For $i = 1,2$, let T_i^ε be the ε-simplification you computed. For each ε, compute E_{avg} of T_1^ε and T_2^ε and plot the histogram as above. Draw all three histograms in one plot.

Appendices

A Datasets

We shall use the Microsoft Research GeoLife data set for this case study. GeoLife consists of a GPS trajectories tracked using a smartphone application between 2008 to 2011. Most of the trajectories from GeoLife are centered around Beijing, although some are from far away. Source: https://www.microsoft.com/en-us/download/details.aspx?id=52367

- The GeoLife dataset consists of trajectories recorded while using different modes of transport such as pedestrian, public transport, cars, etc. In this case study, we shall limit our scope to car trajectories near Beijing. The dataset for this case study is filtered using these criteria.
- The original dataset provides all trajectories in the latitude-longitude format. We have converted the data from latitude-longitude format to cartesian coordinates (x, y) by projecting them to a plane, as these trajectories lie in a small geographic area.
- Download the `geolife-cars.csv` dataset from Sakai. The dataset contains the following fields:

  1. `date`: Timestamp of the observed location point
  2. `id_`: Unique id for a trajectory in `user-start_time` format.
  3. `x`: Projected value in a cartesian plane corresponding to the longitude. Unit: kilometer
  4. `y`: Projected value in a cartesian plane corresponding to the latitude. Unit: kilometer

  For task 1, use all values of columns `x` and `y` for points in $\mathscr{P}$. For tasks 2 & 3, you'll need to use all points of a chosen trajectory using the `id_` field in the same sequence.
- In addition to `geolife-cars.csv`, task 1 requires to run the `hubs` algorithm on subsamples. Download and use `geolife-cars-ten-percent.csv`, `geolife-cars-thirty-percent.csv`, and `geolife-cars-sixty-percent.csv` files to evaluate your algorithm on these subsamples.

B Submission

This section describes what is expected in the slides and the source code.

B.1 Slides

Submit a slide deck in `.ppt` or `.pptx` (PowerPoint) format that includes all algorithm designs, experimental results, and conclusions. All the design descriptions and experimental results should be on the main slides. Any discussion beyond the basics can go in the speaker's notes. Make sure to mention on the slides to check the speaker's notes if any text is included. Google Slides, OpenOffice, LibreOffice, and Keynote can export to the required formats. The title slide should include the names of all group members. We describe the slides expected for each of the tasks:

B.1.1 Task 1 (5 slides):

1. Clearly explain the density function density that you used, the reason for choosing the specified function, and the set of parameters you used. Describe each parameter in the parameter set Θ. Illustrate density calculation with an example. Describe the preprocessing algorithm and the algorithm for computing density(p). State the time complexity of density(p).
2. Describe the hubs($\mathscr{P}; k, r$) procedure. State its time complexity.
3. Include experimental results from Section 3.2.
4. Summarize the benefits and drawbacks of the proposed algorithm for hubs. Describe potential avenues for further improvement.

B.1.2 Task 2 (4 slides):

1. Explain the proposed trajectory simplification algorithm TS-greedy. State the time complexity.
2. Include experimental results from Section 4.2.
3. Describe the advantages and drawbacks of the proposed greedy algorithm. Discuss any future scope for enhancement.

B.1.3 Task 3 (5 slides):

1. Describe the proposed distance algorithm and state its space and time complexity.
2. Include experimental results from Section 5.2.
3. Discuss the proposed algorithm in terms of advantages and drawbacks.

In addition to these slides, include a slide that describes the contribution of each group member.

B.2 Source code

1. ***Programming languages***: All groups must use Java or Python, similar to Homework 4.
2. A single ZIP archive that includes all of your source code and a `README` file that lists group members and describes all compilation instructions, execution instructions, and the organization of your code (e.g., what each file contains). If you use any datasets that we do not provide, include links to them if they are accessible online, or include the dataset with your submission if you generated them.
3. You may only use standard library modules/packages and not external dependencies or other code, except for external packages for visualization, if desired. The grader should be able to reproduce all of your results by running your code on a clean Java or Python 3 installation.

C Grading Rubrics

	Excellent (100%)	Satisfactory (75%)	Unsatisfactory (50%)
Task 1	(i) The `hubs` algorithm running time should be $O(n)$ (ii) The `density` algorithm should be $O(1)$ assuming a finite neighborhood with a $O(n \log n)$ preprocessing time	Either of the algorithms (`density`, `hubs`) does not satisfy the runtime complexity requirements but produces desired results	(i) Incomplete submission, or (ii) algorithm does not identify k distinct hubs, or (iii) There are multiple hubs within a disc of radius r
Task 2	(i) The algorithm produces a small-size simplification (that is a subsequence of a trajectory T) in $O(\|T\| \log \|T\|)$ time	(i) The algorithm produces a correct simplification, but either the algorithm is slow, or the simplification size is not small.	(i) Incomplete submission, (ii) Output is not correct
Task 3	Algorithm finds correct monotone assignments E_{avg} and $E_{\max}$ in $O(\|P\| \cdot \|Q\|)$ time where P and Q are trajectories being compared	The algorithm is slower but finds correct monotone assignments	(i) Incorrect monotone assignments, or (ii) Incomplete submission
Slides	(i) Include all the experimental results and design descriptions from Section B.1. Include member names and contributions (ii) Minimal use of speaker's notes	Missing up to three experimental results, but describe all algorithms	Missing algorithm description or more than three experimental results
Code zip	(i) Reproducible, readable, and well-commented code (ii) `README` file describing files in the archive and member names	(i) Reproducible code that lacks comments/unreadable code, (ii) No use of external dependencies	(i) Use of external dependencies other than visualization, or (ii) non-reproducible code

D Glossary

- **Euclidean distance:** Euclidean distance $d(p,q)$ between points $p=(p_x,p_y)$ and $q=(q_x,q_y)$ in $\mathbb{R}^2$ is given as $d(p,q)=\sqrt{(p_x-q_x)^2+(p_y-q_y)^2}$.
- **Trajectory:** A trajectory T is represented as a sequence of points sampled on the curve, i.e., $T=\langle p_1,p_2,\ldots,p_m\rangle$, where $p_i=(a_i,b_i)\in\mathbb{R}^2$ is a point in the plane. For simplicity, assume the actual curve is reconstructed by performing linear interpolation between two consecutive sample points, i.e., the polygonal curve with vertices $p_1,p_2,\ldots,p_m$. See Figure 1.
- **Density:** Density at a point p can be defined as the number of observed points in a small fixed-size neighborhood (i.e., the number of observed points in a disk (or square) of a small fixed radius (or side length) centered at p) of p.
- **Simplification:** A simplification of a trajectory $T=\langle p_1,\ldots,p_n\rangle$ is a subsequence $T^*=\left\langle p_{i_1},p_{i_2}\ldots,p_{i_k}\right\rangle$ of T where $1=i_1<i_2\ldots<i_k=n$.
- **ε-simplification:** Given an error parameter $\varepsilon>0$, a simplification T^* of T is called an *ε-simplification* if $\mathsf{err}(T^*,T)\leq\varepsilon$ where the $\mathsf{err}(T^*,T)$ is given by Equation 1.
- **Monotone assignment:** Let $P=\langle p_1,\ldots,p_r\rangle$ and $Q=\langle q_1,\ldots,q_s\rangle$ be two trajectories represented by sequences of points in $\mathbb{R}^2$. A *monotone assignment* for P and Q is a sequence of pairs $E=\left\langle(p_{i_1},q_{j_1}),(p_{i_2},q_{j_2}),\ldots,(p_{i_t},q_{j_t})\right\rangle$ such that $i_1\leq i_2\leq\ldots\leq i_t$ and $j_1\leq j_2\leq\ldots\leq j_t$
- **Dynamic time warping (**dtw**):** Dynamic time warping distance between trajectories P and Q is the minimum of $\sigma_{\text{avg}}(P,Q;E)$ over possible assignments E where $\sigma_{\text{avg}}(P,Q;E)$ is calculated as per Equation 2.
- **Frechet distance (**fd**):** Frechet distance between trajectories P and Q is the minimum of $\sigma_{\max}(P,Q;E)$ over possible assignments E where $\sigma_{\max}(P,Q;E)$ is calculated as per Equation 4.