Clustering Moving Entities in Euclidean Space

Stephane Durocher[1]
University of Manitoba, Winnipeg, Canada
durocher@cs.umanitoba.ca

Md Yeakub Hassan
University of Manitoba, Winnipeg, Canada
hassanmy@cs.umanitoba.ca

Abstract

Clustering is a fundamental problem of spatio-temporal data analysis. Given a set $\mathcal{X}$ of n moving entities, each of which corresponds to a sequence of τ time-stamped points in $\mathbb{R}^d$, a *k-clustering* of $\mathcal{X}$ is a partition of $\mathcal{X}$ into k disjoint subsets that optimizes a given objective function. In this paper, we consider two clustering problems, k-Center and k-MM, where the goal is to minimize the maximum value of the objective function over the duration of motion for the worst-case input $\mathcal{X}$. We show that both problems are NP-hard when k is an arbitrary input parameter, even when the motion is restricted to $\mathbb{R}$. We provide an exact algorithm for the 2-MM clustering problem in $\mathbb{R}^d$ that runs in $O(\tau d n^2)$ time. The running time can be improved to $O(\tau n \log n)$ when the motion is restricted to $\mathbb{R}$. We show that the 2-Center clustering problem is NP-hard in $\mathbb{R}^2$. Our 2-MM clustering algorithm provides a 1.15-approximate solution to the 2-Center clustering problem in $\mathbb{R}^2$. Moreover, finding a $(1.15 - \epsilon)$-approximate solution remains NP-hard for any $\epsilon > 0$. For both the k-MM and k-Center clustering problems in $\mathbb{R}^d$, we provide a 2-approximation algorithm that runs in $O(\tau d n k)$ time.

2012 ACM Subject Classification Theory of computation → Facility location and clustering

Keywords and phrases trajectories, clustering, moving entities, k-CENTER, algorithms

Digital Object Identifier 10.4230/LIPIcs.SWAT.2020.22

Funding This research was supported in part by the Natural Sciences and Engineering Research Council of Canada.

1 Introduction

The technology of tracking moving entities (e.g., humans, animals, vehicles, etc.) using GPS, motion capture, and other location-tracking devices has developed rapidly and has become widely adopted across a range of applications in the past decade. Tracking devices record the location of a moving entity over time, corresponding to a sequence of time-stamped spatial coordinates. In many cases, the positions of multiple moving entities are tracked simultaneously. In this paper, we consider a set of entities moving continuously in Euclidean space, such that at any given time, the position of each entity is represented by a point in $\mathbb{R}^d$. Several applications (e.g., animal migration analysis, weather forecasting, and component classification from motion capture data) require clustering moving entities [16]. The goal of clustering is to partition a given set of objects into groups of similar objects, where the degree of similarity, i.e., the quality of the clustering, is measured according to a given objective function. Some previous work on clustering moving entities has examined the problem of updating clusters over time to maintain a good clustering [7, 9, 14, 16, 18]. In these types of clustering, an entity is often required to switch from one cluster to another to maintain the

[1] This research was completed while Stephane Durocher was a visiting researcher with the Morpheo research group at INRIA Grenoble Rhône Alpes in Grenoble, France.

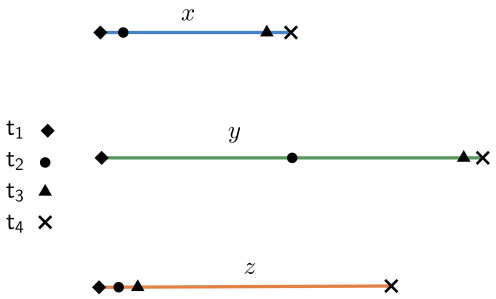

Figure 1 Trajectories of three entities x, y, z in $\mathbb{R}^2$ within time stamps t_1 to t_4.

optimality of the objective function. These types of clustering are not suitable for applications that require an entity to remain in a cluster while optimizing a global objective over the entire motion. For example, consider an input consisting of trajectories of a set of sensors recorded during the motion capture of a person running, for which the goal is to identify different body parts during the motion, e.g., torso, arms, legs, etc. If we allow entities (in this case, sensors) to change clusters over time, then one part of the body (one trajectory) might be classified into one cluster (say an arm) at one time and another cluster (say a leg) later during the same motion. To prevent such inconsistencies, we examine the problem of clustering a set of moving entities where the moving entities cannot switch clusters over time. In particular, the basic unit of clustering is the whole trajectory of each moving entity.

The discrete sequence of time-stamped locations of an entity can be linearly interpolated to obtain a continuous piecewise-linear curve. Depending on the application, we may need to cluster just the curves (e.g., when clustering animal migration routes followed by multiple herds or flocks of animals, possibly at different times and speeds) or the curves along with their associated time stamps (e.g., when clustering animals into groups that moved together during migration). The k-Center clustering of trajectories of moving entities has been studied under the Fréchet distance metric [5, 6]. This type of clustering considers only the piecewise-linear curves of each entity, without factoring the time at which moving entities occupied points on their trajectory. Fréchet distance considers all possible walk sequences between two trajectories. Therefore, input consisting of position as a function of time leads to different clusters than when the input consists of trajectories only. For example, consider three entities x, y and z moving in $\mathbb{R}^2$ whose trajectories from time t_1 to t_4 are given in Figure 1. The Fréchet distance between the trajectories of x and y is greater than the Fréchet distance between the trajectories of y and z. Therefore, if we partition $\{x, y, z\}$ into two clusters to minimize the maximum intra-cluster Fréchet distance, then y and z would be in one cluster, and x would be in another cluster. However, the Euclidean distance between y and z at time t_3 is greater than the distance between x and y at any time. When time is considered, the resulting clustering assigns y and z to different clusters. As a result, when time is important, clustering should optimize the objective function based on the position of entities over the entire motion, i.e., at all times during the motion. In this paper, we examine clustering algorithms for moving entities in Euclidean space where each entity's trajectory has associated time stamps. In particular, we consider k-MM and k-Center clustering of moving entities, where k specifies the number of clusters.

Problem description

We are given a set $\mathcal{X}$ of n entities moving in $\mathbb{R}^d$ that are tracked over a given time interval. The spatial coordinates of entities are recorded at a set T of τ discrete time stamps. Therefore, the location of each entity over time traces a piecewise-linear trajectory in $\mathbb{R}^d$ consisting of τ vertices. At any time $t \in T$, the position of each entity $x_i \in \mathcal{X}$ is a fixed point in $\mathbb{R}^d$, denoted $x_i(t)$. For any time $t \in T$, $\mathcal{X}(t) = \{x(t) \mid x \in \mathcal{X}\}$ denotes the multiset of points corresponding to the positions of the entities in $\mathcal{X}$ at time t. Similarly, for a cluster $C_i \subseteq \mathcal{X}$, $C_i(t)$ denotes the multiset of points corresponding to the positions of entities in C_i at time t.

▶ **Definition 1.** *Given a set T of τ discrete time stamps and a finite set $\mathcal{X}$ of n moving entities in $\mathbb{R}^d$, the goal of the **k-clustering** problem is to partition $\mathcal{X}$ into k disjoint sets $C_1, C_2, \ldots, C_k$ to optimize a given objective function $f : \{C_1, C_2, \ldots, C_k\} \to \mathbb{R}$. The goal is to minimize the maximum value of $f(t)$ over all $t \in T$ for the worst-case input $\mathcal{X}$.*

Kinetic clustering refers to clustering problems on moving or dynamic entities for which entities may switch clusters over time; *static clustering* refers to clustering problems on moving or dynamic entities for which each entity is assigned to a single cluster. Note that static facility location does not imply static entities, i.e., entity positions can change over time in static clustering, but their assignment to clusters cannot. This paper examines the static setting for the problem of clustering moving entities.

The particular objective function depends on the applications of the problem. Common clustering problems are k-Center, k-median, k-MM and k-means clustering. In this paper, we present the results for the k-MM and k-Center clustering of moving entities in $\mathbb{R}^d$. In the k-MM clustering problem, the objective function f measures the maximum distance during the motion between any two entities in any cluster C_i. The "MM" in the objective function refers to "max max". The objective of the k-MM clustering is to find a partition $C_1, \ldots, C_k$ of $\mathcal{X}$ that minimizes

$$\max_{i \in \{1,\ldots,k\}} \max_{\{x,y\} \subseteq C_i} \max_{t \in T} \delta(x(t), y(t)), \tag{1}$$

where δ is the Euclidean (ℓ_2) distance metric. For any set of points in $\mathbb{R}^2$, the k-MM clustering is often denoted k-2MM, where "2" refers to Euclidean (ℓ_2) distance. The k-2MM problem is NP-hard in $\mathbb{R}^2$ when k is an arbitrary input parameter [12]. In the k-Center clustering problem, the objective function f measures the maximum distance during the motion from any $x \in \mathcal{X}$ to its cluster center, where the center of each cluster C_i at time t is the center of the smallest enclosing d-ball of $C_i(t)$. Therefore, the objective is to find a partition $C_1, \ldots, C_k$ of $\mathcal{X}$ that minimizes

$$\max_{i \in \{1,\ldots,k\}} \max_{x \in C_i} \max_{t \in T} \delta(x(t), m_i(t)), \tag{2}$$

where $m_i(t)$ denotes the center of the smallest enclosing d-ball of $C_i(t)$. That minimum value of Equation 2 corresponds to the smallest radius r such that each cluster can be covered at all times by a d-ball of radius r. We consider the *continuous* k-Center clustering problem, where the center of a cluster can be placed anywhere in Euclidean space (i.e., the cluster center is not restricted to $\mathcal{X}(t)$, as is the case in discrete facility location). The k-Center clustering problem is also NP-hard in $\mathbb{R}^2$ when k is an arbitrary input parameter [19].

Results and organization.

In this paper, we study k-MM and k-Center clustering of moving entities in $\mathbb{R}^d$. In Section 2, we discuss related work on clustering moving entities. In Section 3, we present algorithms

that solve the 2-MM clustering problem optimally in $O(\tau dn^2)$ time in $\mathbb{R}^d$, and in $O(\tau n \log n)$ time in $\mathbb{R}$. In Section 4, we show that 2-Center clustering is NP-hard for entities moving in $\mathbb{R}^2$; this differs from the 2-Center problem on a set of points in $\mathbb{R}^d$, which can be solved in polynomial time [2]. The 2-MM clustering algorithm gives an 1.15-approximate solution for the 2-Center clustering problem in $\mathbb{R}^2$. Moreover, computing a $(1.15-\epsilon)$-approximation remains NP-hard for any $\epsilon > 0$. In Section 5, we show that the k-MM and k-Center clustering problems are NP-hard when k is an arbitrary input parameter, even in $\mathbb{R}$; again, this differs from the k-MM and k-Center problems on a set of points, which can be solved in polynomial time in $\mathbb{R}$ [11]. We provide a 2-approximation algorithm for both problems by using the greedy technique of Gonzalez's algorithm [12]. Finally, in Section 6, we conclude with a discussion of possible directions for future research.

2 Related work

The k-Center and k-MM clustering problems have been studied extensively for sets of points in $\mathbb{R}^d$ [1, 12]. Both problems are NP-hard in $\mathbb{R}^2$ when k is an arbitrary input parameter [12, 19]. Gonzalez [12] provides a 2-approximation algorithm for both k-MM and k-Center clustering of a set of points in $\mathbb{R}^2$ which requires $O(nk)$ time. However, the algorithm works for any metric space. Frederickson [11] gives an algorithm for the k-Center problem on a tree that uses $O(n)$ time and space. The algorithm allows centers to be located at any points on the edges. Therefore, the k-Center problem for any set of points in $\mathbb{R}$ can be solved in $O(n)$ time using Frederickson's algorithm. The 2-Center clustering of a set of points in $\mathbb{R}^2$ can be solved exactly in $O(n \log^2 n)$ time [10, 21].

Har-Peled [13] examines the k-Center problem for sets of moving entities in $\mathbb{R}^d$, giving an algorithm that partitions the input into $k^{\mu+1}$ clusters for entities whose motion has algebraic degree μ. The 1-Center, 2-Center, and k-Center clustering problems have been studied for sets of moving entities in the kinetic setting, where entities may switch clusters, with the goal to maintain a good approximation of the objective function while bounding the relative velocity of the cluster centers [3, 7, 9, 8]. Variants of these problems have been studied, such as k-clustering to minimize the average value of the objective function throughout the motion (e.g., [14]). Li et al. [17] examine a movement pattern of moving entities, called swarm, where a group of entities moves together for a set of (possibly nonconsecutive) timestamps.

Previous work also examines clustering trajectories of moving entities under different similarity measures: Fréchet distance, dynamic time warping, and edit distance [22]. Lee et al. [15] propose a two-phase algorithm to cluster similar portions of a set of trajectories. Buchin et al. [4] provide a trajectory grouping framework where the structure of a group of similar trajectories is represented by a Reeb graph. Driemel et al. [6] introduce the (k,l)-Center clustering problem for a set of curves in $\mathbb{R}$ under the Fréchet distance metric, where each of the k-Center curves has complexity at most l. A $(1+\epsilon)$-approximation algorithm is proposed for this problem, which runs in near-linear time. Buchin et al. [5] show that (k,l)-Center clustering under Fréchet distance is NP-hard even if $k = 1$.

3 Algorithms for 2-MM clustering

Given a set $\mathcal{X}$ of n moving entities in $\mathbb{R}^d$, we can construct a weighted undirected complete graph G, where the vertices of G are the moving entities in $\mathcal{X} = \{x_1, x_2, \ldots, x_n\}$ and the weight of the edge between two vertices x_i and x_j is

$$\max_{t \in T} \delta(x_i(t), x_j(t)). \tag{3}$$

▶ **Lemma 2.** *The weights of graph G satisfies the triangular inequality.*

Proof. Let x, y, and z be three moving entities in $\mathbb{R}^d$. At any time t, the positions of these entities satisfy triangle inequality (i.e., $\delta(x(t), y(t)) + \delta(y(t), z(t)) \geq \delta(z(t), x(t))$). For this instance, the graph G has three vertices $\{x, y, z\}$ and three edges $\{e_1(x, y), e_2(y, z), e_3(z, x)\}$. The weights of the edges are

$$e_1 = \delta(x(t_1), y(t_1)); \quad e_2 = \delta(y(t_2), z(t_2)); \quad e_3 = \delta(z(t_3), x(t_3));$$

where, $t_1 = \arg\max_t \delta(x(t), y(t))$; $t_2 = \arg\max_t \delta(y(t), z(t))$; $t_3 = \arg\max_t \delta(z(t), x(t))$. From the weights of the edges we get,

$$\begin{aligned} e_1 + e_2 &= \delta(x(t_1), y(t_1)) + \delta(y(t_2), z(t_2)) \\ &\geq \delta(x(t_3), y(t_3)) + \delta(y(t_3), z(t_3)); \qquad \text{(from the definition of } t_1, t_2 \text{ and } t_3\text{)} \\ &\geq \delta(x(t_3), z(t_3)); \quad \text{(since x, y, z satisfy the triangle inequality at time } t_3\text{)} \\ &\geq e_3 \end{aligned}$$

◀

We can solve the 2-MM clustering of $\mathcal{X}$ by partitioning G into two clusters C_1, C_2 such that the partitioning minimizes

$$\max_{i \in \{1,2\}} \max_{\{x,y\} \subseteq C_i, x \neq y} w(x, y), \tag{4}$$

where $w(x, y)$ is the weight of the edge between x and y. In other words, we want to minimize the maximum intra-cluster edge weight, where an edge having both end vertices in the same cluster is called an intra-cluster edge. To partition the graph G, we first compute a tree T that is a Maximum Spanning Tree of G using Prim's algorithm. Proceed to 2-color T by alternatively coloring vertices red and blue in a breadth-first search. This coloring defines the two clusters, C_1 and C_2.

▶ **Theorem 3.** *C_1 and C_2 define an optimal 2-MM clustering.*

Proof. The algorithm constructs a tree T, i.e., a bipartite graph whose vertices are divided into two disjoint and independent sets C_1 and C_2. Suppose x and y are two vertices in C_1 (or C_2), such that the edge (x, y) in G is a maximum-weight intra-cluster edge (outcome of Equation 4). Since T is bipartite, there exists a path from x to y with an even number of edges. Adding the edge (x, y) to T would create a cycle in T. Since T is a maximum spanning tree of G, each edge in that cycle has a weight at least $w(x, y)$. As a result, there exists at least one vertex z_1 in C_2 (or C_1) having $w(x, z_1) \geq w(x, y)$, and there exists at least one vertex z_2 in C_2 (or C_1) having $w(y, z_2) \geq w(x, y)$. If there exists an edge between (x, z_2) in T, then $w(x, z_2) \geq w(x, y)$ (since x, z_2, y creates a cycle in T). If there is no edge between (x, z_2) in T, then adding the edge (x, z_2) to T would create a cycle where $w(x, z_2) \geq w(y, z_2)$. Since $w(y, z_2) \geq w(x, y)$, we can say that $w(x, z_2) \geq w(x, y)$. Similarly, we can say that $w(y, z_1) \geq w(x, y)$.

In every possible 2-clustering of G where x and y are in different cluster, there is an intra-cluster edge that has weight at least $w(x, y)$ (because z_1 or z_2 can be in any of the two clusters). On the other hand, when x and y are in the same cluster, every possible 2-clustering of G has an intra-cluster edge having weight $w(x, y)$. This proves that, in every possible 2-clustering of G, there exists an intra-cluster edge which has weight at least $w(x, y)$. Therefore, G cannot be partitioned into two clusters where the maximum intra-cluster edge has weight less than $w(x, y)$. Hence, the clustering C_1, C_2 is optimal. ◀

We have n moving entities in $\mathbb{R}^d$, and the pairwise distance of all pair of entities using Equation 3 can be computed in $O(\tau dn^2)$ time. Thus, constructing the graph G requires $O(\tau dn^2)$ time. The running time of Prim's algorithm is $O(n^2)$. Since, the tree T has $n-1$ edges, the breadth-first search in the 2-coloring algorithm requires $\Theta(n)$ time. Therefore, the running time of the 2-MM clustering algorithm is $O(\tau dn^2)$.

▶ **Theorem 4.** *The 2-MM clustering of moving entities in $\mathbb{R}^d$ can be computed in $O(\tau dn^2)$ time.*

3.1 Improved algorithm for entities in $\mathbb{R}$

We consider a set $\mathcal{X}$ of n moving entities in $\mathbb{R}$. We plot the entities' position functions over time, such that horizontal axis represents time and the vertical axis represents the positions of the entities.

▶ **Definition 5.** *The **upper trajectory** of $\mathcal{X}$, denoted $U_{\mathcal{X}}$, is a piecewise-linear trajectory corresponding to the upper envelope of the plots of trajectories of entities in $\mathcal{X}$. That is, at any time $t \in T$, the position of the upper trajectory $U_{\mathcal{X}}(t)$ is $\max\{x(t) | x \in \mathcal{X}\}$. The **lower trajectory** of $\mathcal{X}$, denoted $L_{\mathcal{X}}$, is defined analogously with respect to the lower envelope of the plots of the trajectories of entities in $\mathcal{X}$.*

To compute $U_{\mathcal{X}}$ and $L_{\mathcal{X}}$, at each time t, we need to find the maximum and minimum value of $\mathcal{X}(t)$. Therefore, $U_{\mathcal{X}}$ and $L_{\mathcal{X}}$ can be computed in $\Theta(\tau n)$ time. Examples of the upper and lower trajectories of six entities are shown in Figure 2.

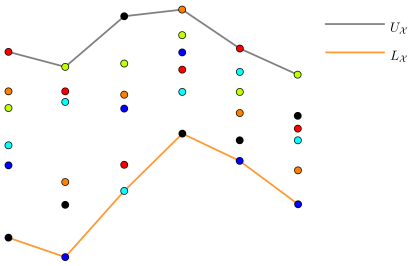

Figure 2 Upper and Lower trajectories of six entities moving in $\mathbb{R}$.

▶ **Observation 6.** *The upper and lower trajectories of $\mathcal{X}$ can be computed in $\Theta(\tau n)$ time.*

We can find the furthest pair of entities in $\mathcal{X}$ by using $U_{\mathcal{X}}$ and $L_{\mathcal{X}}$. A pair $(x, y) \in \mathcal{X}$ is called the furthest pair if

$$\max_{t \in T} \delta(x(t), y(t)) \geq \max_{\substack{\{x', y'\} \subseteq \mathcal{X}, \\ x' \neq x, y' \neq y}} \max_{t \in T} \delta(x'(t), y'(t)).$$

At each time stamp t, we calculate the distance between entities $x(t) \in U_{\mathcal{X}}(t)$ and $y(t) \in L_{\mathcal{X}}(t)$. The furthest pair is the one which has maximum distance over all time.

▶ **Observation 7.** *Given the upper and lower trajectories of $\mathcal{X}$, the furthest pair of entities in $\mathcal{X}$ can be found in $\Theta(\tau)$ time.*

At every time stamp, we consider maintaining the sorted order of the entities where they are sorted according to their positions. This takes $O(\tau n \log n)$ time. If an entity is added to or removed from $\mathcal{X}$, then $U_{\mathcal{X}}$ and $L_{\mathcal{X}}$ need to be updated as well. Consider a new entity x that is added to $\mathcal{X}$. To maintain the sorted order, the insertion of x would take $O(\tau \log n)$ time by using a balanced search tree. After that, we can update the upper and lower trajectories of $\mathcal{X}$ in $\Theta(\tau)$ time. For each time t, if $x(t) \geq U_{\mathcal{X}}(t)$, then we update the value of $U_{\mathcal{X}}(t)$ to $x(t)$. Similarly, if $x(t) \leq L_{\mathcal{X}}(t)$, then we update the value of $L_{\mathcal{X}}(t)$ to $x(t)$. If an entity x is deleted from $\mathcal{X}$, we can update the sorted order of $\mathcal{X}$ in $O(\tau)$ time. $U_{\mathcal{X}}$ and $L_{\mathcal{X}}$ can also be updated in $\Theta(\tau)$ time. For each time t, if $x(t) = U_{\mathcal{X}}(t)$, we update the value of $U_{\mathcal{X}}(t)$ to $y(t)$, where $y \in \mathcal{X}$ and $y(t)$ is the closest neighbor of $x(t)$. Similarly, we can also update the lower trajectory of $\mathcal{X}$.

Consider two sets $\mathcal{X}_1$ and $\mathcal{X}_2$ of moving entities in $\mathbb{R}$. We want to find an entity $x \in \mathcal{X}_1$ that is furthest away from all entities in $\mathcal{X}_2$. Therefore, $x \in \mathcal{X}_1$ is the entity that maximizes

$$\max_{y \in \mathcal{X}_2} \max_{t \in T} \delta(x(t), y(t)).$$

After computing the upper and lower trajectories of $\mathcal{X}_1$ and $\mathcal{X}_2$, the farthest entity can be found in $\Theta(\tau)$ time. At each time t, we need to find the furthest entity $x(t) \in \mathcal{X}_1(t)$ by calculating the maximum of $\{\delta(U_{\mathcal{X}_1}(t), U_{\mathcal{X}_2}(t))$, $\delta(U_{\mathcal{X}_1}(t), L_{\mathcal{X}_2}(t))$, $\delta(L_{\mathcal{X}_1}(t), U_{\mathcal{X}_2}(t))$, $\delta(L_{\mathcal{X}_1}(t), L_{\mathcal{X}_2}(t))\}$.

▶ **Observation 8.** *Given two sets $\mathcal{X}_1$ and $\mathcal{X}_2$ of moving entities in $\mathbb{R}$ and their upper and lower trajectories, the furthest entity $x \in \mathcal{X}_1$ from all entities in $\mathcal{X}_2$ can be computed in $\Theta(\tau)$ time.*

Unlike the 2-MM clustering algorithm in $\mathbb{R}^d$, we do not need to construct the graph G for the 2-MM clustering in $\mathbb{R}$. In this case, we construct a graph G', which is equivalent to the maximum spanning tree of G. We use the idea of Prim's algorithm to construct G'. At every time stamp, we maintain the sorted order of the entities. We then use the following approach to find the 2-MM clustering of $\mathcal{X}$:

1. Calculate $U_{\mathcal{X}}$ and $L_{\mathcal{X}}$.
2. Find the furthest pair (x_1, x_2) of entities in $\mathcal{X}$.
3. Let G' be a graph with two vertices (entities) x_1, x_2.
4. Add an edge between x_1 and x_2.
5. Let $\mathcal{Y} = \{x_1, x_2\}$. Remove x_1 and x_2 from $\mathcal{X}$.
6. Update $U_{\mathcal{X}}$, $L_{\mathcal{X}}$, $U_{\mathcal{Y}}$ and $L_{\mathcal{Y}}$.
7. For $i = 1$ to $n - 2$:
   - Find an entity $x \in \mathcal{X}$ that is farthest away from all entities in $\mathcal{Y}$. Let $y \in \mathcal{Y}$ be the entity for which x becomes the furthest from $\mathcal{Y}$.
   - Add x to $\mathcal{Y}$ and remove x from $\mathcal{X}$. Then, update $U_{\mathcal{X}}$, $L_{\mathcal{X}}$, $U_{\mathcal{Y}}$ and $L_{\mathcal{Y}}$.
   - Add a vertex x to graph G'. Since, y is already added to G', add an edge between y and x.
8. The graph G' is a bipartite graph; run the 2-coloring algorithm in graph G'.

Each vertex of G' represents a moving entity in $\mathbb{R}$. The algorithm initially adds the furthest pair of entities in G' and grows G' by one edge in each iteration. Similar to Prim's algorithm, in each iteration, the algorithm finds an entity that is furthest away from all entities (vertices) in G', and then adds that entity to G'. Therefore the graph G' is equivalent to the maximum spanning tree of the graph G. After constructing G', we apply the 2-coloring algorithm (similar to the 2-MM algorithm in $\mathbb{R}^d$) to find the clusters C_1 and C_2.

▶ **Theorem 9.** *The 2-MM clustering of moving entities in $\mathbb{R}$ can be computed in $O(\tau n \log n)$ time.*

Proof. The sorting operations in all time stamps take total $O(\tau n \log n)$ time. Initially, we compute the upper and lower trajectories in $\Theta(\tau n)$ time. In each iteration, we find the furthest entity from $\mathcal{X}$, and then update the upper and lower trajectories of $\mathcal{X}$ and $\mathcal{Y}$. Since we are maintaining the sorted order of entities in all time stamps, the insert operation in $\mathcal{Y}$ would take $O(\tau \log n)$ time. The upper and lower trajectories of $\mathcal{X}$ and $\mathcal{Y}$ are updated in $\Theta(\tau)$ time. Therefore, constructing G' requires $O(\tau n \log n)$ time. The 2-coloring algorithm requires $\Theta(n)$ time. Thus, the overall running time of the algorithm is $O(\tau n \log n)$. ◀

4 Hardness results of 2-Center clustering

Given any set P of points in $\mathbb{R}$, the diameter of the smallest enclosing circle of P and the maximum possible distance between any two points in P are equal. Therefore, the k-Center and k-MM clustering of P are equivalent. This result also holds for the k-Center and k-MM clustering of a set of moving entities in $\mathbb{R}$. However, the two problems have different solutions in general when entities move in higher dimensions (see Example 11 below).

▶ **Observation 10.** *If $\mathcal{X}$ is a set of moving entities in $\mathbb{R}$, then the k-Center and the k-MM clustering of $\mathcal{X}$ are equivalent.*

▶ **Example 11.** Consider four points $a(0, 33), b(0, -33), c(-56, 0), d(-124, 0)$ in $\mathbb{R}^2$ (see Figure 3). In exact 2-Center clustering, the radius of any cluster would be at most 34 by keeping $\{a, b\}$ in one cluster and $\{c, d\}$ in another cluster. Because the smallest enclosing circle containing any three points among $\{a, b, c, d\}$ would have radius at least 37.7 (the radius of circle passing through $\{a, b, c\}$). However, the exact 2-MM clustering would keep $\{a, b, c\}$ in one cluster, leaving $\{d\}$ for the other cluster, since the maximum distance between any two points in $\{a, b, c\}$ is at most 66.

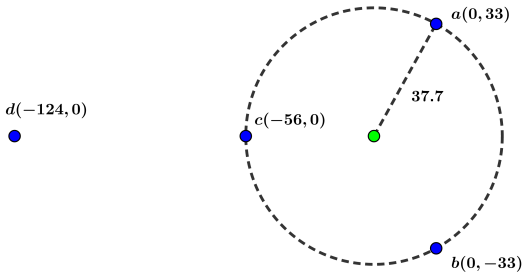

Figure 3 Illustration in support of Example 11.

As we now show, computing the 2-Center clustering of a set $\mathcal{X}$ of n moving entities in $\mathbb{R}^2$ is NP-hard. This differs from the problem of computing a 2-Center of a set of points in $\mathbb{R}^2$ which can be solved in $O(n \log^2 n)$ time [21]. Given a set $\mathcal{X}$ of n moving entities and a real parameter r^*, the decision version of the 2-Center clustering problem asks to determine whether $\mathcal{X}$ can be partitioned into two sets C_1 and C_2, such that each cluster can be covered by a disc of radius r^* at all times. We prove that the decision version of 2-Center clustering in $\mathbb{R}^2$ is NP-complete. We obtain the result by reduction from the Monotone Not-All-Equal

3SAT (NAE-3SAT) problem. The Monotone NAE-3SAT is one of many variants of the 3SAT problem, which is NP-Complete [20]. The Monotone NAE-3SAT problem consists of a set of Boolean variables and a set of monotone clauses, i.e., variables in each clause are never negated. Unlike the 3SAT problem, each clause of the NAE3SAT problem requires to have at least one true and one false Boolean value.

▶ **Theorem 12.** *The decision version of the* 2-Center *clustering problem in* $\mathbb{R}^2$ *is NP-complete.*

Proof. Choose any instance of the Monotone NAE-3SAT problem, let $\mathcal{S}$ denote its set of Boolean variables, and let $n = |\mathcal{S}|$. We describe how to construct a corresponding instance of the decision version of the 2-Center clustering problem with $r^* = 1$. Each element in $\mathcal{S}$ is mapped to a corresponding moving entity in $\mathbb{R}^2$, and the number of time stamps is equal to the number of clauses.

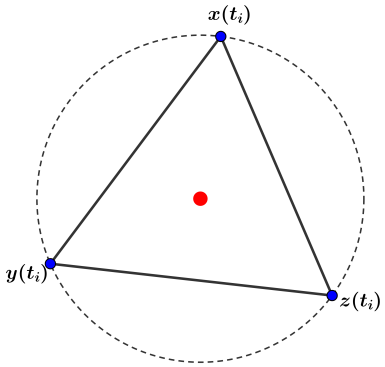

Figure 4 Blue points represent the positions of entities x, y, z at time t_i. All entities other than x, y, and z are placed at the red point. The circumradius of the triangle is $1 + \epsilon$, for some small $\epsilon > 0$.

At each time t_i, we take the i^{th} clause from the instance of the Monotone NAE-3SAT problem and assign coordinates to the moving entities in the plane as follows. Let $x, y, z \in \mathcal{S}$ be the three entities from the clause and let $x(t_i), y(t_i), z(t_i)$ denote their respective positions at time t_i. We place $x(t_i), y(t_i), z(t_i)$ at the vertices of an equilateral triangle such that the circumradius of the triangle is slightly larger than one; thus, x, y, and z cannot be covered by a unit-radius disc at time t_i. The remaining $n - 3$ entities are placed at the circumcenter of the triangle. This transformation can be done in $O(n)$ time. Each cluster in the partitioning of $\mathcal{S}$ can be covered by a unit-radius disc if and only if the three entities x, y, z are not in the same cluster. In the instance of the Monotone NAE-3SAT problem, the values of three variables in each clause cannot be equal to each other. Similarly, the corresponding three entities from each clause cannot be in the same cluster if the instance is a "yes" instance of the decision version of the 2-Center clustering problem. Therefore, the instance of the Monotone NAE-3SAT problem is satisfiable if and only if the corresponding instance of the decision version of 2-Center clustering problem is a "yes" instance. ◀

We can achieve a 1.15-approximate solution for the 2-Center clustering of moving entities in $\mathbb{R}^2$ by applying the 2-MM clustering algorithm from Section 3. We also prove that no polynomial-time algorithm can achieve a better approximation ratio. In what follows, we prove the approximation ratio and its lower bound.

▶ **Theorem 13.** *The exact* 2*-MM clustering of moving entities in* $\mathbb{R}^2$ *gives a* 1.15*-approximate solution for the* 2-Center *clustering problem.*

Proof. We partition $\mathcal{X}$ into two clusters C_1 and C_2 by applying the algorithm described in Section 3. Let l be the maximum distance over all time between any pair of entities from the same cluster (C_1 or C_2). As argued in the proof of Theorem 3, in every possible 2-clustering of $\mathcal{X}$, there exists a pair of entities in some cluster whose maximum distance over all time is at least l. By Jung's theorem, if the largest possible distance between two points from a finite set of points in the plane is l, then there exists a circle enclosing all these points with a radius no greater than $l/\sqrt{3}$. Hence, in any 2-MM clustering of $\mathcal{X}$, the maximum radius of any cluster would be at most $l/\sqrt{3}$. Since every possible 2-clustering of $\mathcal{X}$ consists a pair of entities in any cluster whose maximum distance over all time is at least l, the maximum radius of any cluster in exact 2-Center clustering of $\mathcal{X}$ would be at least $l/2$. Thus, the exact 2-MM clustering gives a $2/\sqrt{3}$-approximation (or 1.15-approximation) for the 2-Center clustering problem. ◀

▶ **Theorem 14.** *If $P \neq NP$, no polynomial-time algorithm can achieve a $(1.15 - \epsilon)$-approximate solution for the 2-Center clustering of moving entities in $\mathbb{R}^2$ for any $\epsilon > 0$.*

Proof. Choose any instance of the Monotone NAE-3SAT problem, let $\mathcal{S}$ denote its set of Boolean variables, and let $n = |\mathcal{S}|$. We show how to construct a corresponding instance of the 2-Center problem in $\mathbb{R}^2$. Each Boolean variable is mapped to a moving entity in $\mathbb{R}^2$. At each time t_i, the three entities in the i^{th} clause from the instance of the Monotone NAE-3SAT problem are placed at the vertices of an equilateral triangle where the length of each side of the triangle is 2. The remaining entities are placed at the circumcenter of the triangle. If we can partition $\mathcal{S}$ into two clusters $\{C_1, C_2\}$ such that all three entities in each clause are split by this partition, then the maximum radius of any partition over all time would be 1. For any $\epsilon > 0$, a $(1.15 - \epsilon)$-approximation algorithm for the 2-Center clustering of $\mathcal{S}$ will give us two clusters whose maximum radius over all time would be less than 1.15, in this case, it has to be 1.

We consider the value of a Boolean variable in $\mathcal{S}$ is true if the corresponding entity is in cluster C_1 in the 2-Center clustering of $\mathcal{S}$. Similarly, the value of a variable is false if the corresponding entity is in cluster C_2. The instance of the Monotone NAE-3SAT problem is satisfiable if and only if all the entities in each clause are not in the same cluster. This can only be done by a $(1.15 - \epsilon)$-approximation algorithm for 2-Center clustering of $\mathcal{S}$. Therefore, a polynomial-time $(1.15 - \epsilon)$-approximation algorithm for 2-Center clustering of $\mathcal{S}$ can be used to decide the satisfiability of the corresponding instance of the Monotone NAE-3SAT problem. This cannot happen assuming $P \neq NP$. ◀

5 k-MM and k-Center clustering

The k-MM and k-Center clustering problems are solvable in polynomial-time for any set of fixed points in $\mathbb{R}$, i.e., without motion [7]. As we now show, both problems are NP-hard for entities moving in $\mathbb{R}$. We reduce an instance of a restricted version of the k-gMM clustering problem to an instance of the k-MM clustering problem for moving entities in $\mathbb{R}$. k-gMM is the more general version of the geometric k-MM problem on arbitrarily weighted graphs; any instance of the k-MM problem directly corresponds to an instance of the k-gMM problem.

Given an arbitrary weighted graph $G = (V, E)$, the goal of the k-gMM clustering problem is to partition V into k disjoint sets (k clusters) such that the maximum weight of any intra-cluster edge is minimized. The decision version of the k-gMM problem has been proved to be NP-complete [12]. The result is obtained by a reduction from a restricted version of the exact cover by 3-sets problem. Given an instance of the restricted version of the exact

cover by 3-sets problem, Gonzalez constructed a weighted complete graph as an instance of the decision version of k-gMM problem, where the weight of each edge is either one or two. Therefore, the decision version of k-gMM problem remains NP-complete even when the input graph is a weighted complete graph with edge weights either one or two. We call this problem the restricted k-gMM problem which we use in our reduction.

▶ **Theorem 15.** *Given a set of moving entities in $\mathbb{R}^d$, for any $d \geq 1$, the k-MM and k-Center clustering problems are NP-hard when k is an arbitrary input parameter.*

Proof. We consider an instance of the restricted k-gMM clustering problem, where we are given a weighted complete graph G with edge weights in $\{1, 2\}$. We construct a corresponding instance of the k-MM clustering problem. Each vertex of G is mapped to a moving entity in $\mathbb{R}$. The number of time stamps and the number of edges in G are equal. For each time t_i, we take the ith edge e_i from G and assign coordinates to its two end vertices x, y (two entities) on the real line such that the Euclidean distance between them is equal to the weight of that edge. At that time, t_i, the remaining entities are positioned at the midpoint of the line segment between x and y. Since the minimum and maximum weight of an edge in G is one and two, respectively, the maximum distance between any two entities over all time is equal to the weight of the longest edge connecting two entities (vertices) in G. Therefore, we can achieve the optimal solution of the restricted k-gMM clustering problem for graph G if and only if we achieve the optimal solution of the k-MM clustering problem for the corresponding transformed instance. Since the k-MM and k-Center clustering problems are equivalent in $\mathbb{R}$ (Observation 10), this hardness result also holds for the k-Center problem as well. ◀

Gonzalez [12] provides a 2-approximation algorithm for the k-MM and k-Center clustering of fixed points in $\mathbb{R}^d$. We provide a modified version of Gonzalez's algorithm to compute an approximate solution for the k-MM and k-Center clustering of a set $\mathcal{X}$ of n moving entities in $\mathbb{R}^d$. The algorithm initially assigns all entities to cluster C_1 and arbitrarily labels one entity as the head of that cluster, denoted $head_1$. The remaining clusters are computed in $k-1$ iterations. In the i^{th} iteration, we compute the head of cluster C_i. An entity $x \in \mathcal{X}$ is considered as the head of the cluster C_i if it maximizes

$$\max_{j\in\{1,\ldots,i\}} \max_{x\in C_j} \max_{t\in T} \delta(head_j(t), x(t)).$$

We label x as $head_i$ and add it to cluster C_i. After computing the i^{th} head, the entities are assigned to a cluster such that, for each entity $y \in \{C_1 \cup \cdots \cup C_{i-1}\}$, we move y to C_i if the maximum distance between y and its current cluster head over all time is larger than the maximum distance between y and $head_i$. The i^{th} iteration of choosing the i^{th} *head* takes $O(\tau n)$ time. Therefore, the total running time of the algorithm is $O(\tau nk)$. Since the algorithm makes its decision by comparing maximum distances between pairs of entities over all time, it can be generalized for moving entities in $\mathbb{R}^d$, where computing the Euclidean distance between two points takes $O(d)$ time. If d is an input parameter, then the algorithm runs in $O(\tau dnk)$ time. The algorithm guarantees a 2-approximation for both the k-MM and k-Center clustering problems. The objective function value of k-MM and k-Center clustering of $\mathcal{X}$ by using the above algorithm is denoted by $\phi_{P1}(\mathcal{X})$ and $\phi_{P2}(\mathcal{X})$ respectively. Let $OPT_{P1}(\mathcal{X})$ and $OPT_{P2}(\mathcal{X})$ be the objective function values of optimal k-MM and k-Center clustering of $\mathcal{X}$ respectively.

▶ **Lemma 16.** *Given a set $\mathcal{X}$ of n moving entities in $\mathbb{R}^d$, where $\mathcal{L} = \{x_1, \ldots, x_{k+1}\}$ is a subset of $\mathcal{X}$, for every pair $\{x, y\} \subseteq \mathcal{L}$, if $\max_{t\in T} \delta(x(t), y(t)) \geq h$, then $OPT_{P1}(\mathcal{X}) \geq h$, for any $h > 0$.*

Proof. We have $k+1$ entities to make k clusters. By the pigeonhole principle, one cluster must contain two entities, and the rest of the $k-1$ clusters contain one entity in each. The maximum distance between any two entities in $\mathcal{X}$ over all time is at least h, for any $h > 0$. We have only one cluster with two entities where the maximum distance between them over all time is at least h, therefore we can say that $OPT_{P1}(\mathcal{X}) \geq h$. ◀

▶ **Theorem 17.** *The modified Gonzalez algorithm is a 2-approximation algorithm for the k-MM and k-Center clustering of $\mathcal{X}$.*

Proof. Let x denote an entity in cluster C_i that maximizes

$$\max_{i\in\{1,\ldots,k\}} \max_{x\in C_i} \max_{t\in T} \delta(head_i(t), x(t)).$$

Let h be the maximum distance between x and $head_i$ over all time. This distance satisfies triangle inequality (Lemma 2). Therefore, $\phi_{P1}(\mathcal{X}) \leq 2 \cdot h$. In every iteration, the maximum distance between x and $head_i$ over all time is at least h. Therefore, when a new cluster is added, the maximum distance between the new cluster's head and any previously added cluster's head over all time is at least h. That is to say, for any $i \neq j$, $\max_{t\in T} \delta(head_i(t), head_j(t)) \geq h$. Let $\mathcal{L} = \{head_1, \ldots, head_k, x\}$ be a set of $k+1$ entities. The maximum distance between any pair of entities in $\mathcal{L}$ over all time is at least h. Since, $\mathcal{L}$ is a subset of $\mathcal{X}$, $OPT_{P1}(\mathcal{X}) \geq h$ (Lemma 16). Thus, we can conclude that $\phi_{P1}(\mathcal{X}) \leq 2 \cdot OPT_{P1}(\mathcal{X})$.

In the algorithm, the head of a cluster can be considered to be the center of that cluster. For k-Center clustering, we assume that $\phi_{P2}(\mathcal{X}) > 2 \cdot OPT_{P2}(\mathcal{X})$. This implies that in $\mathcal{L}$, we have $k+1$ entities, and their pairwise maximum distance over all time is greater than $2 \cdot OPT_{P2}(\mathcal{X})$. By the pigeonhole principle, at least two of these entities (say, $head_i$ and $head_j$) must be in the same cluster in the optimal k-Center clustering. Let s be the center of that cluster. Therefore, we get

$$\max_{t\in T} \delta(head_i(t), s(t)) \leq OPT_{P2}(\mathcal{X}),\ \max_{t\in T} \delta(head_j(t), s(t)) \leq OPT_{P2}(\mathcal{X}).$$

By triangle inequality, $\max_{t\in T} \delta(head_i(t), head_j(t)) \leq 2 \cdot OPT_{P2}(\mathcal{X})$. This contradicts our initial assumption. Therefore, $\phi_{P2}(\mathcal{X})$ cannot be greater than $2 \cdot OPT_{P2}(\mathcal{X})$. ◀

From the above discussion, we can state the following theorem.

▶ **Theorem 18.** *A 2-approximate solution for the k-MM and k-Center clustering of moving entities in $\mathbb{R}^d$ can be computed in $O(\tau dnk)$ time.*

6 Conclusion and possible directions for future research

In this paper, we examine the k-MM and k-Center clustering problems for sets of moving entities in $\mathbb{R}^d$. Both problems have been studied for sets of fixed points in $\mathbb{R}^d$; this paper examines these problems in the mobile setting. We show that both problems are NP-hard, even if entities move in $\mathbb{R}$. The 2-MM clustering problem can be solved exactly in $O(\tau dn^2)$ time in $\mathbb{R}^d$, and $O(\tau n \log n)$ time in $\mathbb{R}$. Unlike the 2-MM clustering problem, the 2-Center clustering problem is NP-hard in $\mathbb{R}^2$. We show that our 2-MM clustering algorithm gives a 1.15-approximate solution for the 2-Center clustering problem. Furthermore, we prove that no polynomial-time algorithm can achieve a better approximation ratio unless $P = NP$. We use the idea of Gonzalez's algorithm [12] and provide a 2-approximation algorithm for the k-MM and k-Center clustering problems. Possible directions for future research include

studying these problems when the time stamps of entities differ. It would be interesting to develop an algorithm for the 2-MM clustering problem that runs in less than $O(\tau dn^2)$ time. Finally, if the distance traveled by an entity between two time stamps is bounded by a constant, can we develop algorithms for these problems whose running time is logarithmic in τ?

References

1. P. K. Agarwal and C. M. Procopiuc. Exact and approximation algorithms for clustering. *Algorithmica*, 33(2):201–226, 2002.
2. P. K. Agarwal and M. Sharir. Efficient algorithms for geometric optimization. *ACM Computing Surveys (CSUR)*, 30(4):412–458, 1998.
3. S. Bespamyatnikh, B. Bhattacharya, D. Kirkpatrick, and M. Segal. Lower and upper bounds for tracking mobile users. In *Foundations of Information Technology in the Era of Network and Mobile Computing*, pages 47–58. Springer, 2002.
4. K. Buchin, M. Buchin, M. van Kreveld, B. Speckmann, and F. Staals. Trajectory grouping structure. In *Proc. Workshop on Algorithms and Data Structures*, volume 8037 of *Lecture Notes in Computer Science*, pages 219–230. Springer, 2013.
5. K. Buchin, A. Driemel, J. Gudmundsson, M. Horton, I. Kostitsyna, and M. Löffler. Approximating (k, l)-center clustering for curves. In *Proc. Symposium on Discrete Algorithms*, pages 2922–2938. SIAM, 2019.
6. A. Driemel, A. Krivošija, and C. Sohler. Clustering time series under the Fréchet distance. In *Proc. Symposium on Discrete Algorithms*, pages 766–785, 2016.
7. S. Durocher. *Geometric facility location under continuous motion*. PhD thesis, University of British Columbia, 2006.
8. S. Durocher and D. Kirkpatrick. The Steiner centre of a set of points: Stability, eccentricity, and applications to mobile facility location. *International Journal of Computational Geometry & Applications*, 16(04):345–371, 2006.
9. S. Durocher and D. Kirkpatrick. Bounded-velocity approximation of mobile Euclidean 2-centres. *International Journal of Computational Geometry & Applications*, 18(03):161–183, 2008.
10. D. Eppstein. Faster construction of planar two-centers. In *Proc. Symposium on Discrete Algorithms*, pages 131–138. ACM/SIAM, 1997.
11. G. N. Frederickson. Parametric search and locating supply centers in trees. In *Proc. Workshop on Algorithms and Data Structures*, volume 519 of *Lecture Notes in Computer Science*, pages 299–319. Springer, 1991.
12. T. F. Gonzalez. Clustering to minimize the maximum intercluster distance. *Theoretical Computer Science*, 38:293–306, 1985.
13. S. Har-Peled. Clustering motion. *Discrete & Computational Geometry*, 31(4):545–565, 2004.
14. C. S. Jensen, D. Lin, and B. C. Ooi. Continuous clustering of moving objects. *IEEE Trans. Knowl. Data Eng.*, 19(9):1161–1174, 2007.
15. J. Lee, J. Han, and K. Whang. Trajectory clustering: a partition-and-group framework. In *Proc. international conference on Management of data*, pages 593–604. ACM, 2007.
16. Y. Li, J. Han, and J. Yang. Clustering moving objects. In *Proc. international conference on Knowledge discovery and data mining*, pages 617–622. ACM, 2004.
17. Z. Li, B. Ding, J. Han, and R. Kays. Swarm: Mining relaxed temporal moving object clusters. *Proceedings of the VLDB Endowment*, 3(1-2):723–734, 2010.
18. T. W. Liao. Clustering of time series data - a survey. *Pattern recognition*, 38(11):1857–1874, 2005.
19. N. Megiddo and K. J. Supowit. On the complexity of some common geometric location problems. *SIAM journal on computing*, 13(1):182–196, 1984.

20 T. J. Schaefer. The complexity of satisfiability problems. In *Proceedings of the tenth annual ACM symposium on Theory of computing*, pages 216–226, 1978.

21 X. Tan and B. Jiang. Simple $O(n \log^2 n)$ algorithms for the planar 2-center problem. In *Proc. Computing and Combinatorics Conference*, volume 10392 of *Lecture Notes in Computer Science*, pages 481–491. Springer, 2017.

22 G. Yuan, P. Sun, J. Zhao, D. Li, and C. Wang. A review of moving object trajectory clustering algorithms. *Artificial Intelligence Review*, 47(1):123–144, 2017.