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

Due Date: April 25, 2023

Directions

- ***Group work***: As Part I, you work on the case study in a group of at most four.
- ***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 C 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

Part I of the case study focused on algorithms for finding hubs (Task 1), trajectory simplification (Task 2), and trajectory comparison (Task 3). Part II consists of two tasks: (i) computing center trajectories (Task 4) and (ii) clustering trajectories (Task 5). You will design and implement algorithms and run some experiments for each task.

Given a set of trajectories, centering allows us to identify a single trajectory to represent the set. This is similar to taking the mean of a set of numbers or choosing a representative value. We shall use the comparison metrics explored in Task 3 to find the averages over trajectories. Like points in the plane, clustering groups similar trajectories into “clusters.” This helps identify sets of similar trajectories given a distance measure. Trajectory clustering can identify common transit paths in road networks, find a set of stocks that show similar trends in the stock markets, etc. Together with Part I, these two tasks give a method for computing a compact summary of a trajectory data set. We continue using the processed GeoLife data set for Part II of the case study.

2 Definitions

Trajectory: As in Part I, 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.

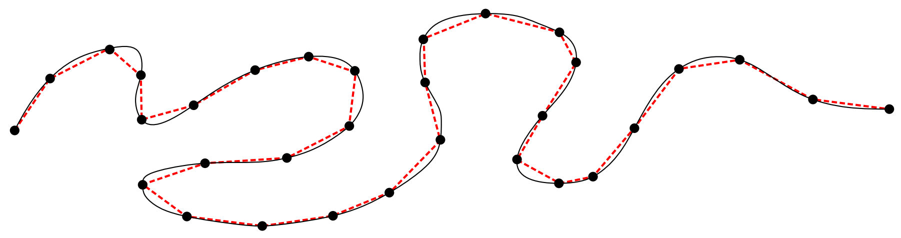

Figure 1: An original trajectory and its sampled points. The dashed trajectory is obtained by linearly interpolating between consecutive points.

Trajectory set: We use $\mathscr{T} = \{T_1, \ldots, T_n\}$ to denote the set of n given trajectories.

Size: Size m of a trajectory $T = \langle p_1, \ldots, p_m \rangle$ is the number of vertices in T. We denote the size of T using $|T|$.

Maximum trajectory size: Given a set of trajectories $\mathscr{T}$, we define the maximum trajectory size M as size of the longest trajectories of $\mathscr{T}$ in terms of number of points. M bounds the size of any trajectory T of $\mathscr{T}$. We calculate M as: $M = \max_{T \in \mathscr{T}} |T|$.

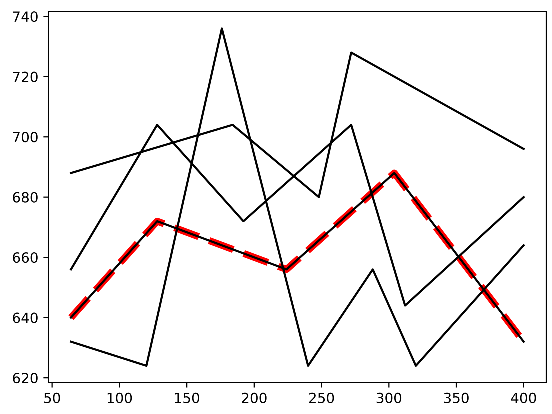

Figure 2: An example $\bar{T}_C$ trajectory (dashed red line) for a set of trajectories (solid black lines). Note that the $\bar{T}_C$ trajectory calculated is one of the input trajectories.

3 Task 4: Center Trajectories

Let Δ be a distance function on trajectories in $\mathscr{T}$, i.e., for two trajectories T, T' in $\mathscr{T}$, $\Delta(T, T')$ returns a measure of distance between them. Dynamic Time Warping (dtw) and Frechet Distance (fd), which we saw in Part I, are two examples of distance functions. The center trajectory of $\mathscr{T}$ is a trajectory $\bar{T}$ that "best" represents the trajectories $\mathscr{T}$ with respect to Δ. $\bar{T}$ need not be one of the trajectories of $\mathscr{T}$. The notion of center trajectory is similar to how the centroid (center of mass) is often used to represent the point set.

Computing the best center trajectory $\bar{T}$ is computationally intractable. Therefore, this task asks to design and implement two approaches for approximating the center trajectory of a given set $\mathscr{T}$ of trajectories. One of the approaches is given below, and you will propose the second one.

Approach I: We define the center trajectory $\bar{T}_C$ of $\mathscr{T}$ as one of its trajectories that minimizes the total distance from all trajectories in $\mathscr{T}$. See Figure 2(a). Formally, $\bar{T}_C$ can be defined as:

$$\bar{T}_C = \operatorname*{arg\,min}_{T \in \mathscr{T}} \sum_{T' \in \mathscr{T}} \Delta(T, T'). \tag{1}$$

Note that the computation of $\bar{T}_C$ requires computing $\Delta(T, T')$ for all pairs of trajectories in $\mathscr{T}$, which is computationally expensive. You can use trajectory simplification to speed up the computation by computing the distances between simplified trajectories.

3.1 What to design

1. Approach II. Computing the center trajectory $\bar{T}_C$ using Approach I, requires evaluating all pair-wise distances between the trajectories of $\mathscr{T}$. This makes evaluating $\bar{T}_C$ a $O(n^2M^2)$ operation where n and M are as defined in Section 2. Design an alternate approach to define a center trajectory, which can be computed in $O(\sum_{i=1}^{n} |T_i|)$ time. The center trajectory need not be one

of the input trajectories, and you can choose the number of points used to represent $\bar{T}$. (**Hint:** *Think of each trajectory $T_i \in \mathscr{T}$ as a function f_i of time, and the goal is to compute a function that computes an average of these functions.*)

3.2 What to code

1. Implement both of the centering approaches. Use dtw as the distance measure for the first approach.

2. Consider trajectory IDs given in file `trajectory-ids.txt`. For these trajectories, calculate the center trajectories using Approach I and Approach II with distance function Δ as dtw. Plot and label (i) all trajectories in `trajectory-ids.txt` and (ii) center trajectories using line plots. Ensure that both center trajectories can be distinguished in the plot. Report the average distance of trajectories in `trajectory-ids.txt` from each center trajectory.

3. Repeat the experiment in Step 2 using ε-simplification from Task 2 of Part I. Evaluate the centering Approach I for $\varepsilon = 0.03,\ 0.1,\ 0.3$ kilometers. See Appendix A.2.

4 Task 5: Clustering Trajectories

A *k-clustering* of $\mathscr{T}$ is a partition $\mathscr{C} = \{\mathscr{T}_1, \ldots, \mathscr{T}_k\}$ of $\mathscr{T}$ into k pairwise-disjoint subsets called clusters, so that each trajectory of $\mathscr{T}$ appears in exactly one cluster. For each cluster $\mathscr{T}_i$, let $\bar{T}^i$ be its *center* trajectory. The centering approaches described in Section 3 can be used for approximating $\bar{T}^i$. Next, we define the cost $\mathsf{cost}(\mathscr{T}_i, \bar{T}^i)$ of a partition $\mathscr{T}_i$ for a given central trajectory $\bar{T}^i$ as:

$$\mathsf{cost}(\mathscr{T}_i, \bar{T}^i) = \sum_{T' \in \mathscr{T}_i} \Delta(\bar{T}^i, T'). \tag{2}$$

Further, we define the *cost* of clustering $\mathscr{C}$ as the sum of costs of all partitions in $\mathscr{C}$.

$$\mathsf{cost}(\mathscr{C}) = \sum_{i=1}^{k} \mathsf{cost}(\mathscr{T}_i, \bar{T}^i). \tag{3}$$

The goal is to compute a k-clustering with the minimum cost and to represent each cluster by its center trajectory. Clustering allows us to identify a few representatives of a larger data set by the *centers* of those clusters. Computing a minimum-cost k-clustering is computationally intractable, so the goal will be to compute a k-clustering of a small cost efficiently. One such approach is Lloyd's Algorithm discussed in class (adapted for trajectory data), and also briefly described in Appendix A.1. Lloyd's algorithm is typically initialized by choosing a random subset of k objects from the object set. However, the algorithm is sensitive to the choice of initial partitions. Therefore, by carefully initializing Lloyd's algorithm, one can achieve a better clustering.

4.1 What to design

1. Propose an algorithm for carefully selecting the initial centers for Lloyd's algorithm. This is often called "seeding."

2. Given n trajectories and k clusters, the runtime of Lloyd's algorithm can be bounded by $O(tknM^2)$, where t is the number of iterations to termination, and M is the maximum size of a trajectory. As defined above, this can be computationally expensive in practice. Use the trajectory simplification method from Task 2 to improve the runtime. There is a tradeoff between the reduction of trajectory size and simplification error induced due to simplification. Carefully choose the simplification parameter.

4.2 What to code

1. Implement Lloyd's algorithm with a randomized seed for a given value of k and with `dtw` as distance Δ. Randomized seeding implies that the initial partitions $\mathcal{T}_1, \ldots, \mathcal{T}_k$ are chosen randomly. See Appendix A.1. You may use the modification from Step2 of Section 4.1 to improve the running time of your implementation. Also, set $t_{\max}$ — the maximum number of iterations, described in Appendix A.1 — to ensure Lloyd's algorithm always terminates.
2. Implement the proposed seeding method. Modify the code from Step 1 such that the seeding method can be specified in a function call to Lloyd's algorithm.
3. Run Lloyd's algorithm on all trajectories in `geolife-cars-upd8.csv`. Evaluate the cost of clustering for $k = 4, 6, 8, 10, 12$ for the random and the proposed seeding method. For robustness, evaluate the cost three times for each value of k, and report the average. Draw a line plot of the average cost of clustering vs. k, for both seeding methods. What value of k would you recommend?
4. Monitor the cost of clustering over iteration. For robustness, we evaluate the costs over r runs of Lloyd's algorithm. Let C be the matrix of the cost of clustering such that C_{ij} represents the cost of clustering in the i^{th} run of Lloyd's algorithm for j^{th} iteration. We denote the average cost of clustering as $\bar{C}_j = \frac{1}{r}\sum_{i=1}^{r} C_{ij}$. Plot the average cost of clustering over iterations for both the seeding methods. Use the value of k you suggested in Step 3.
5. Plot center trajectories of all clusters evaluated using the proposed seeding with k from Step 3 in the same plot. Use colors to distinguish between clusters.

Appendices

A Algorithms

A.1 Lloyd's Algorithm

1. Partition $\mathscr{T}$ into k sets $\mathscr{T}_1, \dots, \mathscr{T}_k$ using the specified seed method.
2. Iteratively repeat the following two steps,

   Center computation. For each $1 \le j \le k$, compute the *center* $\bar{T}_j$ of $\mathscr{T}_j$.

   Re-assignment. For each trajectory $T \in \mathscr{T}$, assign it to the cluster whose center trajectory is closest to T. Alternately, we can see this as a re-definition of the partitions such that each partition only includes trajectories whose closest trajectory is the central trajectory of the partition. Formally, each cluster $\mathscr{T}_j$ is redefined as:

   $$\mathscr{T}_j = \{T \in \mathscr{T} \mid \Delta(\bar{T}_j,\ T) \le \Delta(\bar{T}_l, T),\ \text{ for all } l \le k\}.$$

   Repeat. Until either (i) the partitions remain the same before and after an iteration or (ii) $t_{\max}$ iterations are completed.

A.2 Trajectory Simplification

Here is a simple recursive algorithm for computing an ε-simplification of a trajectory $T = \langle p_1, \dots, p_n \rangle$: Let p_m be the point of T that is farthest from the line segment $p_1 p_n$. If $d(p_m, p_1 p_n) \le \varepsilon$, return $\langle p_1, p_n \rangle$ as the simplification. Otherwise, recursively compute ε-simplification of $\langle p_1, \dots, p_m \rangle$ and $\langle p_m, \dots, p_n \rangle$. Suppose they are $\langle u_1, \dots, u_r \rangle$ and $\langle v_1, \dots, v_s \rangle$. Note that $u_1 = p_1$, $u_r = v_1 = p_m$, and $v_s = p_n$. Then return $\langle u_1, \dots, u_{r-1}, p_m, v_2, \dots, v_s \rangle$ as the desired simplification of T.

A.3 Dynamic Time Warping

Let T_i, T_j be two trajectories. Let $A(T_i, T_j)$ be a monotone assignment of T_i, T_j that minimizes the sum of squared Euclidean distances of the edges in the assignment. Let $C(T_i, T_j)$ denote the cost of assignment, the sum of squared Euclidean distances of the edges in the assignment:

$$C(T_i, T_j) = \sum_{(p,q)\, \in\, A(T_i, T_j)} d(p,q)^2,$$

where, $d(p,q)$ is the Euclidean distance between points $p \in T_i$ and $q \in T_j$. We define $\Delta(T_i, T_j)$ as:

$$\Delta(T_i, T_j) = \sqrt{\frac{C(T_i, T_j)}{|A(T_i, T_j)|}},$$

where $|A(T_i, T_j)|$ is the number of edges in the assignment. Recall that the cost of the best assignment can be computed recursively. Suppose $T_i = \langle u_1, \dots, u_r \rangle$ and $T_j = \langle v_1, \dots, v_s \rangle$. For $1 \le a \le r$ and $1 \le b \le s$, let $C(a, b)$ be the minimum cost of aligning $\langle u_1, \dots, u_a \rangle$ and $\langle v_1, \dots, v_b \rangle$, then,

$$C(a,b) = d^2(u_a, v_b) + \min\{C(a-1, b),\ C(a-1, b-1),\ C(a, b-1)\}.$$

B Dataset

Trajectory data: We use the same dataset we used for Part I. See Appendix A from Part I for more details.

- Download the `geolife-cars-upd8.csv` dataset from Sakai. The dataset contains the following fields:

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

  For both tasks in Part II, you'll need to use all points of a chosen trajectory using the `id_` field in the same sequence.

- Unlike part 1, you do not need the sample datasets, namely: `geolife-cars-ten-percent.csv`, `geolife-cars-thirty-percent.csv`, and `geolife-cars-sixty-percent.csv` for this part.

Trajectory IDs: Task 4 asks to compute the center trajectory of a given set of trajectories using two approaches (Section 3.2). The experiments use trajectory IDs from the `trajectory-ids.txt` file. To run these experiments,

1. Download `trajectory-ids.txt` file from Sakai. The file contains trajectory IDs for Task 4 delimited by newlines.
2. Select the listed trajectory IDs from `geolife-cars-upd8.csv` using the `id_` field as described above.
3. Run the centering experiments from Section 3.2

C Submission

This section describes what is expected in the slides and the source code. The submission format will remain the same as Part 1.

C.1 Slides

Submit a slide deck in `.pdf` format that includes all algorithm designs, experimental results, and conclusions. Google Slides, OpenOffice, LibreOffice, and Keynote can be used to create and export the slides to the required formats. Alternatively, one can create the slides directly using LaTeX. The title slide should include the names of all group members. Make sure to note the contributions of each group member in the slides. We describe the slides expected for each of the tasks:

C.1.1 Task 4 (4 slides)

1. Describe the proposed Approach II.
2. Include the experiment results from Section 3.2
3. Compare Approach II with Approach I and discuss its strengths and weaknesses.

C.1.2 Task 5 (5 slides)

1. State and explain the proposed seeding method.
2. Include the experiment results from Section 4.2. Discuss the advantages of the proposed seeding method over random seeding.
3. Lloyd's algorithm takes the number of clusters k as input. Suppose we don't know the number of clusters in advance. Describe an approach for choosing the number of clusters. State and justify your choice of the value of k.
4. (Optional) Suggest alternate approaches to clustering trajectories in $\mathscr{T}$.

C.2 Source code

1. ***Programming languages***: All groups must use Java or Python, similar to Part 1.
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.

D High-level Rubric

	Excellent (100%)	Satisfactory (75%)	Unsatisfactory (50%)	Incomplete (25%)
Task 4 (100) / **Task 5** (100)	Good model, correct algorithm, Efficient implementation	Reasonably good model, Correct algorithm, possibly inefficient implementation	Deficiencies in the model and/or the algorithm or highly inefficient implementation	Incomplete attempt
Slides (25)	Include all the experimental results, design descriptions from Section C.1, and individual contributions of each group member; clearly labeled figures, easy-to-follow descriptions	Some of the details missing, difficult-to-interpret results	Ambiguous algorithm description or any of the experimental results	Missing major components
Code (25)	Reproducible, readable, and well-commented code; no use of external dependencies except for visualization; `README` file describing files in the archive and member names	Reproducible code that lacks comments/unreadable code; no use of external dependencies except for visualization	Use of external dependencies beyond visualization or non-reproducible code	Non-reproducible code with missing components

E 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.
- **Size of a trajectory:** Size m of a trajectory $T\in\mathscr{T}$ such that $T=\langle p_1,\ldots,p_m\rangle$ is the number of vertices in T. We denote the size of T using $|T|$.
- **Trajectory distance:** For two trajectories T,T' in $\mathscr{T}$, $\Delta(T,T')\in\mathbb{R}^+$ gives a measure of distance between them. Dynamic Time Warping (dtw) implemented in Part I, is an example of a trajectory distance functions.

- **Center trajectory:** The center trajectory of $\mathscr{T}$ is a trajectory $\bar{T}$, denoted by $\bar{T}^i$, that "best" represents the trajectories $\mathscr{T}$ with respect to Δ.

- **Clustering:** A *k-clustering* $\mathscr{C} = \{\mathscr{T}_1, \ldots, \mathscr{T}_k\}$ of $\mathscr{T}$ is a partition of $\mathscr{T}$ into k pairwise-disjoint subsets (clusters), i.e., each trajectory of $\mathscr{T}$ appears in exactly one cluster.

- **Cost of a partition:** We write the cost of a partition as the sum distance of its trajectories from the given center trajectory $\bar{T}$. Formally, the cost of a partition $\mathscr{T}_i$ is: $\mathsf{cost}(\mathscr{T}_i, \bar{T}^i) = \sum_{T' \in \mathscr{T}_i} \Delta(\bar{T}^i, T')$.

- **Cost of clustering:** Cost of clustering $\mathscr{C}$ is the sum of costs of all partitions in $\mathscr{C}$. Formally, $\mathsf{cost}(\mathscr{C}) = \sum_{i=1}^{k} \mathsf{cost}(\mathscr{T}_i, \bar{T}^i)$.

- **Seed:** The initial partitions used for clustering.