Noname manuscript No.
(will be inserted by the editor)

On Discovering Motifs and Frequent Patterns in Spatial Trajectories with Discrete Fréchet Distance

Bo Tang · Man Lung Yiu · Kyriakos Mouratidis · Jiahao Zhang · Kai Wang

Received: date / Accepted: date

Abstract The discrete Fréchet distance (DFD) captures perceptual and geographical similarity between two trajectories. It has been successfully adopted in a multitude of applications, such as signature and handwriting recognition, computer graphics, as well as geographic applications. Spatial applications, e.g., sports analysis, traffic analysis, etc. require discovering similar subtrajectories within a single trajectory or across multiple trajectories. In this paper, we adopt DFD as the similarity measure, and study two representative trajectory analysis problems, namely, motif discovery and frequent pattern discovery. Due to the time complexity of DFD, these tasks are computationally challenging. We address that challenge with a suite of novel lower bound functions and a grouping-based solution. Our techniques apply directly when the analysis tasks are defined within the same or across multiple trajectories. An extensive empirical study on real trajectory datasets reveals that our approaches are 3 orders of magnitude faster than baseline solutions.

Keywords Spatial trajectory · Discrete Fréchet distance · Trajectory analysis

Bo Tang
Department of Computer Science and Engineering, Southern University of Science and Technology
E-mail: tangb3@sustech.edu.cn

Man Lung Yiu and Jiahao Zhang
Department of Computing, The Hong Kong Polytechnic University
E-mail: {csmlyiu,csjhzhang}@comp.polyu.edu.hk

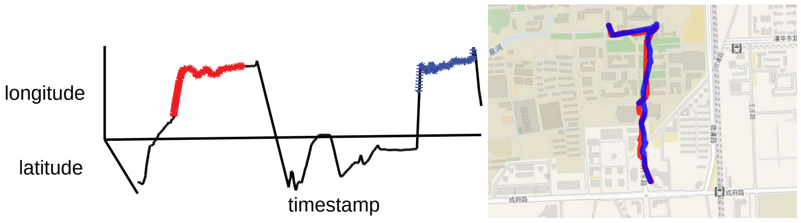
Kyriakos Mouratidis
School of Information Systems, Singapore Management University
E-mail: kyriakos@smu.edu.sg

Kai Wang
School of Computer Science and Engineering, University of New South Wales
E-mail: kai.wang@unsw.edu.au

1 Introduction

With the advancement of location positioning and tracking technologies, the amount of trajectory data generated daily is increasing rapidly. Thus, spatial trajectory analysis is becoming more important and relevant than ever, being prevalent in many applications, e.g., moving object analysis [21,25], traffic estimation/prediction systems [26,43,22, 44], etc.

Intuitively, it is important to choose a suitable similarity measure for the above spatial trajectory analysis problems. The continuous Fréchet metric is amongst the most popular such measures [32,20]. The continuous Fréchet distance between two spatial trajectories, S_a and S_b, is the length of the shortest leash needed to walk a dog when the person walks along S_a and the dog walks along S_b, where both are allowed to control their speed but they are not allowed to go backwards. In the geographic information handbook [20], the authors conclude that *"The most successful fundamental distance measure to this date is probably the Fréchet metric, which is one of the most natural measures to calculate the similarity between two trajectories"*. The Fréchet distance has been used successfully in a number of application domains, such as handwriting recognition [30], bioinformatics [38], measuring similarity between curves [14], as well as geographic applications [6]. As Fréchet distance is computationally expensive to calculate, many recent studies [6,20,10,22,33,39] have adopted the discrete Fréchet distance (DFD), which provides an upper bound on the continuous Fréchet distance, with a deviation no greater than the distance of the trajectory's longest edge. In addition, as we will elaborate in Section 2, DFD is particularly suitable for real-world spatial trajectories which are sufficiently sampled by GPS devices but bear the following properties: (i) varying sampling rate, and (ii) missing samples at some time points. For example, the GeoLife dataset [44], a real spatial trajectory dataset collected by Microsoft, exhibits both the above properties.



(a) A pedestrian's trajectory April 10-12, 2009

(b) Discovered motif red: 07:33-7:48, April 10, 2009 blue: 07:33-7:50, April 12, 2009

Fig. 1 Motif in a trajectory (from GeoLife)

Detecting similar subtrajectories has been used in many applications, such as team sports analysis [21] and traffic analysis [25], as well as a building block for other trajectory mining and analysis tasks [26,43,22]. Using real trajectories, Figure 1(a) visualizes a pedestrian's GPS trajectory (from the GeoLife dataset [44]), by a 3D plot with time as the horizontal axis. The *motif* corresponds to the most similar pair of subtrajectories, shown in red and blue in Figure 1(b), and can be used, among others, as a building block

Table 1 Input and output of the studied problems

	Motif discovery	Frequent pattern discovery
Input	trajectory $\mathcal{S}$, minimum motif length ξ	trajectory $\mathcal{S}$, length ξ, distance threshold d_{thres}, frequency threshold t_{thres}
Output	a subtrajectory pair $(\mathcal{S}_{i,i_e}, \mathcal{S}_{j,j_e})$ where $d_F(i, i_e, j, j_e)$ is minimal	all $(\mathcal{S}_{i,i_e}, \Gamma_i)$ where $\lvert\Gamma_i\rvert \geq t_{thres}$ and $\forall \mathcal{S}_{j,j_e} \in \Gamma_i d_F(i, i_e, j, j_e) \leq d_{thres}$

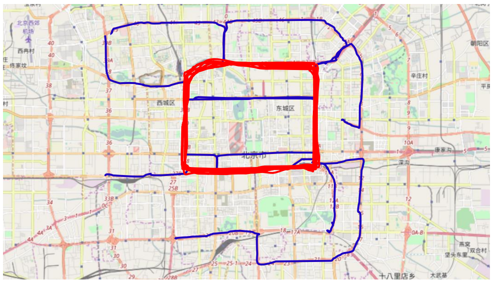


Fig. 2 Frequent pattern in a one-day taxi trajectory (from T-Drive)

in subtrajectory clustering. A *frequent pattern*, as its name suggests, represents a frequently occurring subtrajectory in a given trajectory. For instance, Figure 2 illustrates a frequent pattern (shown in red) in a one-day taxi trajectory from the T-Drive dataset [42]. It indicates a recurrent travel pattern, which can be used in intelligent transportation systems. Specifically, motif is **the most similar subtrajectory pair** in the given trajectory, while a frequent pattern is a **set of subtrajectories** that includes at least a specified number of similar subtrajectories, concentrating around the representative subtrajectory of the set. We summarize the input and output of both studied problems in Table 1; used notation and specific definitions are elaborated in Sections 3 and 4, respectively.

Although the above two examples consider a single trajectory, those problems can also be extended to multiple trajectories too (e.g., a trajectory database). For example, a motif discovered in multiple player trajectories can be used in a sports analysis application [21], and the frequent pattern in multiple trajectories could be applied to intelligent transportation systems [25]. In addition, those problems serve as building blocks in a wide range of trajectory mining tasks (e.g., trajectory clustering, trajectory classification, trajectory prediction, trajectory outlier detection). For instance, they enable us to discover the habits of moving objects (e.g., buses, animals, sports players).

Technical Challenges: We consider spatial trajectory analysis problems with DFD as the similarity measure. Specifically, we study how to compute motifs and frequent patterns. These problems are computationally challenging for two reasons:

1. *Expensive DFD computation.* The computation of DFD between two subtrajectories takes $O(\ell^2)$ time [17,7], where ℓ denotes the subtrajectory length. There have been attempts to speed up DFD computation by using GPUs [22] or a faster algorithm (with $O(\ell^2 \cdot \frac{\log\log\ell}{\log\ell})$ time complexity) [1]. In contrast, we take an orthogonal research direction to reduce the number of DFD computations via various types of pruning techniques.
2. *Huge search space.* The search space of both addressed problems is huge. E.g., motif discovery on a trajectory involves $O(n^4)$ pairs of subtrajectories, where n is the input

trajectory length. Furthermore, DFD exhibits non-monotonicity (see Section 3.1.1), i.e., appending additional vertices to trajectory $\mathcal{S}$ can result in the non-monotonicity of the DFD measures between two (sub)trajectories $\mathcal{S}$ and $\mathcal{P}$. Thus, it prevents us from applying efficient algorithmic paradigms (like binary search) to reduce the search space.

Our Contributions: To overcome the above challenges, we exploit the properties of DFD and devise efficient lower bound functions to reduce the search space and guide the search. Furthermore, we propose a grouping-based approach and optimize it with multi-level pruning techniques. All our proposed techniques and algorithms are exact, i.e., they produce exact results for both studied problems.

Our preliminary work [31] focuses on motif discovery on a single trajectory. In this extension paper, we also consider the frequent pattern problem, and design efficient techniques to solve it. The idea is to transform the expensive DFD computation to a cheaper test on Boolean values. This transformation allows us to save memory and also utilize fast operations on Boolean values. Furthermore, we adapt the lower bound functions for motif discovery to solve the frequent pattern discovery problem. Finally, we also extend our solutions to a multiple-trajectory database.

The rest of the paper is organized as follows. Section 2 reviews related work and provides preliminary information. Sections 3 and 4 study the motif discovery problem and the frequent pattern discovery problem, respectively. Section 5 extends both problems to a trajectory database. Section 6 demonstrates empirically the efficiency of our solutions on real trajectory datasets. Finally, Section 7 concludes the paper with directions for future work.

2 Related Work & Preliminaries

In this section, we first present alternative similarity measures and pinpoint the advantages offered by the (discrete) Fréchet metric that render it an appropriate choice for (sub)trajectory similarity [20]. We then overview the most relevant research in the spatial trajectory analysis area. We discuss existing techniques in trajectory motif discovery and frequent pattern discovery problems, and juxtapose them to ours. In addition, we provide an outlook of other practically relevant trajectory analysis techniques.

Trajectory Similarity Measures: The literature includes a wealth of spatial-temporal similarity measures for trajectories, e.g., Euclidean Distance (ED) [24], Dynamic Time Warping (DTW) [40], Longest Common Subsequence (LCSS) [34], Edit Distance on Real Sequence (EDR) [15], Fréchet Distance (FD) [3] and its discrete variant (DFD) [17, 1]. Sequence measures consider the number of edit operations required to transform one trajectory into the other (e.g., EDR, LCSS). Other measures (e.g., FD) consider both shape and temporal aspects. For a taxonomy of available measures, we refer the reader to surveys [20,32].

Real-world trajectories (e.g., GeoLife dataset) exhibit two key characteristics, namely, varying sampling rate and missing samples for some time points. A desirable similarity measure would account for these characteristics. ED is the fastest metric to compute but it is not robust to missing samples for some time points. More robust measures, such as DTW [40], LCSS [34], EDR [15], are defined as the sum of point-to-point distances, which makes them sensitive to the sampling rate. In contrast, FD and its discrete variant DFD, also known as the "dog-man" distance, can tolerate varying sampling rates [10,32,22] if the GPS points are sampled sufficiently in real-world trajectories. In this work we use the discrete version (DFD) as (i) DFD provides a good approximation of FD [10] for real-world trajectories sampled with sufficient GPS points, and (ii) in real life, a GPS-recorded trajectory is a series of discrete locations.

The parallel computing and computational geometry communities have proposed some techniques to speed up DFD computation [22,1,4,7], In contrast, we take an orthogonal research direction to accelerate the analysis tasks (i.e., motif discovery, frequent pattern discovery) via novel pruning techniques.

Trajectory Analysis Techniques: Several studies [12,13,19,29,35] have considered how to extract motif or frequently occurring patterns from trajectories. They adopt the *transformation approach*. First, they transform each trajectory into a string of symbols with timestamps, by methods like line segmentation [12], clustering of points [19], map matching [29]. This step may incur information loss. Then, they apply existing sequential pattern mining techniques (e.g., LCSS, Suffix Tree, PrefixSpan [23]) on those strings. These techniques differ from our approach in two aspects. First, they operate on the strings obtained from transformation, whereas our approach runs on raw trajectories. Second, they do not apply to the scenario where DFD is used as the similarity measure.

Besides motif discovery and frequent pattern discovery, there are many other spatial trajectory analysis problems, e.g., convoy discovery [24], outlier detection [41,27], trajectory clustering [26,21,22], etc. We refer the interested reader to a recent survey [43].

Range Query with Fréchet Distance: This paper is an extension of our conference piece [31], published in EDBT (March 2017). Since then, there has been work on a related problem, namely, range query with Fréchet Distance [11,5,16,36,37,8]. These subsequent studies address a different problem (range query vs. motif/frequent pattern discovery) and distance measure (Fréchet Distance vs. Discrete Fréchet Distance). They all apply a similar technique to our "cell-based" bound, but we stress that this is by far the simplest of our bounds, which was anyway present in our preliminary publication that precedes them. Clearly, also, the "delta" in this extension paper over the base version [31], i.e., the frequent pattern discovery problem, is an even more distinct problem from these works.

2.1 Preliminaries

We introduce several basic definitions for our problems in the rest of the paper.

Definition 1 (Spatial Trajectory & Subtrajectory) A spatial trajectory $\mathcal{S} = \langle s_0, \cdots, s_i, \cdots, s_{n-1} \rangle$ is a sequence of points. We denote its trajectory length, i.e., the number of points along the trajectory, by $n = |\mathcal{S}|$.

Given a trajectory $\mathcal{S}$, we denote a subtrajectory of $\mathcal{S}$ as $\mathcal{S}_{i,i_e} = \mathcal{S}[i \cdots i_e]$, where $0 \le i < i_e \le n-1$. Let $T(\mathcal{S}) = \langle t_0, \cdots, t_i, \cdots, t_{n-1} \rangle$ be a sequence of timestamps, where t_i is the timestamp of location s_i in $\mathcal{S}$. The timestamps may or may not be spaced uniformly.

We assume each point s_i is a latitude-longitude (φ_i, λ_i) pair. We measure the *ground distance* between two trajectory points $s_i = (\varphi_i, \lambda_i), s_j = (\varphi_j, \lambda_j)$ as the great circle distance on earth [28]:

$$d_G(i,j) = 2R\arcsin\sqrt{\sin^2\left(\frac{\varphi_j-\varphi_i}{2}\right) + \cos\varphi_i\cos\varphi_j\sin^2\left(\frac{\lambda_j-\lambda_i}{2}\right)}$$

where R is the radius of the earth. Nevertheless, our methods are directly applicable to higher dimensions (e.g., 3-d data points) and other types of ground distance (e.g., Euclidean).

As discussed in Section 2, we adopt the discrete Fréchet distance (DFD) to measure the distance between two subtrajectories $\mathcal{S}_{i,i_e}$ and $\mathcal{S}_{j,j_e}$. Eiter et al. [17] define the DFD as the alignment of vertices minimizing the maximum distance. It is equivalent to the following recursive equation:

$$d_F(i,i_e,j,j_e) = \max\begin{cases} d_G(i_e,j_e) \\ \min\begin{cases} d_F(i,i_e-1,j,j_e), \\ d_F(i,i_e,j,j_e-1), \\ d_F(i,i_e-1,j,j_e-1) \end{cases} \end{cases}$$

When $i_e = i$ and $j_e = j$, the recursion terminates and we get $d_F(i,i,j,j) = d_G(i,j)$.

3 Motif Discovery

In this section, we study the motif discovery problem on a trajectory. To produce a meaningful trajectory motif $(\mathcal{S}_{i,i_e}, \mathcal{S}_{j,j_e})$, we require that: (i) subtrajectories $\mathcal{S}_{i,i_e}$ and $\mathcal{S}_{j,j_e}$ are sufficiently long (e.g., each has length at least ξ), and (ii) their timestamp intervals do not overlap.

Problem 1 (Trajectory Motif Discovery) Given a trajectory $\mathcal{S}$ and a minimum motif length ξ (i.e., the minimum number of vertices in a subtrajectory), return the pair of subtrajectories $\mathcal{S}_{i,i_e}$ and $\mathcal{S}_{j,j_e}$ with the smallest DFD distance $d_F(i,i_e,j,j_e)$ among all pairs of non-overlapping subtrajectories (that is, $i < i_e < j < j_e$) with length at least ξ (that is, $i_e \geq i+\xi-1, j_e \geq j+\xi-1$).

Inspired by [3], a straightforward solution for Problem 1 is to enumerate all pairs of subtrajectories $(\mathcal{S}_{i,i_e}, \mathcal{S}_{j,j_e})$ and then compute the DFD value for each pair. Its time complexity is $O(n^6)$, as there are $O(n^4)$ pairs of subtrajectories and each call to DFD takes $O(\ell^2) = O(n^2)$ time. Even if we implement each call to DFD by [1], the time complexity is still $O(n^6 \cdot \frac{\log\log n}{\log n})$. We observe that, for all subtrajectory pairs $(\mathcal{S}_{i,i_e}, \mathcal{S}_{j,j_e})$ with the same start point (i,j), their DFD computation can be shared via dynamic programming. By incorporating this idea into the above solution, we obtain BruteDP (Algorithm 1) – a brute force algorithm that uses dynamic programming.

Algorithm 1 can be adapted to motif discovery between different trajectories easily, i.e., with $\mathcal{S}_{j,j_e}$ playing the role of a subtrajectory in the second input trajectory, and by incrementing i until $n-\xi+1$ (instead of $n-2\xi+1$) at Line 2, and j starting from 0 (instead of $i+\xi$) at Line 3 (as this variant considers separate trajectories, thus not imposing the constraint $i < i_e < j < j_e$).

Analysis: The time complexity of Algorithm 1 is $O(n^4)$, which is attributed to the nested for-loops for variables i, j (at Lines 2-3) and variables i_e, j_e (at Lines 8-9). The

Algorithm 1 BruteDP ($\mathcal{S}$, ξ)

```
    Input: trajectory $\mathcal{S}$, length $n$, minimum motif length $\xi$
    Output: subtrajectory pair $bpair = (\mathcal{S}_{i,i_e}, \mathcal{S}_{j,j_e})$
    compute the ground distance matrix $d_G$
 1: $bsf \leftarrow +\infty$; $bpair \leftarrow \emptyset$
 2: for $i \leftarrow 0$ to $n - 2\xi + 1$ do
 3:     for $j \leftarrow i + \xi$ to $n - \xi + 1$ do
 4:         $d_F[i][j] \leftarrow d_G(i, j)$                                  ▷ initialization
 5:         for $t \leftarrow i + 1$ to $n$ do
 6:             $d_F[i][t] \leftarrow \max(d_G(i,t), d_F[i][t\text{-}1])$
 7:             $d_F[t][j] \leftarrow \max(d_G(t,j), d_F[t\text{-}1][j])$
 8:         for $i_e \leftarrow i + 1$ to $j - 1$ do                         ▷ share DFD computation
 9:             for $j_e \leftarrow j + 1$ to $n$ do
10:                 $tmp \leftarrow \min(d_F[i_e\text{-}1][j_e\text{-}1], d_F[i_e][j_e\text{-}1], d_F[i_e\text{-}1][j_e])$
11:                 $d_F[i_e][j_e] \leftarrow \max(d_G(i_e, j_e), tmp)$
12:                 if $i_e \geq i + \xi - 1, j_e \geq j + \xi - 1$ and $d_F[i_e][j_e] < bsf$ then
13:                     $bsf \leftarrow d_F[i_e][j_e]$; $bpair \leftarrow (\mathcal{S}_{i,i_e}, \mathcal{S}_{j,j_e})$
14: return $bpair$
```

space complexity of the algorithm is $O(n^2)$, as it employs two 2-dimensional matrices: (i) $d_F[\cdot][\cdot]$ for implementing dynamic programming, and (ii) $d_G[\cdot][\cdot]$ for holding all-pair ground distances.

3.1 Advanced Solution

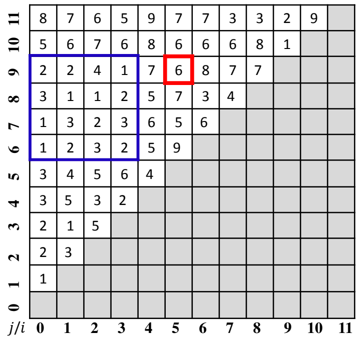
We first analyze the properties of DFD (in Section 3.1.1). We devise novel lower bound functions for DFD (in Section 3.1.2). Our lower bounds can be computed in amortized $O(1)$ time, and guarantee no false negatives (in Section 3.1.3). Finally, we propose a bounding-based solution that applies our lower bound functions to prune unpromising pairs of trajectories and reduce the number of DFD computations (in Section 3.1.4).

3.1.1 Properties of DFD

Non-monotonicity: Typical sequence/string mining algorithms exploit the monotone property to develop efficient Apriori-style algorithms. An example of the monotone property would be: "given a string S, if q_α is a substring of q_β, then the frequency of q_α in S cannot be smaller than the frequency of q_β in S." It would be tempting to adapt such an idea to solve our problem efficiently. Unfortunately, the DFD metric does not satisfy the monotone property. Formally:

Definition 2 (Containment $\subseteq$) $\mathcal{S}_{i,i_e}$ is said to contain $\mathcal{S}_{i',i'_e}$, denoted as $\mathcal{S}_{i',i'_e} \subseteq \mathcal{S}_{i,i_e}$, if and only if $i' \geq i$ and $i'_e \leq i_e$.

Lemma 1 (Non-monotonicity) *Let* $(\mathcal{S}_{i,i_e}, \mathcal{S}_{j,j_e})$ *be a subtrajectory pair of* S. *Let* $\mathcal{S}_{i',i'_e}, \mathcal{S}_{j',j'_e}$ *be subtrajectories that satisfy* $\mathcal{S}_{i',i'_e} \subseteq \mathcal{S}_{i,i_e}, \mathcal{S}_{j',j'_e} \subseteq \mathcal{S}_{j,j_e}$. *It holds that,* $d_F(i, i_e, j, j_e)$ *is neither monotone increasing nor monotone decreasing with respect to* $d_F(i', i'_e, j', j'_e)$.



11	8	7	6	5	9	7	7	3	3	2	9	
10	5	6	7	6	8	6	6	6	8	1		
9	2	2	4	1	7	6	8	7	7			
8	3	1	1	2	5	7	3	4				
7	1	3	2	3	6	5	6					
6	1	2	3	2	5	9						
5	3	4	5	6	4							
4	3	5	3	2								
3	2	1	5									
2	2	3										
1	1											
0												
j/i	**0**	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**

Fig. 3 Example of d_G matrix

For the proof, we refer the interested reader to [3]. However, here we provide a counter-example to demonstrate the non-monotonicity as follows.

Example: Consider a trajectory $\mathcal{S}$ with length $n = 12$. Figure 3 shows the ground distance for each pair $(\mathcal{S}[i], S[j])$. Consider three subtrajectories $\mathcal{S}_{0,2} \subseteq \mathcal{S}_{0,3} \subseteq \mathcal{S}_{0,4}$ and their DFD distances from $\mathcal{S}_{6,9}$. Using Algorithm 1, we can compute these DFD values: $d_F(0,2,6,9) = 4$, $d_F(0,3,6,9) = 1$, $d_F(0,4,6,9) = 7$. When comparing $\mathcal{S}_{0,2}$ and $\mathcal{S}_{0,3}$, the DFD value (from $\mathcal{S}_{6,9}$) decreases from 4 to 1. However, when comparing $\mathcal{S}_{0,3}$ and $\mathcal{S}_{0,4}$, the DFD value (from $\mathcal{S}_{6,9}$) increases from 1 to 7. Thus, DFD does not satisfy the monotone property.

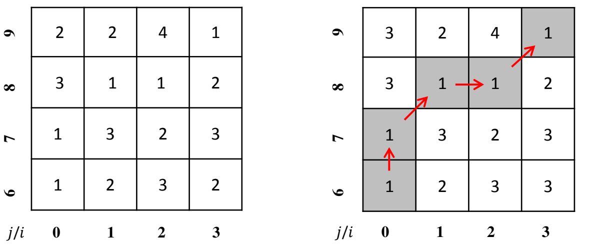
Non-monotonicity aside, a crucial observation mentioned in [2,6] is quintessential to our approach. Specifically, the computation of DFD by recurrence is equivalent to a path finding problem in the d_G matrix as follows.

Observation 1 *The DFD between $\mathcal{S}_{i,i_e}$ and $\mathcal{S}_{j,j_e}$ must be contributed by a path from (i,j) to (i_e, j_e) such that: (i) the path travels along non-decreasing positions, and (ii) the maximum possible ground distance along the path is minimized.*

We illustrate using two subtrajectories $\mathcal{S}_{0,3}$ and $\mathcal{S}_{6,9}$. Figure 4(a) shows the ground distance d_G for each pair of points from $\mathcal{S}_{0,3}$ and $\mathcal{S}_{6,9}$ (note that only the relevant part of the d_G matrix from Figure 3 is shown). We compute the d_F value for each pair of points, as illustrated in Figure 4(b). The DFD distance is $d_F(0,3,6,9) = 1$, which is contributed by the path of gray cells from $(0,6)$ to $(3,9)$, that minimizes the maximum ground distance among the cells it visits.

3.1.2 Pattern-based Lower Bounds

From Observation 1, we devise novel lower bound functions for DFD by traversing the d_G matrix according to different patterns (e.g., a single cell, cells in a cross, cells in a band). Specifically, assuming that matrix d_G is precomputed and that bsf is the DFD of the best subtrajectory pair encountered so far in the search process, we propose a set of lower bound functions that apply to candidate subtrajectory pairs, or entire groups of candidate pairs, such that if the bound is greater than bsf, the candidates are safe to



(a) Relevant part of d_G (b) d_F computation as a path in d_G

Fig. 4 DFD computation for $\mathcal{S}_{0,3}$ and $\mathcal{S}_{6,9}$

prune, i.e., to disqualify without further consideration, because they are guaranteed not to be the motif.

Cell-based Lower Bound: We refer to a subtrajectory pair $(\mathcal{S}_{i,i_e}, \mathcal{S}_{j,j_e})$ as candidate (i, i_e, j, j_e). We define a candidate subset $CS_{i,j}$ to represent all candidates with the same start positions i and j. This compact notation, using a pair (i, j), allows us to represent $O(n^2)$ candidates.

Definition 3 (Candidate Subset) Given two start positions i and j, the candidate subset is defined as $CS_{i,j} = \{(i, i_e, j, j_e) : i_e > i \wedge j_e > j\}$.

The following holds for any $CS_{i,j}$.

Observation 2 *For every $(i, i_e, j, j_e) \in CS_{i,j}$, the path leading to $d_F(i, i_e, j, j_e)$ must start from cell (i, j).*

For example, in Figure 4(a), for each candidate in $CS_{i,j}$, the path leading to DFD must start at cell $(0, 6)$. We thus derive our first bound, which applies to any candidate in $CS_{i,j}$:

$$LB_{cell}(i, j) = d_G(i, j) \tag{1}$$

For every $(i, i_e, j, j_e) \in CS_{i,j}$, $LB_{cell}(i, j) \leq d_F(i, i_e, j, j_e)$.

Example. In Figure 3, for candidate subset $CS_{5,9}$ (i.e., for all candidate pairs that start at the red cell), we obtain $LB_{cell}(5, 9) = d_G(5, 9) = 6$. This is a lower bound for the DFD of any candidate pair in $CS_{5,9}$. E.g., for pair $(\mathcal{S}_{5,6}, \mathcal{S}_{9,11})$, the exact DFD is $d_F(5, 6, 9, 11) = 7$.

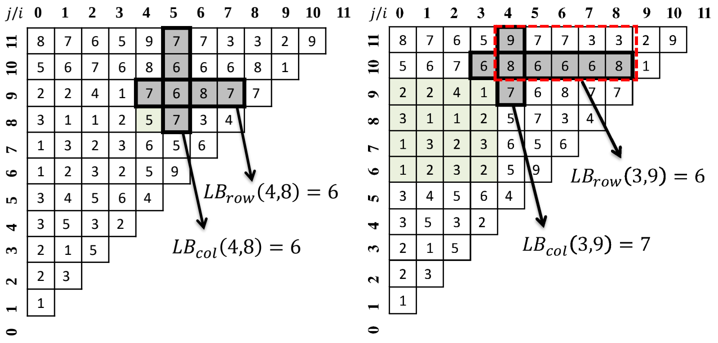
Cross-based Lower Bound: If a candidate subset is not pruned using LB_{cell}, we attempt to prune it with tighter bounds.

Observation 3 *For every $(i, i_e, j, j_e) \in CS_{i,j}$, the path leading to $d_F(i, i_e, j, j_e)$ must pass through the $(i + 1)$-th column and $(j + 1)$-th row.*

We thus define the following row-based and column-based lower bounds.

$$LB_{row}(i, j) = \min_{i' \in [i, j-1]} \{d_G(i', j + 1)\} \tag{2}$$

$$LB_{col}(i, j) = \min_{j' \in [j, n-1]} \{d_G(i + 1, j')\} \tag{3}$$



(a) $LB^{start}_{cross}(4, 8)$ (b) $LB^{end}_{cross}(3, 9)$

Fig. 5 Examples of cross-based bounds

For every $(i, i_e, j, j_e) \in CS_{i,j}$, it holds that $LB_{row}(i, j) \leq d_F(i, i_e, j, j_e)$ and that $LB_{col}(i, j) \leq d_F(i, i_e, j, j_e)$. Thus, we combine the two into the cross-based lower bound below:

$$LB^{start}_{cross}(i, j) = \max\left(LB_{row}(i, j), LB_{col}(i, j)\right) \tag{4}$$

For every $(i, i_c, j, j_c) \in CS_{i,j}$, $LB^{start}_{cross}(i, j) \leq d_F(i, i_c, j, j_c)$.

Example. Consider cell (4,8) in Figure 5(a), and assume that $n = 12$. $LB^{start}_{cross}(4, 8)$ is computed over the gray cells as follows:

$$\begin{aligned}
LB^{start}_{cross}(4, 8) &= \max(LB_{row}(4, 8), LB_{col}(4, 8)) \\
&= \max\left(\min_{i' \in [4,7]}\{d_G(i', 9)\}, \min_{j' \in [8,11]}\{d_G(5, j')\}\right) \\
&= \max(6, 6) = 6
\end{aligned}$$

Band-based Lower Bound: Our problem definition considers only subtrajectories with length at least ξ. Based on that, we extend Observation 3 to:

Observation 4 *For every $(i, i_e, j, j_e) \in CS_{i,j}$ that satisfies the constraint $i_e > i + \xi$ and $j_e > j + \xi$, the path leading to $d_F(i, i_e, j, j_e)$ must pass through columns $i + 1$ to $i + \xi$ and through rows $j + 1$ to $j + \xi$.*

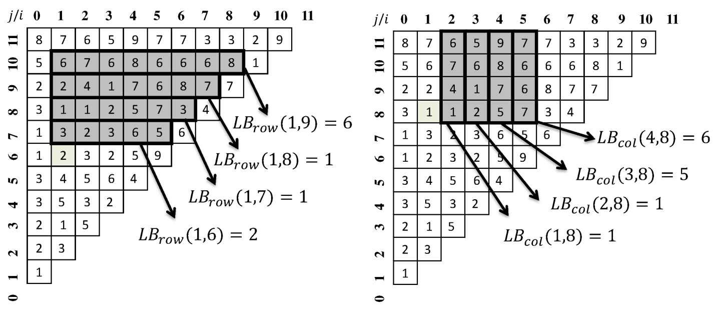
Hence, we define the following band-based lower bounds:

$$LB^{row}_{band}(i, j) = \max_{j' \in [j, j+\xi-1]}\{LB_{row}(i, j')\} \tag{5}$$

$$LB^{col}_{band}(i, j) = \max_{i' \in [i, i+\xi-1]}\{LB_{col}(i', j)\} \tag{6}$$

For every $(i, i_e, j, j_e) \in CS_{i,j}$ where $i_e > i + \xi$ and $j_e > j + \xi$, it holds that:

$$\max\{LB^{row}_{band}(i, j), LB^{col}_{band}(i, j)\} \leq d_F(i, i_e, j, j_e) \tag{7}$$



(a) $LB^{row}_{band}(1, 6)$ (b) $LB^{col}_{band}(1, 8)$

Fig. 6 Example of band-based bound

If $LB^{row}_{band}(i, j) \geq bsf$ or $LB^{col}_{band}(i, j) \geq bsf$ we can safely prune $CS_{i,j}$.

Example. Consider candidate subset $CS_{1,6}$ in Figure 6(a). Suppose the minimum motif length is $\xi = 4$ and $n = 12$. By the definition of $LB_{row}(i, j)$, the minimum values in the 7-th, 8-th, 9-th and 10-th row are 2, 1, 1 and 6, respectively. Hence, $LB^{row}_{band}(1, 6) = \max(2, 1, 1, 6) = 6$. Similarly, consider candidate subset $CS_{1,8}$ in Figure 6(b). By the definition of $LB_{col}(i, j)$, the minimum value of the 2-nd, 3-rd, 4-th and 5-th column are 1, 1, 5 and 6, respectively, as shown in Figure 6(b). Hence, $LB^{col}_{band}(1, 8) = \max(1, 1, 5, 6) = 6$.

Pruning within Candidate Subset: The bounds presented so far prune entire candidate subsets. If a candidate subset $CS_{i,j}$ survives these bounds, we need to consider candidate pairs in it. To avoid considering all candidate pairs in $CS_{i,j}$, we introduce a cross-based bound that applies to candidate pairs.

As described in Algorithm 1, for all candidate pairs (i.e., $\mathcal{S}_{i,i_e}, \mathcal{S}_{j,j_e}$) in candidate set $CS_{i,j}$, their DFD computation can be shared via dynamic programming. Assume that at some point, the dynamic programming reaches end-cell (i_e, j_e), where $i_e - i > \xi$, $j_e - j > \xi$ and $bsf = d_F(i, i_e, j, j_e)$. We define the following cross-based lower bound for the end-cell:

$$LB^{end}_{cross}(i_e, j_e) = \max\left(LB_{row}(i_e, j_e), LB_{col}(i_e, j_e)\right) \tag{8}$$

If (i, i_c, j, j_c) is a candidate in $CS_{i,j}$ where $i_c > i_e$ and $j_c > j_e$, it holds that $LB^{end}_{cross}(i_e, j_e) \leq d_F(i, i_c, j, j_c)$. Hence, if $LB^{end}_{cross}(i_e, j_e) \geq bsf$, we can safely avoid expanding cell (i_e, j_e), i.e., eliminate paths within $CS_{i,j}$ that pass via cell (i_e, j_e).

Example. In Figure 5(b), suppose $\xi = 2$, $i = 0$, $j = 6$, $i_e = 3$ and $j_e = 9$. $LB^{end}_{cross}(i_e, j_e)$ is computed over the gray cells:

$$\begin{aligned} LB^{end}_{cross}(3, 9) &= \max(LB_{row}(3, 9), LB_{col}(3, 9)) \\ &= \max\Big(\min_{i'_e \in [3,8]}\{d_G(i'_e, 10)\}, \min_{j'_e \in [9,11]}\{d_G(4, j'_e)\}\Big) \\ &= \max(6, 7) = 7 \end{aligned}$$

If $LB^{end}_{cross}(3, 9) \geq bsf$, we prune the candidates (i.e., subtrajectory pairs) in $CS_{0,6}$ whose end-cells fall in the red dotted box.

3.1.3 Relaxed Lower Bounds

If we follow the aforementioned equations directly, a cross-based bound takes $O(n)$ time to compute and a band-based bound takes $O(\xi n)$ time. Although both of them are more efficient than raw DFD computation (i.e., $O(n^2)$), in this section, we drop their amortized time complexity to $O(1)$ by relaxing them slightly. These relaxed bounds incur no false negatives, i.e., they are guaranteed not to miss the motif. For brevity, we illustrate our relaxation approach for band-based bounds only. The relaxation of cross-based bounds follows the same lines.

The key idea is to employ one parameter per bound, and keep them in matrices for rapid access. First, we compute the minimum value for each column i and each row j:

$$C_{min}[i] = \min_{j' \in [0, j-1]} (d_G(i+1, j')) \tag{9}$$

$$R_{min}[j] = \min_{i' \in [i, n-1]} (d_G(i', j+1)) \tag{10}$$

This step takes $O(2 \cdot n \cdot n) = O(n^2)$ time as it computes every pairwise ground distance in two subtrajectories.

We define the relaxed version of cross-based bounds as:

$$rLB_{cross}^{start}(i, j) = \max\{C_{min}[i], R_{min}[j]\} \tag{11}$$

$$rLB_{cross}^{end}(i_e, j_e) = \max\{C_{min}[i_e], R_{min}[j_e]\} \tag{12}$$

In turn, the relaxed band-based bounds are defined as:

$$rLB_{band}^{row}(j) = \max_{j' \in [j, j+\xi-1]} \{R_{min}[j']\} \tag{13}$$

$$rLB_{band}^{col}(i) = \max_{i' \in [i, i+\xi-1]} \{C_{min}[i']\} \tag{14}$$

Lemma 2 shows that the relaxed cross-base bounds also satisfy the lower bound property.

Lemma 2 *It holds that:*

$$rLB_{cross}^{start}(i, j) \le LB_{cross}^{start}(i, j)$$
$$rLB_{cross}^{end}(i_e, j_e) \le LB_{cross}^{end}(i_e, j_e)$$

We compute the relaxed version of cross-based bounds by calculating $C_{min}[i]$ and $R_{min}[j]$ for each column i and each row j. This step takes $O(n)$ time per column/row. Similarly, we compute relaxed band-based bounds for each column i and each row j. This step takes $O(\xi n)$ time per row/column. Thus, the total computation time of cross-based and band-based lower bounds is $O(n \cdot n) = O(n^2)$ and $O(\xi n \cdot n) = O(\xi n^2)$, respectively. By amortizing the computation time over all candidate subsets $CS_{i,j}$ (i.e., $O(n^2)$ of them), the computation time per $CS_{i,j}$ for cross-based and band-based relaxed bounds is only $O(n^2/n^2) = O(1)$ and $O(\xi n^2/n^2) = O(\xi)$, respectively.

The following lemma proves the correctness of the relaxed band-based bounds.

Table 2 Summary of lower bounds

Lower bound	Time	Relaxed bound	Amortized time	Prunes
$LB_{cell}(i,j)$	$O(1)$			entire candidate subset
$LB_{cross}^{start}(i,j)$	$O(n)$	$rLB_{cross}^{start}(i,j)$	$O(1)$	entire candidate subset
$LB_{cross}^{end}(i_e,j_e)$	$O(n)$	$rLB_{cross}^{end}(i_e,j_e)$	$O(1)$	within candidate subset
$LB_{band}^{row}(i,j)$	$O(\xi n)$	$rLB_{band}^{row}(j)$	$O(\xi)$	entire candidate subset
$LB_{band}^{col}(i,j)$	$O(\xi n)$	$rLB_{band}^{col}(i)$	$O(\xi)$	entire candidate subset

Lemma 3 *It holds that:*

$$rLB_{band}^{row}(j) \le LB_{band}^{row}(i,j) \text{ and } rLB_{band}^{col}(i) \le LB_{band}^{col}(i,j)$$

Proof

$$\begin{aligned}
&\min_{i'\in[0,j-1]}(d_G(i',j+1)) \le \min_{i'\in[i,j-1]}(d_G(i',j+1))\\
&\Rightarrow R_{min}[j] \le LB_{row}(i,j)\\
&\Rightarrow \max_{j'\in[j,j+\xi-1]}\{R_{min}[j']\} \le \max_{j'\in[j,j+\xi-1]}\{LB_{row}(i,j')\}\\
&\Rightarrow rLB_{band}^{row}(j) = LB_{band}^{row}(i,j)
\end{aligned}$$

Similarly, $rLB_{band}^{col}(i) \le LB_{band}^{col}(i,j)$.

In the experiments, we compare the effectiveness of the original bounds with the relaxed ones. We summarize the time requirements of all lower bounds in Table 2.

3.1.4 Optimized Solution

Combining All Bounds: Given a candidate subset $CS_{i,j}$, we compute a tighter lower bound for $CS_{i,j}$, denoted by $CS_{i,j}.LB$, using:

$$\max\{LB_{cell}(i,j), rLB_{cross}^{start}(i,j), rLB_{band}^{row}(j), rLB_{band}^{col}(i)\}.$$

This lower bound takes $O(1)$ time because each term can be obtained in $O(1)$ time, as shown in Table 2. The minimum motif length ξ affects the tightness of $CS_{i,j}.LB$ as $rLB_{band}^{row}(j)$ and $rLB_{band}^{col}(i)$ take ξ into account.

Prioritizing Search Order: To support effective pruning of $CS_{i,j}$ by lower bounds, it is desirable to obtain a small bsf (i.e., a good temporary motif) as early as possible. Intuitively, a candidate subset with small $CS_{i,j}.LB$ tends to contain a candidate with small DFD value. Thus, we propose to process $CS_{i,j}$ in ascending order of $CS_{i,j}.LB$.

Putting It All Together: Algorithm 2 presents the pseudocode for *bounding-based trajectory motif* (BTM), which incorporates all above ideas for the trajectory motif discovery problem.

At Line 2, we first compute all lower bounds (and store them in matrices). Then, we insert each candidate subset $CS_{i,j}$ with its bound $CS_{i,j}.LB$ into a list (at Line 3), and sort that list (at Line 4). Next, we process the elements of the list in the sorted order. For each candidate subset, we examine its candidates via nested loops (at Lines

Algorithm 2 BTM ($\mathcal{S}$, ξ)

```
    Input: trajectory $\mathcal{S}$, length $n$, minimum motif length $\xi$
    Output: subtrajectory pair $bpair = (\mathcal{S}_{i,i_e}, \mathcal{S}_{j,j_e})$
    compute the ground distance matrix $d_G$
 1: $bsf \leftarrow +\infty$; $bpair \leftarrow \emptyset$; $j_{end} \leftarrow n$
 2: Compute { $LB_{cell}, rLB^{start}_{cross}, rLB^{end}_{cross}, rLB^{row}_{band}, rLB^{col}_{band}$ }
 3: Construct a list $\mathbf{A}$ with one element $\mathbf{a}$ per candidate subset
 4: Sort $\mathbf{A}$ in ascending order of $\mathbf{a}.LB$
 5: **for** each $\mathbf{a}$ in $\mathbf{A}$ with $bsf > \mathbf{a}.LB$ **do**
 6:     **for** $i_e \leftarrow \mathbf{a}.i + 1$ to $\mathbf{a}.j$ **do**
 7:         **for** $j_e \leftarrow \mathbf{a}.j + 1$ to $j_{end}$ **do**
 8:             $tmp \leftarrow \min(d_F[i_e\text{-}1][j_e\text{-}1], d_F[i_e][j_e\text{-}1], d_F[i_e\text{-}1][j_e])$
 9:             $d_F[i_e][j_e] \leftarrow \max(d_G(i_e, j_e), tmp)$
10:             **if** $i_e \geq \mathbf{a}.i + \xi - 1, j_e \geq \mathbf{a}.j + \xi - 1$ and $d_F[i_e][j_e] < bsf$ **then**
11:                 $bsf \leftarrow d_F[i_e][j_e]; bpair \leftarrow (\mathcal{S}_{i,i_e}, \mathcal{S}_{j,j_e})$
12:         **if** $bsf \leq rLB^{end}_{cross}(bpair.i_e, bpair.j_e)$ **then**
13:             $j_{end} \leftarrow bpair.j_e$                ▷ Pruning by $LB^{end}_{cross}(i_e, j_e)$ from Equation 8
14: return $bpair$
```

6-7), and compute the DFD of each candidate (at Line 8-9). Finally, we update bsf and the temporary motif pair (at Lines 10-11). Note that Lines 12-13 implement pruning by $LB^{end}_{cross}(i_e, j_e)$, as defined in Equation 8; this essentially performs pruning within the candidate subset currently considered, by disqualifying some of the candidate pairs it contains.

Analysis: The time complexity of Algorithm 2 is $O(n^4)$ in the worst case, which is attributed to the nested for-loops for variables $\mathbf{a}$ [that is, $O(n^2)$ iterations], i_e [that is, $O(n)$ iterations] and j_e [that is, $O(n)$ iterations] at Lines 5-7. The space complexity of Algorithm 2 is $O(n^2)$.

Algorithm 2 follows the best-first search paradigm with several effective lower bounds. As we show in the experiments, it outperforms Algorithm 1 by two orders of magnitude.

3.2 Grouping-based Solution

In this section, we enhance the scalability of our techniques for long trajectories. Our idea is to organize trajectory points into groups, then attempt pruning unpromising pairs of groups, before applying our solution from Section 3.1. To enable pruning, we design novel bounding functions for DFD on groups.

We outline our grouping-based computation framework in Figure 7. First, we divide a trajectory into groups with group size τ (where τ is a tunable parameter), and compute a ground distance bound for each group pair (Steps 1 and 2, in Section 3.2.1). Next, we apply $O(1)$-time lower bounds (Step 3, in Section 3.2.2) to prune group pairs, before using tighter bounds for pruning (Step 4, in Section 3.2.3). For the surviving group pairs, we repeat the above steps by halving the group size, until τ reaches 1. Finally, we compute the exact DFD of candidates in the surviving groups (Step 5).

By combining the advantages of all techniques in Section 3.1 and in the current one, our grouping-based computation framework outperforms the baseline solution by

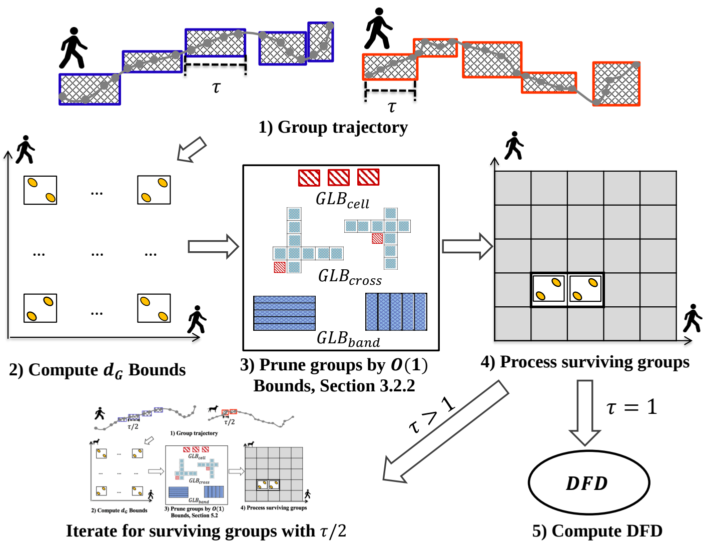


Fig. 7 Grouping-based computation framework

over 3 orders of magnitude. Importantly, all our techniques (lower bounds and grouping framework) produce exact answers (motifs).

3.2.1 Grouping Trajectory Points

We employ a group size parameter τ in order to partition a long trajectory into small groups. We proceed to define a group and the ground distances between groups.

Definition 4 (**τ-grouping**) Given the group size τ, we define the u-th group as the interval $g_u = [u\tau, (u+1)\tau - 1]$. For two groups g_u and g_v, we define the minimum and the maximum ground distance between them as:

$$d_G^{min}(g_u, g_v) = \min_{i \in g_u, j \in g_v} d_G(i, j) \tag{15}$$

$$d_G^{max}(g_u, g_v) = \max_{i \in g_u, j \in g_v} d_G(i, j) \tag{16}$$

By Definition 4, the ground distances between two groups satisfy the following property:

Corollary 1 *For every $i \in g_u,\ j \in g_v$, it holds that:*

$$d_G^{min}(g_u, g_v) \le d_G(i, j) \le d_G^{max}(g_u, g_v)$$

Example. Consider a trajectory $\mathcal{S}$ with $n = 12$ points. Given $\tau = 2$, we obtain six groups: $g_0, g_1, g_2, g_3, g_4, g_5$, as illustrated in Figure 8(a). For example, for groups $g_2 = [4, 5]$ and $g_5 = [10, 11]$, we compute the minimum ground distance as $d_G^{min}(g_2, g_5) = \min(d_G(4, 10), d_G(4, 11), d_G(5, 10), d_G(5, 11)) = 6$, and the maximum ground distance as $d_G^{max}(g_2, g_5) = \max(8, 9, 6, 7) = 9$. In Figure 8(b), we show

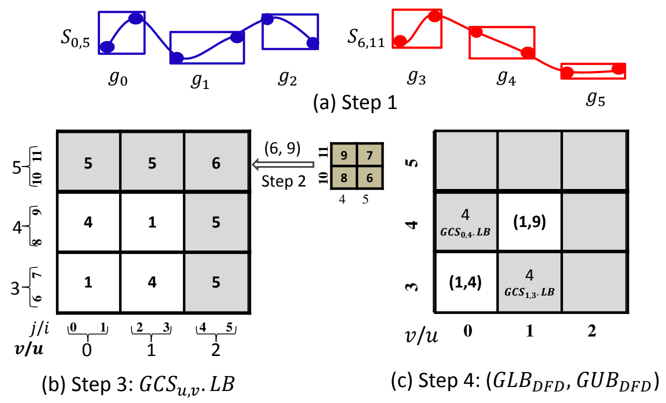


Fig. 8 Example of 2-grouped trajectory

the minimum and maximum ground distances for group pair g_2, g_5 (i.e., values (6, 9)) on top of the "Step 2" arrow.

We utilize Corollary 1 to devise lower bound functions in Sections 3.2.2 and 3.2.3.

3.2.2 Pattern-based Bounds for Groups

To enable pruning on unpromising pairs of groups, we adapt our lower bounds from Section 3.1 to groups. We denote the corresponding lower bounds with prefix $\mathcal{G}$, i.e., $\mathcal{G}LB_{cell}(u,v)$, $\mathcal{G}LB^{start}_{cross}(u,v)$, $\mathcal{G}LB^{end}_{cross}(u_e,v_e)$, $\mathcal{G}LB^{row}_{band}(u,v)$, and $\mathcal{G}LB^{col}_{band}(u,v)$. We first define the cell-based lower bound for groups, denoted by $\mathcal{G}LB_{cell}$, as:

$$\mathcal{G}LB_{cell}(u,v) = d_G^{min}(g_u, g_v) \tag{17}$$

In Figure 8(b), $\mathcal{G}LB_{cell}(2,5) = d_G^{min}(2,5) = 6$. For any $i \in u$ and $j \in v$, it holds that $\mathcal{G}LB_{cell}(u,v) \leq d_F(i, i_e, j, j_e)$.

Similarly, the cross-based lower bounds and band-based lower bounds can be expressed in terms of $\mathcal{G}LB_{cell}(u,v)$. The concept of relaxed bounds, introduced in Section 3.1.3, can be adapted directly to the above pattern-based bounds for groups. This allows us to obtain relaxed lower bounds for groups in $O(1)$ time. We omit them and refer the reader to our preliminary paper [31].

3.2.3 Bounding by DFD Computation

By exploiting the recurrence of DFD, we devise a tighter lower bound and a tighter upper bound for pairs of groups. While the lower bound is used to prune unpromising pairs of groups, the upper bound can be used to tighten bsf and thus improve the effectiveness of pruning.

Below we define a subtrajectory group, together with group-based DFD bounds.

Definition 5 (Group-based DFD) Let subtrajectory group $\mathcal{G}_{t,t_e}$ correspond to the interval $[t\tau, (t_e+1)\tau - 1]$, i.e., it covers group t to group t_e. Given two subtrajectory groups $\mathcal{G}_{u,u_e}$ and $\mathcal{G}_{v,v_e}$, we define the group-based DFD lower bound $d_{Fmin}(u, u_e, v, v_e)$ as:

$$d_{Fmin}(u, u_e, v, v_e) = \max \begin{cases} d_G^{min}(g_{u_e}, g_{v_e}) \\ \min \begin{cases} d_{Fmin}(u, u_e - 1, v, v_e) \\ d_{Fmin}(u, u_e - 1, v, v_e - 1) \\ d_{Fmin}(u, u_e, v, v_e - 1) \end{cases} \end{cases}$$

The definition of group-based DFD upper bound $d_{Fmax}(u, u_e, v, v_e)$ is similar to the above lower bound, i.e., it replaces $d_G^{min}()$ with $d_G^{max}()$ and $d_{Fmin}()$ with $d_{Fmax}()$, respectively. The following lemma proves the bounding property of $d_{Fmin}(u, u_e, v, v_e)$ and $d_{Fmax}(u, u_e, v, v_e)$.

Lemma 4 *Let $\mathcal{G}_{u,u_e}$ and $\mathcal{G}_{v,v_e}$ be two subtrajectory groups. If a pair of subtrajectories $\mathcal{S}_{i,i_e}, \mathcal{S}_{j,j_e}$ satisfies $i \in g_u, j \in g_v, i_e \in g_{u_e}$ and $j_e \in g_{v_e}$, it holds that:*

$$d_{Fmin}(u, u_e, v, v_e) \leq d_F(i, i_e, j, j_e) \leq d_{Fmax}(u, u_e, v, v_e)$$

Recall that our problem definition enforces a minimum motif length ξ. To comply with it, we define the following lower and upper bounds between two groups g_u and g_v:

$$\mathcal{G}LB_{DFD}(u, v) = \min_{u_e, v_e} \{d_{Fmin}(u, u_e, v, v_e) : u_e - u > \frac{\xi}{\tau} \wedge v_e - v > \frac{\xi}{\tau}\} \tag{18}$$

$$\mathcal{G}UB_{DFD}(u, v) = \min_{u_e, v_e} \{d_{Fmax}(u, u_e, v, v_e) : u_e - 1 - u > \frac{\xi}{\tau} \wedge v_e - 1 - v > \frac{\xi}{\tau}\} \tag{19}$$

The following lemma shows their correctness. It is derived by applying the $\min_{u_e, v_e}$ function to both sides of Lemma 4.

Lemma 5 $\forall\, i \in g_u, \forall\, j \in g_v$*, and* $i_e > i + \xi, j_e > j + \xi$ *it holds that:*

$$\mathcal{G}LB_{DFD}(u, v) \leq d_F(i, i_e, j, j_e) \leq \mathcal{G}UB_{DFD}(u, v)$$

$\mathcal{G}UB_{DFD}(u, v)$ allows us to tighten bsf, which in turn boosts the effectiveness of pruning. Both $\mathcal{G}LB_{DFD}(u, v)$ and $\mathcal{G}UB_{DFD}(u, v)$ can be computed in $O((\frac{n}{\tau})^2)$. We can reduce their computation cost by early termination. Specifically, if at some point during the computation of $\mathcal{G}LB_{DFD}(u, v)$, it holds that $\mathcal{G}LB_{cross}^{end}(u_e, v_e) \geq \mathcal{G}LB_{DFD}(u, v)$ with $u_e - u > \frac{\xi}{\tau} \wedge v_e - v > \frac{\xi}{\tau}$, we may safely terminate the computation because $\forall u_{e'} > u_e$ and $\forall v_{e'} > v_e$ it must be that $d_{Fmin}(u, u_{e'}, v, v_{e'}) > d_{Fmin}(u, u_e, v, v_e)$ (i.e., it cannot further tighten the bound). Similarly, early termination is possible in the calculation of $\mathcal{G}UB_{DFD}(u, v)$ too.

3.2.4 GTM Algorithm

Algorithm 3 presents the pseudocode for *grouping-based trajectory motif* (GTM), which implements the computation framework depicted in Figure 7. We first construct groups at Line 3, then we compute the pattern-based lower bounds of group pairs at Lines 4-5. Next, we insert each grouping-based candidate subset $\mathcal{G}CS_{u,v}$ with its bound $\mathcal{G}CS_{u,v}.LB = \max(\mathcal{G}LB_{cell}(u,v), r\mathcal{G}LB^{start}_{cross}(u,v), r\mathcal{G}LB^{row}_{band}(u,v), r\mathcal{G}LB^{col}_{band}(u,v))$ into a list. Then, we process the list in ascending order of $\mathcal{G}CS_{u,v}.LB$, and apply DFD bounds for pruning (Lines 10-11) or for tightening the bsf (Lines 12-13). After that, we halve the group size and repeat the above procedure on the set of surviving groups $\mathcal{S}_{survive}$ until the group size drops to 1. When this happens (i.e., $\tau = 1$), each element in $\mathcal{S}_{survive}$ is a candidate subset $CS_{i,j}$. We invoke Algorithm 2 on $\mathcal{S}_{survive}$ to obtain the final result.

Algorithm 3 GTM ($\mathcal{S}$, ξ, τ)

```
    Input: trajectory $\mathcal{S}$, length $n$, minimum motif length $\xi$, group size $\tau$
    Output: subtrajectory pair $bpair = (\mathcal{S}_{i,i_e}, \mathcal{S}_{j,j_e})$
    compute the ground distance matrix $d_G$
 1: $bsf \leftarrow +\infty$; $bpair \leftarrow \emptyset$
 2: while $\tau > 1$ do
 3:     Group trajectory $\mathcal{S}$ to $\mathcal{G}$                                    ▷ Section 3.2.1
 4:     Compute $\mathcal{G}LB_{cell}$, $r\mathcal{G}LB^{start}_{cross}$, $r\mathcal{G}LB^{end}_{cross}$
 5:         and $r\mathcal{G}LB^{row}_{band}$, $r\mathcal{G}LB^{col}_{band}$                    ▷ Section 3.2.2
 6:     Construct a list $\mathcal{G}\mathbf{A}$ of grouping-based candidate subsets
 7:     Sort $\mathcal{G}\mathbf{A}$ in ascending order of $\mathcal{G}\mathbf{a}.LB$
 8:     $\mathcal{S}_{survive} \leftarrow \emptyset$          ▷ set of surviving grouping-based candidate subsets
 9:     for each $\mathcal{G}\mathbf{a}$ in $\mathcal{G}\mathbf{A}$ with $bsf > \mathcal{G}\mathbf{a}.LB$ do           ▷ Section 3.2.3
10:         if $bsf > \mathcal{G}LB_{DFD}(\mathcal{G}\mathbf{a}.u, \mathcal{G}\mathbf{a}.v)$ then
11:             $\mathcal{S}_{survive} \leftarrow \mathcal{S}_{survive} \cup \mathcal{G}\mathbf{a}$
12:         if $bsf > \mathcal{G}UB_{DFD}(\mathcal{G}\mathbf{a}.u, \mathcal{G}\mathbf{a}.v)$ then
13:             $bsf \leftarrow \mathcal{G}UB_{DFD}(\mathcal{G}\mathbf{a}.u, \mathcal{G}\mathbf{a}.v)$
14:     $\tau \leftarrow \tau/2$, $\mathcal{S} \leftarrow \mathcal{S}_{survive}$
15: Invoke Lines 5-13 in Alg. 2 on $\mathcal{S}_{survive}$ to compute the result $bpair$
```

Example. We demonstrate the grouping-based computation framework in Figure 8, considering that the motif is sought between subtrajectories $\mathcal{S}_{0,5}$ and $\mathcal{S}_{6,11}$, and that the minimum motif length is $\xi = 2$. We first map $\mathcal{S}_{0,5}$ and $\mathcal{S}_{6,11}$ onto groups (with $\tau = 2$), e.g., $\mathcal{S}_{0,5}$ is mapped onto g_0, g_1 and g_2 (see Figure 8(a)). We next compute pattern-based bounds to prune group pairs. For example, referring to Figure 8(b), the grouping-based cell bound $\mathcal{G}LB_{cell}(2,5)$ is 6, as derived by Equation (17). If at the current point of execution $bsf = 5$, the respective pair of groups (i.e., g_2, g_5) can be pruned, because $\mathcal{G}LB_{cell}(2,5) = 6 \geq bsf = 5$. The same applies for all group pairs shown in gray in Figure 8(b). Next, we apply Equations (18) and (19) to compute the $\mathcal{G}LB_{DFD}, \mathcal{G}UB_{DFD}$ bounds for the remaining (i.e., un-pruned) group pairs. For instance, as illustrated in Figure 8(c), the $\mathcal{G}LB_{DFD}$ and $\mathcal{G}UB_{DFD}$ bounds for group pair $(0,3)$ are 1 and 4, respectively. This fact tightens bsf (from 5, previously) down to $\mathcal{G}UB_{DFD}(0,3) = 4$. The tightened bsf can be used to prune grouping-based cells; in our example, it prunes the grouping-based cells $(0,4)$ and $(1,3)$, since the grouping-based lower bound for both of them is 4, i.e., no smaller than the current bsf. This means that only subtrajectory pairs starting at cell (g_0, g_3) or at cell (g_1, g_4) could produce the motif. We half τ, which becomes 1. That splits each group to its resulting points, e.g., cell (g_0, g_3) is split to candidate subsets $CS_{0,6}, CS_{0,7}, CS_{1,6}, CS_{1,7}$, and a similar splitting is applied for cell (g_1, g_4). Since now $\tau = 1$, the eight produced candidate subsets are fed to Algorithm 2, comprising list $\mathbf{A}$ in its Line 5.

Lemma 6 *Algorithm 3 solves Problem 1 exactly, i.e., it returns the correct motif.*

Proof The first point that Algorithm 3 performs pruning is at Line 9, where it utilizes grouping-based lower bounds to eliminate candidate subsets with DFD larger than bsf. The correctness of this step is guaranteed by the validity of these bounds (specifically, $\mathcal{G}LB_{cell}(u,v)$, $r\mathcal{G}LB^{start}_{cross}(u,v)$, $r\mathcal{G}LB^{row}_{band}(u,v)$, $r\mathcal{G}LB^{col}_{band}(u,v)$) as described in Section 3.2.2. The next pruning point is at Lines 10-11, where candidate subsets are filtered according to the lower bound in Lemma 5, whose validity has been substantiated in Section 3.2.3. The last pruning point is at Lines 12-13 that indirectly help eliminate candidate subsets by decreasing (i.e., tightening) bsf, again according to Lemma 5, utilizing its upper bound this time. In other words, none of the three pruning points in Algorithm 3 may eliminate the actual motif, which is bound to be discovered by the rest of processing (Line 15) that draws directly from Algorithm 2.

Analysis: The computation cost of the while-loop (Lines 2–14) is $O(\sum_{i=1}^{log(\tau)} (\frac{c_i}{\tau})^4)$, where $c_1 = n$ and c_i is the number of surviving groups in iteration i. Line 16 takes $O(c\tau^2 n^2)$ time, where c is the number of surviving groups after the while-loop. In summary, the time complexity of Algorithm 3 is $O(\sum_{i=1}^{log(\tau)} (\frac{c_i}{\tau})^4 + c\tau^2 n^2)$. In the worst case, Algorithm 3 degenerates to Algorithm 2, with time complexity $O(n^4)$.

The space complexity of the algorithm is $O(n^2)$ as it employs two 2-dimensional matrices for precomputed ground distances (i.e., $d_G[\cdot][\cdot]$) and DFD values (i.e., $d_F[\cdot][\cdot]$). In addition, it takes $O((\frac{n}{\tau})^2)$ space for precomputed grouping-based lower bounds in $\mathcal{G}\mathbf{A}$ at Line 6.

4 Frequent Pattern Discovery

In this section, we study the frequent pattern discovery problem on a trajectory. We first define a pattern as follows.

Definition 6 (Pattern) A pattern is expressed as $(\mathcal{S}_{i,i_e}, \Gamma_i)$, where (i) $\mathcal{S}_{i,i_e}$ is a representative subtrajectory, and (ii) Γ_i is the set of subtrajectories that are similar to $\mathcal{S}_{i,i_e}$.

Given a subtrajectory length ξ, similarity threshold d_{thres}, and frequency threshold t_{thres}, we are set to compute all frequent patterns $(\mathcal{S}_{i,i_e}, \Gamma_i)$ such that: (i) $\mathcal{S}_{i,i_e}$ and subtrajectories in Γ_i have length ξ, (ii) the timestamps of these subtrajectories do not overlap, (iii) $\mathcal{S}_{i,i_e}$ is similar to each subtrajectory $\mathcal{S}_{j,j_e}$ in Γ_i, i.e., $d_F(i, i_e, j, j_e) \leq d_{thres}$, and (iv) the size of Γ_i is above a frequency threshold, i.e., $|\Gamma_i| \geq t_{thres}$. Formally:

Problem 2 (Frequent Pattern Discovery) Given a trajectory $\mathcal{S}$ and two positive thresholds d_{thres} and t_{thres}, find each frequent pattern $(\mathcal{S}_{i,i_e}, \Gamma_i)$ such that (1) $\forall\ \mathcal{S}_{j,j_e} \in \Gamma_i, d_F(i, i_e, j, j_e) \leq d_{thres}$ and (2) $|\Gamma_i| \geq t_{thres}$. We require that these subtrajectories have length ξ.

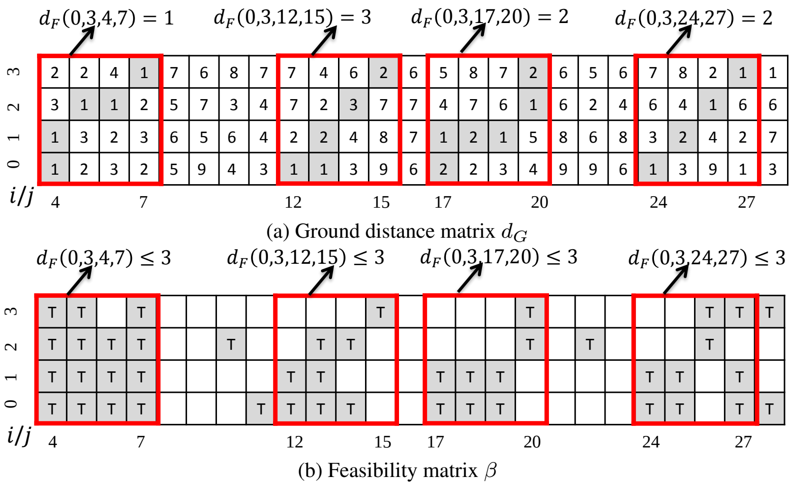


Fig. 9 Frequent pattern discovery example

Example. Figure 9(a) illustrates the ground distance matrix d_G for a trajectory $\mathcal{S}$. Suppose that the parameters of the above problem are: subtrajectory length $\xi = 3$, distance threshold $d_{thres} = 3$, and frequency threshold $t_{thres} = 3$. In this figure, the red rectangles indicate that $\mathcal{S}_{0,3}$ is similar to four subtrajectories ($\mathcal{S}_{4,7}, \mathcal{S}_{12,15}, \mathcal{S}_{17,20}, \mathcal{S}_{24,27}$), i.e., the DFD distances between them are below d_{thres}. Thus, the result contains the frequent pattern $(\mathcal{S}_{0,3}, \Gamma_0)$, where $\Gamma_0 = \{\mathcal{S}_{4,7}, \mathcal{S}_{12,15}, \mathcal{S}_{17,20}, \mathcal{S}_{24,27}\}$.

We note that Buchin et al. [10] studied a subtrajectory clustering problem with some similarity to Problem 2. Specifically, given a trajectory $\mathcal{S}$, an integer $t_{thres} > 0$ and two positive real values ξ and d_{thres}, a subtrajectory cluster $C(t_{thres}, \xi, d_{thres})$ is defined as a set of t_{thres} subtrajectories in T, where (1) **every pair of subtrajectories** in $C(t_{thres}, \xi, d_{thres})$ is within distance d_{thres}, and (2) **at least one of the subtrajectories** in $C(t_{thres}, \xi, d_{thres})$ has length ξ.

The problem in [10] differs from Problem 2 in two aspects: (i) it requires every pair of subtrajectories to be within distance d_{thres} from each other, whereas the distance reference in Problem 2 is a single pivot subtrajectory $\mathcal{S}_{i,i_e}$; (ii) it allows for subtrajectories of any length, except for only one of length ξ, whereas Problem 2 requires length ξ for all subtrajectories. Due to the NP-hardness of the problem in [10], they proposed an approximate solution. In contrast, we devise exact solutions to Problem 2, as we elaborate shortly. Even if an adaptation of the algorithms in [10] to Problem 2 were possible, comparing that hypothetical, approximate approach to our exact solutions would not be meaningful. In Table 3, we summarize the differences of the subtrajectory clustering problem in [10] from our frequent pattern discovery problem. Regarding the complexity row in the table, the following naïve algorithm (is exact and) already has a polynomial complexity.

A brute force solution to Problem 2 is to consider each subtrajectory $\mathcal{S}_{i,i_e}$ as a representative. Then, for each $\mathcal{S}_{i,i_e}$, compute its DFD distance to every (non-overlapping) subtrajectory $\mathcal{S}_{j,j_e}$, and count the number of subtrajectories with DFD distance below

Table 3 Differences between the subtrajectory clustering problem in [10] and our Problem 2

Problem	Subtrajectory clustering in [10]	Frequent pattern discovery
Output	$C(t_{thres}, \xi, d_{thres})$	all $(\mathcal{S}_{i,i_e}, \mathit{\Gamma}_i)$ pairs
Subtrajectory length	$\exists \mathcal{S}_{j,j_e} \in C(t_{thres}, \xi, d_{thres})$, $\lvert\mathcal{S}_{j,j_e}\rvert \geq \xi$	$\forall \mathcal{S}_{j,j_e} \in \mathit{\Gamma}_i$, $\lvert\mathcal{S}_{j,j_e}\rvert = \xi$
Subtrajectory distance	$\forall \mathcal{S}_{i,i_e}, \mathcal{S}_{j,j_e} \in C(t_{thres}, \xi, d_{thres})$, $d_F(i, i_e, j, j_e) \leq d_{thres}$	$\forall\ \mathcal{S}_{j,j_e} \in \mathit{\Gamma}_i$, $d_F(i, i_e, j, j_e) \leq d_{thres}$
Complexity	NP-hard	polynomial
Solution	approximate	exact

d_{thres}. When the count is at least t_{thres}, we insert the corresponding frequent pattern into the result set. We describe this brute force approach in Algorithm 4.

Algorithm 4 BruteFP ($\mathcal{S}$, ξ, d_{thres}, t_{thres})

```
    Input: trajectory $\mathcal{S}$, length $n$, minimum motif length $\xi$
           distance threshold $d_{thres}$, frequency threshold $t_{thres}$
    Output: frequent pattern set $\mathbf{R}$
    compute the ground distance matrix $d_G$
 1: $\mathbf{results} \leftarrow \emptyset$
 2: for $i \leftarrow 0$ to $n - \xi + 1$ do
 3:     $\mathit{\Gamma}_i \leftarrow \emptyset$
 4:     for $j \leftarrow 0$ to $n - \xi + 1$ and $j \notin [i, i + \xi]$ do
 5:         $d_F[i][j] \leftarrow d_G(i, j)$                                   ▷ initialization
 6:         for $t \leftarrow i + 1$ to $n$ do
 7:             $d_F[i][t] \leftarrow \max(d_G(i, t), d_F[i][t\text{-}1])$
 8:             $d_F[t][j] \leftarrow \max(d_G(t, j), d_F[t\text{-}1][j])$
 9:         for $i_e \leftarrow i + 1$ to $i + \xi$ do
10:             for $j_e \leftarrow j + 1$ to $j + \xi$ do                    ▷ share DFD computation
11:                 $tmp \leftarrow \min(d_F[i_e\text{-}1][j_e\text{-}1], d_F[i_e][j_e\text{-}1], d_F[i_e\text{-}1][j_e])$
12:                 $d_F[i_e][j_e] \leftarrow \max(d_G(i_e, j_e), tmp)$
13:             if $d_F[i_e][j_e] \leq d_{thres}$ then
14:                 $\mathit{\Gamma}_i \leftarrow \mathit{\Gamma}_i \cup \mathcal{S}_{j,j_e}$
15:                 $j \leftarrow j_e + 1$
16:     if $|\mathit{\Gamma}_i| \geq t_{thres}$ then
17:         $\mathbf{results} \leftarrow \mathbf{results} \cup (\mathcal{S}_{i,i+\xi}, \mathit{\Gamma}_i)$
18: return $\mathbf{results}$
```

Analysis: Algorithm 4 takes $O(n^3 + n^2 \cdot \xi^2)$ time, which is attributed to initiation (i.e., $O(n^3)$), the nested for-loops for variables i, j (at Lines 2 and 4) and variables i_e (at Line 9) and j_e (at Line 10) (i.e., $O(n^2 \cdot \xi^2)$). The space complexity of the algorithm is $O(n^2)$ (due to the matrices $d_F[\cdot][\cdot]$ and $d_G[\cdot][\cdot]$).

4.1 Advanced Solution

Problem 2 relies on the *distance threshold test*, i.e., testing whether $d_F(i, i_e, j, j_e) \leq d_{thres}$, incurring numerous accesses to the ground distance matrix d_G. To accelerate the test, we replace the numeric matrix d_G by a Boolean matrix (in Section 4.1.1). Besides saving memory, this idea enables us to utilize fast operations on Boolean values. Specifically, Boolean values reduce memory consumption by 4 times, and save CPU cycles by almost 3 times compared to floating-point values [1]. Then, we adapt the lower bound functions in the previous section to support the distance threshold test (in Section 4.1.2), and discuss their optimized implementation (in Section 4.1.3). Finally, we present the algorithm for frequent pattern discovery (in Section 4.1.4) and propose a space-efficient version (in Section 4.2).

4.1.1 Distance Matrix Reduction

The following observation is derived by the properties of the Free Space Diagram [9,3]:

Observation 5 *We have $d_F(i, i_e, j, j_e) \leq d_{thres}$ if and only if there exists a path from (i, j) to (i_e, j_e) in the d_G matrix such that the maximum ground distance along the path is less than or equal to d_{thres}.*

This observation suggests that if a cell (i, j) satisfies $d_G(i, j) > d_{thres}$, then any path via cell (i, j) must have DFD larger than d_{thres} (i.e., infeasible path). For example, $d_G(0, 8) = 5 \geq d_{thres}$ means that the DFD of any path passing via cell $(0, 8)$ is larger than d_{thres}.

Inspired by the Free Space Diagram [9,3] and utilizing Observation 5, we convert the ground distance matrix d_G into a Boolean *feasibility matrix* β as follows:

$$\beta(i, j) = \begin{cases} \mathsf{False} & \text{if} \quad d_G(i, j) > d_{thres} \\ \mathsf{True} & \text{otherwise} \end{cases}$$

For the distance matrix d_G in Figure 9(a), we show its Boolean feasibility matrix β in Figure 9(b). For example, since $d_G(0, 4) = 1 \leq d_{thres} = 3$, the cell in β is labeled as T; otherwise, the cell is labeled as F.

This feasibility matrix β saves memory and utilizes fast operations on Boolean values. We note that the distance threshold test $d_F(i, i_e, j, j_e) \leq d_{thres}$ is equivalent to finding a path of T cells from (i, j) to (i_e, j_e) in β. In the subsequent sections, we devise several pruning techniques based on β.

4.1.2 Pattern-based Pruning Rules

In this section, we adapt the lower bounds from Section 3.1.2 to accelerate the distance threshold test.

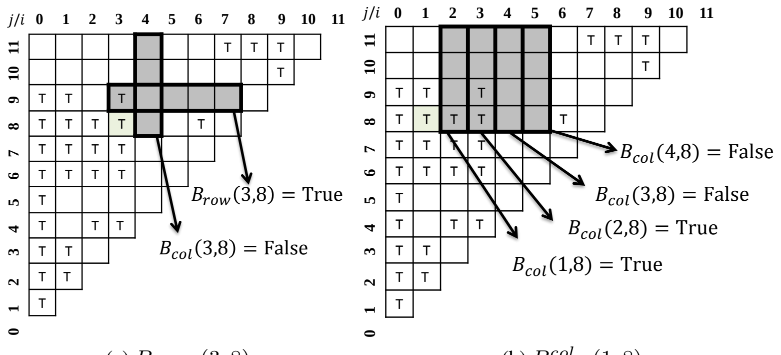
Cell-based Pruning Rule: By using Observation 2 on the feasibility matrix β, we can prune a candidate set $CS_{i,j}$ if $\beta(i, j) = \mathsf{False}$. This pruning rule takes $O(1)$ time.

Example. In Figure 9(b), we have $\beta(0, 8) = \mathsf{False}$. Thus, the candidate set $CS_{0,8}$ cannot produce any result. Recall that the candidate $CS_{0,8}$ is a set of pairs of trajectories with start cell $(0, 8)$ (see Definition 3).

Cross-based Pruning Rule: By applying Observation 3, we can define two Boolean indicators for a candidate set $CS_{i,j}$:

$$B_{row}(i, j) = \vee_{i' \in [i, i+\xi-1]} \beta(i', j + 1)$$
$$B_{col}(i, j) = \vee_{j' \in [j, j+\xi-1]} \beta(i + 1, j')$$

[1] Intel architectures optimization manual: `http://intel.ly/2lgN4rc`



(a) $B_{cross}(3,8)$ (b) $B^{col}_{band}(1,8)$

Fig. 10 Example of cross/band-based pruning rule

For two subtrajectories $\mathcal{S}_{i,i_e}$ and $\mathcal{S}_{j,j_e}$, if there exists a feasible path from (i,j) to (i_e,j_e), it must hold that $B_{row}(i,j) = \mathsf{True}$ and $B_{col}(i,j) = \mathsf{True}$. Thus, we combine them into a single Boolean indicator:

$$B_{cross}(i,j) = B_{row}(i,j) \wedge B_{col}(i,j)$$

With this indicator, we can prune a candidate set $CS_{i,j}$ if $B_{cross}(i,j) = \mathsf{False}$. This pruning rule takes $O(\xi)$ time.

Example. We illustrate the cross based pruning rule by the example in Figure 10(a). Consider the candidate set $CS_{3,8}$, for which $B_{cross}(3,8) = B_{row}(3,8) \wedge B_{col}(3,8) = \mathsf{True} \wedge \mathsf{False} = \mathsf{False}$. Thus, the candidate set $CS_{3,8}$ will be pruned.

Band-based Pruning Rule: According to Observation 4, we define the following band-based indicators:

$$B^{row}_{band}(i,j) = \wedge_{j' \in [j, j+\xi-1]} B_{row}(i,j')$$
$$B^{col}_{band}(i,j) = \wedge_{i' \in [i, i+\xi-1]} B_{col}(i',j)$$

For two subtrajectories $\mathcal{S}_{i,i_e}$ and $\mathcal{S}_{j,j_e}$, if there exists a feasible path from (i,j) to (i_e,j_e), it must hold that $B^{row}_{band}(i,j) = \mathsf{True}$ and $B^{col}_{band}(i,j) = \mathsf{True}$. This idea enables us to prune a candidate set $CS_{i,j}$ if $B^{row}_{band}(i,j) = \mathsf{False}$ or $B^{col}_{band}(i,j) = \mathsf{False}$. It takes $O(\xi^2)$ time to execute this rule.

Example. Consider the example in Figure 10(b) with $\xi = 4$. For the candidate set $CS_{1,8}$, we have: $B^{col}_{band}(1,8) = B_{col}(1,8) \wedge B_{col}(2,8) \wedge B_{col}(3,8) \wedge B_{col}(4,8) = \mathsf{False}$. Thus, the candidate set $CS_{1,8}$ can be safely pruned.

4.1.3 Efficient Implementation of Pruning Rules

The cross-based and the band-based pruning rules require $O(\xi)$ time and $O(\xi^2)$ time, respectively. We propose a technique to support these operations in $O(1)$ amortized time.

Consider $B_{row}(i,j)$ first. We observe that the computation of $B_{row}(i,j)$ is equivalent to a counting problem. Let $\mathsf{Count}^{\beta}_{row}(i,j)$ denote the number of terms in

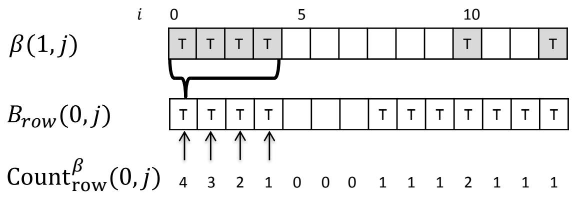


Fig. 11 Optimization for B_{row} Computation

$B_{row}(i,j)$ that hold the True value in matrix β. Formally, we have: $\mathsf{Count}^{\beta}_{row}(i,j) = \sum_{i'=i}^{i+\xi-1} \mathsf{Int}(\beta(i',j+1))$, where the function $\mathsf{Int}()$ converts a Boolean value to an integer as follows:

$$\mathsf{Int}(x) = \begin{cases} 1 & \text{if } \quad x = \mathsf{True} \\ 0 & \text{otherwise} \end{cases}$$

Observation 6 *$B_{row}(i,j)$ is equivalent to Boolean value ($\mathsf{Count}^{\beta}_{row}(i,j) > 0$).*

It remains to discuss how to compute $\mathsf{Count}^{\beta}_{row}(i,j)$ in $O(1)$ time. Observe that:

$$\mathsf{Count}^{\beta}_{row}(i+1,j) = \mathsf{Count}^{\beta}_{row}(i,j) - \mathsf{Int}(\beta(i,j)) + \mathsf{Int}(\beta(i+\xi,j)) \qquad (20)$$

Generally, $\mathsf{Count}^{\beta}_{row}(i,j)$ will be computed before $\mathsf{Count}^{\beta}_{row}(i+1,j)$. Then Equation 20 can be applied to obtain $\mathsf{Count}^{\beta}_{row}(i+1,j)$ in $O(1)$ time. We illustrate this idea in Figure 11. For example, $\mathsf{Count}^{\beta}_{row}(0,1) = \mathsf{Count}^{\beta}_{row}(0,0) - \mathsf{Int}(\beta(0,0)) + \mathsf{Int}(\beta(0+4,0)) = 4-1+0 = 3$. We have computed $\mathsf{Count}^{\beta}_{row}(0,0)$, thus $\mathsf{Count}^{\beta}_{row}(0,1)$ can be updated in $O(1)$ time.

This idea can also be used to compute $B_{col}(i,j)$ in $O(1)$ time, except we apply the following equation to update each count.

$$\mathsf{Count}^{\beta}_{col}(i,j+1) = \mathsf{Count}^{\beta}_{col}(i,j) - \mathsf{Int}(\beta(i,j)) + \mathsf{Int}(\beta(i,j+\xi)) \qquad (21)$$

A similar idea can be used for the band-based pruning rule. We use $\mathsf{Count}^{row}_{band}(i,j)$ and $\mathsf{Count}^{col}_{band}(i,j)$ to represent the number of True terms in $B^{row}_{band}(i,j)$ and $B^{col}_{band}(i,j)$, respectively.

Observation 7 *$B^{row}_{band}(i,j)$ is equivalent to Boolean value ($\mathsf{Count}^{row}_{band}(i,j) = \xi$).*

By using the following equations, we can compute $\mathsf{Count}^{row}_{band}(i,j)$ and $\mathsf{Count}^{col}_{band}(i,j)$ in $O(1)$ time.

$$\mathsf{Count}^{row}_{band}(i,j+1) = \mathsf{Count}^{row}_{band}(i,j) - \mathsf{Int}(B_{row}(i,j)) + \mathsf{Int}(B_{row}(i,j+\xi))$$
$$\mathsf{Count}^{col}_{band}(i+1,j) = \mathsf{Count}^{col}_{band}(i,j) - \mathsf{Int}(B_{col}(i,j)) + \mathsf{Int}(B_{col}(i+\xi,j))$$

We summarize time complexities of all pruning rules in Table 4.

Regarding grouping-based techniques, we mention that our internal testing showed that they are not useful in the frequent pattern problem. The main reason is that the frequent pattern discovery has been reduced to a feasibility testing problem, for which the group-based techniques are not effective. Specifically, each group in the motif discovery problem has a lower and upper ground distance bound, which can be used in our proposed cell-, cross-, and band-based bounds. However, for frequent pattern discovery, each group only has True or False value. The group cannot be pruned if its value is True.

Table 4 Summary of pruning rules

Pruning rule	Cost	Optimized cost
$\beta(i,j)$	$O(1)$	$O(1)$
$B_{row}(i,j)$	$O(\xi)$	$O(1)$
$B_{col}(i,j)$	$O(\xi)$	$O(1)$
$B^{row}_{band}(i,j)$	$O(\xi^2)$	$O(1)$
$B^{col}_{band}(i,j)$	$O(\xi^2)$	$O(1)$

4.1.4 Optimized Solution

Algorithm 5 presents the pseudocode for the trajectory frequent pattern discovery problem with *bounding-based pruning rules* (BFP).

Algorithm 5 BFP ($\mathcal{S}$, ξ, d_{thres}, t_{thres})

```
    Input: trajectory $\mathcal{S}$, length $n$, subtrajectory length $\xi$
        DFD distance threshold $d_{thres}$, frequency threshold $t_{thres}$
    Output: frequent pattern set **results**
    compute the feasibility matrix $\beta$ by $d_G$ matrix
1:  **results** $\leftarrow \emptyset$
2:  **for** $i \leftarrow 0$ to $n - \xi + 1$ **do**
3:      **results**($\mathcal{S}_{i,i+\xi}$) $\leftarrow \emptyset$
4:      **for** $j \leftarrow 0$ to $n - \xi + 1$ and $j \notin [i, i+\xi]$ **do**
5:          **if** $\beta[i][j] \wedge B_{cross}(i,j) \wedge B_{band}(i,j) =$ True **then**
6:              **for** $t \leftarrow i+1$ to $i+\xi$ **do**
7:                  $d_F[i][t] \leftarrow \beta(i,t) \wedge d_F[i][t\text{-}1]$
8:                  $d_F[t][j] \leftarrow d_G(t,j) \wedge d_F[t\text{-}1][j]$
9:              **for** $i_e \leftarrow i+1$ to $i+\xi$ **do**
10:                 **for** $j_e \leftarrow j+1$ to $j+\xi$ **do**
11:                     **if** $\beta(i_e, j_e) =$ True **then**
12:                         $d_F[i_e][j_e] \leftarrow d_F[i_e\text{-}1][j_e\text{-}1] \vee d_F[i_e][j_e\text{-}1] \vee d_F[i_e\text{-}1][j_e]$
13:                     **else**
14:                         $d_F[i_e][j_e] \leftarrow$ False
15:             **if** $d_F[i_e][j_e] =$ True **then**
16:                 **results**($\mathcal{S}_{i,i+\xi}$) $\leftarrow$ **results**($\mathcal{S}_{i,i+\xi}$) $\cup \mathcal{S}_{j,j_e}$
17:                 $j \leftarrow j_e + 1$;
18:     **if** $|$**results**($\mathcal{S}_{i,i+\xi}$)$| \geq t$ **then**
19:         **results** $\leftarrow$ **results**($\mathcal{S}_{i,i+\xi}$)
20: return **results**
```

It incorporates all pruning ideas in Line 5. Both $B_{cross}(i,j)$ and $B_{band}(i,j)$ can be obtained in $O(1)$ time as discussed in Section 4.1.3. To support this, we store $B_{row}(i,j)$, $B_{col}(i,j)$, $B^{row}_{band}(i,j)$ and $B^{col}_{band}(i,j)$ in temporary arrays, so that we can update them in $O(1)$ time by using the equations in Section 4.1.3.

Analysis: The time complexity of Algorithm 5 is $O(n^2 \cdot \xi^2)$ in the worst case, which is attributed to the nested for-loops for variables i [that is, $O(n)$ iterations] , j [that is, $O(n)$ iterations], i_e and j_e [these are, $O(\xi)$ iterations]. The space complexity of Algorithm 5 is $O(n^2)$. It performs faster than Algorithm 4 by a constant factor, because we have replaced expensive mathematic functions by Boolean operators.

4.2 Space-efficient Variant: BFP^*

We present a space-efficient variant of BFP, called BFP^*. It incorporates two ideas: (i) it employs a rolling array for β (i.e., $O(n)$ space cost), and (ii) it computes d_G, β, B_{cross}, B_{band}, and d_F on-the-fly. Idea (i) is feasible because, in Line 12 of Algorithm 5, we examine at most two rows of β at the same time. Idea (ii) eliminates the need for precomputed d_G and β. Thus, the space complexity of BFP^* is $O(n \cdot \xi)$.

The time complexity of the space-efficient solution BFP^* is $O(n^2 \cdot \xi^2)$, the same as BFP. Moreover, our space enhancements do not affect the pruning ability of vanilla BFP. Thus, BFP^* performs comparably to or, most often, slightly better than BFP, as we show in the experiments (see Figure 21 in Section 6.3).

5 Extension to Trajectory Database

In this section, we extend our solutions for Problems 1 and 2 to multiple trajectories (i.e., a trajectory database). We use the notation $d_F(\mathcal{S}_{i,i_e}, \mathcal{T}_{j,j_e})$ to represent the DFD distance between two subtrajectories from different trajectories (i.e., $\mathcal{S}, \mathcal{T}$).

Motif Discovery in Trajectory Database: A variant of Problem 1 is to discover a motif among multiple trajectories. As an application example in weather prediction and typhoon analysis, Figure 12 illustrates the trajectories of five typhoons in Hong Kong, in July 2016. The motif corresponds to the pair of subtrajectories (shown in red) from the typhoons called Nepartak and Chaba. Formally, the problem is defined as follows:

Problem 3 (Motif Discovery in Trajectory Database) Given a trajectory dataset $\mathcal{D}$, find the pair of subtrajectories $\mathcal{S}_{i,i_e}$ and $\mathcal{T}_{j,j_e}$, where $\mathcal{S} \in \mathcal{D}$ and $\mathcal{T} \in \mathcal{D}$, with the smallest DFD distance $d_F(\mathcal{S}_{i,i_e}, \mathcal{T}_{j,j_e})$ among all possible pairs of subtrajectories. We require that $\mathcal{S}_{i,i_e}$ and $\mathcal{T}_{j,j_e}$ have length at least ξ.

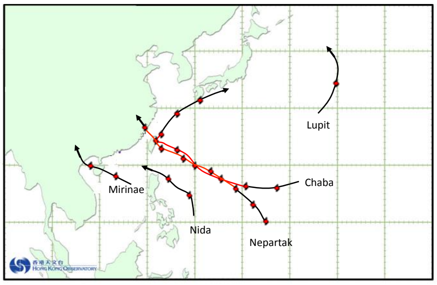


Fig. 12 Example of motif discovery among multiple trajectories (adapted from http://www.hko.gov.hk/informtc/tcMain.htm)

An intuitive solution is to concatenate all trajectories in trajectory dataset $\mathcal{D}$ into a single trajectory $\mathcal{S}_{all}$. For example, for the trajectories in Figure 12, we obtain the single trajectory $\mathcal{S}_{all}$ as shown in Figure 13. Then, we can apply the algorithms in Section 3 to compute the motif on this single trajectory $\mathcal{S}_{all}$.

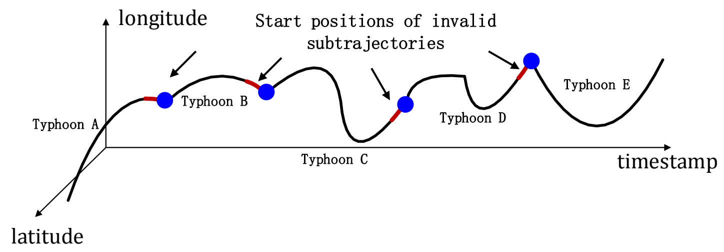


Fig. 13 Example of multiple trajectories concatenation

To ensure the correctness of the above solution, we must exclude invalid subtrajectories from consideration. We call a subtrajectory (of $\mathcal{S}_{all}$) to be *invalid* if its start position belongs to one of the last $\xi - 1$ points in an original trajectory. We illustrate some invalid subtrajectories in red color in Figure 13. In addition, for each $\mathcal{S}_{i,i_e}$, we also exclude any subtrajectories which belong to the same original trajectory.

Frequent Pattern Discovery in Trajectory Database: Similarly, a variant of Problem 2 is to discover frequent patterns among multiple trajectories. Given distance threshold d_{thres} and frequency threshold t_{thres}, the frequent pattern discovery problem in a trajectory database $\mathcal{D}$, is to find all the frequent patterns $(\mathcal{S}_{i,i_e}, \Gamma_i)$ in $\mathcal{D}$, where $\mathcal{S}_{i,i_e}$ belongs to one of the trajectories in $\mathcal{D}$, and each subtrajectory $\mathcal{T}_{j,j_e} \in \Gamma_i$ is similar to $\mathcal{S}_{i,i_e}$ (i.e., $d_F(\mathcal{S}_{i,i_e}, \mathcal{T}_{j,j_e}) \leq d_{thres}$), where $\mathcal{T} \neq \mathcal{S}$, and the number of subtrajectories in Γ_i is not smaller than t_{thres}. By applying the above concatenation idea, we can reduce the frequent pattern discovery in trajectory database problem to Problem 2. Then, we can apply the algorithms in Section 4 to compute the result. Again, we need to filter out invalid trajectories.

6 Empirical Evaluation

In this section, we evaluate the performance of our solutions on real data sets. First, we introduce the experimental setting in Section 6.1. Then, Sections 6.2 and 6.3 evaluate the performance of different methods on the motif discovery problem and the frequent pattern discovery problem, respectively.

6.1 Experimental Setup

Dataset: We used four real trajectory datasets from moving people, vehicles and animals in our experimental study. We note that these datasets have different characteristics (such as sampling frequency and data distribution) thus helping us verify the generality of our findings. We used all datasets in [31], i.e., **GeoLife**, **Truck**, and **Wild-Baboon**. For each dataset, we concatenate raw trajectories in order to build longer trajectories. As the **Truck** dataset is not long enough for frequent pattern discovery problem, we use the **Pigeons**[2] dataset instead, which was collected from homing pigeons [18] in Pisa, Italy. It contains 24 trajectories of pigeons with GPS that recorded a location every second from 2nd July to 21st July, 2010. Table 5 lists the used datasets in each problem.

[2] `http://chorochronos.datastories.org/`

Table 5 Datasets in each problem

Problem	Datasets
Motif discovery	**GeoLife**, **Truck**, **Wild-Baboon**
Frequent pattern discovery	**GeoLife**, **Pigeons**, **Wild-Baboon**

Methods and Implementation: We used C++ for the implementation and conducted all experiments (with single thread) on a machine with an Intel Core i7-4770 3.40GHz processor. In our experiments, we report the average measurements over 10 different trajectories of the same length. The response times reported include the precomputation time of distances and lower bounds.

Motif Discovery Problem: We compare the following methods: (i) the baseline solution BruteDP (see Algorithm 1), (ii) the bounding-based solution BTM (see Algorithm 2), (iii) the grouping-based solution GTM (see Algorithm 3). By default, we fix the motif length threshold ξ to 100, and the trajectory length n to 5000.

Frequent Pattern Discovery Problem: We compare the following methods: (i) the baseline solution BruteFP (see Algorithm 4), (ii) the bounding-based solution BFP (see Algorithm 5), (iii) the space-efficient solution BFP* (see Section 4.2), and (iv) CUP, adapted from the 1st place solution in SIGSPATIAL CUP 2017 [5]. CUP was proposed to work with Fréchet distance, consequently, its adaptation to DFD is straightforward. However, to fairly interpret the comparison with CUP, we note that, being proposed for range queries, its adaptation to frequent pattern discovery requires performing a range query for each possible subtrajectory of length ξ to determine whether at least t_{thres} subtrajectories are within distance d_{thres} from it. Unlike our methods, it is non-trivial (if possible at all) to enable those range queries to reuse computations in a fashion similar to the computation sharing that is organically achieved within our framework[3].

Table 6 Experiment parameters, tested values, and defaults

Parameter	Tested and default values
Trajectory length (n)	5K, **10K**, 50K, 100K, 500K
Frequent pattern length (ξ)	100, **200**, 300, 400, 500
Frequency threshold (t_{thres})	2, **4**, 6, 8, 10
Distance threshold (d_{thres}) Unit: (meters)	**GeoLife:** 50, **100**, 150, 200, 250 **Pigeons:** 1, **2**, 3, 4, 5 **Wild-Baboon:** 0.5, **1**, 1.5, 2, 2.5

Table 6 provides the value ranges for each tested parameter for frequent pattern discovery problem, with their default values indicated in bold. In each experiment, we vary one parameter while keeping the remaining ones to their default values.

[3] We note that recently the authors of [5] have produced a more efficient approach [8], still considering the range query.

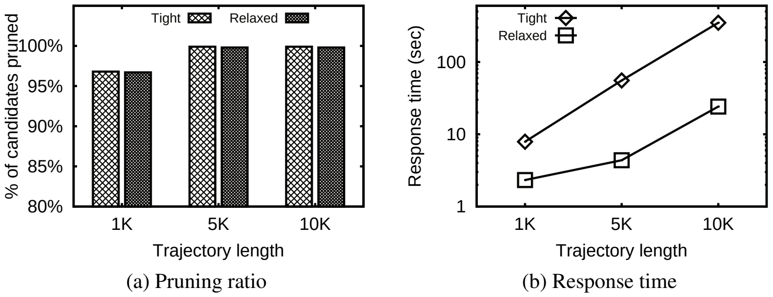


Fig. 14 BTM, effect of trajectory length n

6.2 Results on Motif Discovery

We first study the effectiveness of our pruning techniques (e.g., lower bounds and grouping) in Section 6.2.1. We then test the performance of methods with respect to different parameters in Section 6.2.2.

6.2.1 Pruning Effectiveness

We first assess the effectiveness of our pruning techniques, particularly of our lower bounds and grouping. We present results on the GeoLife dataset. Results on Truck and Wild-Baboon are similar and are omitted for brevity.

Effectiveness of Relaxed Bounds: We first compare two variants of BTM that use: (i) only the *tight lower bounds* from Section 3.1.2, and (ii) only the *relaxed lower bounds* from Section 3.1.3.

In Figure 14, we compare the tight with the relaxed bounds by varying the trajectory length n, with ξ fixed to 100. The pruning percentage in Figure 14(a) corresponds to the ratio of candidate pairs successfully pruned to the total number of candidate pairs. Note that because the percentage is high, and in order to show enough detail, we truncated the y-axis of the plot to start from 80%. In Figure 14(b), we show the overall response time to compute the motif. We observe that the relaxed bounds are only slightly weaker in pruning power, but they are orders of magnitude faster computation-wise.

In Figure 15, we investigate the effectiveness and performance of tight and relaxed bounds as a function of the minimum motif length ξ, with n fixed to 5000. Again, although the tight bounds have slightly higher pruning ratio (in Figure 15(a)), the relaxed bounds render motif computation 10 times faster (in Figure 15(b)). Since the relaxed bounds perform much better, we adopt them in our framework (instead of the tight ones) and use them in the subsequent experiments.

Effectiveness of Lower Bounds: In the next experiment, we compare the pruning effectiveness of the different lower bound functions ($LB_{cell}, rLB_{cross}, rLB_{band}$) using BTM. Each bar in Figure 16 corresponds to the total number of candidate pairs, broken down into the fraction pruned by each of the 3 types of bounds, and the fraction of the surviving pairs that required exact DFD computation (labeled as DFD in the bar charts). In Figures 16(a) and 16(b), we vary the trajectory length n and the minimum

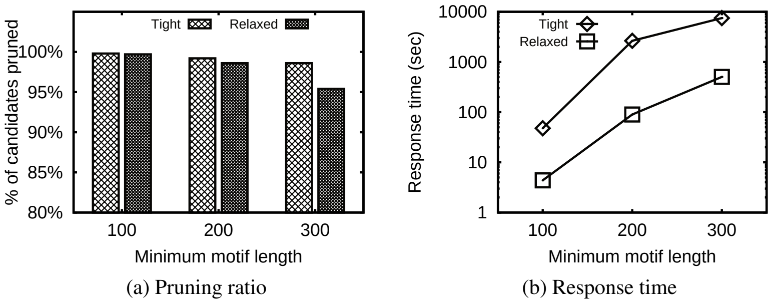


Fig. 15 BTM, effect of minimum motif length ξ

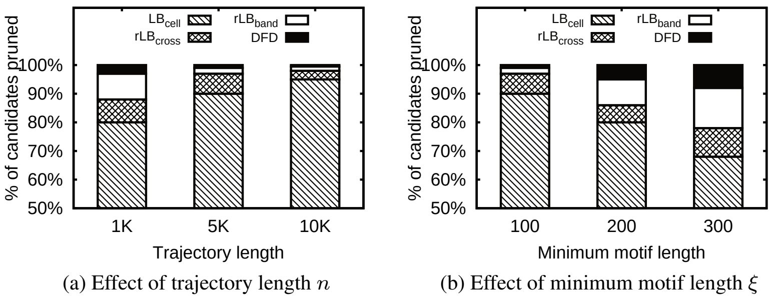


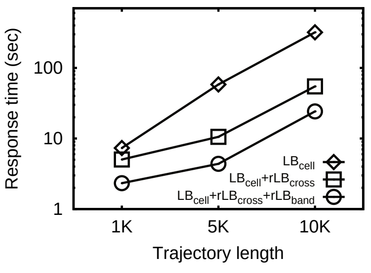
Fig. 16 BTM, pruning ratio breakdown

motif length ξ, respectively. The bars are truncated to start at ratio 50% to retain detail, because the percentage of LB_{cell} hugely dominates the rest.

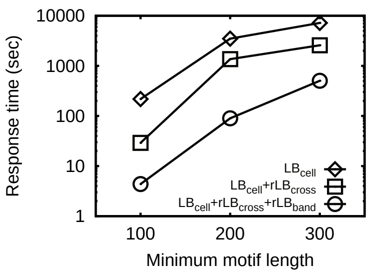
Over 92% of the candidates can be collectively pruned by our lower bounds. An interesting observation is that the bounds complement each other. For instance, when ξ increases (in Figure 16(b)), although LB_{cell} deteriorates, rLB_{band} becomes stronger, thus eliminating many of the candidates that survived LB_{cell}. This renders our methodology robust to different problem settings.

Next, we compare three variants of BTM that use: (i) LB_{cell} only, (ii) LB_{cell}, rLB_{cross} only, and (iii) $LB_{cell}, rLB_{cross}, rLB_{band}$. We vary the trajectory length n and the minimum motif length ξ in Figures 17(a) and (b), respectively. The results verify that the bounds complement each other gracefully, and that the performance gains achieved are not due to just one or some of them.

Effect of Group Size τ: In GTM (Algorithm 3), the initial group size τ influences the pruning effectiveness and the computation cost of the algorithm. Generally, when τ is small, group-based pruning has a high pruning power but it requires high computation cost. In contrast, when τ is large, group-based pruning becomes faster but it becomes less effective. Figure 18 plots the response time of GTM for different values of τ (x-axis) and trajectory length n (as indicated by the label of each line). We observe that the response time is not overly sensitive to τ. In the following experiments, we set $\tau = 32$ by default as it seems to work well in all cases.



(a) Effect of trajectory length n



(b) Effect of minimum motif length ξ

Fig. 17 BTM, response time

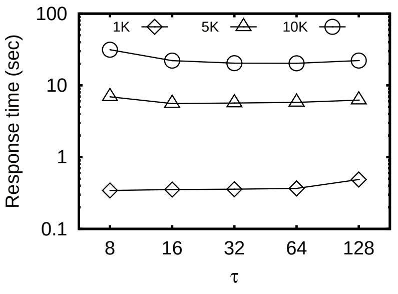
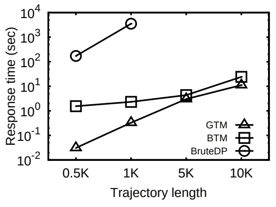
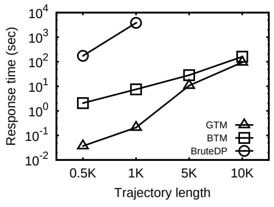


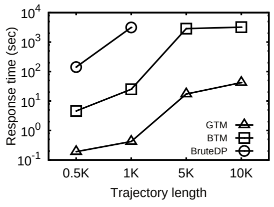
Fig. 18 GTM, effect of group size τ



(a) GeoLife



(b) Truck



(c) Wild-Baboon

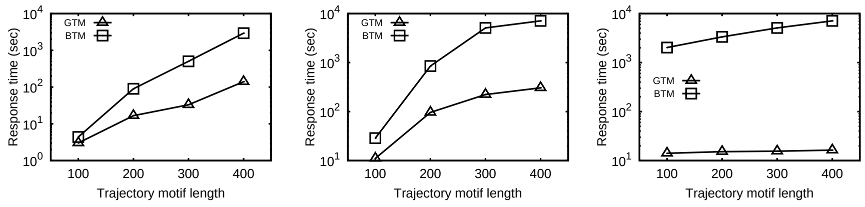
Fig. 19 Response time vs. trajectory length n, motif discovery problem

6.2.2 Performance Experiments

We compare the performance of our solutions (BTM, GTM) with the baseline (BruteDP) on the real datasets (GeoLife, Truck, and Wild-Baboon).

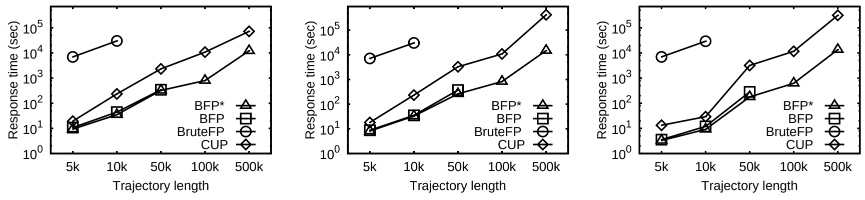
Figure 19 plots the average response time for different trajectory lengths n while fixing $\xi = 100$. BruteDP is prohibitively slow even for small trajectories (e.g., $n = 1000$), thus, we terminate it when its response time exceeds 2 hours. For the settings where it does terminate within reasonable time, our advanced solution (i.e., GTM) outperforms it by 3 orders of magnitude. GTM is the fastest algorithm. Due to the clear inefficiency of BruteDP, we exclude it from the following experiments.

In Figure 20, we measure response time as we vary the minimum trajectory motif length ξ (with n fixed to 5000). The relative performance of the methods is the same as in the previous experiment. The response time of all solutions increases with ξ. That is because a large ξ disqualifies short motifs with small DFDs, thus making it harder to identify early on a small bsf to enable aggressive pruning (see also Figure 15(a)).



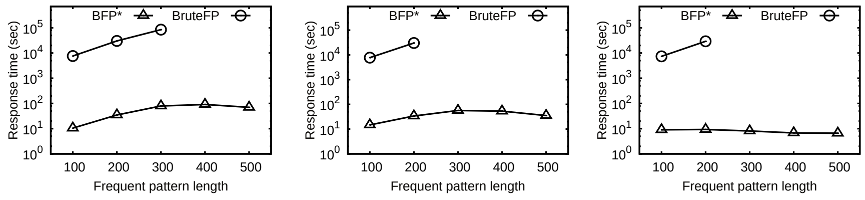
(a) GeoLife (b) Truck (c) Wild-Baboon

Fig. 20 Response time vs. minimum motif length ξ, motif discovery problem



(a) GeoLife (b) Pigeons (c) Wild-Baboon

Fig. 21 Response time vs. trajectory length n, frequent pattern discovery problem



(a) GeoLife (b) Pigeons (c) Wild-Baboon

Fig. 22 Response time vs. frequent pattern length ξ, frequent pattern discovery problem

6.3 Results on Frequent Pattern Discovery

We now turn to the frequent pattern discovery problem. We compare the performance of our solutions (BFP and BFP*) with the baseline solution (BruteFP) on three real datasets (i.e., GeoLife, Pigeons, and Wild-Baboon). Recall that BFP* is the space-efficient version of BFP.

Figure 21 plots the average response time for different trajectory lengths n. Our solutions (BFP and BFP*) outperform the baseline solution and CUP by at least one order of magnitude. Since BruteFP is too slow, we terminate it when its response time exceeds 6 hours. BFP and BFP* exhibit similar performance for small data lengths (e.g., n from $5k$ to $50k$). However, when n exceeds $50k$, BFP runs out of memory and thus gets terminated. Furthermore, BFP* performs slightly better than BFP as we discussed in Section 4.2. We will elaborate the memory consumption of the methods shortly.

In Figure 22, we measure the response time as a function of the frequent pattern length ξ. BFP* again outperforms BruteFP. Note that the response time of BFP* decreases slightly with large ξ (i.e., $\xi = 500$). To investigate this issue, in Figure 23 we plot the feasible cells (in black) of a sample trajectory from the GeoLife dataset. When ξ is too large, it becomes more difficult to find a path of feasible cells, and thus the number of surviving candidates drops.

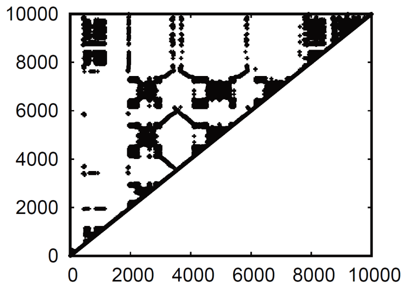


Fig. 23 Feasible cells in one trajectory in GeoLife

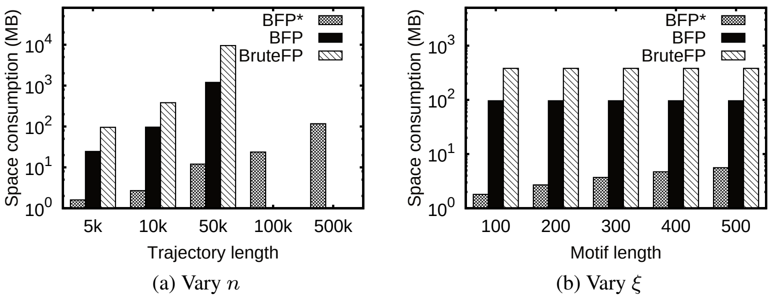


Fig. 24 Space consumption

Figure 24 shows the memory consumption of the solutions on the GeoLife dataset. We omit the experiments on the other datasets as the findings are similar. Figure 24(a) plots the memory consumption of the solutions versus the trajectory length n. BFP* performs much better than the others. Both BruteFP and BFP run out of memory when trajectory length exceeds $n = 100k$ and $500k$, respectively. BFP requires much less memory than BruteFP because BFP employs Boolean matrices instead of floating-point matrices. Figure 24(b) illustrates the memory consumption of these solutions by varying the frequent pattern length ξ. BFP* occupies much less memory than the other solutions.

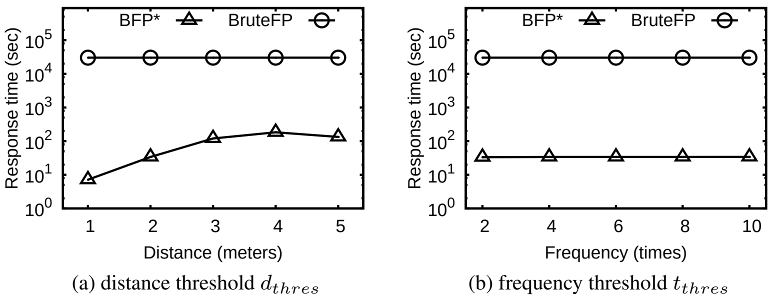
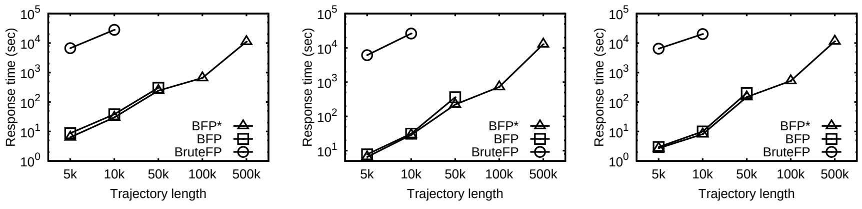


Fig. 25 Response time on Pigeons



(a) GeoLife (b) Pigeons (c) Wild-Baboon

Fig. 26 Response time vs. concatenated trajectory length n, multiple trajectories, frequent pattern discovery problem

We study the effect of the distance threshold (d_{thres}) on Pigeons in Figure 25(a). Since BruteFP does not apply pruning rules, its response time is independent of d_{thres}. The response time of BFP* rises with the distance threshold d_{thres}. Similarly, we study the effect of the frequency threshold t_{thres} on Pigeons in Figure 25(b). With the increase of t_{thres}, the response time of BFP* rises and then stabilizes. When t_{thres} is low, we could quickly determine a pattern as part of the result when its frequency exceeds t_{thres}. The experimental results on GeoLife and Wild-Baboon are similar; we omit them for brevity.

Finally, we evaluate our algorithms for frequent pattern discovery on multiple trajectories. In this experiment, we randomly select input trajectories (from the corresponding real dataset) until the concatenated trajectory's length exceeds n. Figure 26 reports the average response time as a function of the concatenated trajectory length n. It demonstrates the efficiency of our approaches. The performance trend is similar to the case of single input trajectory (see Figure 21).

7 Conclusion

In this paper, we study spatial trajectory analysis problems using the discrete Fréchet distance (DFD). Our contributions include: (i) a suite of novel lower bound functions for DFD, and (ii) a space-optimized approach that is both time- and space-efficient. Our fastest methods are over 3 orders of magnitude faster than the baseline solutions. In addition, our solutions can be extended to motif discovery and trajectory frequent pattern discovery problems on multiple trajectories (i.e., a trajectory database). Promising directions for future work include: devising approximate solutions that trade exactness for shorter running times; and extending our techniques to other trajectory similarity measures.

References

1. Agarwal, P.K., Avraham, R.B., Kaplan, H., Sharir, M.: Computing the discrete fréchet distance in subquadratic time. SIAM Journal on Computing **43**(2) (2014)
2. Alt, H., Efrat, A., Rote, G., Wenk, C.: Matching planar maps. In: SODA (2003)
3. Alt, H., Godau, M.: Computing the fréchet distance between two polygonal curves. International Journal of Computational Geometry & Applications **5**, 75–91 (1995)
4. Astefanoaei, M., Cesaretti, P., Katsikouli, P., Goswami, M., Sarkar, R.: Multi-resolution sketches and locality sensitive hashing for fast trajectory processing. In: SIGSPATIAL (2018)

5. Baldus, J., Bringmann, K.: A fast implementation of near neighbors queries for fréchet distance (gis cup). In: SIGSPATIAL (2017)
6. Brakatsoulas, S., Pfoser, D., Salas, R., Wenk, C.: On map-matching vehicle tracking data. In: VLDB (2005)
7. Bringmann, K.: Why walking the dog takes time: Frechet distance has no strongly subquadratic algorithms unless seth fails. In: FOCS (2014)
8. Bringmann, K., Künnemann, M., Nusser, A.: Walking the Dog Fast in Practice: Algorithm Engineering of the Fréchet Distance. In: SoCG, vol. 129 (2019)
9. Buchin, K., Buchin, M., Gudmundsson, J.: Constrained free space diagrams: a tool for trajectory analysis. IJGIS **24**(7), 1101–1125 (2010)
10. Buchin, K., Buchin, M., Gudmundsson, J., Löffler, M., Luo, J.: Detecting commuting patterns by clustering subtrajectories. International Journal of Computational Geometry & Applications **21**(03) (2011)
11. Buchin, K., Diez, Y., van Diggelen, T., Meulemans, W.: Efficient trajectory queries under the fréchet distance (gis cup). In: SIGSPATIAL (2017)
12. Cao, H., Mamoulis, N., Cheung, D.W.: Mining frequent spatio-temporal sequential patterns. In: ICDM (2005)
13. Cao, H., Mamoulis, N., Cheung, D.W.: Discovery of periodic patterns in spatiotemporal sequences. TKDE **19**(4), 453–467 (2007)
14. Chambers, E.W., Wang, Y.: Measuring similarity between curves on 2-manifolds via homotopy area. In: SoCG (2013)
15. Chen, L., Özsu, M.T., Oria, V.: Robust and fast similarity search for moving object trajectories. In: SIGMOD (2005)
16. Dütsch, F., Vahrenhold, J.: A filter-and-refinement-algorithm for range queries based on the fréchet distance (gis cup). In: SIGSPATIAL (2017)
17. Eiter, T., Mannila, H.: Computing discrete fréchet distance. Tech. rep., Information Systems Department, Technical University of Vienna (1994)
18. Gagliardo, A., Ioalè, P., Filannino, C., Wikelski, M.: Homing pigeons only navigate in air with intact environmental odours: a test of the olfactory activation hypothesis with gps data loggers. PLoS One **6**(8) (2011)
19. Giannotti, F., Nanni, M., Pinelli, F., Pedreschi, D.: Trajectory pattern mining. In: SIGKDD (2007)
20. Gudmundsson, J., Laube, P., Wolle, T.: Computational movement analysis. In: Springer handbook of geographic information (2011)
21. Gudmundsson, J., Thom, A., Vahrenhold, J.: Of motifs and goals: mining trajectory data. In: SIGSPATIAL (2012)
22. Gudmundsson, J., Valladares, N.: A gpu approach to subtrajectory clustering using the fréchet distance. TPDS **26**(4) (2015)
23. Han, J., Pei, J., Mortazavi-Asl, B., Pinto, H., Chen, Q., Dayal, U., Hsu, M.: Prefixspan: Mining sequential patterns efficiently by prefix-projected pattern growth. In: ICDE (2001)
24. Jeung, H., Yiu, M.L., Zhou, X., Jensen, C.S., Shen, H.T.: Discovery of convoys in trajectory databases. PVLDB **1**(1) (2008)
25. Kwan, M.P.: Interactive geovisualization of activity-travel patterns using three-dimensional geographical information systems: a methodological exploration with a large data set. Transportation Research Part C: Emerging Technologies **8**(1) (2000)
26. Lee, J.G., Han, J., Whang, K.Y.: Trajectory clustering: a partition-and-group framework. In: SIGMOD (2007)
27. Li, X., Han, J., Kim, S., Gonzalez, H.: Roam: Rule-and motif-based anomaly detection in massive moving object data sets. In: SDM, vol. 7 (2007)
28. Sinnott, R.W.: Virtues of the haversine. Sky and Telescope **68**(2) (1984)
29. Song, R., Sun, W., Zheng, B., Zheng, Y.: Press: A novel framework of trajectory compression in road networks. PVLDB **7**(9), 661–672 (2014)
30. Sriraghavendra, R., Karthik, K., Bhattacharyya, C.: Fréchet distance based approach for searching online handwritten documents. In: 9th International Conference on Document Analysis and Recognition, vol. 1 (2007)
31. Tang, B., Yiu, M.L., Mouratidis, K., Wang, K.: Efficient motif discovery in spatial trajectories using discrete fréchet distance. In: EDBT, pp. 378–389 (2017)
32. Toohey, K., Duckham, M.: Trajectory similarity measures. SIGSPATIAL Special **7**(1), 43–50 (2015)
33. Trajcevski, G., Ding, H., Scheuermann, P., Tamassia, R., Vaccaro, D.: Dynamics-aware similarity of moving objects trajectories. In: GIS (2007)

34. Vlachos, M., Kollios, G., Gunopulos, D.: Discovering similar multidimensional trajectories. In: ICDE (2002)
35. Wang, Y., Zheng, Y., Xue, Y.: Travel time estimation of a path using sparse trajectories. In: SIGKDD (2014)
36. Wei, H., Fellegara, R., Wang, Y., De Floriani, L., Samet, H.: Multi-level filtering to retrieve similar trajectories under the fréchet distance. In: SIGSPATIAL (2018)
37. Werner, M., Oliver, D.: Acm sigspatial gis cup 2017: range queries under fréchet distance. SIGSPATIAL Special **10**(1), 24–27 (2018)
38. Wylie, T., Zhu, B.: Protein chain pair simplification under the discrete fréchet distance. IEEE/ACM Transactions on Computational Biology and Bioinformatics **10**(6) (2013)
39. Xie, D., Li, F., Phillips, J.M.: Distributed trajectory similarity search. PVLDB **10**(11), 1478–1489 (2017)
40. Yi, B.K., Jagadish, H., Faloutsos, C.: Efficient retrieval of similar time sequences under time warping. In: ICDE (1998)
41. Yu, Y., Cao, L., Rundensteiner, E.A., Wang, Q.: Detecting moving object outliers in massive-scale trajectory streams. In: SIGKDD (2014)
42. Yuan, J., Zheng, Y., Zhang, C., Xie, W., Xie, X., Sun, G., Huang, Y.: T-drive: driving directions based on taxi trajectories. In: SIGSPATIAL (2010)
43. Zheng, Y.: Trajectory data mining: an overview. TIST **6**(3) (2015)
44. Zheng, Y., Zhang, L., Xie, X., Ma, W.Y.: Mining interesting locations and travel sequences from gps trajectories. In: WWW (2009)