Efficient Greedy Discrete Subtrajectory Clustering

Ivor van der Hoog
Technical University of Denmark, Lyngby, Denmark

Lara Ost
University of Vienna, Austria

Eva Rotenberg
Technical University of Denmark, Lyngby, Denmark

Daniel Rutschmann
Technical University of Denmark, Lyngby, Denmark

Abstract

We cluster a set of trajectories $\mathcal{T}$ using subtrajectories of $\mathcal{T}$. We require for a clustering C that any two subtrajectories $(\mathcal{T}[a,b], \mathcal{T}[c,d])$ in a cluster have disjoint intervals $[a,b]$ and $[c,d]$. Clustering quality may be measured by the number of clusters, the number of vertices of $\mathcal{T}$ that are absent from the clustering, and by the Fréchet distance between subtrajectories in a cluster.

A Δ-cluster of $\mathcal{T}$ is a cluster $\mathcal{P}$ of subtrajectories of $\mathcal{T}$ with a centre $P \in \mathcal{P}$, where all subtrajectories in $\mathcal{P}$ have Fréchet distance at most Δ to P. Buchin, Buchin, Gudmundsson, Löffler and Luo present two $O(n^2 + nm\ell)$-time algorithms: $\mathbf{SC}(\max,\ \ell,\ \Delta,\ \mathcal{T})$ computes a *single* Δ-cluster where P has at least ℓ vertices and maximises the cardinality m of $\mathcal{P}$. $\mathbf{SC}(m,\ \max,\ \Delta,\ \mathcal{T})$ computes a *single* Δ-cluster where $\mathcal{P}$ has cardinality m and maximises the complexity ℓ of P.

In this paper, which is a mixture of algorithms engineering and theoretical insights, we use such maximum-cardinality clusters in a greedy clustering algorithm. We first provide an efficient implementation of $\mathbf{SC}(\max,\ \ell,\ \Delta,\ \mathcal{T})$ and $\mathbf{SC}(m,\ \max,\ \Delta,\ \mathcal{T})$ that significantly outperforms previous implementations. Next, we use these functions as a subroutine in a greedy clustering algorithm, which performs well when compared to existing subtrajectory clustering algorithms on real-world data. Finally, we observe that, for fixed Δ and $\mathcal{T}$, these two functions always output a point on the Pareto front of some bivariate function $\theta(\ell, m)$. We design a new algorithm $\mathbf{PSC}(\Delta,\ \mathcal{T})$ that in $O(n^2 \log^4 n)$ time computes a 2-approximation of this Pareto front. This yields a broader set of candidate clusters, with comparable quality to the output of the previous functions. We show that using $\mathbf{PSC}(\Delta,\ \mathcal{T})$ as a subroutine improves the clustering quality and performance even further.

2012 ACM Subject Classification Theory of computation → Computational geometry

Keywords and phrases Algorithms engineering, Fréchet distance, subtrajectory clustering

Digital Object Identifier 10.4230/LIPIcs.SoCG.2025.78

Related Version *Full Version*: `http://arxiv.org/abs/2503.14115`

Supplementary Material
Software (Source code): `https://github.com/laraost/DiscreteSubtrajectoryClustering` archived at `swh:1:dir:d00278b75c11b80bce00c51dbf8ac904972d8586`
Software (Test Suite): `https://github.com/dacin21/discrete-subtrajectory-clustering-test-suite`, archived at `swh:1:dir:e8f5c54ffb22bda19e4ca7858b7a6ea5b5ab2090`

Funding *Ivor van der Hoog*, *Eva Rotenberg*, and *Daniel Rutschmann* are grateful to the Carlsberg Foundation for supporting this research via Eva Rotenberg's Young Researcher Fellowship CF21-0302 "Graph Algorithms with Geometric Applications".
Ivor van der Hoog: This project has received funding from the European Union's Horizon 2020 research and innovation programme under the Marie Skłodowska-Curie grant agreement No 899987
Lara Ost: Funded by the Vienna Graduate School on Computational Optimization (VGSCO), FWF-Project No. W1260-N35

1 Introduction

Trajectories describe the movement of objects. The moving objects are modeled as a sequence of locations. Trajectory data often comes from GPS samples, which are being generated on an incredible scale through cars, cellphones, and other trackers. We focus on *subtrajectory clustering* where clusters are sets of subtrajectories. The goal is to create few clusters and to ensure that subtrajectories within a cluster have low pairwise discrete Fréchet distance. This algorithmic problem has been studied extensively [1, 8, 10, 12, 13, 15, 16, 19, 21, 22, 20, 24, 27, 30] in both theoretical and applied settings. For surveys, see [7] and [28].

Subtrajectory clustering. Comparing whole trajectories gives little information about shared structures. Consider two trajectories T_1 and T_2 that represent individuals who drive the same route to work, but live in different locations. For almost all distance metrics, the distance between T_1 and T_2 is large because their endpoints are (relatively) far apart. However, T_1 and T_2 share similar subtrajectories. Subtrajectory clustering allows us to model a wide range of behaviours: from detecting commuting patterns, to flocking behaviour, to congestion areas [12]. In the special case where trajectories correspond to traffic on a known road network, trajectories can be mapped to the road network. In many cases, the underlying road network might not be known (or it may be too dense to represent). In such cases, it is interesting to construct a sparse representation of the underlying road network based on the trajectories. The latter algorithmic problem is called *map construction*, and it has also been studied extensively [2, 6, 3, 14, 17, 25, 26]. We note that subtrajectory clustering algorithms may be used for map construction: if each cluster is assigned one centre, then the collection of centres can be used to construct an underlying road network [9, 10].

Related work. BBGLL [12] introduce the following problem. Rather than clustering $\mathcal{T}$, their goal is to compute a *single* cluster. Given $\mathcal{T}$, a Δ-*cluster* is any pair $(P, \mathcal{P})$ where:

- P is a subtrajectory of a trajectory in $\mathcal{T}$,
- $\mathcal{P}$ is a set of subtrajectories, where for all $\mathcal{T}[a, b], \mathcal{T}[c, d] \in \mathcal{P}$, $[a, b] \cap [c, d] = \emptyset$,
- and, for all $P' \in \mathcal{P}$, the Fréchet distance between P and P' is at most Δ.

See Figure 1. They define two functions that each output a single Δ-cluster $(P, \mathcal{P})$:

- $\mathbf{SC}(\max, \ell, \Delta, \mathcal{T})$ requires that $|P| = \ell$ and maximises $|\mathcal{P}|$.
- $\mathbf{SC}(m, \max, \Delta, \mathcal{T})$ requires that $|\mathcal{P}| = m$ and maximises $|P|$.

If $|P| = \ell$ and $|\mathcal{P}| = m$, their algorithms use $O(n^2 + nm\ell)$ time and $O(n\ell)$ space.

AFMNPT [1] propose a clustering framework. They define a clustering C as any set of clusters. They propose a scoring function with constants (c_1, c_2, c_3) that respectively weigh: the number of clusters, the radius of each cluster, and the number of uncovered vertices. Their implement a heuristic algorithm that uses this scoring function.

BBDFJSSSW [9] do map construction. They compute a set of Δ-clusters C for various Δ. For each $(P, \mathcal{P}) \in C$, they use P as a road on the map they are constructing. Thus, it may be argued that [9] does subtrajectory clustering. As a preprocessing step, they implement $\mathbf{SC}(\max, \ell, \Delta, \mathcal{T})$. Choosing various ℓ and Δ, they compute a collection S of "candidate" clusters. They greedily select a cluster $(P, \mathcal{P}) \in S$, add it to the clustering, and remove all subtrajectories in $\mathcal{P}$ from all remaining clusters in S. For a cluster $(Q, \mathcal{Q}) \in S$, this may split a $Q' \in \mathcal{Q}$ into two subtrajectories. BBGHSSSSSW [10] give improved implementations.

ABCD [4] do not require that for all $\mathcal{T}[a, b], \mathcal{T}[c, d] \in \mathcal{P}$, $[a, b] \cap [c, d] = \emptyset$. They do require that the final clustering C has no uncovered vertices, and that for each cluster $(P, \mathcal{P})$, $|P| \leq \ell$ for some given ℓ. For any Δ, denote by k_Δ the minimum-sized clustering that meets

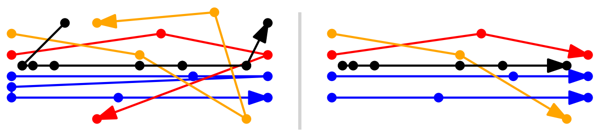

Figure 1 (left) Four trajectories. (right) Five subtrajectories $\mathcal{P}$. Depending on the choice of $P \in \mathcal{P}$, we may get five different clusters $(P, \mathcal{P})$.

their requirements. Their randomised polynomial-time algorithm computes a clustering of $O(\Delta)$-clusters of size $\tilde{O}(k_\Delta \ell)$. BCD [8] give polynomial-time clustering algorithm of $O(\Delta)$-clusters of size $\tilde{O}(k_\Delta)$. Conradi and Driemel [16] implement an $\tilde{O}(\ell^2 n^4 + k_\Delta \ell n^4)$-time algorithm that computes a set of $O(\Delta)$-clusters of size $O(k_\Delta \log n)$.

Contribution. Our contribution is threefold. First, in Section 5, we describe an efficient implementation of $\mathbf{SC}(\max,\ \ell,\ \Delta,\ \mathcal{T})$ and $\mathbf{SC}(m,\ \max,\ \Delta,\ \mathcal{T})$. We compare our implementation to all existing single-core implementations.

In Section 5 we propose that $\mathbf{SC}(\max,\ \ell,\ \Delta,\ \mathcal{T})$ produces high-quality clusters across all clustering metrics. We create a greedy clustering algorithm that fixes (ℓ, Δ) and iteratively adds the solution $(P, \mathcal{P}) = \mathbf{SC}(\max,\ \ell,\ \Delta,\ \mathcal{T})$ to the clustering (removing all vertices in $\mathcal{P}$ from $\mathcal{T}$). An analogous approach for $\mathbf{SC}(m,\ \max,\ \Delta,\ \mathcal{T})$ gives two clustering algorithms.

In Section 6 we note that the functions $\mathbf{SC}(\max,\ \ell,\ \Delta,\ \mathcal{T})$ and $\mathbf{SC}(m,\ \max,\ \Delta,\ \mathcal{T})$ search a highly restricted solution space, having fixed either the minimum length of the centre of a cluster or the minimum cardinality. We observe that these functions always output a cluster on the Pareto front of some bivariate function $\theta(m, \ell)$. Since points on this Pareto front give high-quality clusters, we argue that it may be beneficial to approximate all vertices of this Pareto front directly – sacrificing some cluster quality for cluster flexibility. We create a new algorithm $\mathbf{PSC}(\Delta,\ \mathcal{T})$ that computes in $O(n^2 \log^4 n)$ time and $O(n)$ space a set of clusters that is a 2-approximation of the Pareto font of $\theta(m, \ell)$. We show that using $\mathbf{PSC}(\Delta,\ \mathcal{T})$ as a subroutine for greedy clustering gives even better results.

In Section 7 we do experimental analysis. We compare our clustering algorithms to AFMNPT [1] and BBGHSSSSSW [10]. We adapt the algorithm in [10] so that it terminates immediately after computing the subtrajectory clustering. We do not compare to Conradi and Driemel [16] as their setting is so different that it produces incomparable clusters. We compare the algorithms based on their overall run time and space usage. We evaluate the clustering quality along various metrics such as the number of clusters, the average cardinality of each cluster, and the Fréchet distance between subtrajectories in a cluster. Finally, we consider the scoring function by AFMNPT [1].

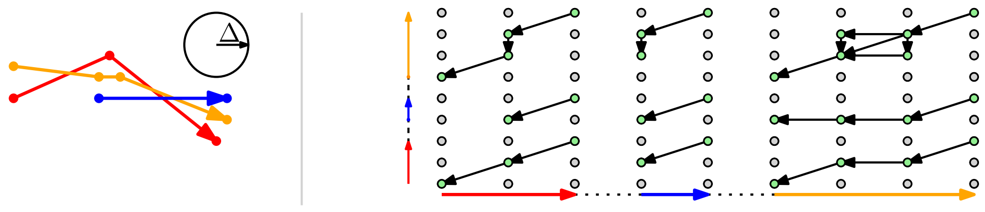

Figure 2 A set of trajectories $\mathcal{T}$ and the matrix $M_\Delta(\mathcal{T}, \mathcal{T})$ as a graph.

2 Preliminaries

The input is a set $\mathcal{T}$ of trajectories where $n = \sum_{T \in \mathcal{T}} |T|$. We denote by $\mathcal{X}$ all vertices of $T \in \mathcal{T}$. We view any trajectory T with n_T vertices as a map from $[n_T]$ to $\mathbb{R}^2$ where $T(i)$ is the i'th vertex on T. For brevity, we refer to $\mathcal{T}$ as a single trajectory obtained by concatenating all trajectories in $\mathcal{T}$ arbitrarily. A *boundary edge* in $\mathcal{T}$ is any edge in $\mathcal{T}$ between the endpoints of two consecutive trajectories. A *subtrajectory* $\mathcal{T}[a, b]$ of $\mathcal{T}$ is defined by $a, b \in [n]$ with $a \leq b$, where we require that the subcurve from $\mathcal{T}(a)$ to $\mathcal{T}(b)$ contains no boundary edge.

Fréchet distance. We first define discrete walks for pair of trajectories $P = (p_1, \ldots, p_x)$ and $T = (t_1, \ldots, t_y)$ in $\mathbb{R}^2$. We say that an ordered sequence F of points in $[x] \times [y]$ is a *discrete walk* if consecutive pair $(i, j), (s, l) \in F$ have $s \in \{i, i-1\}$ and $l \in \{j, j-1\}$. The *cost* of a discrete walk F is the maximum distance $d(P(i), Q(j))$ over $(i, j) \in F$. The discrete Fréchet distance is the minimum-cost discrete walk from (x, y) to $(1, 1)$:

$$\mathcal{D}_F(P, Q) := \min_F \operatorname{cost}(F) = \min_F \max_{(i,j) \in F} d(P(i), Q(j)).$$

Free-space matrix. Given $\mathcal{T}$ and a value Δ, the *Free-space matrix* $M_\Delta(\mathcal{T}, \mathcal{T})$ is an $n \times n$ 01-matrix where for all $(i, j) \in [n] \times [n]$ the matrix has a zero at position i, j if $d(\mathcal{T}(i), \mathcal{T}(j)) \leq \Delta$. We use matrix notation where (i, j) denotes row i and column j. In the plane, $(0, 0)$ denotes the top left corner. We also use $M_\Delta(\mathcal{T}, \mathcal{T})$ to denote the digraph where the vertices are all $(i, j) \in [n] \times [n]$ and there exists an edge from (i, j) to (s, k) whenever (see Figure 2):

- $s \in \{i, i-1\}$ and $k \in \{j, j-1\}$, and
- $M_\Delta(\mathcal{T}, \mathcal{T})[i, j] = M_\Delta(\mathcal{T}, \mathcal{T})[s, k] = 0$.

Alt and Godeau [5] show that for any two subtrajectories $\mathcal{T}[a, b]$ and $\mathcal{T}[c, d]$: $\mathcal{D}_F(T[a, b], T[c, d]) \leq \Delta$ if and only if there is a directed path from (b, d) to (a, c).

▶ **Definition 1.** *Given Δ and $\mathcal{T}$, we say that a vertex (b, x) in $M_\Delta(\mathcal{T}, \mathcal{T})$ can* reach *row a if there exists an integer $x' \leq x$ such that there is a directed path from (b, x) to (a, x'). We denote by $\mathtt{row}(b, x)$ the minimum integer a' such that (b, x) can reach row a'.*

▶ **Definition 2** (Figure 1)**.** *A cluster is any pair $(P, \mathcal{P})$ where P is a subtrajectory and $\mathcal{P}$ is a set subtrajectories with $P \in \mathcal{P}$. We require that for all $\mathcal{T}[a, b], \mathcal{T}[c, d] \in \mathcal{T}$, $[a, b] \cap [c, d] = \emptyset$. We define its* length $|P|$, cardinality $|\mathcal{P}|$, *and* coverage $\mathtt{cov}(P, \mathcal{P}) = \bigcup_{S \in \mathcal{P}} S$.

Note that we follow [1, 9, 10, 12] and consider only *disjoint* clusterings, meaning $\forall (P, \mathcal{P}') \in C$, $\forall \mathcal{T}[a, b], \mathcal{T}[c, d] \in \mathcal{P}$, the intervals $[a, b]$ and $[c, d]$ are disjoint. For brevity, we say that a Δ-cluster is any cluster $(P, \mathcal{P})$ where $\forall P' \in \mathcal{P} : \mathcal{D}_F(P, P') \leq \Delta$.

▶ **Definition 3.** *A* clustering *C is a set of clusters. Its* coverage *is* $\mathtt{cov}(C) := \bigcup_{(P,\mathcal{P}) \in C} \mathtt{cov}(P, \mathcal{P})$.

AFMNPT [1] propose a cost function to determine the quality of a clustering C. We distinguish between *k-centre* and *k-means* clustering:

▶ **Definition 4.** *Given constants* (c_1, c_2, c_3) *,* $\texttt{Score}(C)$ *is a sum that weights three terms:*

1. *The number of clusters in* C*:* $c_1|C|$.
2. *The chosen cost function for the clustering. I.e.,*
   $c_2 \max_{(P,\mathcal{P})\in C} \operatorname{cen}(P,\mathcal{P}) = c_2 \max_{(P,\mathcal{P})\in C} \max_{P'\in\mathcal{P}} \mathcal{D}_F(P,P')$ *when doing* k*-centre clustering, or*
   $c_2 \sum_{(P,\mathcal{P})\in C} \operatorname{mean}(P,\mathcal{P}) = c_2 \sum_{(P,\mathcal{P})\in C} \sum_{P'\in\mathcal{P}} \mathcal{D}_F(P,P')$ *when doing* k*-means clustering.*
3. *The fraction of uncovered points:* $c_3 \frac{|\mathcal{X}\setminus \texttt{cov}(C)|}{|\mathcal{X}|}$.

We also consider unweighted metrics: the maximal and average Fréchet distance between subtrajectories in a cluster, and the maximal and average cardinality of the clusters.

3 Data

We use five real-world data sets (see Table 1). For some of these data sets we include a subsampled version that only retains a $\frac{1}{c}$ fraction of the trajectories.

- The first three data sets are Athens-small, Chicago and Berlin, consisting of GPS samples from car traffic across these cities. These data sets are widely used benchmark tests for road map construction [2, 9, 31, 23].
- The fourth data set is the *Drifter* data set that was used for subtrajectory clustering by Driemel and Conradi [16]. It consists of GPS data from ocean drifters.
- The fifth data set is the UnID data set, which is the smallest data set used in [29].

Unfortunately, we cannot use any of the data sets in [1]. Most of these are no longer available. Those that are available are far too large. It is unknown how [1] preprocessed these.

Synthetic data. All competitor algorithms do not terminate on the larger real-world data sets, even when the time limit is 24 hours. Thus, we primarily use these sets to compare algorithmic efficiency. Our qualitative analysis uses the subsampled data sets and synthetic data. We generate data by creating a random domain X. Then, for some parameter c, we randomly select $c\%$ of the domain and invoke an approximate TSP solver. We add the result as a trajectory. This results in a collection of trajectories with many similar subtrajectories. Synthetic-A has a random domain of 200 vertices. Synthetic-B has a random domain of 100 vertices. For both of these, we consider $c \in \{50, 90, 95\}$.

Table 1 Our data sets, their sizes, and whether we subsampled a fraction of their trajectories.

Name	**Real world**	**# of trajectories**	**# of vertices**	**Subsampled**
Athens-small	yes	128	2840	no
Chicago-4	yes	222	29344	$\frac{1}{4}$'th
Chicago	yes	888	118360	no
Berlin-10	yes	2717	19130	$\frac{1}{10}$'th
Berlin	yes	27188	192223	no
Drifter	yes	2167	1931692	no
UniD	yes	362	214077	no
Synthetic-A	no	50	5000 - 9500	no
Synthetic-B	no	100	5000 - 9500	no

4 Introducing and solving SC(m, ℓ, Δ, $\mathcal{T}$)

BBGLL [12] introduce the problem **SC**(m, ℓ, Δ, $\mathcal{T}$). The input are integers m, ℓ, Δ and a set of trajectories $\mathcal{T}$. The goal is to find a Δ-cluster $(P, \mathcal{P})$ where $|\mathcal{P}| \geq m$ and $|P| = \ell$. The problem statement may be altered by maximising m or ℓ instead. This yields the problems **SC**(max, ℓ, Δ, $\mathcal{T}$) and **SC**(m, max, Δ, $\mathcal{T}$). There exist three single-core implementations of these algorithms (the *MOVETK library* [18], *map-construct* [9] and *map-construct-rtrees* [10], a version of *map-construct* that uses r-trees). [18] uses $O(n^2)$ space and $O(n^2 + nm\ell)$ time. Both [9, 10] use $O(n\ell)$ space and $O(n^2 + nm\ell)$ time, and [10] offers running time improvements over [10]. The latter uses the semi-continuous instead of discrete Fréchet distance. On the experimental input, switching between these metrics does not change the output. We do not compare to the existing GPU-based implementation [21].

Contribution. We sketch how we optimise the BBGLL algorithm in the full version Considers two integers a, b with $\ell = b - a$ (Figure 3). It alternates between `StepFirst` which increments a, `StepSecond` which increments b, and `Query` that given $(\Delta, \ell = b - a, m)$ aims to find a Δ-cluster $(P, \mathcal{P})$ with $P = \mathcal{T}[a, b]$ and $|\mathcal{P}| = m$. We apply two ideas:

1. **Windowing.** By [12], it suffices to keep only $M_\Delta(\mathcal{T}[a, b], \mathcal{T})$ in memory. Furthermore, one may store only the $z' \in O(n\ell)$ zeroes in this sub-matrix. This is implemented in [9, 10]. By cleverly abandoning memory, our `StepFirst` deletes a row in constant time. The `StepSecond` function is then the bottleneck, as it adds a new row.
2. **Row generation.** In [12], the `StepSecond` and `Query` functions iterate over all n cells in the bottommost row of $M_\Delta(\mathcal{T}, \mathcal{T})$. By applying a range searching data structure, we instead only iterate over all zeroes in this row.

We note that map-construct-rtree [10] uses r-trees for row generation (we refer to the journal version [11]). However, we believe that the size of their r-tree implementation dominates the time and space used by their algorithm in our experiments. In the full version, we describe how to use a light-weight range tree to implement this principle.

Experiments. We compare the implementations across our data sets for a variety of choices of Δ and m. We use the implementations of **SC**(m, max, Δ, $\mathcal{T}$) so that our experiments have to fix fewer parameters. Figures 4 and 5 shows the time and space usage across a subset of our experiments. The data clearly shows that our implementation is significantly more efficient than the previous single-core implementations. It is often more than a factor 1000 more efficient in terms of runtime and memory usage. We do not provide a more extensive analysis of these results since this is not our main contribution.

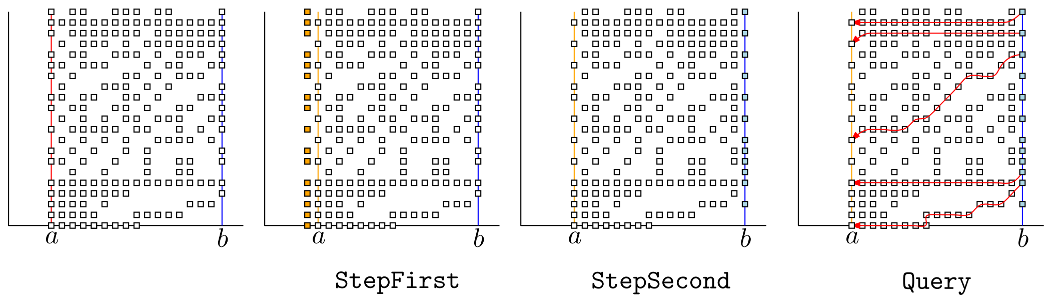

Figure 3 An overview of the algorithm in [12]. We illustrate zeroes in $M_\Delta(\mathcal{T}, \mathcal{T})$ with a square.

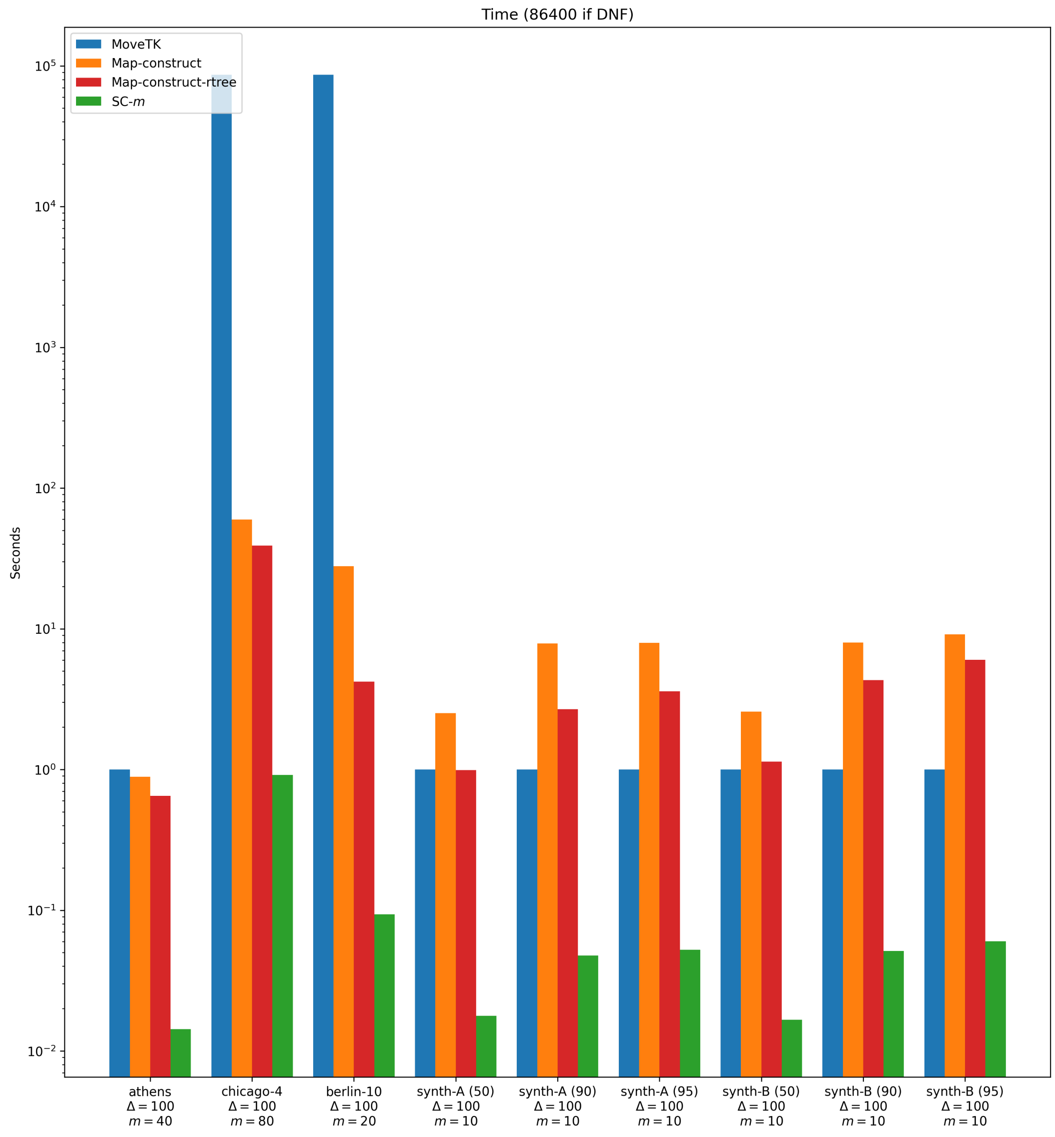

Figure 4 ■ BBGLL implementation comparison ■ Running time ■ Logarithmic scaling.

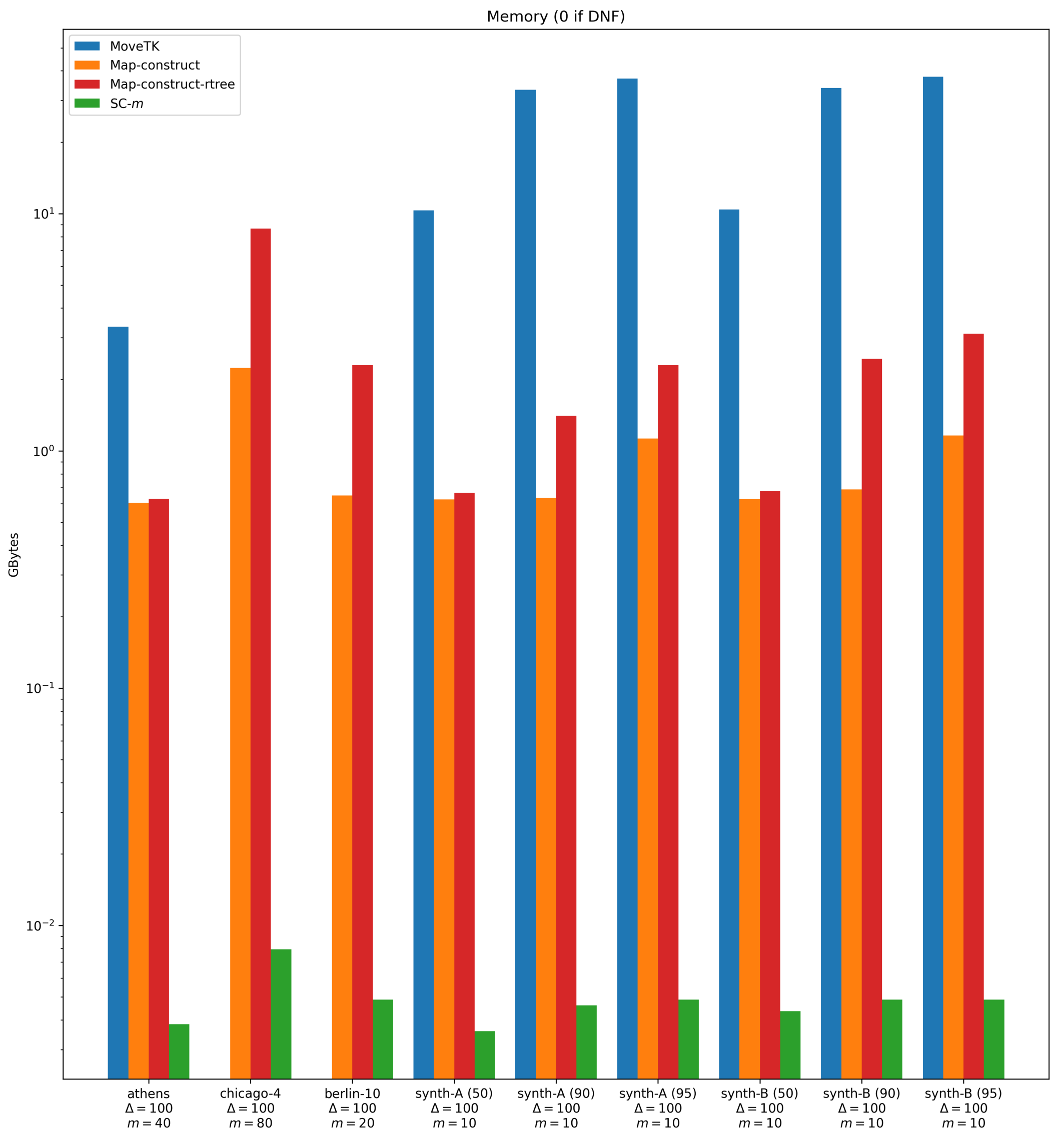

Figure 5 ■ BBGLL implementation comparison ■ Memory usage ■ Logarithmic scaling.

5 Greedy clustering using SC(max, ℓ, Δ, $\mathcal{T}$)

Our primary contribution is that we propose to use **SC**(max, ℓ, Δ, $\mathcal{T}$) in a greedy clustering algorithm. On a high level, our clustering algorithm does the following:

- We first choose the scoring function and constants (see Definition 4).
- Our input is the set of trajectories $\mathcal{T}$, some Δ, some ℓ, and an empty clustering C.
- We obtain a cluster $(P, \mathcal{P}) =$ **SC**(max, ℓ, Δ, $\mathcal{T}$), and add it to C.
- We then remove all subtrajectories in $\mathcal{P}$ from $\mathcal{T}$, and recurse.
- We continue this process until the addition of $(P, \mathcal{P})$ to C no longer increases $\texttt{Score}(C)$.

The implementation details depend on whether we are doing k-centre or k-median clustering.

5.1 k-centre clustering.

Given are (c_1, c_2, c_3) and a set of trajectories $\mathcal{T}$. We choose some integer ℓ. The k-centre clustering scoring function weighs the value: $\max_{(P,\mathcal{P})\in C} \text{cen}(P, \mathcal{P}) = \max_{(P,\mathcal{P})\in C} \max_{P'\in\mathcal{P}} \mathcal{D}_F(P, P')$.
If C is a set of Δ-clusters (we assume there exists a cluster in C whose radius is Δ) then:

$$\texttt{Score}(C) = c_1 \cdot |C| + c_2 \cdot \Delta + c_3 \cdot \frac{|\mathcal{X} \backslash \texttt{cov}(C)|}{|\mathcal{X}|}.$$

Let C be a set of Δ-clusters and $(P, \mathcal{P}) =$ **SC**(max, ℓ, Δ, $\mathcal{T} - \texttt{cov}(C)$). Per construction of our algorithm, $\texttt{cov}(P, \mathcal{P}) \cap \texttt{cov}(C) = \emptyset$ and so the contribution of $(P, \mathcal{P})$ to $\texttt{Score}(C \cup \{(P, \mathcal{P})\})$ equals $c_1 - c_3 \cdot \frac{|\texttt{cov}(P,\mathcal{P})|}{|\mathcal{X}|}$. We obtain $|\texttt{cov}(P, \mathcal{P})|$ using constant additional time throughout our algorithm. This leads to the following algorithm:

▶ **Clustering 1** (SC-ℓ((c_1, c_2, c_3), $\mathcal{T}$) under k-centre clustering). *We run Algorithm 1 (c_1, c_2, c_3) for exponentially increasing $\Delta \in (2, 4, \dots)$ where the subroutine f is **SC**(max, ℓ, Δ, $\mathcal{T}$). We then output the maximum-score clustering C over all runs.*

Algorithm 1 `k_centre(vector` (c_1, c_2, c_3), `value` Δ, `trajectory` $\mathcal{T}$, `subroutine` f).

```
$C \leftarrow \emptyset$, $T \leftarrow \mathcal{T}$.
while $|T| > 0$ do
    $(P, \mathcal{P}) \leftarrow$ the result of applying $f$ to $(\Delta, T)$
    if $\texttt{Score}(C \cup \{(P, \mathcal{P})\}) > \texttt{Score}(C)$ then
        $C$.add($(P, \mathcal{P})$).
        $T \leftarrow T - \mathcal{P}$.
    else
        Break.
    end if
end while
return $C$
```

We may also fix m instead of ℓ. Or, formally:

▶ **Clustering 2** (SC-m((c_1, c_2, c_3), $\mathcal{T}$) under k-centre clustering). *We run Algorithm 1 (c_1, c_2, c_3) for exponentially increasing $\Delta \in (2, 4, \dots)$ where the subroutine f is **SC**(m, max, Δ, $\mathcal{T}$). We then output the maximum-score clustering C over all runs.*

5.2 k-means clustering.

Our algorithm for k-means clustering is less straightforward. To get good performance we must iterate over different choices of Δ across the algorithm. Consider a clustering C and $(P, \mathcal{P}) = \mathbf{SC}(\max,\ \ell,\ \Delta,\ \mathcal{T} - \texttt{cov}(C))$ for some Δ. Under k-centre clustering, the more points a Δ-cluster covers, the better its scoring contribution. For k-means clustering this is no longer true as the contribution of $(P, \mathcal{P})$ to $\texttt{Score}(C \cup \{(P, \mathcal{P})\})$ equals:

$$\text{Contribution of } (P, \mathcal{P}) \text{ to } \texttt{Score}(C) = c_1 - c_3 \cdot \frac{|\texttt{cov}(P, \mathcal{P})|}{|\mathcal{X}|} + \sum_{P' \in \mathcal{P}} c_2 \cdot \mathcal{D}_F(P, P').$$

We abuse the fact that subtrajectories share no vertices to simplify the above expression to:

$$\text{Contribution of } (P, \mathcal{P}) \text{ to } \texttt{Score}(C) = c_1 + \sum_{P' \in \mathcal{P}} \left(c_2 \cdot \mathcal{D}_F(P, P') - c_3 \frac{|P'|}{|\mathcal{X}|} \right).$$

Consider a set of "candidate" clusters S. To select the best cluster from S, we use the same greedy set cover technique as [1]. We assign to every cluster a evaluation value. This is the ratio between how much it reduces the score, and how much it increases the score. I.e.,

$$\texttt{evaluation}(P, \mathcal{P}) = \frac{\sum_{P' \in \mathcal{P}} c_3 \frac{|P'|}{|\mathcal{X}|}}{c_1 + \sum_{P' \in \mathcal{P}} c_2 \cdot \mathcal{D}_F(P, P')}$$

We then select the cluster with the best evaluation among all $(P, \mathcal{P}) \in S$.

▶ **Clustering 3** (SC-$\ell((c_1, c_2, c_3),\ \mathcal{T})$ under k-means clustering)**.** *We run Algorithm 2 where the subroutine f is **SC**(*$\max$*, ℓ, Δ, $\mathcal{T}$).*

▶ **Clustering 4** (SC-$m((c_1, c_2, c_3),\ \mathcal{T})$ under k-means clustering)**.** *We run Algorithm 2 where the subroutine f is **SC**(m,* $\max$*, Δ, $\mathcal{T}$).*

Algorithm 2 `k_means(vector` (c_1, c_2, c_3), `trajectory` $\mathcal{T}$, `subroutine` f).

```
$C \leftarrow \emptyset$, $T \leftarrow \mathcal{T}$.
while $|T| > 0$ do
    $(P, \mathcal{P}) \leftarrow$ null
    for $\Delta \in (2, 4, \ldots)$ do
        $(Q, \mathcal{Q}) \leftarrow$ the result of applying $f$ to $(\Delta, \mathcal{T})$
        if evaluation$(Q, \mathcal{Q}) >$ evaluation$(P, \mathcal{P})$ then
            $(P, \mathcal{P}) \leftarrow (Q, \mathcal{Q})$.
        end if
    end for
    if Score$(C \cup \{(P, \mathcal{P})\}) >$ Score$(C)$ then
        $C$.Add$((P, \mathcal{P}))$.
        $T \leftarrow T - \mathcal{P}$.
    else
        Break.
    end if
end while
return $C$
```

6 A new algorithm as a subroutine

Our previous clusterings 1-4 fix a parameter ℓ (or m) and subsequently *only* produce clusters $(P, \mathcal{P})$ of maximal cardinality with $|P| = \ell$ (or of maximum length with $|\mathcal{P}| = m$).

This approach has three clear downsides:

1. To find a suitable ℓ (or m) you have to train the algorithm on a smaller data set.
2. Using multiple values of ℓ (or m) to broaden the search space is computationally expensive.
3. Even if we fix constantly many choices for ℓ and m, the corresponding algorithm has a heavily restricted solution space which affects the quality of the output.

To alleviate these downsides, we design a new algorithm based on the following observation:

▶ **Definition 5.** *For fixed $(\Delta, \mathcal{T})$, we define a bivariate Boolean function $\theta(m, \ell)$ where $\theta(m, \ell)$ is 1 if and only if there exists a Δ-cluster $(P, \mathcal{P})$ of $\mathcal{T}$ with $m = |\mathcal{P}|$ and $\ell = |P|$.*

The functions **SC(**max**,** ℓ**,** Δ**,** $\mathcal{T}$**)** and **SC(**m**,** max**,** Δ**,** $\mathcal{T}$**)** always output a Δ-cluster $(P, \mathcal{P})$ where $\theta(|P|, |\mathcal{P}|)$ lies on the Pareto Front of $\theta(\cdot\,, \cdot)$. If clusters on the Pareto front of $\theta(\cdot\,, \cdot)$ are good candidates to include in a clustering, then it may be worthwhile to directly compute an approximation of this Pareto front as a candidate set:

▶ **Definition 6.** *Given $(\Delta, \mathcal{T})$, a set of Δ-clusters S is a 2-approximate Pareto front if for every Δ-cluster $(P, \mathcal{P})$ there exists a $(Q, \mathcal{Q}) \in S$ with: $|\mathcal{P}| \leq |\mathcal{Q}|$ and $|P| \leq 2|Q|$.*

Our new algorithm, **PSC(**Δ**,** $\mathcal{T}$**)**, *iterates* over a 2-approximate Pareto front. Storing this front explicitly uses $O(n^2 \log n)$ space. Instead, we generate these clusters on the fly.

High-level description. Let $B(\mathcal{T})$ be a balanced binary tree over $\mathcal{T}$. Let $S(\Delta, \mathcal{T})$ start out as empty. Our algorithm **PSC**$(\Delta, \mathcal{T})$ (defined by Algorithm 3) does the following:

- For each inner node in B, denote by $\mathcal{T}[a, b]$ the corresponding subtrajectory of $\mathcal{T}$.
- For every prefix or suffix P of $\mathcal{T}[a, b]$, let $(P, \mathcal{P})$ be a Δ-cluster maximising $|\mathcal{P}|$.
- We add $(P, \mathcal{P})$ to $S(\Delta, \mathcal{T})$.

Computing all prefix clusters for $\mathcal{T}[a, b]$. Given a trajectory $\mathcal{T}[a, b]$ and our current set $S(\Delta, \mathcal{T})$, we simultaneously compute for all prefixes P of $\mathcal{T}[a, b]$ a maximum-cardinality cluster $(P, \mathcal{P})$ and add this cluster to $S(\Delta, \mathcal{T})$. We define an analogue of Definition 1:

▶ **Definition 7.** *Consider row a in $M_\Delta(\mathcal{T}, \mathcal{T})$. For $(x, y) \in M_\Delta(\mathcal{T}, \mathcal{T})$ let $\texttt{col}_a(x, y)$ be the maximum column $d \leq y$ such that there exists a directed path from (x, y) to (a, d).*

Given $\mathcal{T}[a, b]$ we initialize our algorithm by storing integers a and $c \leftarrow a$. We increment c and maintain for all $y \in [n]$ the value $\texttt{col}_a(c, y)$. This uses $O(n)$ space, as does storing row c. Given this data structure, we alternate between a `StepSecond` and `Query` subroutine:

`StepSecond`: We increment c by 1. For each $y \in [n]$, we check if $d(\mathcal{T}(c), \mathcal{T}(y)) > \Delta$. If it is then (c, y) is an isolated vertex in the graph $M_\Delta(\mathcal{T}, \mathcal{T})$. Thus, there exists no directed path from (c, y) to any vertex in row a and we set $\texttt{col}_a(c, y) = -\infty$. Otherwise, $\texttt{col}_a(c, y) = \max\{\texttt{col}_a(\alpha, \beta) \mid \alpha \in \{c, c-1\}, \beta \in \{y, y-1\}\}$ which we compute in $O(1)$ time.

`Query`: For a prefix $P = \mathcal{T}[a, c]$ of $\mathcal{T}[a, b]$ we compute a maximum-cardinality Δ-cluster $(P, \mathcal{P})$ using the values $\texttt{col}_a(c, y)$ (see Algorithm 4).

Note that Algorithm 4 is essentially the same as the algorithm in [12] that we described in Section 5. Thus, it computes a maximum-cardinality cluster. The key difference is that we, through $\texttt{col}_a(b, j_2)$, avoid walking through the free-space matrix in $O(\ell)$ time to compute j_1. We denote by `Query`'$(T[c, b], \Delta, \mathcal{T}, S(\Delta, \mathcal{T}))$ the function which computes for a suffix $P = \mathcal{T}[c, b]$ of $\mathcal{T}[a, b]$ a maximum-cardinality cluster $(P, \mathcal{P})$ in an analogous manner.

▶ **Lemma 8.** *The set $S(\Delta, \mathcal{T})$ is a 2-approximate Pareto front.*

Proof. Consider any Δ-cluster $(P, \mathcal{P})$. Let $P = \mathcal{T}[a, b]$. Let $\mathcal{T}[i, k]$ be the minimal subtrajectory in our tree that has $\mathcal{T}[a, b]$ as a subtrajectory. Let the children of $\mathcal{T}[i, k]$ in our tree be $\mathcal{T}[i, j]$ and $\mathcal{T}[j + 1, k]$. Then, $a \leq j < b$ by minimality of $\mathcal{T}[i, k]$. Hence, $\mathcal{T}[a, j]$ is a suffix of $\mathcal{T}[i, j]$, and $\mathcal{T}[j + 1, b]$ is a prefix of $\mathcal{T}[j + 1, k]$. There must exist a trajectory $P' \in \{\mathcal{T}[a, j], \mathcal{T}[j + 1, b])\}$ which contains at least half as many vertices as $T[a, b]$. Since P' is either a prefix or suffix of a subtrajectory in our tree, we must have added some maximum-cardinality cluster $(P', \mathcal{P}')$ to $S(\Delta, \mathcal{T})$. Moreover, since P' is a subtrajectory of P, $|\mathcal{P}'| \geq |\mathcal{P}|$ (since any discrete walk from row b to row a is also a discrete walk in the Free-space matrix from row d to row c for $[c, d] \subseteq [a, b]$). ◀

▶ **Theorem 9.** *The algorithm PSC(Δ, $\mathcal{T}$) iterates over a 2-approximate Pareto front of $\mathcal{T}$ in $O(z \log^4 n)$ time using $O(n \log n$ space. Here, z denotes the number of zeroes in $M_\Delta(\mathcal{T}, \mathcal{T})$.*

Proof. Consider the balanced binary tree $B(\mathcal{T})$. Each level of depth of $B(\mathcal{T})$ corresponds to a set of subtrajectories that together cover $\mathcal{T}$. For every such subtrajectory $\mathcal{T}[a, b]$, both `StepSecond` and `Query` get invoked for each row $c \in [a, b]$. Since the tree has depth $\log n$, it follows that we invoke these functions $O(\log n)$ times per row c of $M_\Delta(\mathcal{T}, \mathcal{T})$ (i.e. $O(n \log n)$ times in total). Naively, both the `StepSecond` and `Query` functions iterate over all columns in their given row c. However, just as in Section 5, we note that our functions skip over all (j_2, c) for which $d(\mathcal{T}(j_2), \mathcal{T}(c)) > \Delta$. It follows that we may use our row generation trick to spend $O(Z_c \log^3 n)$ time instead (Z_c denotes the number of zeroes in row c). Since each row c gets processed $O(\log n)$ times, this upper bounds our running time by $O(z \log^4 n)$.

For space, we note that invoking `StepSecond` and `Query` for a row c need only rows c and $c - 1$ in memory. So the total memory usage is $O(n)$. Explicitly storing $S(\Delta, \mathcal{T})$ takes $O(n^2 \log n)$ space. However, since we only iterate over $S(\Delta, \mathcal{T})$, the total space usage remains $O(n)$. The range tree therefore dominates the space with its $O(n \log n)$ space usage. ◀

6.1 Greedy clustering using PSC(Δ, $\mathcal{T}$)

We reconsider Algorithms 1 and 2 in Section 5 for k-centre and k-means clustering. As written, these algorithms invoke a subroutine f that returns a single cluster (P, $\mathcal{P}$). Then, the algorithm computes whether it wants to add (P, $\mathcal{P}$) to the current clustering. We can

Algorithm 3 PSC(Δ, $\mathcal{T}$).

```
$S(\Delta, \mathcal{T}) \leftarrow \emptyset$.
$B(\mathcal{T}) \leftarrow$ balanced binary tree over $\mathcal{T}$.
for all $T[a, b] \in B(\mathcal{T})$ do
    for all $c \in [a, b]$ do
        StepSecond($\Delta$, $\mathcal{T}$, $c$).
        Query($T[a, c]$, $\Delta$, $\mathcal{T}$, $S(\Delta, \mathcal{T})$).
    end for
    for all $c \in [a, b]$ in decreasing order do
        StepSecond($\Delta$, $\mathcal{T}$, $c$).
        Query'($T[c, b]$, $\Delta$, $\mathcal{T}$, $S(\Delta, \mathcal{T})$).
    end for
end for
return $S(\Delta, \mathcal{T})$
```

Algorithm 4 Query($P = \mathcal{T}[a, c]$, Δ, $\mathcal{T}, S(\Delta, \mathcal{T})$).

```
$(c, j_2) \leftarrow M_\Delta(\mathcal{T}, \mathcal{T})[c, n]$, $\mathcal{P} \leftarrow \emptyset$
while $j_2 > 0$ do
    if $\texttt{col}_a(c, j_2) \neq -\infty$ then
        $j_1 \leftarrow \texttt{col}_a(b, j_2)$.
        Add $\mathcal{T}[j_1, j_2]$ to $\mathcal{P}$.
        $(c, j_2) \leftarrow (c, j_1 - 1)$.
    else
        $(c, j_2) \leftarrow (c, j_2 - 1)$.
    end if
end while
Add $(P, \mathcal{P})$ to $S(\Delta, \mathcal{T})$.
```

adapt these algorithms to this new function. For f, we invoke **PSC**(Δ, $\mathcal{T}$). This function iterates over a set of clusters (P, $\mathcal{P}$). For each cluster (P, $\mathcal{P}$) that we encounter, we let either Algorithm 1 or 2 compute whether it wants to add (P, $\mathcal{P}$) to the current clustering.

7 Experiments

We propose three new greedy clustering algorithms for k-centre clustering:

1. SC-ℓ runs Algorithm 1, for exponentially scaling Δ, with $f \leftarrow$**SC(**max**,** ℓ**,** Δ**,** $\mathcal{T}$**)**,
2. SC-m runs Algorithm 1, for exponentially scaling Δ, with $f \leftarrow$ **SC(**m**,** max**,** Δ**,** $\mathcal{T}$**)**, and
3. PSC runs Algorithm 1, for exponentially scaling Δ, with $f \leftarrow$ **PSC**($\Delta, \mathcal{T}$).

Our algorithms SC-ℓ and SC-m need to fix a parameter ℓ and m respectively. We choose $\ell \in \{4, 8, 16, 32, 64\}$ and $m \in \{2, 4, 8, 16\}$ and do a separate run for each choice. Thus, together with PSC, we run ten different algorithm configurations. We additionally compare to the subtrajectory clustering algorithm from [10], which is oblivious to the scoring function.

- `Map-construction` runs the algorithm in [10]. We terminate this map construction algorithm as soon as it has constructed a clustering of $\mathcal{T}$.

k-means clustering. For k-means clustering, we obtain three algorithms by running Algorithm 2 instead. Note that our clustering algorithms invoked Algorithm 1 across several runs for exponentially scaling Δ. Our clustering algorithms do not invoke Algorithm 2 in this manner. Instead, Algorithm 2 considers exponentially scaling Δ's internally. Our experiments show that this heavily impacts the clustering's performance on large input.

We again obtain ten algorithm configurations. In addition, we run the map construction algorithm from [10] and the k-means subtrajectory clustering algorithms from [1]:

- `Envelope` runs the algorithm from [1] with the scoring parameters (c_1, c_2, c_3).

Experimental setup. We design different experiments to measure performance and the scoring function. For performance, we run fourteen experiments across the five real-world data sets. Seven experiments for k-centre clustering, and seven for k-means clustering. Each experiment ran three times, using a different vector (c_1, c_2, c_3). Each experiment runs 11 programs for k-centre clustering, or 12 for k-means clustering. If a program fails to terminate after 24 hours, we do not extract any output or memory usage.

We restrict scoring measurements to the three smaller or subsampled real-world data sets. This is necessary to make `Map-construct` and `Envelope` terminate. We add six synthetic data sets. We only compare under k-means clustering since only our algorithms can optimise for k-centre clustering. The experiments were conducted on a machine with a 4.2GHz AMD Ryzen 9 7950X3D and 128GB memory. Over 90 runs hit the timeout. Running the total set of experiments sequentially takes over 100 days on this machine.

The full results per experiment are organised in tables in the full version. Here, we restrict our attention to only three plots.

Choosing (c_1, c_2, c_3). Recall that c_1 weighs the number of clusters in the clustering and that c_3 weighs the fraction of uncovered points. We follow the precedent set in [1] and choose $c_1 = 1$, and c_3 equal to the number of trajectories in $\mathcal{T}$. Finally, c_2 weighs the Fréchet distance across clusters. For each data set, we first find a choice for c_2 such the algorithms lead to a non-trivial clustering. We then also run the algorithms using the scoring vectors $(c_1, 10 \cdot c_2, c_3)$ and $(c_1, 0.1 \cdot c_2, c_3)$, for a total of three scoring vectors per data set.

Comparing quality. To compare solution quality, we measure the scoring function and other qualities such as: the total number of clusters in the clustering, the maximum Fréchet distance from a subtrajectory to its centre, and the average Fréchet distance across clusters. Our competitor algorithms frequently time out on the real-world data sets. Hence, the qualitative comparison also makes use of our synthetic data set. We use three different configurations of our synthetic set to create input of different size. We choose to compare only under k-means clustering for a more fair comparison to other algorithms. Indeed, the algorithm by [1] was designed for the k-means metric ([10] is scoring function oblivious). Consequently, comparing under k-centre would only give us an unfair advantage.

7.1 Results for k-means clustering

The bar plot in Figure 6 visualizes how the algorithms score. Figures 7 and 8 show, respectively, a logarithmic-scale plot of the time and space usage. These plots use the "medium" choice for c_2 for each data set. Note that our choice of c_2 is very similar across the data sets in the figure. This is because *Athens*, *Berlin* and *Chicago* come from the same source, and thereby have the same scaling. We subsequently designed our synthetic data to match this scaling. Plots for the other scoring vector choices are contained in the full version.

Clustering quality. Figure 6 shows that `Map-construct` (which is oblivious to the scoring function) scores poorly on all inputs. One reasons for this is that it always outputs a large number of clusters. The `Envelope` algorithm, which was specifically designed for this scoring function, almost always obtains the best score.

For any data set, for any scoring vector, we always find a choice for ℓ (or m) where SC-ℓ and SC-m obtain a comparable score to `Envelope`. The algorithm `PSC` scores best out of all newly proposed clustering algorithms. Across all data sets and scoring functions it frequently competes with, and sometimes even improves upon, the score of `Envelope`.

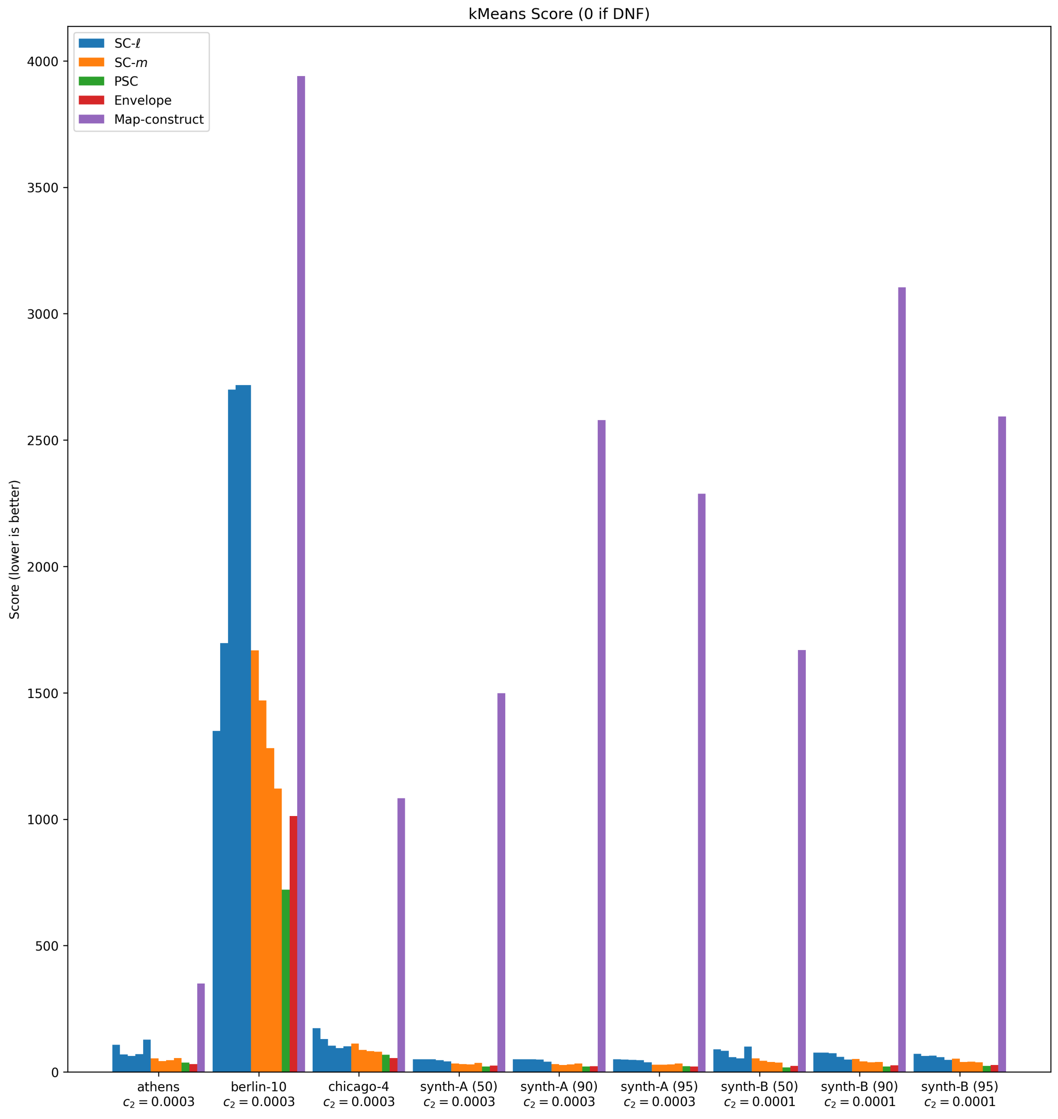

Figure 6 ■ k-means clustering ■ k-means score ■.

Other quality metrics. We briefly note that the full data in the appendix also considers other quality metrics: the number of clusters and the maximum and average Fréchet distance within a cluster. Perhaps surprisingly, `Map-construct` does not perform better on the latter two metrics even though it outputs many clusters. `Envelope` scores worse on these metrics across nearly all data sets and scoring vectors. However, it stays within a competitive range.

Runtime comparison. Figure 7 shows the runtime performance in logarithmic scale. On the *Berlin* data set, no algorithm terminates in time. On *Drifter* and *UniD*, only `PSC` terminates within the time limit. In the figure, we consider the synthetic data sets their place.

`Envelope` may produce high-quality clusterings, but takes significant time to do so. Despite using considerably less memory than `Map-construct`, it is consistently more than a factor two slower. Our algorithms `SC-`ℓ and `SC-`m can always find a choice of ℓ or m such that previous algorithms are orders of magnitude slower. Our `PSC` algorithm is always orders of magnitude faster than the existing clustering algorithms. This is especially true on larger data sets, which illustrates that our algorithms have better scaling.

When we compare our own algorithms, we note that we can frequently find a choice for ℓ such that `SC`-ℓ is significantly faster than `SC`-m and `PSC`. However, when considering the runtime to quality ratio, `PSC` appears to be a clear winner.

Memory consumption. Figure 8 shows memory performance on a logarithmic scale. The figure excludes *Drifter*, *UniD* and *Berlin* for the same reason as when comparing running times. We record zero memory usage whenever an algorithm times out.

We observe that `Map-construct` uses an order of magnitude more memory than its competitors. This is because it stores up to $O(n^2)$ candidate clusters in memory, together with the entire free-space matrix $M_\Delta(\mathcal{T}, \mathcal{T})$ for various choices of Δ. `Envelope` also has a large memory footprint and uses at least a factor 100 more memory on all of the inputs.

When comparing our own implementations, we observe that `SC`-ℓ and `PSC` have a very small memory footprint. These algorithms use $O(n\ell)$ and $O(n)$ space. Since we fix ℓ to be a small constant, both algorithms use near linear space. The space usage of `SC`-m is $O(n\ell)$ where ℓ is the length of the longest subtrajectory P such that there exists a Δ-cluster $(P, \mathcal{P})$ with $|\mathcal{P}| \geq m$. The data shows that for many choices of (m, Δ), this $O(n\ell)$ space is considerable. Yet, it is still considerably lower than pre-existing algorithms.

7.2 Results for k-centre clustering

When comparing score, runtime, or memory usage under k-centre clustering, we reach the same conclusions. The only difference is that the performance of `SC`-m becomes more competitive to `SC`-ℓ and `PSC` on large data sets. This is because **SC**(m, max, Δ, $\mathcal{T}$) uses much time and space whenever both the data set and Δ are large. Our k-means clustering meta-algorithm considers a large Δ each round, and invokes **SC**(m, max, Δ, $\mathcal{T}$) every round.

Our k-centre clustering meta-algorithm has a separate run for each Δ. If Δ is large, it terminates quickly because it adds very high-cardinality clusters to the clustering. This speeds up all our implementations, but `SC`-m the most.

8 Conclusion

We proposed new subtrajectory clustering algorithms. Our first two approaches use the algorithms **SC**(max, ℓ, Δ, $\mathcal{T}$) and **SC**(m, max, Δ, $\mathcal{T}$) by BBGLL [12]. We use these algorithms as a subroutine in a greedy clustering algorithm to cluster a set of trajectories $\mathcal{T}$.

Consider the Boolean function $\theta(\ell, m)$ that outputs `true` if there exists a Δ-cluster $(P, \mathcal{P})$ of $\mathcal{T}$ with $|P| = \ell$ and $|\mathcal{P}| = m$. We observe that these functions always output a point on the Pareto front of this function. We show a new algorithm **PSC**(Δ, $\mathcal{T}$) that, instead of returning a single cluster on this Pareto front, iterates over a 2-approximation of this Pareto front instead. Intuitively, this function iterates over a broader set of candidate clusters that have comparable quality to any output of **SC**(max, ℓ, Δ, $\mathcal{T}$) and **SC**(m, max, Δ, $\mathcal{T}$). We create a greedy clustering algorithm `PSC` using this algorithm as a subroutine.

Our analysis shows that our new clustering algorithms significantly outperform previous algorithms in running time and space usage. We primarily use the scoring function from [1] to measure clustering quality and observe that our algorithms give competitive scores. For other quality metrics, our algorithms often perform better than their competitors. We observe that `PSC` is the best algorithm when comparing the ratio between score and performance.

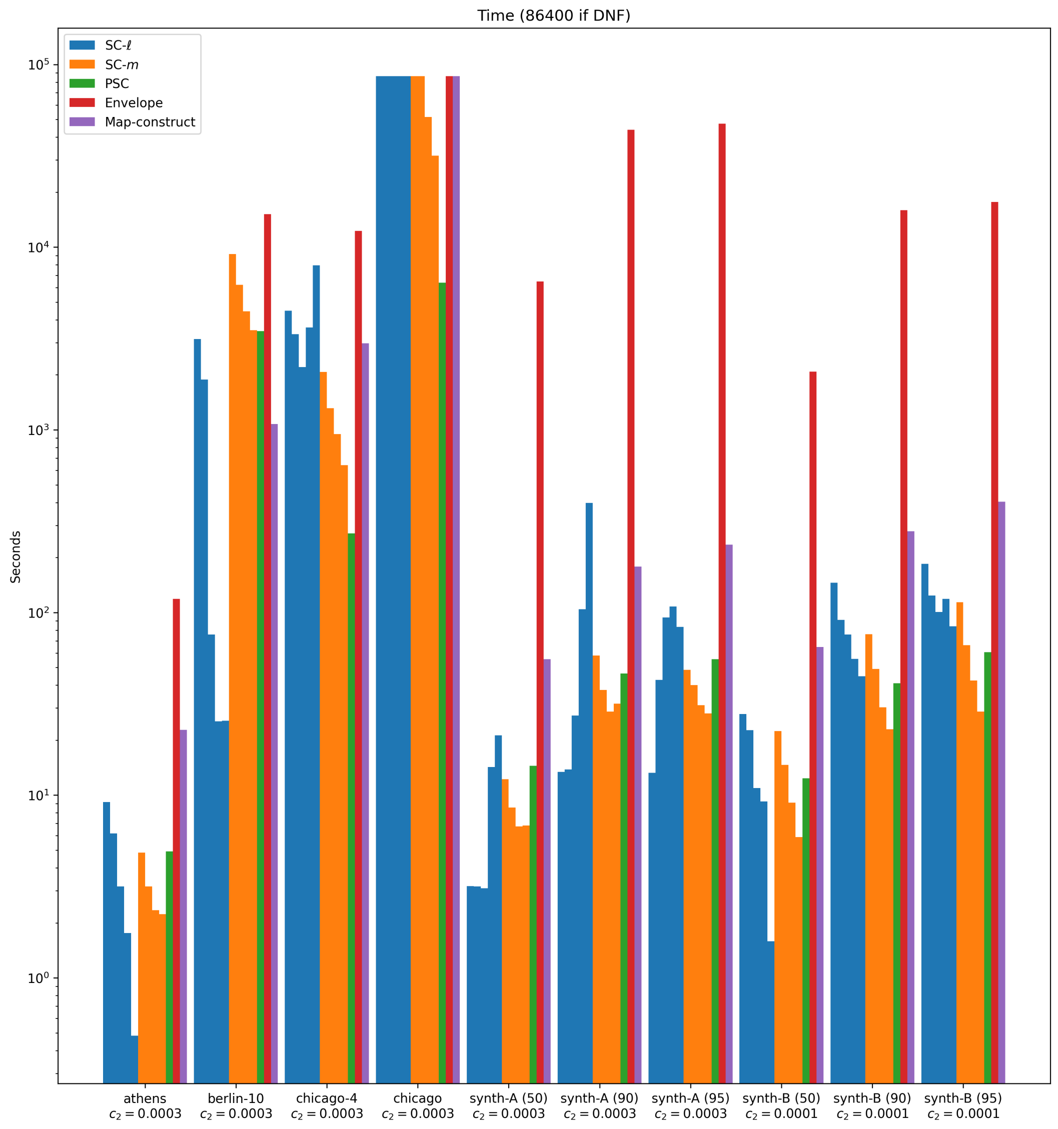

Figure 7 ■ k-means clustering ■ Running time ■ Logarithmic scaling.

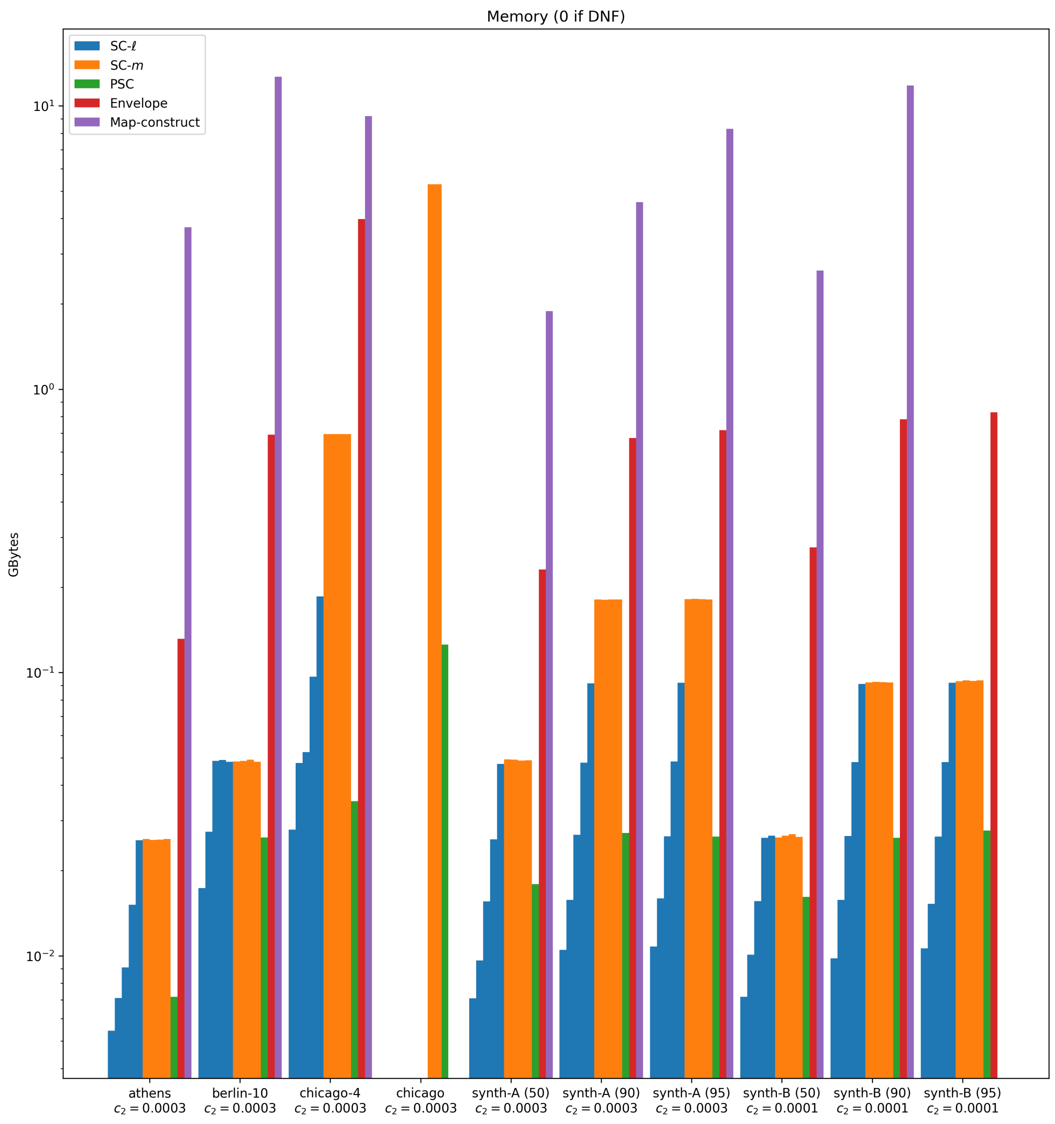

Figure 8 ■ k-means clustering ■ Memory usage ■ Logarithmic scaling
Whenever a run did not terminate, we display a memory usage of zero.

References

1 Pankaj K Agarwal, Kyle Fox, Kamesh Munagala, Abhinandan Nath, Jiangwei Pan, and Erin Taylor. Subtrajectory clustering: Models and algorithms. *ACM SIGMOD-SIGACT-SIGAI Symposium on Principles of Database Systems (PODS)*, pages 75–87, 2018. doi:10.1145/3196959.3196972.

2 Mahmuda Ahmed, Sophia Karagiorgou, Dieter Pfoser, and Carola Wenk. A comparison and evaluation of map construction algorithms using vehicle tracking data. *GeoInformatica*, 19:601–632, 2015. doi:10.1007/s10707-014-0222-6.

3 Mahmuda Ahmed and Carola Wenk. Constructing street networks from gps trajectories. *European Symposium on Algorithms (ESA)*, pages 60–71, 2012. doi:10.1007/978-3-642-33090-2_7.

4 Hugo Akitaya, Frederik Brüning, Erin Chambers, and Anne Driemel. Subtrajectory Clustering: Finding Set Covers for Set Systems of Subcurves. *Computing in Geometry and Topology*, 2(1):1:1–1:48, 2023. doi:10.57717/cgt.v2i1.7.

5 Helmut Alt and Michael Godau. Computing the fréchet distance between two polygonal curves. *International Journal of Computational Geometry & Applications*, 5:75–91, 1995. doi:10.1142/S0218195995000064.

6 James Biagioni and Jakob Eriksson. Map inference in the face of noise and disparity. *ACM International Conference on Advances in Geographic Information Systems (SIGSPATIAL)*, pages 79–88, 2012. doi:10.1145/2424321.2424333.

7 Jiang Bian, Dayong Tian, Yuanyan Tang, and Dacheng Tao. A survey on trajectory clustering analysis. *arXiv preprint arXiv:1802.06971*, 2018. arXiv:1802.06971.

8 Frederik Brüning, Jacobus Conradi, and Anne Driemel. Faster Approximate Covering of Subcurves Under the Fréchet Distance. In Shiri Chechik, Gonzalo Navarro, Eva Rotenberg, and Grzegorz Herman, editors, *European Symposium on Algorithms (ESA 2022)*, volume 244 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 28:1–28:16, Dagstuhl, Germany, 2022. Schloss Dagstuhl – Leibniz-Zentrum für Informatik. doi:10.4230/LIPIcs.ESA.2022.28.

9 Kevin Buchin, Maike Buchin, David Duran, Brittany Terese Fasy, Roel Jacobs, Vera Sacristan, Rodrigo I Silveira, Frank Staals, and Carola Wenk. Clustering trajectories for map construction. *ACM International Conference on Advances in Geographic Information Systems (SIGSPATIAL)*, pages 1–10, 2017. doi:10.1145/3139958.3139964.

10 Kevin Buchin, Maike Buchin, Joachim Gudmundsson, Jorren Hendriks, Erfan Hosseini Sereshgi, Vera Sacristán, Rodrigo I Silveira, Jorrick Sleijster, Frank Staals, and Carola Wenk. Improved map construction using subtrajectory clustering. *ACM Workshop on Location-Based Recommendations, Geosocial Networks, and Geoadvertising (SIGSPATIAL)*, pages 1–4, 2020. doi:10.1145/3423334.3431451.

11 Kevin Buchin, Maike Buchin, Joachim Gudmundsson, Jorren Hendriks, Erfan Hosseini Sereshgi, Rodrigo I Silveira, Jorrick Sleijster, Frank Staals, and Carola Wenk. Roadster: Improved algorithms for subtrajectory clustering and map construction. *Computers & Geosciences*, 196:105845, 2025. doi:10.1016/j.cageo.2024.105845.

12 Kevin Buchin, Maike Buchin, Joachim Gudmundsson, Maarten Löffler, and Jun Luo. Detecting commuting patterns by clustering subtrajectories. *International Journal of Computational Geometry & Applications*, 21(03):253–282, 2011. doi:10.1142/S0218195911003638.

13 Kevin Buchin, Maike Buchin, Marc Van Kreveld, Maarten Löffler, Rodrigo I Silveira, Carola Wenk, and Lionov Wiratma. Median trajectories. *Algorithmica*, 66:595–614, 2013. doi:10.1007/s00453-012-9654-2.

14 Lili Cao and John Krumm. From gps traces to a routable road map. *ACM international conference on advances in geographic information systems (SIGSPATIAL)*, pages 3–12, 2009. doi:10.1145/1653771.1653776.

15 Chen Chen, Hao Su, Qixing Huang, Lin Zhang, and Leonidas Guibas. Pathlet learning for compressing and planning trajectories. *ACM International Conference on Advances in Geographic Information Systems (SIGSPATIAL)*, pages 392–395, 2013. doi:10.1145/2525314.2525443.

16 Jacobus Conradi and Anne Driemel. Finding complex patterns in trajectory data via geometric set cover. *arXiv preprint arXiv:2308.14865*, 2023. doi:10.48550/arXiv.2308.14865.

17 Jonathan J Davies, Alastair R Beresford, and Andy Hopper. Scalable, distributed, real-time map generation. *IEEE Pervasive Computing*, 5(4):47–54, 2006. doi:10.1109/MPRV.2006.83.

18 university of Technology Eindhoven. MoveTK: the movement toolkit. http://https://github.com/movetk/movetk, 2020.

19 Scott Gaffney and Padhraic Smyth. Trajectory clustering with mixtures of regression models. *ACM International Conference on Knowledge Discovery and Data mining (SIGKDD)*, pages 63–72, 1999. doi:10.1145/312129.312197.

20 Joachim Gudmundsson, Andreas Thom, and Jan Vahrenhold. Of motifs and goals: mining trajectory data. *ACM International Conference on Advances in Geographic Information Systems (SIGSPATIAL)*, pages 129–138, 2012. doi:10.1145/2424321.2424339.

21 Joachim Gudmundsson and Nacho Valladares. A GPU approach to subtrajectory clustering using the fréchet distance. *ACM nternational Conference on Advances in Geographic Information Systems (SIGSPATIAL)*, pages 259–268, 2012. doi:10.1145/2424321.2424355.

22 Joachim Gudmundsson and Sampson Wong. Cubic upper and lower bounds for subtrajectory clustering under the continuous fréchet distance. *ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 173–189, 2022. doi:10.1137/1.9781611977073.9.

23 Jincai Huang, Min Deng, Jianbo Tang, Shuling Hu, Huimin Liu, Sembeto Wariyo, and Jinqiang He. Automatic generation of road maps from low quality gps trajectory data via structure learning. *IEEE Access*, 6:71965–71975, 2018. doi:10.1109/ACCESS.2018.2882581.

24 Chih-Chieh Hung, Wen-Chih Peng, and Wang-Chien Lee. Clustering and aggregating clues of trajectories for mining trajectory patterns and routes. *The VLDB Journal*, 24:169–192, 2015. doi:10.1007/s00778-011-0262-6.

25 Sophia Karagiorgou and Dieter Pfoser. On vehicle tracking data-based road network generation. *ACM International Conference on Advances in Geographic Information Systems (SIGSPATIAL)*, pages 89–98, 2012. doi:10.1145/2424321.2424334.

26 Sophia Karagiorgou, Dieter Pfoser, and Dimitrios Skoutas. Segmentation-based road network construction. *ACM International Conference on Advances in Geographic Information Systems (SIGSPATIAL)*, pages 460–463, 2013. doi:10.1145/2525314.2525460.

27 Jae-Gil Lee, Jiawei Han, and Kyu-Young Whang. Trajectory clustering: a partition-and-group framework. *ACM International Conference on Management of Data (SIGMOD)*, pages 593–604, 2007. doi:10.1145/1247480.1247546.

28 Vanessa Lago Machado, Ronaldo dos Santos Mello, Vânia Bogorny, and Geomar André Schreiner. A survey on the computation of representative trajectories. *GeoInformatica*, pages 1–26, 2024. doi:10.1007/s10707-024-00514-y.

29 Tobias Moers, Lennart Vater, Robert Krajewski, Julian Bock, Adrian Zlocki, and Lutz Eckstein. The exid dataset: A real-world trajectory dataset of highly interactive highway scenarios in germany. *2022 IEEE Intelligent Vehicles Symposium (IV)*, pages 958–964, 2022. doi:10.1109/IV51971.2022.9827305.

30 Cynthia Sung, Dan Feldman, and Daniela Rus. Trajectory clustering for motion prediction. *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 1547–1552, 2012. doi:10.1109/IROS.2012.6386017.

31 Suyi Wang, Yusu Wang, and Yanjie Li. Efficient map reconstruction and augmentation via topological methods. *ACM International Conference on Advances in Geographic Information Systems (SIGSPATIAL)*, pages 1–10, 2015. doi:10.1145/2820783.2820833.