Subtrajectory Clustering: Models and Algorithms*

Pankaj K. Agarwal
Duke University
pankaj@cs.duke.edu

Kyle Fox
The University of Texas at Dallas
kyle.fox@utdallas.edu

Kamesh Munagala
Duke University
kamesh@cs.duke.edu

Abhinandan Nath
Duke University
abhinath@cs.duke.edu

Jiangwei Pan
Facebook
panjiangwei@gmail.com

Erin Taylor
Duke University
erin.c.taylor@duke.edu

ABSTRACT

We propose a model for *subtrajectory clustering*—the clustering of subsequences of trajectories; each cluster of subtrajectories is represented as a *pathlet*, a sequence of points that is not necessarily a subsequence of an input trajectory. Given a set of trajectories, our clustering model attempts to capture the shared portions between them by assuming each trajectory is a concatenation of a small set of pathlets, with possible gaps in between.

We present a single objective function for finding the optimal collection of pathlets that best represents the trajectories taking into account noise and other artifacts of the data. We show that the subtrajectory clustering problem is NP-Hard and present fast approximation algorithms for subtrajectory clustering. We further improve the running time of our algorithm if the input trajectories are "well-behaved."

Finally, we present experimental results on both real and synthetic data sets. We show via visualization and quantitative analysis that the algorithm indeed handles the desiderata of being robust to variations, being efficient and accurate, and being data-driven.

CCS CONCEPTS

• **Theory of computation → Facility location and clustering**; **Computational geometry**; *Packing and covering problems*; Data compression; Data structures and algorithms for data management;

KEYWORDS

Trajectory data; subtrajectory clustering; greedy algorithm

ACM Reference Format:
Pankaj K. Agarwal, Kyle Fox, Kamesh Munagala, Abhinandan Nath, Jiangwei Pan, and Erin Taylor. 2018. Subtrajectory Clustering: Models and Algorithms. In *PODS'18: 35th ACM SIGMOD-SIGACT-SIGAI Symposium on Principles of Database Systems, June 10–15, 2018, Houston, TX, USA.* ACM, New York, NY, USA, 13 pages. https://doi.org/10.1145/3196959.3196972

*Work on this paper is supported by NSF under grants CCF-15-13816, CCF-15-46392, CCF-14-08784, CCF-16-37397, IIS-14-47554, and IIS-14-08846, by ARO grant W911NF-15-1-0408, and by grant 2012/229 from the U.S.-Israel Binational Science Foundation. Work by Kyle Fox (resp. Jiangwei Pan) was done when he was a postdoc (resp. graduate student) at Duke University.

PODS'18, June 10–15, 2018, Houston, TX, USA

ACM ISBN 978-1-4503-4706-8/18/06…$15.00
https://doi.org/10.1145/3196959.3196972

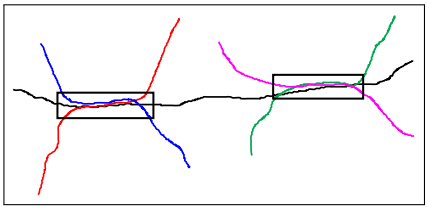

Figure 1. The middle black trajectory shares commonalities with different sets of trajectories, as shown in the boxed regions. The subtrajectories inside these boxes can be clustered together, and a representative pathlet for each cluster computed so that the black trajectory can be (almost) obtained by a concatenation of these pathlets, with gaps in between.

1 INTRODUCTION

Trajectories arise in the description of any dynamic physical system. They are being recorded or inferred from millions of sensors at an unprecedented scale. To take full benefit of the enormous opportunities provided by these datasets for extracting useful information and improving decision making, several computational challenges need to be addressed. This paper focuses on one such computational problem, namely extracting high-level shared structure that encodes much of the information present in a large trajectory dataset. For specificity, we focus on GPS traces of moving objects, though much of the work here can be extended to many other domains. We assume that each trajectory is observed as a sequence of points. For simplicity, we refer to the point sequences themselves as trajectories. For such trajectories, common but unknown constraints and objectives generate patterns of *subtrajectories*, portions of trajectories that are commonly shared. See Figure 1 for an example. Our goal is to cluster such shared subtrajectories and represent each cluster as a *pathlet*, a sequence of points that is not necessarily a subsequence of an input trajectory. For example, a road network contributes to shared subtrajectories for vehicle trajectories. Intuitively, a pathlet is a portion of a road/path that tends to be traversed *as a whole* by many trajectories. The *subtrajectory clustering* problem is to partition shared subtrajectories into a small number of clusters such that the pathlet for each cluster is a high-quality representation of subtrajectories in each cluster.

Subtrajectory clustering is an interesting trajectory segmentation problem from both a modeling and algorithmic standpoint. Individual trajectories carry little information about shared structures, and it is only in the context of a collection of trajectories that the importance of certain subtrajectories becomes evident. This

situation is different from some other cases where shared structures (motifs) are extracted from a set of curves, such as protein backbones or speech signals. In both of these cases, motifs are more structured, and domain knowledge provides enough information to identify common substructures in each curve, such as secondary structures (α-helices and β-sheets) in a protein backbone.

If trajectories correspond to traffic on a known road network, then these trajectories can be mapped to paths on a graph that models the road network. However, in many cases there might not be an underlying road network or it might not be known, e.g., trajectories corresponding to pedestrian movement in an urban environment or pixels in video data. Therefore, we focus on unmapped trajectory point sequences in $\mathbb{R}^2$ sampled at discrete time stamps.

Not only do pathlets provide a compression of large trajectory data, effectively performing non-linear dimension reduction, the semantic information provided by pathlets have been useful for trajectory analysis applications such as similarity search and anomaly detection [34]. Furthermore, a pathlet representation of trajectories reduces uncertainty in individual trajectories; it reduces noise, fills missing data, and identifies outliers (see Figure 8) [25, 26]. Pathlets have also been used for many other applications such as reconstructing a road network from trajectory data [9, 25].

1.1 Our Contributions

There are three main contributions of the paper – a simple model for subtrajectory clustering; efficient approximation algorithms with provable guarantees on the quality of solution and running time under this model; and experimental results that demonstrate the efficacy and efficiency of our model and algorithms, respectively.

Modeling. Our first main contribution, given in Section 2, is a simple model for subtrajectory clustering. In our model, each cluster of subtrajectories is represented as a *pathlet*, a sequence of points. A subtrajectory clustering is a set $\mathcal{P}$ of pathlets along with an assignment of subtrajectories to pathlets in $\mathcal{P}$. We use a distance function between two point sequences (e.g., discrete Fréchet distance [6]) to measure how well a pathlet represents (covers) a subtrajectory.

Such a model is faithful to the following three desiderata. As discussed in Section 1.2, though related work has considered some of these desiderata, we believe we are the first to address all of these simultaneously.

Robustness to variations. We allow trajectories to have *gaps*, i.e., we do not require that the entire trajectory is covered by pathlets. This serves two purposes – in addition to handling noisy portions of trajectories better (akin to how gaps in DNA sequence alignment model mutations [14]), these gaps also model those portions of a trajectory that may be unique to it and not shared with other trajectories (see Figure 1 and Section 7).

Theoretical guarantees. We define a single objective function that is a weighted sum of the number of pathlets chosen, the cluster quality for each pathlet, and the gap penalty for each trajectory. By varying weights, one can obtain a trade-off between the three criteria. An optimal subtrajectory clustering that minimizes the objective should result in a small number of good quality clusters (pathlets) and few gaps. Further, one should be able to cover shared portions of an input trajectory by a short sequence of pathlets (see Figure 9). As we show, such an objective admits to efficient approximation algorithms.

Data-driven clustering. Given the above objective, the number of pathlets chosen, the subtrajectory assignments, and the gaps in the trajectories are all decided by the algorithm in a data-driven fashion, largely independent of any user input. Further, we assume that the set $\mathbb{P}$ of candidate pathlets from which the set $\mathcal{P}$ of pathlets is constructed is the set of all possible point sequences, and we do not make any *a priori* restrictions on what the pathlets can look like. In this sense, our model is entirely *data-driven*.

In the subtrajectory-clustering problem, we assume that the candidate pathlets is the set of all possible point sequences. We also consider the variant in which $\mathcal{P}$ is chosen from a given finite set $\mathbb{P}$ of b pathlets. We refer to this variant as the *pathlet-cover* problem. Besides being interesting in its own right, our subtrajectory-clustering algorithm will need an algorithm for this problem. For both variants, we are given a set $\mathcal{T}$ of n trajectories containing m points in total.

Algorithms. We show (Section 3) that both pathlet cover and subtrajectory clustering are NP-Hard by a simple reduction from the standard Facility-Location problem in Euclidean space.

We then turn to our main algorithmic results – approximation algorithms for pathlet cover and subtrajectory clustering. Our algorithm for pathlet-cover (Section 4) finds an $O(\log m)$-approximate pathlet dictionary in $\tilde{O}(bm^3)$ time[1]. To obtain this result, we show that the construction of $\mathcal{P}$ can be formulated as an instance of the weighted Set-Cover problem so that a greedy algorithm can be used to construct a pathlet dictionary of the appropriate cost. However, the size of the set system can be as large as $\Omega(2^n)$. The main technical contribution is to show how the greedy algorithm can be executed efficiently without constructing the set system explicitly; a similar idea was employed in inventory management [27].

Next we describe approximation algorithms for subtrajectory clustering (Section 5). Here we use discrete Fréchet distance, a widely used method for measuring similarity between two point sequences, to measure the quality of a pathlet covering a subtrajectory; see Section 5 for the formal definition. By a simple reduction to the pathlet-cover problem, we obtain a polynomial-time $O(\log m)$-approximation. We simply use the set of all $O(m^2)$ contiguous subtrajectories of input trajectories as the candidate set $\mathbb{P}$. However, such a "brute-force" enumeration of candidate pathlets makes the running time prohibitive. In essence, this algorithm considers too many possible assignments between subtrajectories and candidate pathlets, and neither takes full advantage of the freedom in our choice of pathlets nor does it exploit the underlying geometry fully.

We first show via a geometric grouping argument that by paying an extra $\log m$ factor in the approximation quality, we can work with a set $\mathbb{P}$ of $O(m)$ candidate pathlets. We then describe how to expedite the algorithm further and prove that if the input trajectories are κ-packed [2] for a constant $\kappa > 0$, then the algorithm runs in $\tilde{O}(m^2)$ time. Intuitively, a trajectory is κ-packed if it is not a space-filling

[1] We use the notation $\tilde{O}$ to hide polylogarithmic factors in n, m, b, and σ, the *spread* of the trajectories' points. The spread of a point set is defined as the ratio of largest to smallest pairwise distance. The exponent on the polylog may depend upon the dimension d of the underlying space when our algorithms are generalized beyond $\mathbb{R}^2$.

[2] A trajectory is κ-packed if the length of its intersection with any disk of radius r is at most κr. See Section 5 for details.

curve or a fractal. In practice, GPS trajectories are κ-packed with small values of κ; see Section 7 and [38]. The main technical result, which is quite surprising, is to show that for κ-packed curves for constant κ, the number of subtrajectories of any trajectory that can be mapped to a single pathlet is $\tilde{O}(1)$; this pruning of subtrajectories also leads to a more robust solution. By use of appropriate data structures, the corresponding steps of the greedy algorithm can therefore be significantly speeded up.

Preprocessing and Experiments. We briefly discuss two additional preprocessing steps that improve our algorithms' performance in practice (Section 6). The first step is designed to handle lossy or sparse data sets for which many trajectories are only partially observed. We describe a method to fill in the missing observations not through a simple linear interpolation, but instead using observations from other input trajectories. The second step is designed to even further reduce the size of the set $\mathbb{P}$ of candidate pathlets. Our method sparsifies the set $\mathbb{P}$ by first projecting them to an Euclidean space and then applying a clustering technique to remove redundant candidate pathlets.

We conclude with an empirical evaluation of our algorithms and the properties of our model in Section 7. We perform our experiments on both synthetic and real data sets, and show that the algorithm handles the desiderata of being robust to variations, being efficient and accurate, and being data-driven. In particular, it finds pathlets that cover the dense areas of the data sets, and individual trajectories can be recovered from a subset of the pathlets with few gaps that capture variations in individual trajectories. We also show that meaningful pathlets can be found even on instances where there is no underlying road network, and further, the resulting pathlets gracefully handle noise in the individual trajectories. In addition to examining the quality of our algorithm's output visually, we also demonstrate that the parameters can be tuned for different balances between dictionary size, coverage of trajectories, and similarity between pathlets and their assigned subtrajectories.

1.2 Prior Work

Though there has been a recent line of work on subtrajectory clustering [10, 15, 21, 24, 34, 40], all of this only considers a subset of the desiderata we consider. In particular, none of these algorithms provide any guarantee on their performance in the worst case. In contrast, our goal was to develop an algorithm with provable guarantees on its performance. Further, either the works aim for complete coverage without the notion of gaps, or are not data-driven, imposing particular structure on either the trajectories or pathlets. In some more detail, the algorithm in Chen *et al.* [15] assumes the trajectories to be paths in a graph with no noise and they do not consider gaps. Though the approach in Panagiotakis *et al.* [29] appears superficially similar to ours, their method segments trajectories based on the representativeness of individual trajectory edges, which is based on density near the edge (irrespective of which trajectories contribute to the density). The algorithm in Lee *et al.* [24] requires pathlets to be line segments, and the algorithm in Sankararaman *et al.* [33] is effective only if data is dense, not too noisy, and not too big. The algorithm in Buchin *et al.* [10] requires the user to specify two of the following three parameters – size, length or diameter of the cluster; their algorithm finds the single cluster that optimizes for the third. Their approach is slightly modified in [9] to detect multiple subtrajectory clusters from trajectories; however they assume the trajectories are points sampled from an unknown, underlying road network; and they do not have a notion of gaps. In summary, our work encompasses the desiderata that previous work either partially covers or omits entirely.

There is a fair bit of work on extracting common movement patterns. This line of work either pre-specifies the portion of trajectories where to find a pattern or pre-specifies the pattern. For example, given a set of trajectories and a query time-interval, Pelekis *et al.* [30, 31] cluster the subtrajectories having points lying in the query interval. This is equivalent to clustering trajectories, obtained by restricting each trajectory in the query interval. The algorithms in [19, 35, 36, 39] search for pre-defined patterns, such as groups or crowds, in trajectory data. In contrast, our goal is to identify shared structures determined by the data, while being robust to noise, missing data, and non-uniform sampling.

The work on multiple sequence alignment (MSA) in computational biology [28], on functional clustering in statistics [32], and on topic modeling [7] or dictionary learning [18] in machine learning and signal processing focus on identifying shared structures, and thus relate to the problem studied here. However, MSA deals with one dimensional sequences over a finite alphabet, and thus the setting is much simpler. The work on functional clustering assumes a (known) global parametrization over the data, and the range of functions is one dimensional which makes the problem simpler. Topic modeling/dictionary learning if applied to trajectory data will not return subtrajectories as "topics" or words in the dictionary, as they do not concern themselves with locality.

Finally, there is work on computing a mean or a median trajectory as a representative of a given set of "similar" trajectories [11], computing basis trajectories to recover structure from motion [5], and segmenting individual trajectories using certain geometric criteria [13]. The work on trajectory clustering [20, 22, 37] focuses on partitioning a set of trajectories into clusters of similar trajectories, and possibly computing a representative trajectory for each cluster using the aforementioned work. This line of work, however, does not discover shared structures among otherwise dissimilar trajectories. There is some work on partial matching between a pair of trajectories, i.e., finding most similar subtrajectories between two trajectories [12].

2 THE CLUSTERING MODEL

A trajectory or pathlet is a polygonal curve defined by a finite sequence of points $\langle p_1, p_2, \ldots \rangle$ in $\mathbb{R}^2$ (again, our results can be generalized to trajectories in $\mathbb{R}^d$ for any constant d). Let $\mathcal{T} = \{T_1, \ldots, T_n\}$ be a set of n trajectories and $\mathbb{P} = \{P_1, \ldots, P_b\}$ be a set of b candidate pathlets. For simplicity, we assume the points in each trajectory are distinct and let $\mathbb{X} = \bigcup_{i=1}^{n} T_i$ be the set of all trajectory points. Set $m = |\mathbb{X}|$. Let $T[p, q]$ denote the subtrajectory of T lying between points p and q of T. For positive integers i, j, let $T(i, j)$ denote $T[p_i, p_j]$.

A subtrajectory *clustering* is a *pathlet dictionary*, that is, a (multi) subset $\mathcal{P} \subseteq \mathbb{P}$, along with an assignment of a subset $\mathcal{T}(P)$ of subtrajectories to each $P \in \mathcal{P}$ such that there is at most one subtrajectory

$S \in \mathcal{T}(P)$ of each trajectory $T \in \mathcal{T}$. In turn, for each trajectory $T \in \mathcal{T}$, we let $T_P \in \mathcal{T}(P)$ denote the subtrajectory of T assigned to P, assuming one exists. We say that each $S \in \mathcal{T}(P)$ is *assigned to* or *covered by* P. We want to compute a multi set of pathlets $\mathcal{P}$ (and their assignments)[3] that is succinct and captures the shared portions between trajectories.

In our model, a good clustering minimizes an objective function consisting of three terms. The first term is proportional to $|\mathcal{P}|$, the number of pathlets. The second term consists of the fraction of points of each trajectory that are not assigned to any pathlet in $\mathcal{P}$, summed over all trajectories in $\mathcal{T}$. In other words, the second term measures the size of the *gaps* left uncovered by $\mathcal{P}$. We focus on the fraction of uncovered points in each trajectory instead of the absolute number, because we do not wish to optimize only for a small number of especially long or densely sampled trajectories. The third term captures how well the assigned subtrajectories resemble the pathlets in the dictionary.

To formally define the third term in our objective, we use a distance function, denoted by $\mathsf{d}(T_1, T_2)$, to measure the distance between two point sequences T_1 and T_2. To eliminate the possibility of certain assignments, we may have $\mathsf{d}(S, P) = \infty$ in some cases. We say an assignment is *permissible* if $\mathsf{d}(S, P) \neq \infty$.

For a trajectory $T \in \mathcal{T}$, let $\tau(T)$ be the fraction of the trajectory's points that is not covered any pathlet. The cost of covering $\mathcal{T}$ by $\mathcal{P}$ is defined as:

$$\mu(\mathcal{T}, \mathcal{P}, \mathsf{d}) = c_1|\mathcal{P}| + c_2 \sum_{T \in \mathcal{T}} \tau(T) + c_3 \sum_{P \in \mathcal{P}} \sum_{S \in \mathcal{T}(P)} \mathsf{d}(S, P),$$

where c_1, c_2 and c_3 are user-defined parameters.

Given a tuple $(\mathcal{T}, \mathbb{P}, \mathsf{d})$, the *pathlet-cover problem* is to compute $\mathcal{P}^* \subseteq \mathbb{P}$ and permissible assignments of subtrajectories to pathlets, to minimize $\mu(\mathcal{T}, \mathcal{P}, \mathsf{d})$. We let $\pi(\mathcal{T}, \mathbb{P}, \mathsf{d}) = \mu(\mathcal{T}, \mathcal{P}^*, \mathsf{d})$.

The *subtrajectory-clustering problem* is to solve the pathlet-cover instance $(\mathcal{T}, \mathbb{P}, \mathsf{d})$ with $\mathbb{P}$ being the (uncountably infinite) set of all point sequences and d again being an arbitrary distance function between point sequences.

The subtrajectory-clustering problem is hard for arbitrary distance functions, even from an approximation point of view, because of the infinitely large set of candidate pathlets. It is helpful to consider distance functions that are metrics, i.e., distance functions satisfying the triangle inequality, as it allows us to restrict ourselves to a finite set of candidate pathlets for computing an approximate solution of the above objective function[4]. In particular, we use the discrete Fréchet distance fr, a widely used metric for point sequences. Given two point sequences A and B, a *correspondence* is a subset $C \subseteq A \times B$ such that for all $a \in A$ (resp. $b' \in B$), there exists $b \in B$ (resp. $a' \in A$) such that $(a, b) \in C$ (resp. $(a', b') \in C$). Let $T_1 = \langle p_1, p_2, \ldots, p_k \rangle$ and $T_2 = \langle q_1, q_2, \ldots, q_\ell \rangle$ be a pair of point sequences. A correspondence C between T_1 and T_2 is monotone if for $(p_i, q_j), (p_{i'}, q_{j'}) \in C$ with $i \leq i'$ we have $j \leq j'$. The discrete Fréchet distance between T_1 and T_2 is defined as

$$\mathrm{fr}(T_1, T_2) = \min_{C \in \Xi} \max_{(p,q) \in C} ||p - q||,$$

where Ξ is the set of all monotone correspondences between $\{p_1, p_2, \ldots, p_k\}$ and $\{q_1, q_2, \ldots, q_\ell\}$.

3 HARDNESS

In this section, we show that the pathlet-cover and subtrajectory-clustering problems are both NP-Hard, even when we restrict the distance function d to be the discrete Fréchet distance fr. The decision version of the pathlet-cover problem can be formulated as follows: given an instance $(\mathcal{T}, \mathbb{P}, \mathsf{d})$ of pathlet-cover and a value k, determine whether $\pi(\mathcal{T}, \mathbb{P}, \mathsf{d}) \leq k$. We define the decision version of subtrajectory-clustering similarly.

We reduce the Facility-Location problem, known to be NP-Complete [23], to the pathlet-cover problem. Given two sets of points F and C referred to as the facilities and customers, respectively, a facility cost $c > 0$, and a real number $k' > 0$, the Facility-Location problem asks if there exists a subset $F' \subseteq F$ of facilities and an assignment of customers to facilities $f : C \to F'$ such that $c|F'| + \sum_{p \in C} ||f(p) - p|| \leq k'$.

Consider the following reduction to the pathlet-cover problem from Facility-Location. Initially, we set $\mathcal{T}$ and $\mathbb{P}$ to be empty. For each customer $p \in C$, we add a trajectory $T_p = \langle p \rangle$ to $\mathcal{T}$. For each facility $f \in F$, we add a candidate pathlet $P_f = \langle f \rangle$ to $\mathbb{P}$. Finally, we let $c_1 = c$, $c_2 > k'$, and $c_3 = 1$, and we solve the decision version of pathlet-cover over $(\mathcal{T}, \mathbb{P}, \mathrm{fr})$ with $k = k'$. Observe $\mathrm{fr}(T_p, P_f) = ||p - f||$. We derive the following theorem[5].

Theorem 3.1. *The pathlet-cover problem is NP-Complete.*

Similarly, we can reduce the variant of Facility-Location where F is implicitly defined as all points in $\mathbb{R}^2$ to the subtrajectory-clustering problem using essentially the same reduction[6].

Theorem 3.2. *The subtrajectory-clustering problem is NP-Hard.*

4 PATHLET-COVER

In this section, we describe an approximation algorithm for the pathlet-cover problem: the algorithm relies on a reduction to the standard Set-Cover problem. We define a set system as a pair $(X, \mathcal{S})$, where $X = \{e_1, \ldots, e_\ell\}$ is the ground set of elements and $\mathcal{S}$ is a family of subsets of X. Given a set system $(X, \mathcal{S})$ and a weight function $\mathrm{w} : \mathcal{S} \to \mathbb{R}^+$, the optimization version of the Set-Cover problem asks for a subset $\mathcal{C} \subseteq \mathcal{S}$ of minimum total weight such that $\bigcup \mathcal{C} = X$. Below, we describe an approximation-preserving reduction from the pathlet-cover problem to the Set-Cover problem. Running the classic greedy algorithm for Set-Cover as described below results in an $O(\log m)$-approximation for both the Set-Cover and original pathlet-cover instances. Unfortunately, the reduction constructs an exponentially large set system. Our main algorithmic

[3] We allow $\mathcal{P}$ to contain the same point sequence multiple times so that each copy may be assigned to different portions of a single trajectory.

[4] Using a metric as a distance function makes it possible to cover every point in $\mathbb{X}$ by simply making each of the n trajectories its own pathlet. Our goal of course is to find a much smaller pathlet dictionary that pulls pathlets from the commonly traversed portions of the trajectory collection.

[5] The proof uses the discrete Fréchet distance for the distance function d used to measure similarity between pathlets and subtrajectories. If we allow arbitrary distance functions instead, then we can show the stronger result that there exists no $o(\log m)$-approximation for pathlet-cover unless P = NP.

[6] While we are able to prove hardness for subtrajectory-clustering, we do not have a proof that the problem is in NP because we do not know if the number of bits required to describe optimal dictionary pathlets is polynomial in the input size.

challenge is to implicitly run the greedy algorithm without having to explicitly create the whole set system.

4.1 From pathlet-cover to set-cover

Fix an instance $(\mathcal{T}, \mathbb{P}, \mathsf{d})$ of the pathlet-cover problem. Let $\mathbb{X}$ be the set of points in $\mathcal{T}$ as defined earlier; $|\mathbb{X}| = m$. We define a family of subsets $\mathcal{S}$ to create a weighted set system $(\mathbb{X}, \mathcal{S})$. There are two types of subsets in our family; the first represents trajectory points being covered by pathlets, and the second represents uncovered trajectory points. Formally, the sets are defined as follows.

(1) For every $P \in \mathbb{P}$ and for any set of input subtrajectories $\mathcal{R}$ drawn from distinct trajectories of $\mathcal{T}$ such that $\mathsf{d}(S, P) \neq \infty$ for all $S \in \mathcal{R}$, family $\mathcal{S}$ contains a set $S(P, \mathcal{R}) = \{p \in S \mid S \in \mathcal{R}\}$ with $\mathrm{w}(S(P, \mathcal{R})) = c_1 + c_3 \sum_{S \in \mathcal{R}} \mathsf{d}(S, P)$.
(2) For a point $p \in \mathbb{X}$, let $T^{(p)} \in \mathcal{T}$ denote the trajectory containing p. Then for every $p \in \mathbb{X}$, family $\mathcal{S}$ contains a singleton set $\{p\}$ with $\mathrm{w}(\{p\}) = c_2/|T^{(p)}|$.

As mentioned above, the first step of this reduction constructs a set system of exponential size. The following lemma expresses the correctness of our reduction.

Lemma 4.1. *There exists a bijection between set covers of $(\mathbb{X}, \mathcal{S})$ and solutions to the pathlet-cover problem for $(\mathcal{T}, \mathbb{P}, \mathsf{d})$ so that cost and weight remain equal across the bijection.*

Proof. Consider a solution to the pathlet-cover problem, consisting of the pathlet dictionary $\mathcal{P}$ and assignments $\mathcal{T}(P)$, for all $P \in \mathcal{P}$. We will create a solution $\mathcal{C}$ to the Set-Cover instance $(\mathbb{X}, \mathcal{S})$. For each uncovered trajectory point p, we add the singleton set $\{p\}$ to $\mathcal{C}$. For each $P \in \mathcal{P}$, we add the set $S(P, \mathcal{T}(P))$ to $\mathcal{C}$. One may easily verify that the total weight of $\mathcal{C}$ is equal to the cost of $\mathcal{P}$.

Conversely, consider a solution $\mathcal{C}$ to the Set-Cover instance. For each set in $\mathcal{C}$ of the form $S(P, \mathcal{R})$, we add the pathlet P to $\mathcal{P}$ and assign all the subtrajectories of $\mathcal{R}$ to P. We leave any points p such that $\{p\} \in \mathcal{C}$ uncovered by our pathlet dictionary. Again, the cost of our pathlet dictionary is the same as the total weight of $\mathcal{C}$. □

4.2 Greedy set cover

We would like to use the standard greedy algorithm to solve the Set-Cover instance described above. The greedy algorithm picks the set that maximizes the ratio of newly covered elements to the weight of the set. We refer to these ratios as the sets' coverage-cost ratios. The algorithm continues picking sets in this manner until every element is covered. It is well-known that the greedy algorithm has an approximation ratio of $O(\log m)$ when run on the set system $(\mathbb{X}, \mathcal{S})$.

We now describe the process in more detail for our setting. In the Set-Cover instance above, each set is either a pathlet with corresponding subtrajectory assignments or a singleton set containing a trajectory point. Choosing a set $S(P, \mathcal{R})$ of the first type results in *covering* the trajectory points within subtrajectories $\mathcal{R}$ assigned to the pathlet P. Choosing a set $\{p\}$ of the second type is implemented as marking a trajectory point $p \in \mathbb{X}$ as *permanently uncovered*. We refer to all trajectory points covered by some pathlet or marked permanently uncovered as *processed*.

Consider the state of the greedy algorithm immediately following an iteration. Let $\mathcal{C} \subseteq \mathcal{S}$ be the family of sets chosen so far by the greedy algorithm, and let $\hat{\mathbb{X}} := \mathbb{X}(\mathcal{C}) \subseteq \mathbb{X}$ be the subset of points not processed by $\mathcal{C}$. For a subtrajectory S, let $\hat{S} = S \cap \hat{\mathbb{X}}$ be the set of unprocessed points in S.

We now consider the next iteration of the greedy algorithm. For a family of subtrajectories $\mathcal{R}$ and a pathlet P, define its *coverage-cost ratio* (with respect to $\mathcal{C}$), $\rho(P, \mathcal{R})$, as

$$\rho(P, \mathcal{R}) = \frac{\sum_{S \in \mathcal{R}} |\hat{S}|}{c_1 + \sum_{S \in \mathcal{R}} c_3 \mathsf{d}(S, P)},$$

and set

$$\mathcal{T}_P = \operatorname*{arg\,max}_{\mathcal{R}: S(P,\mathcal{R}) \in \mathcal{S}} \rho(P, \mathcal{R}); \qquad P^* = \operatorname*{arg\,max}_{P \in \mathbb{P}} \rho(P, \mathcal{T}_P).$$

Similarly, for each unprocessed point $p \in \hat{\mathbb{X}}$, we define its coverage-cost ratio as $\rho(p) = |T^{(p)}|/c_2$ and set $p^* = \arg\max_{p \in \hat{\mathbb{X}}} \rho(p)$. Note that $\rho(P, \mathcal{R})$ depends on $\hat{\mathbb{X}}$ and thus on $\mathcal{C}$, while $\rho(p)$ is independent of $\mathcal{C}$.

In the next iteration, the algorithm chooses the set $S(P^*, \mathcal{T}_{P^*})$ or $\{p^*\}$, whichever has higher coverage-cost ratio. After adding the set to $\mathcal{C}$, we update the set $\hat{\mathbb{X}}$ of unprocessed points, the values $\rho(P, \mathcal{R})$, and the sets $\mathcal{T}_P$. To implement each step of the greedy algorithm, we store all pathlets and unprocessed points in a (max) priority queue with their coverage-cost ratios as the keys. At each step, we delete newly processed points from the priority queue and update the priority queue as pathlets' keys get updated. Note that pathlets remain in the priority queue even when they are added to the dictionary so that multiple copies of the same pathlet may be added to the dictionary with distinct assignments.

The main challenge in implementing a step of the greedy algorithm efficiently is the computation of $\mathcal{T}_P$ for each pathlet P since there are an exponential number of sets $S(P, \mathcal{R})$ in $\mathcal{S}$. Notwithstanding $|\mathcal{S}| = \Omega(2^n)$, we describe below an efficient procedure for computing $\mathcal{T}_P$.

4.3 Computing $\mathcal{T}_P$

Let $S_P(T)$ denote the set of subtrajectories S of trajectory T where $\mathsf{d}(P, S) \neq \infty$. For a set $S(P, \mathcal{R})$, we define the set of variables $x_{S,T} \in \{0, 1\}$ for each $S \in S_P(T)$ and $T \in \mathcal{T}$ which indicate whether $S \in \mathcal{R}$. We also have $\sum_{S \in S_P(T)} x_{S,T} \leq 1$ for all $T \in \mathcal{T}$, i.e., at most one subtrajectory of each trajectory can be in $\mathcal{R}$.

Thus, for a fixed pathlet P, computing the set $\mathcal{T}_P$ is equivalent to solving the following optimization problem.

$$\max \frac{\sum_{T \in \mathcal{T}} \sum_{S \in S(T)} |\hat{S}| x_{S,T}}{c_1 + c_3 \sum_{T \in \mathcal{T}} \sum_{S \in S(T)} \mathsf{d}(S, P) x_{S,T}}.$$

$$\text{s.t.} \sum_{S \in S(T)} x_{S,T} \leq 1 \quad \forall T \in \mathcal{T}.$$

$$x_{S,T} \in \{0, 1\} \quad \forall S \in S(T), T \in \mathcal{T}.$$

The maximum objective value is $\geq \gamma$ iff there exist feasible values of the variables x such that

$$\sum_{T \in \mathcal{T}} \sum_{S \in S(T)} \left(|\hat{S}| - c_3 \gamma \mathsf{d}(S, P) \right) x_{S,T} \geq c_1 \gamma. \tag{1}$$

Define γ^* as the maximum value of γ such that there exists a valid assignment of values to the variables x that satisfy (1). We have $\rho(P, \mathcal{T}_P) = \gamma^*$.

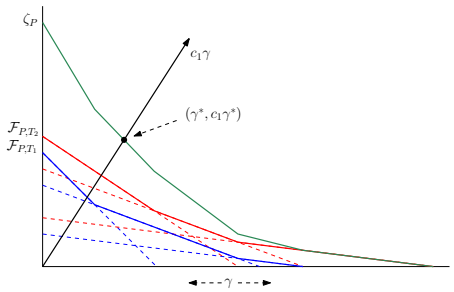

Figure 2. Functions $f_{S,P}$ for subtrajectories S of T_1 (dashed blue) and T_2 (dashed red), functions $\mathcal{F}_{P,T_1}$ (blue) and $\mathcal{F}_{P,T_2}$ (red), function ζ_P (green), and its intersection with the line of slope c_1 at γ^*.

For a fixed value of γ, in order to maximize the left hand side of 1, for each trajectory T we should pick the subtrajectory $S \in S_P(T)$ maximizing the quantity $\left(|\hat{S}| - c_3\gamma \mathsf{d}(S,P)\right)$ provided it is greater than 0. We do not pick any subtrajectory of T if all of them make the quantity less than 0.

For a subtrajectory S, let $f_{S,P}$ be a real-valued function of γ defined as $f_{S,P}(\gamma) = |\hat{S}| - c_3\gamma \mathsf{d}(S,P)$. For each pathlet P and trajectory T, we define the function $\mathcal{F}_{P,T} : \mathbb{R}_{\geq 0} \to \mathbb{R}_{\geq 0}$ as

$$\mathcal{F}_{P,T}(\gamma) = \max\{0, \max_{S \in S_P(T)} f_{S,P}(\gamma)\}.$$

Function $\mathcal{F}_{P,T}$ is a monotonically non-increasing, piecewise-linear, convex function with at most $|S_P(T)| + 1$ linear pieces. Next, we define $\zeta_P(\gamma) = \sum_{T \in \mathcal{T}} \mathcal{F}_{P,T}(\gamma)$. Function ζ_P is also a monotonically non-increasing, piecewise-linear, convex function with at most $\sum_{T \in \mathcal{T}} |S_P(T)| + m$ pieces.

Since the graph of ζ_P, which we also denote by ζ_P, is a monotonically non-increasing, convex chain, the optimal point γ^* is the intersection point of ζ_P with the line of slope c_1 and passing through the origin. See Figure 2. Furthermore, $\mathcal{T}_P = \{T \in \mathcal{T} \mid \mathcal{F}_{P,T}(\gamma^*) > 0\}$. We now describe a data structure that will aid us in a binary search for γ^*.

4.4 Data structure

The projection of $\mathcal{F}_{P,T}$ on the γ-axis partitions it into $|S_P(T)| + 1$ intervals. Let $I_{P,T}$ denote the set of these intervals, and let $\mathcal{I}_P = \bigcup_T I_{P,T}$. We store $\mathcal{I}_P$ in a segment tree Ψ_P; see [16] for details on the segment tree. Roughly speaking, Ψ_P is a balanced binary tree. Each node v of Ψ_P is associated with an interval δ_v—the leaves are associated with "atomic" intervals between two consecutive endpoints of intervals in $\mathcal{I}_P$, and for an interval node v with children w and z, $\delta_v = \delta_w \cup \delta_z$. An interval $I \in \mathcal{I}_P$ is stored at a node v if $\delta_v \subseteq I$ and $\delta_{p(v)} \not\subseteq I$, where $p(v)$ is the parent of v. An interval is stored at $O(\log |\mathcal{I}_P|)$ nodes. Let $\mathcal{I}_{P,v} \subseteq \mathcal{I}_P$ denote the subset of intervals stored at v. We define the function $\zeta_{P,v}(\gamma) = \sum_{T : I_{P,T} \cap \mathcal{I}_{P,v} \neq \emptyset} \mathcal{F}_{P,T}(\gamma)$. The restriction of $\zeta_{P,v}$ in the interval δ_v is a linear function. We store this linear function at v. Let $\alpha(w)$ denote the set of nodes encountered on the path in Ψ_P to w. Finally, for a given leaf w, let $\zeta'_{P,w}(\gamma) = \sum_{v \in \alpha(w)} \zeta_{P,v}(\gamma)$. We have $\zeta'_{P,w}(\gamma) = \zeta_P(\gamma)$ for all $\gamma \in \delta_w$. Computing $\zeta'_{P,w}$ takes $O(\log |\mathcal{I}_P|)$ time. An interval can be inserted into or deleted from Ψ_P in $O(\log |\mathcal{I}_P|)$ time.

Returning to the greedy algorithm, whenever a new point is processed for some trajectory $T \in S_P(T)$, we update $\mathcal{F}_{P,T}$, the set $I_{P,T}$, and the segment tree Ψ_P. In the worst case, these updates may take $O(|S_P(T)| \log |\mathcal{I}_P|)$ time. We then perform a binary search for γ^* as mentioned earlier over the $O(|\mathcal{I}_P|)$ leaves. When searching over a leaf w of Ψ_P, we compute the intersection of $\zeta'_{P,w}$ with the line of slope c_1 passing through the origin. If the intersection occurs to the left (resp. right) of δ_w, then we continue the search over leaves to the left (resp. right) of w. Otherwise, the intersection occurs in δ_w. The binary search takes $O(\log^2 |\mathcal{I}_P|)$ time.

4.5 Analysis

We now analyze the running time of our algorithm. We will do so in terms of a value $\chi = \max_{P \in \mathbb{P}, T \in \mathcal{T}} |S_P(T)|$. While this value can be as high as $O(m^2)$, we show in the next section how to reduce the value greatly when solving the subtrajectory-clustering problem for collections of κ-packed trajectories. Selecting a pathlet or point from the priority queue takes $O(\log(b+m))$ time. Updating the pathlets in the queue at the end of each greedy step takes $O(b \log(b+m))$ time, and each point member of the queue is updated at most once in $O(\log(b+m))$ time. Therefore, the algorithm spends $O(bm \log(b+m))$ time total updating and searching the queue.

We also have $|\mathcal{I}_P| = O(n\chi)$ for any P. If a pathlet P is selected at the beginning of a greedy step, the subtrajectories it covers can be computed in $O(\log(n\chi) + k)$ time, where k is the number of newly covered subtrajectories. Therefore, finding these subtrajectories takes $O(m \log(n\chi) + m)$ time total. It takes $O(b \log^2(n\chi))$ time to run the binary searches to recompute $\rho(P, \mathcal{T}_P)$ for each candidate pathlet P at the end of a greedy step.

Finally, at the end of a greedy step, it takes $O(\chi \log \chi)$ time to recompute the function $\mathcal{F}_{P,T}$ for each pathlet P and trajectory T that has a newly processed point. There are at most m instances where a trajectory sees one or more of its points covered at the end of a greedy step, so the total time spent updating $\mathcal{F}_{P,T}$ functions is $O(bm\chi \log \chi)$. By the same argument, the total number of updates needed for each segment tree Ψ_P over all iterations is $O(m\chi)$, for a total update time of $O(bm\chi \log(n\chi))$ for all pathlets. This later bound is the bottleneck of our algorithm.

We thus have the following theorem.

Theorem 4.2. *Let $\mathcal{T}$ be a set of n trajectories having m points in total, $\mathbb{P}$ a set of b candidate pathlets, and d any distance function between point sequences. Let χ be the maximum number of subtrajectories of any one trajectory $T \in \mathcal{T}$ for which there are permissible assignments for any one $P \in \mathbb{P}$. Then there is an $O(\log m)$-approximation algorithm for the pathlet-cover problem that runs in $\tilde{O}(bm\chi)$ time, provided there is an oracle that computes the distance function d in constant time.*

5 SUBTRAJECTORY CLUSTERING

We now describe our approximation algorithm for the subtrajectory-clustering problem. As mentioned in the introduction, we assume trajectories to be κ-packed and the underlying distance function to be the discrete Fréchet distance. A curve is said to be κ-packed if the length of its intersection with any disk of radius r is at most κr. A point sequence is κ-packed if the polygonal curve obtained by joining its points in sequence is κ-packed.

The algorithm relies on two main ideas. First, using the fact that discrete Fréchet distance is a metric, we quickly construct a small set $\mathbb{S}$ of candidate pathlets so that the cost of the optimal

pathlet cover of $\mathcal{T}$ with respect to $\mathbb{S}$ is close to that of an optimal subtrajectory clustering of $\mathcal{T}$ (Section 5.1)[7]. Using properties more specific to discrete Fréchet distance, we then reduce the set of candidate pathlets further to a subset $\mathbb{C}$. Next, we take advantage of our use of Fréchet distance and the input trajectories being κ-packed to quickly compute an approximation of the Fréchet distance for pathlet-cover's distance function (Section 5.2). This approximation enables us to consider only a small number of assignments between each remaining pathlet and trajectory, thereby greatly speeding up the algorithm without sacrificing the quality of the clustering.

5.1 Candidate pathlets

Let $\mathcal{T}$ be the set of n input trajectories with a total of m points, and let $\mathbb{P}$ be the set of all point sequences in $\mathbb{R}^2$. We first show how to construct a candidate set $\mathbb{S}$ of $O(m^2)$ pathlets such that $\pi(\mathcal{T}, \mathbb{S}, \mathsf{fr}) \leq 2\pi(\mathcal{T}, \mathbb{P}, \mathsf{fr})$, and then show how to construct a candidate set $\mathbb{C}$ of $O(m)$ pathlets such that $\pi(\mathcal{T}, \mathbb{C}, \mathsf{fr}) = O(\log m)\pi(\mathcal{T}, \mathbb{P}, \mathsf{fr})$.

Let $\mathcal{P} \subset \mathbb{P}$ be an optimal pathlet dictionary of $\mathcal{T}$, i.e., $\mathcal{P} = \arg\min_{\mathcal{P} \subset \mathbb{P}} \mu(\mathcal{T}, \mathbb{P}, \mathsf{fr})$. Recall that for any $P \in \mathcal{P}$, $\mathcal{T}(P)$ is the family of subtrajectories assigned to P. Let $\mathsf{d}(P, \mathcal{T}(P))$ denote the cost of assigning subtrajectories in $\mathcal{T}(P)$ to P, i.e.,

$$\mathsf{d}(P, \mathcal{T}(P)) = c_3 \sum_{S \in \mathcal{T}(P)} \mathsf{fr}(S, P).$$

The lemma below states that P can be replaced by a subtrajectory of $\mathcal{T}(P)$ while only doubling the cost.

Lemma 5.1. *There exists a subtrajectory $S' \in \mathcal{T}(P)$ such that $\mathsf{d}(S', \mathcal{T}(P)) \leq 2\mathsf{d}(P, \mathcal{T}(P))$.*

Proof. Let $S' = \arg\min_{S \in \mathcal{T}(P)} \mathsf{fr}(S, P)$. We then have for any $S \in \mathcal{T}(P)$,

$$\mathsf{fr}(S, S') \leq \mathsf{fr}(S, P) + \mathsf{fr}(S', P) \leq 2\,\mathsf{fr}(S, P).$$

Thus replacing P by S' increases the cost by at most a factor of 2. □

In any solution to the pathlet-cover instance $(\mathcal{T}, \mathbb{P}, \mathsf{fr})$, we can replace all dictionary pathlets by input subtrajectories using Lemma 5.1, increasing the total cost by a factor of at most 2.

Corollary 5.2. *Let $\mathbb{P}$ be the set of all point sequences in $\mathbb{R}^2$, and let $\mathbb{S}$ be the set of all subtrajectories of $\mathcal{T}$. We have $|\mathbb{S}| = O(m^2)$ and $\pi(\mathcal{T}, \mathbb{S}, \mathsf{fr}) \leq 2\pi(\mathcal{T}, \mathbb{P}, \mathsf{fr})$.*

Corollary 5.2 immediately implies a polynomial time approximation algorithm for subtrajectory-clustering. However, we can further reduce the number of candidate pathlets by another factor of m with only an $O(\log m)$-factor increase in the approximation ratio.

Canonical pathlets. To further restrict the set of candidate pathlets beyond the set of all subtrajectories, we introduce the notion of *canonical pathlets*. Consider a trajectory $T \in \mathcal{T}$. For simplicity, we assume that $|T|$ is a power of 2. Consider a balanced binary tree over the interval $[1, |T|]$. Each node in the tree corresponds to an interval $[i, j]$, with its left and right children associated with intervals $[i, \lfloor (i + j + 1)/2 \rfloor]$ and $[\lfloor (i + j + 1/2 \rfloor + 1, j]$ respectively. The leaves correspond to singleton intervals. The height of the tree is $\log_2 |T|$, and the total number of nodes is $2|T|$. Each interval induces a subtrajectory $T(i, j)$ of T. These subtrajectories of T corresponding to the intervals of the tree nodes are called the canonical pathlets of T, which we denote by $\mathbb{C}(T)$. Applying the same procedure to all the trajectories, we get a set $\mathbb{C}$ of $O(m)$ canonical pathlets.

We can replace a subtrajectory pathlet P by a set of $O(\log m)$ canonical pathlets $C(P)$ while increasing the assignment cost by at most a factor of $O(\log m)$. Recall $\mathsf{d}(P, \mathcal{T}(P)) = \sum_{S \in \mathcal{T}(P)} \mathsf{fr}(S, P)$. The lemma below formalizes the prior observation.

Lemma 5.3. *For a pathlet P and any set of subtrajectories $\mathcal{T}(P)$, there exists a set of $O(\log m)$ canonical pathlets $C(P) = \{P_1, \ldots, P_{|C(P)|}\}$ and a family of subtrajectory sets $\{\mathcal{T}(P_1), \ldots, \mathcal{T}(P_{|C(P)|})\}$, the union of whose points equals the points in $\mathcal{T}(P)$, such that*

$$\sum_{P_i \in C(P)} \mathsf{d}(P_i, \mathcal{T}(P_i)) = O(\log m) \cdot \mathsf{d}(P, \mathcal{T}(P)).$$

Proof. Let P be a subtrajectory of trajectory T. It is not hard to see that P can be written as the concatenation of pathlets $P_1, P_2, \ldots, P_{|C(P)|}$ in order, where each $P_i \in \mathcal{C}(T)$ and $|C(P)| = O(\log |T|)$. Consider a subtrajectory $S \in \mathcal{T}(P)$. Since there is a monotone correspondence between P and S with discrete Fréchet distance $\mathsf{fr}(P, S)$, there exist monotone correspondences between each P_i and some subtrajectory S_i of S such that $\mathsf{fr}(P_i, S_i) \leq \mathsf{fr}(P, S)$ and the S_i's concatenated in order cover S. We construct the family of sets $\mathcal{T}(P_i)$ by including the subtrajectory S_i of S in $\mathcal{T}(P_i)$, for all $S \in \mathcal{T}(P)$ and $i \in \{1, \ldots, |C(P)|\}$. We then have

$$\begin{aligned}
\sum_{P_i \in C(P)} \mathsf{d}(P_i, \mathcal{T}(P_i)) &= \sum_{S \in \mathcal{T}(P)} \sum_{i=1}^{|C(P)|} \mathsf{fr}(P_i, S_i) \\
&\leq |C(P)| \sum_{S \in \mathcal{T}(P)} \mathsf{fr}(P, S) \\
&= O(\log m) \cdot \mathsf{d}(P, \mathcal{T}(P)).
\end{aligned}$$

□

As before, we can perform the above substitution for every subtrajectory pathlet in a (nearly) optimal dictionary with little cost in the approximation ratio.

Corollary 5.4. *Let $\mathbb{P}$ be the set of all point sequences in $\mathbb{R}^2$ and let $\mathbb{C}$ be the set of all canonical pathlets of $\mathcal{T}$. We have $\pi(\mathcal{T}, \mathbb{C}, \mathsf{fr}) = O(\log m) \cdot \pi(\mathcal{T}, \mathbb{P}, \mathsf{fr})$.*

5.2 Approximate distances

We use the set of canonical pathlets $\mathbb{C}$ as the set of candidate pathlets in our reduction to pathlet-cover. Rather than use the discrete Fréchet distance as our distance function directly, however, we instead compute a distance function d that approximates the Fréchet distances between a subset of pairs of canonical pathlets and subtrajectories, without decreasing the cluster quality by much. Other pairs are given a distance of ∞ to mark them as not being permissible assignments. Using d dramatically reduces the number of permissible assignments considered by the pathlet-cover approximation algorithm, and we are able to compute approximate Fréchet distances much faster than we can compute them exactly. Our

[7] This step works for any distance function that is a metric, but for simplicity we focus on the discrete Fréchet distance.

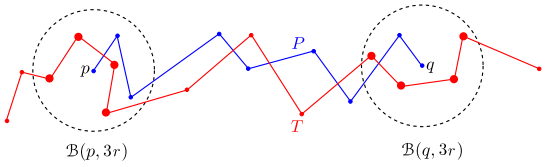

Figure 3. Sets Q^1_{TPr} and Q^2_{TPr} as the thick red points inside the left and right balls, respectively.

algorithm for computing these distances is based upon a simple trajectory simplification scheme as described below.

For any $r > 0$, an *r-simplification* of a trajectory T is a trajectory S consisting of a subsequence of points from T such that $\mathsf{fr}(T, S) \leq r$. The following is a well-known linear-time greedy algorithm to compute an r-simplification of T. We include the first point of T. We then iterate along the points of T in order, and include in our simplification each point that is at a distance of at least r from the previously included point. Finally, we include the last point of T. We denote the resulting trajectory by $\overrightarrow{T_r}$. We can similarly start from the last point of T and proceed in reverse. We denote this trajectory by $\overleftarrow{T_r}$. Finally, let T_r denote the merger of $\overrightarrow{T_r}$ and $\overleftarrow{T_r}$ obtained by including each point that appears in either trajectory in the order they originally appeared in T. The following lemma follows from the construction:

Lemma 5.5. (i) Let p' and q' appear in T_r in order. We have $\mathsf{fr}(T[p', q'], T_r[p', q']) \leq r$.

(ii) Let p and q appear in T in order. There exist $p', q' \in T_r$ such that $T[p, q]$ is a subsequence of $T[p', q']$ and $\mathsf{fr}(T[p, q], T_r[p', q']) \leq r$.

(iii) The lengths of the edges in $\overrightarrow{T_r}$ (resp. $\overleftarrow{T_r}$), except possibly the last (resp. first) edge, is at least r.

As described below, computing r-simplifications of trajectories allows us to quickly compute Fréchet distances within an additive error of $O(r)$. In addition, for any pair $P \in \mathbb{C}, T \in \mathcal{T}$, the number of subtrajectories of T_r, the r-simplification of T, within distance $O(r)$ of P is a constant, assuming κ is a constant.

Let σ denote the *spread* of the trajectories' point sets, i.e., the ratio between the maximum and the minimum pairwise point distances. (Recall our assumption that points are distinct; while the assumption aids in our presentation, the algorithm can be easily adapted to the case when a point appears in multiple trajectories.)

We now construct a distance function d that gives a 4-approximation for the discrete Fréchet distance fr for some pathlet-subtrajectory pairs. For others, their distance will be set so corresponding assignments are not permissible. Let $\underline{r}$ (resp. $\bar{r}$) be the minimum (resp. maximum) pairwise distance between trajectory points: $\bar{r} = \sigma \underline{r}$.

For each $T \in \mathcal{T}, r \in \langle \bar{r}, \bar{r}/2, \ldots, \underline{r} \rangle$, we do the following. We compute T_r in $|T|$ time as described above. We want to efficiently decide which subtrajectories of T_r are within distance $O(r)$ of each pathlet P. To do so, we preprocess the set of points in T_r into an approximate spherical range query data structure of size $O(|T_r|)$ in $O(|T_r|)$ time; see Aronov *et al.* [6]. The data structure can answer queries of the following form: Let $\mathcal{B}(p, r) = \{x \in \mathbb{R}^2 \mid ||x - p|| \leq r\}$ denote a ball of radius r around p. Given a point p, the data structure returns a subset of points of $\mathcal{B}(p, 2r) \cap T_r$, including all points of $\mathcal{B}(p, r) \cap T_r$. It returns no points outside $\mathcal{B}(p, 2r) \cap T_r$. Each query takes constant time[8], plus additional time proportional to the number of points returned. For each $P \in \mathbb{C}$, let p and q be the start and end points of P, respectively. Let $Q^1_{TPr} = \mathcal{B}(p, 3r) \cap T_r$ and $Q^2_{TPr} = \mathcal{B}(q, 3r) \cap T_r$ (see Figure 3). We compute Q^1_{TPr} and Q^2_{TPr} using the approximate spherical range query data structure, and then for every pair (p', q') such that $p' \in Q^1_{TPr}$ and $q' \in Q^2_{TPr}$, we invoke a decision procedure that checks if $\mathsf{fr}(T_r[p', q'], P)$ is at most $3r$; see [17, Lemma 3.1]. Suppose $\mathsf{fr}(T_r[p', q'], P) \leq 3r$. We set $\mathsf{d}(T[p', q'], P) = 4r$. We do the above for all triples (T, r, P), and we let all other values of $\mathsf{d}(\cdot, \cdot)$ not assigned above be equal to ∞. For our algorithm, we return an $O(\log m)$-approximate pathlet dictionary for the pathlet-cover instance $(\mathcal{T}, \mathbb{C}, \mathsf{d})$ using Theorem 4.2.

5.3 Analysis

We now turn to analyzing the running time and approximation factor of our subtrajectory-clustering algorithm.

Running time. Our main goals are to bound the number of permissible assignments per pathlet-trajectory pair and the time it takes to compute their Fréchet distances. The following lemma helps with both goals.

Lemma 5.6. *For each $P \in \mathbb{C}$, $T \in \mathcal{T}$, and $r \in \{\bar{r}, \bar{r}/2, \ldots, \underline{r}\}$, we have $|Q^1_{TPr}|, |Q^2_{TPr}| \leq O(\kappa)$.*

Proof. Let p be the starting point of P. By Lemma 5.5(iii), each pair of consecutive points along $\overrightarrow{T_r}$ (resp. $\overleftarrow{T_r}$) are distance at least r apart (except possibly at the endpoints). In particular, there is at least r length of curve from $\overrightarrow{T_r}$ (resp. $\overleftarrow{T_r}$) lying in $\mathcal{B}(p, 4r)$ between every pair of consecutive points in $Q^1_{TPr} \cap \overrightarrow{T_r}$ (resp $Q^1_{TPr} \cap \overleftarrow{T_r}$). By the definition of κ-packed curves, there are only $O(\kappa)$ such curve portions within that ball. A similar argument holds for Q^2_{TPr}. □

As in the previous section, for each canonical pathlet $P \in \mathbb{C}$ and trajectory $T \in \mathcal{T}$, let $S_P(T)$ denote the set of subtrajectories S of trajectory T where $\mathsf{d}(P, S) \neq \infty$. Recall the value $\chi = \max_{P \in \mathbb{C}, T \in \mathcal{T}} |S_P(T)|$. The following lemma bounds χ.

Lemma 5.7. *For each $P \in \mathbb{C}, T \in \mathcal{T}$, we have $|S_P(T)| \leq O(\kappa^2 \log \sigma)$.*

Proof. By Lemma 5.6, for a fixed value $r \in \{\bar{r}, \bar{r}/2, \ldots, \underline{r}\}$, the total number of subtrajectories for which $\mathsf{d}(T, P)$ is set to be finite is $O(|Q^1_{TPr}|.|Q^2_{TPr}|) = O(\kappa^2)$. There are $O(\log \sigma)$ values of r considered by the algorithm. □

The algorithm spends $O(m \log \sigma)$ time total simplifying trajectories. Fix a canonical pathelet P, trajectory T, and resolution r. Let p and q be the first and last points of P. One can show $\mathcal{B}(p, 6r) \cap T_r$ and $\mathcal{B}(q, 6r) \cap T_r$ both contain $O(\kappa)$ points. Therefore, it takes $O(\kappa)$ time to compute Q^1_{TPr} and Q^2_{TPr} using the approximate spherical range-query data structure. For any subtrajectory S of T_r, testing if $\mathsf{fr}(S, P) \leq 3r$ can be done in $O(\kappa|P|)$ time [17]. There are $b = O(m)$ canonical pathlets, and their total length (i.e., number of points) is $O(m \log m)$. Summing over all permissible assignments between pathlets and subtrajectories, it takes $O(\kappa mn\chi \log m) = O(\kappa^3 mn \log \sigma \log m)$ time to compute approximate Fréchet distances. Finally, it takes $\tilde{O}(bm\chi) = \tilde{O}(\kappa^2 m^2 \log \sigma)$ time to run our algorithm

[8] This constant is exponential in the ambient dimension.

for pathlet-cover on the instance $(\mathcal{T}, \mathbb{C}, \mathsf{d})$. Assuming κ is a small constant, we get the following lemma.[9]

Lemma 5.8. *The subtrajectory-clustering algorithm runs in $\tilde{O}(m^2)$ time.*

Approximation factor. We now bound the approximation factor for our subtrajectory-clustering algorithm. In the following lemma, we claim we can replace an arbitrary assignment to a canonical pathlet with a permissible assignment offering the same coverage at approximately the same cost.

Lemma 5.9. *Let $P \in \mathbb{C}, T \in \mathcal{T}$. For any subtrajectory $T[p,q]$, there exist $p', q' \in \mathcal{T}$ such that $T[p,q]$ is a subsequence of $T[p',q']$ and $\mathsf{d}(T[p',q'],P) \leq 4\,\mathsf{fr}(T[p,q],P)$.*

Proof. Let $r \in \{\bar{r}, \bar{r}/2, \ldots, \underline{r}\}$ be such that $r \leq \mathsf{fr}(T[p,q],P) < 2r$. By Lemma 5.5(ii), there exist $p', q' \in T_r$ such that $T[p,q]$ is a subsequence of $T[p',q']$ and $\mathsf{fr}(T[p,q], T_r[p',q']) \leq r$. By the triangle inequality,

$$\mathsf{fr}(T_r[p',q'],P) \leq \mathsf{fr}(T_r[p',q'],T[p,q]) + \mathsf{fr}(T[p,q],P) < 3r.$$

By definition of the discrete Fréchet distance, $p' \in Q^1_{TPr}$, and $q' \in Q^2_{TPr}$. Therefore, when the algorithm creates T_r, it successfully verifies that $\mathsf{fr}(T_r[p',q'],P) \leq 3r$ and sets $\mathsf{d}(T[p',q'],P) = 4r$. We have $\mathsf{d}(T[p',q'],P) \leq 4\,\mathsf{fr}(T[p,q],P)$ as promised. Note that the algorithm may later set $\mathsf{d}(T[p',q'],P)$ to be a smaller value, but doing so will not invalidate the inequality. □

We also have the following lemma establishing that the function d gives an upper bound on the discrete Fréchet distance.

Lemma 5.10. *Let $P \in \mathbb{C}$, and let S be a subtrajectory. We have $\mathsf{fr}(S,P) \leq \mathsf{d}(S,P)$.*

Proof. Suppose $\mathsf{d}(S,P) \neq \infty$. Let T be the trajectory containing S. Let r be the smallest value such that $S = T_r[p',q']$ and $\mathsf{fr}(T_r[p',q'],P) \leq 3r$. By Lemma 5.5(i),
$\mathsf{fr}(T[p',q'], T_r[p',q']) \leq r$. Therefore, by triangle inequality,

$$\mathsf{fr}(S,P) \leq \mathsf{fr}(T[p',q'], T_r[p',q']) + \mathsf{fr}(T_r[p',q'],P) \leq 4r = \mathsf{d}(S,P).$$

□

Now, let $\mathcal{P}^*$ be the optimal solution to the subtrajectory-clustering problem. By Corollary 5.4, there exists a pathlet dictionary $\bar{\mathcal{P}} \subseteq \mathbb{C}$ of cost at most $O(\log m)\mu(\mathcal{T}, \mathcal{P}^*, \mathsf{fr})$. By Lemma 5.9, we can further modify $\bar{\mathcal{P}}$ without changing the set of covered points so that every subtrajectory assignment is permissible according to d. Let $\hat{\mathcal{P}}$ be this new dictionary. We have $\mu(\mathcal{T}, \hat{\mathcal{P}}, \mathsf{d}) \leq O(\log m)\mu(\mathcal{T}, \mathcal{P}^*, \mathsf{fr})$. Let $\mathcal{P} \subseteq \mathbb{C}$ be the pathlet dictionary returned by the approximate pathlet-cover algorithm run on $(\mathcal{T}, \mathbb{C}, \mathsf{d})$. By Lemma 5.10,

$$\begin{aligned}\mu(\mathcal{T}, \mathcal{P}, \mathsf{fr}) \leq \mu(\mathcal{T}, \mathcal{P}, \mathsf{d}) &\leq O(\log m)\mu(\mathcal{T}, \hat{\mathcal{P}}, \mathsf{d}) \\ &\leq O(\log^2 m)\mu(\mathcal{T}, \mathcal{P}^*, \mathsf{fr}).\end{aligned}$$

We reach our main theorem for this section.

[9]The constant inside $\tilde{O}$ depends exponentially on the ambient dimension, due to a similar dependence for the query time of the approximate spherical range query data structure of [6] .

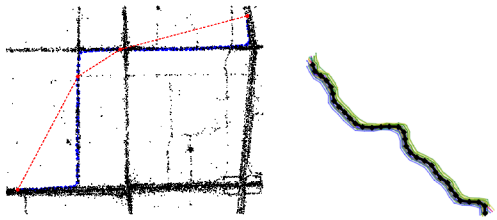

Figure 4. Left: linear (dashed red) versus our graph based (dotted blue) interpolation schemes for sparse trajectories. Right: cluster of pathlets and chosen representative pathlet in black.

Theorem 5.11. *Let $\mathcal{T}$ be a set of n trajectories in $\mathbb{R}^2$ with m points in total such that each trajectory is κ-packed for some constant κ. There is an $O(\log^2 m)$-approximation algorithm for computing a subtrajectory-clustering of $\mathcal{T}$ using the discrete Fréchet distance* fr *as the pathlet cost distance function that runs in $\tilde{O}(m^2)$ time.*

6 PREPROCESSING

In this section, we describe two preprocessing steps that are applied before running the greedy algorithm for pathlet-cover. First, we describe a method to interpolate missing points from sparse or lossy trajectory data. Next, we describe an approach to reduce the number of candidate pathlets considered by our pathlet-cover algorithm beyond even the canonical pathlets.

Handling sparse data. A major challenge we face in clustering, particularly when dealing with GPS data, is that individual trajectories may contain long spans where the data contains no observations. As a consequence, we lose information on what route the trajectory takes between data points. A natural approach to filling in missing observations is to compute a simple interpolation such as a line segment between pairs of consecutive data points and then add additional points to the data set along that interpolation. Unfortunately, objects do not move along straight lines even in a road network, e.g., the trajectory between two consecutive observed points may contain a turn.

Instead of computing simple interpolations based only on the pairs of consecutive trajectory data points, we use the location of observed points throughout the entire collection of trajectories to produce reasonable routes for the unknown parts of individual trajectories. Intuitively, an unknown route between two consecutive data points likely passes close by to many observed data points from other trajectories. Let δ, Δ, and ϵ be parameters. We create a graph $G = (V, E)$ as follows. We begin by creating a grid over the trajectory points using cells of side-length δ. For each grid-cell v containing at least one observed trajectory point, we add v to the vertex set V. Then, we add an edge (u, v) to E for each pair of cells u and v whose centers are within distance Δ to one-another. We give each edge (u, v) a weight equal to the inverse of the number of points lying in u and v so that edges between densely populated cells have low weight. Finally, for each pair of consecutive trajectory points p and q lying distance at least ϵ apart, we compute a shortest path in G between p and q's cells, adding the cell centers encountered along the path to the trajectory between p and q. See Figure 4 for a comparison of our method to the linear interpolation approach.

Sparsifying candidate pathlets. The subtrajectory clustering problem allows us to select pathlets from the full space of point

sequences. By focusing just on the subtrajectories or even canonical pathlets, we are sampling from the space of candidate pathlets. Recall that the primary motivation behind our work is that large groups of subtrajectories will tend to cluster within the same proximity. Since we construct candidate pathlets from each input trajectory independently, we tend to over sample in the neighborhood of a pathlet (a point in the pathlet space) that is shared by many trajectories. As our pathlet-clustering algorithm requires time proportional to the number of candidate pathlets, we perform a sparsifying step to remove redundant canonical pathlets from overly sampled areas.

The high level idea behind our procedure is to project each canonical pathlet to Euclidean space. Similar canonical pathlets should project to closeby points. We then apply standard clustering techniques to find similar points from which only one pathlet's point will remain a candidate. Let d and ω be parameters. For each canonical pathlet $P \in \mathbb{C}$, we select d points lying equidistant along the piecewise linear curve between P's points. Let these points be $\langle (x_1, y_1), (x_2, y_2), \ldots, (x_d, y_d) \rangle$. We create a $2d$-dimensional point $p_P = (x_1, y_1, x_2, y_2, \ldots, x_d, y_d)$. We iterate over the pathlets again, marking some as *removed* as we do so. Initially, none of the pathlets are marked. Now, consider a pathlet $P \in \mathbb{C}$ chosen during the iteration. If P is already marked as removed, we ignore it and continue to the next pathlet in the iteration. Otherwise, we consider the ball of radius ω centered at p_P and mark all other pathlets' points within the ball as removed. When we have finished iterating over the canonical pathlets, we remove the marked ones from the set of candidate pathlets used in the pathlet-cover algorithm. See Figure 4 for an example of a cluster of 32 similar pathlets and the one chosen in the sparsifying step.

7 EXPERIMENTS

We describe experimental results to highlight the various desirable properties of our model mentioned earlier, and to demonstrate the efficiency of our algorithm. In particular, we show that our model can handle noisy and diverse data sets, without compromising on cluster quality. Further, our algorithm is fast, the pathlets found by our algorithm are purely data-driven (hence can be complex), and they capture the underlying shared structure. We also show the sensitivity of our results to varying parameters. All our experiments were run on an Intel Xeon 2.4 GHz computer.

Data sets. We have used both real and synthetic data sets for our experiments. The real data sets used are as follows.

(i) The BEIJING data set contains taxi trajectories in Beijing, China [3]. It has month-long trajectory data of 28,000 cabs in Beijing (about 60 million points). Each trajectory originally consists of multiple trips of a taxi; we break it up so that each trip is its own trajectory. We run experiments on a subset of this data having 9 million points, spanning four days.

(ii) The GEOLIFE data set was collected by Microsoft Research Asia [2]. It tracks 182 users over four years, recording a broad range of outdoor movements, such as walking, biking, and hiking. Widely distributed in over 30 cities in China and some cities in USA and Europe, we only took trajectories in Beijing since it contained most of the data. We focus on pedestrian trajectories only, which result in 2,657 trajectories containing 1,473,115 points.

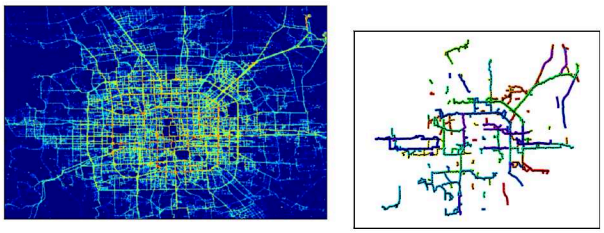

Figure 5. The heatmap (left) and major pathlets (right) of the BEIJING data set.

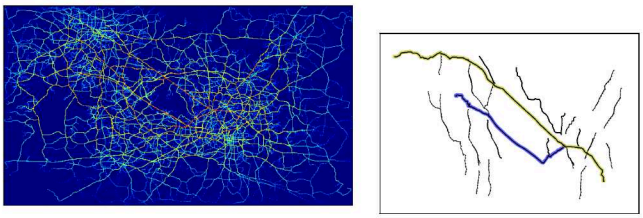

Figure 6. The heatmap (left) and major pathlets (right) of the RTP data set.

(iii) The CYCLING data set is a set of cycling trajectories in Durham, North Carolina. It consists of 37 trajectories, with 106,791 points in total. The trajectories are quite complicated, containing self-intersections and repeating portions.

Besides the above real data sets, we also used a synthetic data set, generated by the University of Minnesota Web-based Traffic Generator [4], which generates traffic data from the underlying road network in any arbitrary region of the world. The model used to generate the traffic data was given by Brinkhoff [8], which accounts for real-world conditions such as varying speeds, road congestion, etc. Denoted RTP, the data set covers the Research Triangle region [1] of North Carolina. It has around 20,000 trajectories and around 1 million points.

Visualization. We visually inspect the trajectories and the pathlets generated, demonstrating various desirable properties of our model and algorithm. [10]

Dense and popular portions. Our algorithm chooses pathlets from areas that have a high density of trajectory points. Figure 5 shows the BEIJING data set and the top 100 pathlets out of around 8,000 picked. The first pathlet chosen for BEIJING is located at the Beijing Capital International Airport (PEK), which has a high point density on the heat map. Other top pathlets include the S12 (the highway to the airport) and other highways leading into the city as well as shorter pathlets closer to the city center.

Figure 6 shows a portion of the RTP data set and the top 35 pathlets out of around 8,500 picked by our algorithm. The two long highlighted pathlets follow two popular highways which have a high point density on the heat map, but were chosen 26th and 32nd. Most trips within the region tend to use relatively short portions of each highway as opposed to traversing their entire length, thus other shorter pathlets were chosen earlier.

Robustness to variations. Our algorithm works well even if there is no underlying road network and data is noisy. For instance, the GEOLIFE data set is much more varied and noisy. Figure 8 shows the top 50 pathlets for GEOLIFE; we can see that these are in general shorter than the pathlets for BEIJING, probably because we focus

[10] The algorithm was run with these parameters to generate the visualizations – (c_1, c_2, c_3) values for BEIJING: $(1, 1.5, 0.005)$; GEOLIFE, CYCLING and RTP: $(1, 1, 0.005)$.

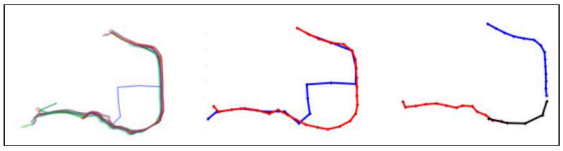

Figure 7. A bunch of trajectories from RTP (left). The blue trajectory takes a detour from the remaining trajectories, before merging again. Without gaps, the blue trajectory is picked as a pathlet on its own, with the remaining trajectories being assigned to the red pathlet (center); whereas having gaps produces three pathlets (right) with the detour being considered a gap.

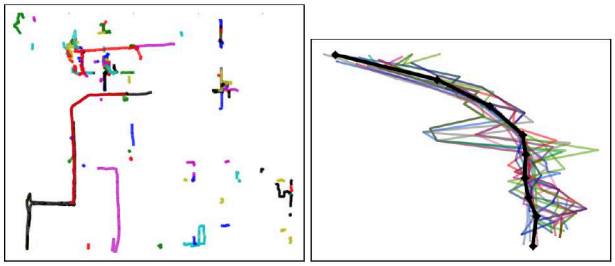

Figure 8. Major pathlets of the GeoLife data set (left); a cluster of subtrajectories (right) with the darker pathlet.

on pedestrian data, and the trajectories are not as long. Figure 8 also shows a cluster of subtrajectories, which look very similar. The nice subtrajectory in the middle is picked as a pathlet, and serves as a less noisy representative for the entire cluster.

Our algorithm successfully finds common portions of trajectories that are otherwise diverse. Figure 10 shows trajectories from the RTP (left) and Beijing (right) data sets passing through a common region that is captured by the highlighted pathlet.

The notion of gaps in our model gracefully handles individual variations in trajectories. We ran the algorithm on the RTP data set with a very large value of c_2 (highly penalizing gaps, and effectively removing them). See Figure 7.

Cluster quality. Our objective function measures cluster quality by taking the sum of distances of the subtrajectories in a cluster to the pathlet instead of other measures, such as the maximum distance to the pathlet. This results in tighter clusters (see right picture of Figure 9).

Trajectory reconstruction. We can approximately reconstruct individual trajectories by simply concatenating the pathlets they are assigned to (see left picture of Figure 9).

Data-driven pathlets. The pathlets found are completely data-driven and can be quite complex. Figure 11 shows a trajectory of the Cycling data set, and the first two pathlets assigned to it. The trajectory contains loops, reflecting the cyclist going around a track and changing directions. Pathlets do indicate direction of travel; in particular, the first two pathlets assigned to the cycling trajectory go in each direction of the loop as shown in the figure.

Quantitative analysis. The running time of our algorithm depends upon the number of points and candidate pathlets, and complexity of the trajectories. While RTP and GeoLife had similar number of points, their runtimes were 1 hour and 2 hours respectively, see Table 1. The GeoLife trajectories tended to be more complex and had more intersections. The much larger Beijing data set ran in 12 hours. It is possible to reduce runtimes by using a

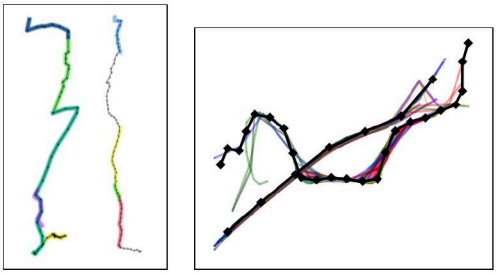

Figure 9. Left : trajectories (in black) from Beijing and RTP resp., and the pathlets (highlighted in color) to which they are assigned. Right : a single cluster found in RTP while using the max distance to measure cluster quality, picking out only the middle darker pathlet. Using the sum of distances splits the above cluster into two – the middle pathlet's cluster, and the second pathlet's cluster.

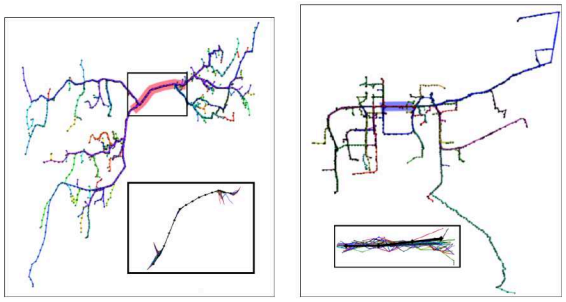

Figure 10. Full trajectories sharing a common portion, captured by pathlets with assigned subtrajectories.

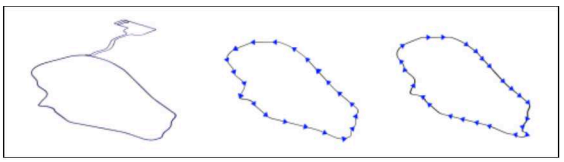

Figure 11. A self-intersecting trajectory from Cycling (left), and the first two pathlets assigned to it.

tighter bound on the maximum Fréchet distance, not penalizing the Fréchet distance (setting $c_3 = 0$), stopping the greedy algorithm when the increase in coverage becomes too small, and sparsifying the candidate pathlets (Section 6).

Table 1 shows the effects of sparsifying the candidate pathlets. Intuitively, the sparsifying step is meant to remove redundant pathlets from the set of candidates. Accordingly, the number of candidates after sparsifying stayed roughly constant even as we took larger samples of the Beijing data set. In general, the speed up increases as the proportion of pathlets remaining after sparsification reduces.

The pathlets picked at the beginning cover more points than those picked later. The left plot in Figure 12 shows the cumulative coverage (as a fraction of the total points) of pathlets picked in iterations of the greedy algorithm for the Beijing data set, with and without sparsifiying the candidate pathlets; one can see that the plots are almost identical. Further, the marginal increase in coverage goes down drastically for later pathlets.

We also investigated the number of pathlets assigned per trajectory. The left plot in Figure 13 shows the frequency of trajectories vs. the number of pathlets assigned per trajectory for the Beijing data set. There is a very long tail, meaning that there exist trajectories with a lot of pathlets assigned to them. However, the tail mostly consists of pathlets added later by the algorithm; these are

Data Set	# points	Runtime (hours)	Runtime after sparsifying	# pathlets	# pathlets after sparsifying
RTP	1,084,257	1.02	0.54	150,142	10,217
GeoLife	1,473,115	2.05	0.83	213,478	11,456
Beijing	9,121,035	12.26	8.12	520,138	18,277
Beijing	20,012,380	50.39	21.08	1,268,598	22,651

Table 1. Runtimes showing speed-up with pathlet sparsification. (c_1, c_2, c_3) values: GeoLife and RTP: (1, 1, 0.005), smaller Beijing: (1, 1.5, 0.005), larger Beijing: (1, 1.5, 0);

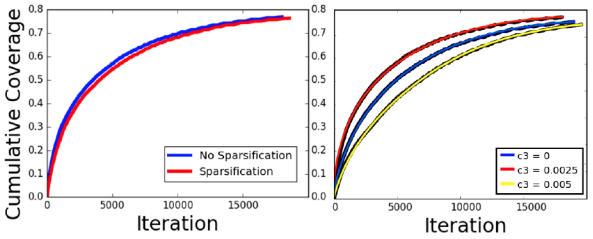

Figure 12. The cumulative coverage of chosen pathlets with and without sparsification (left). Cumulative coverage plots for different values of c_3 (with $c_1 = 1$, $c_2 = 1.75$) (right). Both plots are for Beijing.

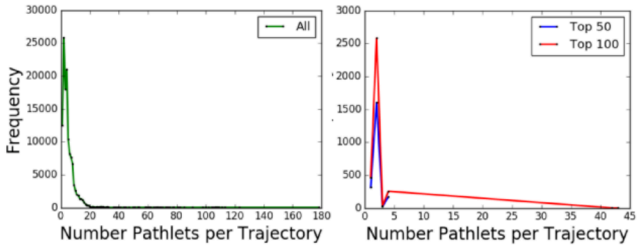

Figure 13. Plots showing the number of pathlets assigned per trajectory.

shorter and many of these are assigned to a single trajectory on average. The right plot in Figure 13 shows the same plot for the top 50 and 100 pathlets. The top pathlets usually concatenate in smaller numbers.

Parameter sensitivity. We conclude by considering the effect each parameter has on the pathlets chosen as well as the coverage of the trajectories' points.

Keeping c_1 and c_3 fixed, we vary c_2. A lower value of c_2 means it is less expensive to leave points unassigned to any pathlet; thus increasing c_2 increases the percentage of assigned points, e.g., from around 64% ($c_2 = 1.5$) to 84% ($c_2 = 2.5$) for Beijing. However, the data itself also affects how much of its points get assigned. To get the same percentage of assigned points of around 74%, we need $c_2 = 2$ for Beijing and $c_2 = 1$ for RTP and GeoLife. Urban centers lead to more route diversity (and somewhat lesser shared structure), hence we need larger c_2 values for Beijing to get the same percentage of assigned points, whereas traffic tends to cluster along the relatively few long highways in RTP. For GeoLife, although the c_2 values are similar to that of RTP, in general the number of pathlets picked is less than RTP (perhaps because in RTP there are more shared portions, and there are a lot less trajectories and hence more points per trajectory). The average number of subtrajectories assigned per pathlet reduces as c_2 increases. Thus the pathlets cover less shared structure.

We now vary c_3 while keeping c_1 and c_2 fixed. The right plot in Figure 12 shows the impact on the cumulative coverage in Beijing as pathlets are added for different values of c_3. The number of pathlets chosen increases with c_3, from 8, 169 for $c_3 = 0$, to 10, 021 for $c_3 = 0.0025$, to 11, 223 for $c_3 = 0.005$. Similar trends are observed for other data sets. As our experiments suggest, increasing c_3 discourages the assignment of many long subtrajectories to individual pathlets as the distance between these subtrajectories and pathlets has a larger and larger influence on our objective function.

8 CONCLUSION

We presented a model for constructing a pathlet dictionary and considered two optimization problems based on our model, pathlet-cover and subtrajectory clustering. After presenting hardness results, we described polynomial time polylogarithmic approximation algorithms for both problems. Finally, we evaluated our model and algorithms through experiments, and visualization and quantitative analysis of the results.

It would be interesting to see how our techniques could be generalized or applied to other settings. In particular, we would like to know if our model can be applied in contexts such as motion capture or if our techniques could be used to fit generative models to trajectory data. It would also be good to know if our algorithms could be implemented in parallel or I/O efficiently without sacrificing much in solution quality.

REFERENCES

[1] 2003. Research Triangle. (2003). http://en.wikipedia.org/wiki/Research_Triangle
[2] 2009. GeoLife. http://research.microsoft.com/en-us/projects/GeoLife/. (2009).
[3] 2009. Taxi trajectory open dataset, Tsinghua University, China. (2009). http://sensor.ee.tsinghua.edu.cn.
[4] 2013. MNTG: Minnesota web-based traffic generator. (2013). http://mntg.cs.umn.edu/tg/index.php.
[5] Ijaz Akhter, Yaser Sheikh, Sohaib Khan, and Takeo Kanade. 2011. Trajectory space: A dual representation for nonrigid structure from motion. *IEEE Trans. Patt. Analy. Mach. Intell.* 33, 7 (2011), 1442–1456.
[6] Boris Aronov, Sariel Har-Peled, Christian Knauer, Yusu Wang, and Carola Wenk. 2006. Fréchet Distance for Curves, Revisited. In *Proc. 14th Annu. Euro. Symp. Alg.* 52–63.
[7] David M Blei, Andrew Y Ng, and Michael I Jordan. 2003. Latent dirichlet allocation. *J. Mach. Learn. Res.* 3 (2003), 993–1022.
[8] Thomas Brinkhoff. 2000. Generating Network-Based Moving Objects. In *Proc. 12th Int. Conf. Scien. Statis. Database Manag.* IEEE, 253–255.
[9] Kevin Buchin, Maike Buchin, David Duran, Brittany Terese Fasy, Roel Jacobs, Vera Sacristan, Rodrigo I Silvera, Frank Staals, and Carola Wenk. 2017. Clustering trajectories for map construction. In *Proc. 25th ACM SIGSPATIAL Int. Conf. Adv. Geog. Info. Sys.* ACM.
[10] Kevin Buchin, Maike Buchin, Joachim Gudmundsson, Maarten Löffler, and Jun Luo. 2011. Detecting commuting patterns by clustering subtrajectories. *Int. J. Comput. Geom. & Appl.* 21, 03 (2011), 253–282.
[11] Kevin Buchin, Maike Buchin, Marc Van Kreveld, Maarten Löffler, Rodrigo I Silveira, Carola Wenk, and Lionov Wiratma. 2013. Median trajectories. *Algorithmica* 66, 3 (2013), 595–614.
[12] Kevin Buchin, Maike Buchin, Marc Van Kreveld, and Jun Luo. 2011. Finding long and similar parts of trajectories. *Comput. Geom.* 44, 9 (2011), 465–476.
[13] Maike Buchin, Anne Driemel, Marc van Kreveld, and Vera Sacristán. 2014. Segmenting trajectories: A framework and algorithms using spatiotemporal criteria. *J. Spat. Inf. Sc.* 3 (2014), 33–63.
[14] Hyrum Carroll, Mark J Clement, Perry Ridge, and Quinn O Snell. 2006. Effects of gap open and gap extension penalties. (2006).
[15] Chen Chen, Hao Su, Qixing Huang, Lin Zhang, and Leonidas Guibas. 2013. Pathlet learning for compressing and planning trajectories. In *Proc. 21st ACM SIGSPATIAL Int. Conf. Adv. Geo. Inf. Sys.* ACM, 382–385.
[16] Mark de Berg, Marc van Kreveld, Mark Overmars, and Otfried Cheong. 2000. *Computational Geometry* (3rd ed.). Springer.

[17] Anne Driemel, Sariel Har-Peled, and Carola Wenk. 2012. Approximating the Fréchet Distance for Realistic Curves in Near Linear Time. *Disc. Comp. Geom.* 48, 1 (2012), 94–127.

[18] Michael Elad and Michal Aharon. 2006. Image denoising via sparse and redundant representations over learned dictionaries. *IEEE Trans. Image Process.* 15, 12 (2006), 3736–3745.

[19] Qi Fan, Dongxiang Zhang, Huayu Wu, and Kian-Lee Tan. 2016. A general and parallel platform for mining co-movement patterns over large-scale trajectories. *Proc. VLDB Endow.* 10, 4 (2016), 313–324.

[20] Scott Gaffney and Padhraic Smyth. 1999. Trajectory clustering with mixtures of regression models. In *Proc. 5th ACM SIGKDD Int. Conf. Know. Disco. Data Mining*. ACM, 63–72.

[21] Joachim Gudmundsson, Andreas Thom, and Jan Vahrenhold. 2012. Of motifs and goals: mining trajectory data. In *Proc. 20th ACM SIGSPATIAL Int. Conf. Adv. Geo. Inf. Sys.* ACM, 129–138.

[22] Chih-Chieh Hung, Wen-Chih Peng, and Wang-Chien Lee. 2015. Clustering and aggregating clues of trajectories for mining trajectory patterns and routes. *VLDB J.* 24, 2 (2015), 169–192.

[23] Jakob Krarup and Peter Mark Pruzan. 1983. The simple plant location problem: survey and synthesis. *European journal of operational research* 12, 1 (1983), 36–81.

[24] Jae-Gil Lee, Jiawei Han, and Kyu-Young Whang. 2007. Trajectory clustering: a partition-and-group framework. In *Proc. 2007 ACM SIGMOD Int. Conf. Manage. Data*. ACM, 593–604.

[25] Yang Li, Qixing Huang, Michael Kerber, Lin Zhang, and Leonidas Guibas. 2013. Large-scale joint map matching of GPS traces. In *Proc. 21st ACM SIGSPATIAL Int. Conf. Adv. Geog. Info. Sys.* ACM, 214–223.

[26] Yang Li, Yangyan Li, Dimitrios Gunopulos, and Leonidas Guibas. 2016. Knowledge-based trajectory completion from sparse GPS samples. In *Proc. 24th ACM SIGSPATIAL Int. Conf. Adv. Geog. Info. Sys.* ACM, 33.

[27] Qian Liu and Garrett van Ryzin. 2008. On the Choice-Based Linear Programming Model for Network Revenue Management. *Manufac. Serv. Op. Manage.* 10, 2 (2008), 233–310.

[28] Cédric Notredame. 2007. Recent evolutions of multiple sequence alignment algorithms. *PLoS Comput. Bio.* 3, 8 (2007), e123.

[29] Costas Panagiotakis, Nikos Pelekis, Ioannis Kopanakis, Emmanuel Ramasso, and Yannis Theodoridis. 2012. Segmentation and sampling of moving object trajectories based on representativeness. *IEEE Trans. Know. Data Engg.* 24, 7 (2012), 1328–1343.

[30] Nikos Pelekis, Panagiotis Tampakis, Marios Vodas, Christos Doulkeridis, and Yannis Theodoridis. 2017. On temporal-constrained sub-trajectory cluster analysis. *Data Mining and Know. Disc.* (2017), 1–37.

[31] Nikos Pelekis, Panagiotis Tampakis, Marios Vodas, Costas Panagiotakis, and Yannis Theodoridis. 2017. In-DBMS Sampling-based Sub-trajectory Clustering.. In *Proc. 20th Int. Conf. Extend. Database Tech.* 632–643.

[32] James O Ramsay. 2006. *Functional data analysis*. Wiley Online Library.

[33] Swaminathan Sankararaman, Pankaj K Agarwal, Thomas Mølhave, Jiangwei Pan, and Arnold P Boedihardjo. 2013. Model-driven matching and segmentation of trajectories. In *Proc. 21st ACM SIGSPATIAL Int. Conf. Adv. Geo. Inf. Sys.* ACM, 234–243.

[34] Cynthia Sung, Dan Feldman, and Daniela Rus. 2012. Trajectory clustering for motion prediction. In *Proc. IEEE/RSJ Int. Conf. Intell. Robots Syst.* IEEE, 1547–1552.

[35] Lu-An Tang, Yu Zheng, Jing Yuan, Jiawei Han, Alice Leung, Wen-Chih Peng, and Thomas La Porta. 2013. A framework of traveling companion discovery on trajectory data streams. *ACM Trans. Intell. Sys. Tech.* 5, 1 (2013), 3.

[36] Shuang Wang, Lina Wu, Fuchai Zhou, Cuicui Zheng, and Haibo Wang. 2015. Group pattern mining algorithm of moving objects' uncertain trajectories. *Int. J. Comp. Comm. Control* (2015), 428–440.

[37] Hongteng Xu, Yang Zhou, Weiyao Lin, and Hongyuan Zha. 2015. Unsupervised trajectory clustering via adaptive multi-kernel-based shrinkage. In *Proc. IEEE Int. Conf. Comp. Vis.* 4328–4336.

[38] Rex Ying, Jiangwei Pan, Kyle Fox, and Pankaj K. Agarwal. 2016. A simple efficient approximation algorithm for dynamic time warping. In *Proc. 24th ACM SIGSPATIAL Int. Conf. Adv. Geog. Info. Sys.* ACM.

[39] Kai Zheng, Yu Zheng, Nicholas Jing Yuan, and Shuo Shang. 2013. On discovery of gathering patterns from trajectories. In *Proc. 29th Int. Conf. Data Engng.* IEEE, 242–253.

[40] Yue Zhou and Thomas S Huang. 2008. 'Bag of segments' for motion trajectory analysis. In *Proc. 15th Int. Conf. Image Process.* IEEE, 757–760.