Proc. the 5th International Conference on Knowledge Discovery and Information Retrieval (KDIR2013), IC3K Conference, September 2013 (to appear).

Trajectory Pattern Mining in Practice: Algorithms for Mining Flock Patterns from Trajectories

Xiaoliang Geng[1], Takeaki Uno[2] and Hiroki Arimura[1]
[1]*Hokkaido University, N14 W9, Kita-ku, Sapporo 060-0814, Japan*
[2]*National Institute of Informatics, 2-1-2, Hitotsubashi, Chiyoda-ku, Japan*
{*gengxiaoliang, arim*}*@ist.hokudai.ac.jp, uno@nii.jp*

Keywords: trajectory mining, spatio-temporal sequence mining, frequent itemset mining, geometric index

Abstract: In this paper, we implement recent theoretical progress of depth-first algorithms for mining flock patterns (Arimura et al., 2013) based on depth-first frequent itemset mining approach, such as Eclat (Zaki, 2000) or LCM (Uno et al., 2004). Flock patterns are a class of spatio-temporal patterns that represent a groups of moving objects close each other in a given time segment (Gudmundsson and van Kreveld, Proc. ACM GIS'06; Benkert, Gudmundsson, Hubner, Wolle, Computational Geometry, 41:11, 2008). We implemented two extensions of a basic algorithm, one for a class of closed patterns, called *rightward length-maximal* flock patterns, and the other with a *speed-up technique using geometric indexes*. To evalute these extensions, we ran experiments on synthesis datasets. The experiments demonstrate that the modified algorithms with the above extensions are several order of magnitude faster than the original algorithm in most parameter settings.

1 Introduction

1.1 Background

By the rapid progress of mobile devices and positional sensors, a massive amount of trajectory data, which are collections of sequences of real-valued locations with errors and missing values, have been accumulated. Since mining of trajectory data have different characteristics from traditional transaction data mining (Pei et al., 2004), research of trajectory mining has attracted a great deal of attention for recent years (Giannotti et al., 2007; Vieira et al., 2009).

A *trajectory database* on a time domain $\mathbb{T} = [1, T]$ is a collection S of n trajectories for n *moving objects*, such as wild animals, walking people, or floating cars, where each trajectory is a sequence of T points on the 2-dimensional space $\mathbb{R}^2$ to which an index called a trajectory ID is associated.

For a positive number $r > 0$, called a *max-width*, and non-negative integers $k, m \geq 0$, called *min-len* and *min-sup*, an (r,k,m)*-flock pattern* in a trajectory database S is a pair $P = (X, [b,e])$ of a set X of trajectory ids and a time interval $I = [b,e]$ in $\mathbb{T}$ that represents a set of at least m moving objects that move together in a continuous interval of length at least k time points in mutual distance at most r in L_∞-norm (the largest of the x- and y-distances). [1] Flock patterns are useful in detecting a group of highly correlated entities combining spatio-tempral features.

There have been not many, but some of existing researches on finding flock patterns in a given trajectory database patterns (Benkert et al., 2008; Gudmundsson and van Kreveld, 2006; Laube et al., 2005). However, most of them deals with *searching* flock patterns in a given trajectory data, while there have been a few work (Vieira et al., 2009; Romero, 2011) on *mining* flock patterns meaning to compute the complete set of flock patterns satisfying given costraints.

It is curious that there do not seem to exist no straightforward adoptation of the *pattern-growth approach* (Pei et al., 2004) to flock pattern mining so far, which is a most influential approach in conventional frequent itemset mining studies (Uno et al., 2004; Zaki and Hsiao, 2005), partly due to difficulties of handling spatio-temporal constraints in continuous multi-dimensional space.

In this paper, we focus on pattern-growth approach for the problem of finding all (r,k)-flock patterns, where our purpose is to make complete mining

[1] Our (r,k,m)-flock patterns use L_∞-distance on $\mathbb{R}^2$, while the original (m,k,r)-flock patterns of Benkert *et al.* (Benkert et al., 2008) used L_2-distance on $\mathbb{R}^2$.

of all patterns that satisfy a given constraint in an input database. For the purpose, we have been developping our algorithm FPM for complete mining of flock patterns (Arimura et al., 2013), the first pure pattern-growth style algorithm. Particularly, this paper focuses on two extensions of of FPM, called RFPM and G-RFPM. The former RFPM finds a class of closed patterns, called *rightward length-maximal flock patterns*, while the latter G-RFPM uses *speed-up technique using a geometric index* (Arimura et al., 2013).

We implemented the basic and the improved algorithms above based on pattern-growth mining approach. To evalute these extensions, we then ran experiments on inplanted synthesis datasets. The experiments demonstrate that both of extensions significantly improve on the efficiency of the original algorithm FPM in a wide range of parameter settings.

In the case of a trajectory database with 200K points, for example, where $C = 5$ copies of $K = 6$ hidden patterns are embedded into 200 trajectories of length 1K points, the running times for FPM, RFPM, and G-RFPM found all patterns in 61.61, 0.96, and 0.03 seconds, respectively. From these results, we obtained around 60 and 30 times speed-ups by the first and second extensions, respectively, and finally, the total speed-up becomes around 2,000 times.

This paper is organized as follows. Sec.2 gives definitions for flock pattern mining including our rightward length-maximal flock patterns. Sec.3 presents the basic algorithm as well as two improvements. Sec.4 is a main section of this paper that shows experimental results. Finally, Sec.5 concludes.

2 Preliminaries

2.1 Basic definitions

Let $\mathbb{R}$ and $\mathbb{N}$ be the set of all real numbers and all nonnegative integers, respectively. For integers a, b ($a \leq b$), we denote by $[a,b] = \{a, a+1, \ldots, b\}$ the discrete interval between a and b. If $a \leq b$ are real numbers, then $[a,b]$ denotes a continuous interval in $\mathbb{R}$ as usual. For a set A, $|A|$ denotes the cardinality of A, and A^* denotes the set of all possibly empty, finite sequences over A.

2.2 Trajectory Database

Let n and $T \geq 0$ are pre-determined nonnegative integers, which indicate the number of moving objects and the maximum value for discrete time stamps, respectively. Let $\mathbb{R}^2$ be the 2-dimensional continuous space, or the *plane*.

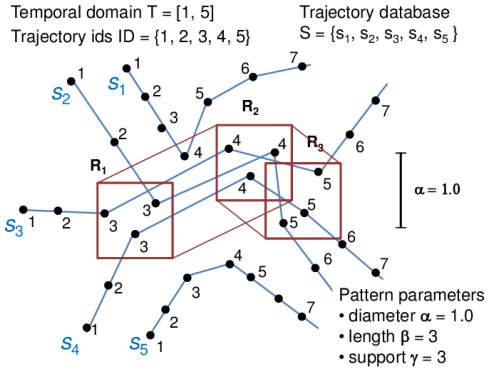

Figure 1: Examples of a trajectory database S_1 on $ID = \{1,\ldots,5\}$ and $\mathbb{T} = [1,7]$ and a $(1.0,2,2)$-flock pattern $P_1 = (X_1, I_1) = (\{2,3,4\},[3,5])$ with diameter $||P_1||_\infty^{S_1} \leq 1.0$, length $len(P_1) = 3$, and support $supp(P_1) = 3$. Here, each line indicates a trajectory and the numbers attached to points are time stamps.

A *trajectory database* on the space domain $\mathbb{R}^2$ and the time domain $\mathbb{T} = \{1,\ldots,T\}$ is a finite set

$$S = \{\, s_i \,|\, i = 1,\ldots,n \,\} \subseteq (\mathbb{R}^2)^T \qquad (1)$$

of the trajectories for n moving objects $o_1,\ldots,o_n$, where for every $i = 1,\ldots,n$,

- the index i, called the *trajectory ID*, is drawn from a set of n identifiers $ID = \{1,\ldots,n\}$, and
- the i-th trajectory s_i is a sequence

  $$s_i = s_i[1]\cdots s_i[T] \in (\mathbb{R}^2)^T$$

  of T points on the 2-dimensional space $\mathbb{R}^2$ such that its t-th point is $s_i[t] = (x_{it}, y_{it}) \in \mathbb{R}^2$.

Example 1. In Fig. 1, we show an example of a trajectory database S, which consists of five trajectories of length $T = 7$.

For example, GPS-trajectories of wild animals, walking people with Wifi device, Probe car data (or floating car data) are instances of such trajectory databases.

2.3 The class of flock patterns

For such trajectory databases, we introduce the class $\mathcal{FP}$ of spatio-tempral patterns, called flock patterns, based on L_∞-norm as follows [2]. Formally, the class of flock patterns is defined as follows.

Definition 1 (FP)**.** A *flock pattern* on $\mathbb{T}$ is a pair $P = (X,[b,e])$, where

- $X \subseteq ID$ is a finite set of ids, called the *ID set* of P, and

[2]The original version of flock patterns are defined based on L_2-norm in (Laube et al., 2005; Gudmundsson and van Kreveld, 2006).

- $I = [b,e]$ is a discrete interval in $[0,T]$ with $b \leq e \leq T$, where b and e are called the *start* and *end time* of P.

We define the support, length, and width of a flock pattern as follows.

- The *support* of P, denoted by *supp*(P), is defined by the number of trajectory (ID) contained in X, that is, $supp(P) = |X|$.
- The *length* of P, denoted by $len(P)$, is the width of the interval I, that is, $len(P) = e - b + 1$.

Clearly, we have $0 \leq supp(P) \leq n$ and $0 \leq len(P) \leq T$.

Example 2. In Fig. 1, we show an example of a flock pattern $P_1 = (X_1, I_1)$, where the ID set is $X_1 = \{2,3,4\}$ and the interval is $I_1 = [3,5]$.

To define the width, we require some definitions below. For a point $p = (x,y)$ on 2-dimensional plane $\mathbb{R}^2$, the x- and y-coordinates of p are denoted by by $p.x = x$ and $p.y = y$, respectively. For two points p and p' on $\mathbb{R}^2$, we denote the *L_∞-distance between p and p'* by $L_\infty(p,p') = \max\{|p.x - p'.x|, |p.y - p'.y|\}$. By definition, $L_\infty(p,p')$ is nonnegative, and coincides zero if and only if $p = p'$.

The *diameter* of a set $A = \{p_1, \ldots, p_n\}$ of points, denoted by $||A||_\infty$, is the maximum L_∞-distance between any two points in A, defined by

$$||A||_\infty = \max_{p,p' \in A} L_\infty(p,p'), \tag{2}$$

The width $||A||_\infty$ of a set A is always nonnegative, and equals zero if and only if A consists of a single point. We can show that $||A||_\infty$ is linear time computable in $n = |A|$ on $\mathbb{R}^2$. For any $d \geq 2$, $||A||_\infty$ can be computed $O(dn)$ time in $\mathbb{R}^d$, which is still linear in n for fixed d.

In an input database S, the t-th time slice, denoted by $S[X][t]$, is the set of all points that appear in the trajectories of X with time stamp t.

- The *width* $||P||_\infty^S$ of a flock pattern $P = (X,I) = (X,[b,e])$ is defined by the maximum diameter of the t-th time slice of the trajectories in X over all $t \in [b,e]$.

Actually, we have the next lemma.

Lemma 1. *The width of P can be computed by Algorithm 1 in $O(m\ell)$ time, where $m = supp(X)$ is the support of P and $\ell = len(P)$ is the length of P.*

Let $r > 0$ be a positive number, and $k, m \geq 0$ are non-negative integers, respectively, called a *maximum width* (max-width), a *minimum length* (min-len), and a *minimum support* (min-sup) parameters. Then, we define:

Algorithm 1 Computing the width $||P||_\infty^S$ of a flock pattern $P = (X,[b,e])$ in a database $S = \{ s_i \mid i = 1, \ldots, n \}$

```
1: width ← 0;
2: for t ← b, b+1, ..., e do
3:     S_t ← { s_i[t] | i ∈ X };          ▷ the t-th slice
4:     width ← max{width, ||S_t||_∞};
5: return width;
```

- an *r-flock pattern* is any flock pattern P such that $||P||_\infty \leq r$,

Consider the class of r-flock patterns in a trajectory database S.

- An *(r,k)-flock pattern* is any r-flock pattern P with $len(P) \geq k$.

Example 3. The pattern P_1 of Fig. 1 in the last example has diameter $||P_1||_\infty^{S_1} \leq 1.0$, length $len(P_1) = 3$, and support $supp(P_1) = 3$. Thus, it is a $(1.0,2,3)$-flock pattern for $r = 1.0$, $k = 2$, and $m = 3$.

In this paper, we consider all (r,k)-flock patterns in a given trajectory database.

2.4 Rightward length-maximal patterns

For a given max-width parameter $r \geq 0$, it is often useful to find only (r,k)-flock patterns $P = (X,[b,e])$ whose time interval $[b,e]$ are extended rightward along time line as long as possible preserving the diameter r (See (Gudmundsson and van Kreveld, 2006)). This idea of *length-maximal mining* is expected to reduce the number of solutions and running time than just finding all (r,k)-patterns.

A flock pattern $P = (X,[b,e])$ is said to be a *rightward length-maximal* flock pattern in S if its interval cannot be extended rightward without changing the width of P in S.

Formally, it is defined as follows.

Definition 2 (RFP)**.** A flock pattern $P = (X,[b,e])$ in S is a *rightward length-maximal* flock pattern (RFP, for short) if there is no other flock pattern $P' = (X,[b,e'])$ in S such that (i) P' has the same ID set X as P, and (ii) the right end of P' is strictly more larger than that of P.

By definition, any RFP in S is an FP. However, the converse does not hold in general. Thus, we have the inclusion $\mathcal{RFP}(r,k) \subseteq \mathcal{FP}(r,k)$.

Example 4. In the example of Fig. 1, the flock pattern $P_1 = (X_1,[3,5])$ of length three is an RFP in S_1, while $P_2 = (X_1,[3,4])$ and $P_3 = (X_1,[3])$ are non-rightward length-maximal FPs, where $X_1 = \{2,3,4\}$.

On the other hand, P_1 has RFPs $P_4 = (X_1, [4,5])$ and $P_5 = (X_1, [5])$.

2.5 The data mining problems

For any class name $\mathcal{C} \in \{\mathcal{FP}, \mathcal{RFP}, \ldots\}$ and any parameter values $r, k \geq 0$, we denote by $\mathcal{C}(r,k)$ the class of all (r,k)-flock patterns within the class $\mathcal{C}$. Similarly, we define the classes $\mathcal{C}(r)$, and $\mathcal{C}(r,k,m)$ as well. From now on, we consider the classes $\mathcal{FP}(r,k)$ and $\mathcal{RFP}(r,k)$.

We state our data mining problem as follows.

Definition 3. (FLOCK PATTERN MINING PROBLEM FOR PATTERN CLASS $\mathcal{C}$) Let $\mathcal{C}$ be a class of flock patterns. An input is a tuple (S,r,k) of an input trajectory database S, and parameter values r and $k \geq 0$. The task is to find all flock patterns P in S within class $\mathcal{C}$ without repetition that have width at most r and length at least k.

Similarly, we can consider the flock pattern mining problem with paramters (r,k,m).

We evaluate the performance of a flock pattern mining algorithm $\mathcal{A}$ in terms of enumeration algorithms (Avis and Fukuda, 1993). Let N and M be the input size and the number of patterns as solutions. A pattern mining algorithm $\mathcal{A}$ is said to have *polynomial delay* (poly-delay) if the *delay*, which is the maximum computation time between two consecutive outputs, is bounded by a polynomial $p(N)$ in N. $\mathcal{A}$ is of *polynomial space* (poly-space) if the maximum size of its working space, in addition to that of output stream O, is bounded by a polynomial $p(N)$.

3 Algorithms

In this section, we present our pattern mining algorithms for FPs and RFPs. We also give a speed-up technique using geometric indexes to prune redundant candidates.

3.1 A basic DFS algorithm for FPs

We first present a basic mining algorithm FPM (basic flock pattern miner) for FPs. In Algorithm 2 we present the algorithm FPM with its subprocedure RecFPM for mining (r,k)-FPs.

In the overall design of our algorithm FPM, we employ DFS (depth-first search) procedure according to pattern growth approach (e.g., (Pei et al., 2004)) approach, as in PrefixSpan (Pei et al., 2004), Eclat (Zaki, 2000) and LCM (Uno et al., 2004).

Algorithm 2 A basic DFS algorithm FPM for finding all (r,k)-flock patterns in an input trajectory database S given maximum width r and minimum length k.

```
1:  procedure FPM(ID, S, r, k)
2:    for ℓ ← k, ..., T do                          ▷ Every length
3:      for b_0 ← 1, ..., T do                      ▷ Each start time in 𝕋
4:        ℓ_0 ← b_0 + ℓ − 1;
5:        ID_1 ← ID;
6:        while ID_1 ≠ ∅ do                         ▷ Each id in ID
7:          i_0 = deletemin(ID_1);
8:          P_0 ← ({i_0}, [b_0, ℓ_0]);              ▷ Initial pattern
9:          RecFPM(P_0, ID_1, S, r, k);

10: procedure RecFPM(P = (X, [b, e]), ID, S, r, k)
11:   if ||P||_∞^S > r then
12:     return ;                                    ▷ P is too wide
13:   output P;
14:   ID_1 ← ID;
15:   while ID_1 ≠ ∅ do
16:     i = deletemin(ID_1);
17:     RecFPM(Q = (X ∪ {i}, [b, e]), ID_1, S, r, k);
18:   end while
```

In DFS (or pattern growth) approach, a recursive mining procedure searches for all descendant of the current pattern from smaller to larger in depth-first manner using backtracking. The advantage of DFS approach is that DFS miners are proven fast in main memory environment and can be easily implemented as a simple recusive procedure.

At the top-level of FPM, for each possible length $\ell \in [k,T]$ no less than k, it invokes the recursive subprocedure RecFPM given as arguments an initial pattern $P_0 = (X_0, [b_0, e_0])$ consisting of a singleton ID set $X_0 = \{i_0\}$ and an interval $[b_0, e_0]$ for every possible combination of $i_0 \in ID$ and $b_0 \in [0,T]$. The end time e_0 is calculated by b_0 and ℓ.

The recursive subprocedure RecFPM is a DFS algorithm (or a backtracking algorithm) that searches the hypthesis space of all r-flock patterns with length exactly ℓ as follow.

Starting from the initial pattern $P_0 = (X_0, [b_0, e_0])$ consisting of a singleton ID set $X_0 = \{i_0\}$, the procedure enumerates all subsets X of ID using a backtracking algorithm similar to depth-first search algorithms for frequent itemset mining, such as Eclat (Zaki, 2000) and LCM (Uno et al., 2004).

For each generated subset X, the procedure forms a candidate (r,k)-flock pattern $P = (X, [b,e])$ with a specified interval $[b,e]$. Then, the algorithm computes the width $||P||_\infty$ of the pattern P by accessing the trajectories in S, and checks if P satisfies the width con-

straint $||P||_\infty \leq r$. If the condition is violated, then it prunes the search for P and all of its descendants.

This *width-based pruning rule* is justified by the following lemma, which says the class of (r,k)-patterns has the anti-monotonicity w.r.t. set inclusion of their ID sets.

Lemma 2 (anti-monotonicity)**.** *Let $P_i = (X_i, I_i)$ are two flock patterns, where $i = 1,2$. If P_2 is an (r,k)-flock pattern in S and if $X_1 \subseteq X_2$ and $I_1 \subseteq I_2$ hold, then P_1 is also an (r,k)-flock pattern in S.*

From this lemma, once a candidate pattern $P = (X,I)$ does not satisfy the width and length constrants, any descendant of P obtained by adding new trajectory (ids) to X no longer satisfies the constraints. Therefore, we can prune the whole search sub-space for descendants of P for (r,k)-flock patterns.

On the running time and space of the algorithm FPM, The following proposition is easily derived from our manuscript (Arimura et al., 2013).

Proposition 1. *(Arimura et al., 2013) Let S be an input trajectory database S of n trajectories with length T. Then, the algorithm FPM in Algorithm 2 solves the flock pattern mining problem for the class $\mathcal{FPM}(r,k)$ of (r,k)-flock patterns in S. It uses $O(knT^2)$ time per pattern and $O(k^2)$ words of space, respectively, where $k = supp(X) = |X|$ is the support of the pattern X being enumerated.*

From the practical view, $O(T^2)$ term in the time complexity of FPM is too large to apply it to long trajectories with large T. We point our that this $O(T^2)$ term come from the doubly nested for-loop in Lines 2 and 3 of Algorithm 2. In the next subsection, we will see how we can remove this $O(T^2)$ term by focusing on mining of RFPMs.

3.2 A modified algorithm for RFPs

Next, we present a modified mining algorithm RFPM (rightward flock pattern miner) for RFPs (rightward length-maximal flock patterns), the class of rightward length-maximal flock patterns. In Algorithm 4, we present the algorithm RFPM with its subprocedure RecRFPM for mining (r,k)-RFPs.

3.2.1 Rightward horizontal closure

From the view of frequent pattern mining, RFPs in a trajectory database are a sort of *closed patterns*, which have been extensively studied in frequent itemset mining (FIM) field (Uno et al., 2004; Zaki and Hsiao, 2005) as well as formal concept analysis (FCA) field. Many efficient closed pattern mining algorithms use a class of operation, called *closure* operation, which enlarge a given, possibly non-closed pattern to obtain its *closed* version.

Algorithm 3 An algorithm for computing the unique rightward length-maximal flock pattern. Note that $||S[X][t]||_\infty$ is defined to be ∞ for $t \notin [1,T]$.

```
1: procedure RH_CLOSURE((X,[b0,e0]);S,r)
2:     t ← b0;
3:     while ||S[X][t]||∞ ≤ r do
4:         t ← t+1;
5:     b ← b0; e ← t−1;
6:     return (X,[b,e]);
```

For RFPs, we actually have a *rightward horizontal closure* operation that extends the interval of a given non RFPs to obtain a proper RFP.

Definition 4 (rightward horizontal closure)**.** Let $P = (X, I = [b,e])$ be any flock pattern in a database S. Then, the *rightward horizontal closure* of P in S, denoted by RH_Closure$(P;S,r)$, is the unique flock pattern $P_{max} = (X, I = [b, e_{max}])$ such that $e_{max} \in [0,T]$ is the maximum value of end position e' satisfying the equality

$$||P' = (X,[b,e'])||_\infty = ||P||_\infty. \tag{3}$$

Note that the rightward horizontal closure operation only change the end position e, but not change the ID set X or starting time b of the original P at all.

In Algorithm 3, we show the procedure RH_Closure that computes the rightward horizontal closure of non-RFP P in $O(k\ell)$ time, where $k = supp(P) = |X| = O(n)$ and $\ell = len(P_{max}) = O(T)$.

The following lemmas show the correctness of the rightward horizontal closure. First, the key of the correctness is the following characterization, which can be easily shown from definition of RFPs.

Lemma 3 (characterization)**.** *Let $P = (X,[b,e])$ be an (r,k)-flock pattern in S. Then, P is rightward length-maximal if and only if*

- $||S[X][t]||_\infty \leq r$ *for all* $t \in [b,e]$, *and*
- $||S[X][e+1]||_\infty > r$,

where we extend the t-th time slice $||S[X][t]||_\infty$ to be ∞ if either $t < 1$ or $t > T$ holds for convenience.

From the above lemma, we have the correctness below.

Lemma 4. *(Arimura et al., 2013) The rightward horizontal closure P_{max} of a possibly non-rightward length-maximal r-FP P is the unique longest r-RFP such that the ID sets and the start time are identical to those of P.*

Algorithm 4 An algorithm FPM for finding all length-maximal (r,k)-flock patterns appearing in a given trajectory database S with ID for maximum width r and minimum length k.

```
1: procedure RFPM(ID, S, r, k)
2:   for b_0 ← 1, ..., T do        ▷ Each start time in 𝕋
3:     for i_0 ← 1, ..., n do          ▷ Each id in ID
4:       P_0 = ({i_0}, [b_0, *]);
5:       RecRFPM(P_0, ID, S, r, k);

6: procedure RecRFPM(P = (X, [b, *]), ID, S, r, k)
7:   P = (X, [b, e]) ← RH_Closure((X, [b, *]); S, r);
8:   if len(P) < k then
9:     return ;        ▷ P is not an (r,k)-flock pattern
10:  output P;
11:  ID_1 ← ID;
12:  while ID_1 ≠ ∅ do
13:    i = deletemin(ID_1);
14:    P_1 = (X ∪ {i}, [b, *]);
15:    RecRFPM(P_1, ID_1, S, r, k);
16:  end while
```

Since $len(P_{max}) \geq len(P)$ always holds for P_{max}, we see that if P satisfies the (r,k)-constraint then so does the obtained RFP P_{max}. Hence, P_{max} is the unique longest (r,k)-RFP version of P that share the ID set and start time.

3.2.2 Putting them together

We describe the computation done by the algorithm RFPM. The overall structure of RFPM is almost identical to the basic algorithm FPM. Given a database S, the main algorithm RFPM invokes the recusive subprocedure RecFPM with an initial pattern P_0 as before.

Only the difference in the top level is that RFPM iterates only $O(T)$ iteration here for the start position b_0 rather than $O(T^2)$ iteration in FPM using an initial pattern $P_0 = (\{i_0\}, b_0, *)$ with missing end position $e_0 = *$, called a *partial pattern* here.

The computation of the recursive subprocedure RecRFPM proceeds in the following steps.

- Receiving a partial RFP $P_* = (X, b, *)$ as arguments, the recursive procedure RecFPM computes the rightward horizontal closure $P = (X, [b, e])$ from P_* by the procedure RH_Closure with max-width r.
- Next, if the obtained RFP P satisfies (r,k)-constraints, then output it. Otherwise, we sefely prune all descendants as before.

Algorithm 5 An algorithm G-RFPM for finding all length-maximal (r,k)-flock patterns appearing in a given trajectory database S with ID for maximum width r and minimum length k.

```
1: procedure G-RFPM(X, b, k, ID, S, r, k)
2:   Let S = { s_i | i = 1, ..., n };
3:   for b_0 ← 1, ..., T do        ▷ Each start time in 𝕋
4:     Build a grid index for point set U ← S[b_0];
5:                          ▷ The time slice at time b_0
6:     for i_0 ← 1, ..., n do          ▷ Each id in ID
7:       p ← s_{i_0}[b_0];  δ ← r;        ▷ initial point p
8:       R ← [p.x−δ, p.x+δ] × [p.y−δ, p.y+δ];
9:             ▷ 2r×2r-query rectangle at center p
10:      ID_0 ← U.Range(R); P_0 ← ({i_0}, [b_0, *]);
11:      RecRFPM(P_0, ID_0, S, r, k);
12:    end
13:  end
```

- Finally, RecFPM recursively calls its copy with an extended pattern $P_1 = (X \cup \{i\}, [b, *])$. To avoid duplicated generation of patterns, the id i is removed from the universe ID.

From a similar argument to (Uno et al., 2004) based on reverse search technique of (Avis and Fukuda, 1993), we have the following time and space complexities of RFPM.

Theorem 2. *(Arimura et al., 2013) Let S be an input trajectory database S of n trajectories with length T. Then, the algorithm RFPM in Algorithm 4 solves the flock pattern mining problem for the class $\mathcal{RFPM}(r,k)$ of (r,k)-flock patterns in S. It uses $O(knT)$ time per pattern and $O(k^2)$ words of space, respectively, where $k = supp(X) = |X|$ is the support of the pattern X being enumerated.*

We can generalize RFPM for the case of the d-dimensional space $\mathbb{R}^d$ for every $d \geq 1$ with extra $O(d)$ factor in time and space by only modifying the procedure RH_Closure for $\mathbb{R}^d$.

3.3 Speed-up using geometric index

In this subsection, we present a speed-up technique using geometric index in $\mathbb{R}^2$, called *geometric database reduction*, which achieve order of magnitude accelaration of both of FPM and RFPM algorithms, which is orthogonal to the rightward horizontal closure technique.

In Algorithm 5, we present our modified mining algorithm G-RFPM (grid-based flock pattern miner) based on RFPM using geometric constraint on the 2-dimensional plane for (r,k)-patterns. The algorithm uses RecFPM in Algorithm 5 as subprocedure.

Given a trajectory database S, maximum width $r > 0$ and minimum length k as arguments, the algorithm G-RFPM starts with selecting a combination of a trajectory id i_0 in ID and a starting time b_0 in $\mathbb{T} = [1,T]$ as in the original FPM or RFPM.

Let $i_0 \in ID$ and $b_0 \in [1,T]$ be any pair of trajectory ID and start time. Then, we know that the trajectory s_{i_0} starts from the point $c = s_{i_0}[b_0]$ in the database. Now, we assume to find any (r,k)-flock pattern of the form $P = (X,[b_0,*])$ be any (r,k)-pattern such that $i_0 \in X$ in the database.

From the L_∞-geometry of the plane $\mathbb{R}^2$, we can show that any trajectory i in ID must be contained in the rectangle $R = R(c,2r)$ of size $2r \times 2r$ given by

$$R(c,2r) = [x-\delta,x+\delta] \times [y-\delta,y+\delta] \subseteq \mathbb{R}^2, \qquad (4)$$

where $x = p.x$, $y = p.y$, and $\delta = r$. Consider the $t = b_0$ time slice U of S, that is, the set U of all points with the specified time $t = b_0$, given by

$$U = \{\, p = s_i[t] \,|\, s_i \in S, i \in ID, t = b_0 \,\}. \qquad (5)$$

Let $P = (X,[b,e])$ with $b = b_0$ be any target (r,k)-flock patterns in S. For any trajectory ID i, if the ID i belongs to X then the corresponding trajectory s_i starts from any point in $U \cap R(c,2r)$. Therefore, we can reduce the original domain ID of candidate IDs for X to the following smaller sub-domain

$$ID(R) = \{\, i \in ID \,|\, p_i = s_i[b_0] \in U \cap R \,\}. \qquad (6)$$

By using an appropriate geometric index, such as quad trees or range trees, we can compute $ID(R)$ by making the range query

$$ID(R) = U.Range(R) \qquad (7)$$

in $q = O(\log^2 \sigma)$ time by quad trees, or $O(\log^2 n)$ time by range trees using $O(n \log n)$ time preprocessing of S, where $n = |U| = |S|$ and $\sigma = ||U||_\infty$ are the L_∞-diameter of points in U. Then, the total overhead becomes $O(N \log^2 n)$ time (Arimura et al., 2013), which is linear in input size N with polylogarithmic factor.

As shown in Sec. 4, the above modification on RFPM to obtain G-RFPM greatly reduces the time complexity of the algorithm.

4 Experiments

We ran experiments on synthesis datasets to evaluate the efficiency of our algorithms.

4.1 Data

We generated a sets of inplanted synthesis trajectory datasets using our data generator implemented in C++ as follows. Let $n = 200$ and $T = 200$. Our data set is a collection of random trajectories in which C copies of random patterns are implanted as follows. We first fixed $a \times a$ area $\mathbb{A}$ in the plane, where $a = 40.0$, and then generated a set of n trajectories of length T by uniform distribution on $\mathbb{A}$. Then, we embbed C copies of each of K random short trajectories of length L_* are implanted in some of generated trajectories, where location of the copies are randomly perturbated within width r_*. In our experiments, we set $C = 5$, $K = 6$, $L_* = 20$, and $r_* = 1.0$. The other parameters are varied in experiments.

4.2 Methods

We implemented our algorithms FPM (BFPM), RFPM (BFPM R), and G-RFPM (GFPM R) of Sec. 3 in C++. We also implemented a simple grid-based geometric index in C++, where the plane is devided into $b \times b$ grid cells, and cells are looked up by constant time random access followed by sequential scan of a point list, where $b = 5$ most time.

We compiled the above programs by g++ of GNU, version 4.6.3. We used a PC with Intel(R) Xeon(R) CPU E5-1620, 3.60GHz with 32GB of memory on OS Ubuntu Linux, version 12.04. We used the following default parameters otherwise stated: Data mining algorithm use width $r = 1.0$, length $k = 20$, and and min-sup is $m = 5$ for patterns.

In the experiments, we varied as data parameters, the number n and length T of input trajectories, and as mining parameters, the minlen k, minsup m, and minwid r. We used default values for other values. We note that in Exp 1a and Exp 1b, only the number of false random trajectories is varied, while the numbers C and K of the copies and the true patterns are kept constant. In plots below, each line indicates the running time, while the number attached to each mark indicates the number of solutions.

4.3 Results A: the speed-up by rightward length-maximal flock patterns

In this subsection, we examine the effect of mining of RFPs (rightward length-maximal flock patterns) introduced in Sec. 3.2, compared to mining of FPs (orginary flock patterns). For the purpose, we measure the number of solutions and the running time by running RFPM (BFPM R, in plots) of Sec. 3.2 for mining all RFPs with length $\geq k$, compared to basic FPM (BFPM) of Sec. 3.1 for mining all FPs with length $\geq k$.

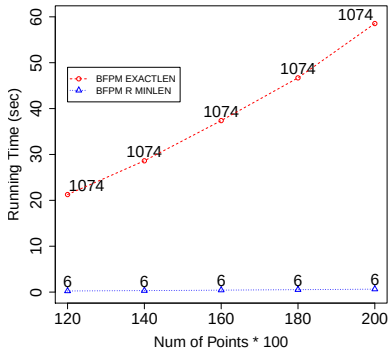

Figure 2: Exp 1a: The running time (and the number of patterns by mark) by algorithms FPM (BFPM) for FPs and RFPM (BFPM R) for RFPs by varying the the total number n of input points from 12K to 20K points, where a number attached to each mark indicates the number of soliutions.

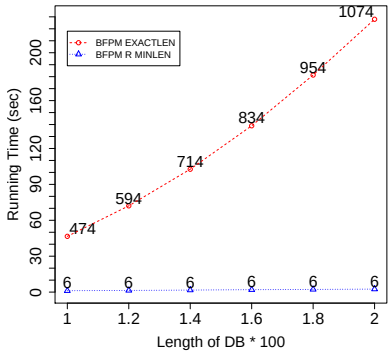

Figure 3: Exp 2a: The running time (and the number of patterns by mark) by algorithm FPM (BFPM) for FPs and RFPM (BFPM R) for RFPs varying the length T of input trajectories from 100 to 200 points, where a number attached to each mark indicates the number of soliutions.

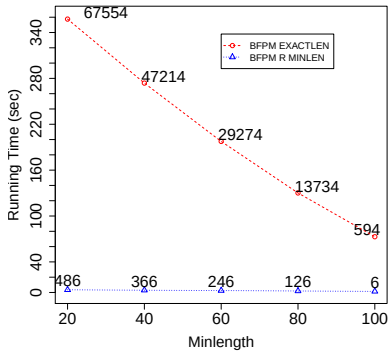

Figure 4: Exp 3a: The running time (and the number of patterns by mark) by algorithm FPM (BFPM) for FPs and RFPM (BFPM R) for RFPs by varying the min-len k from 20 to 100 points.

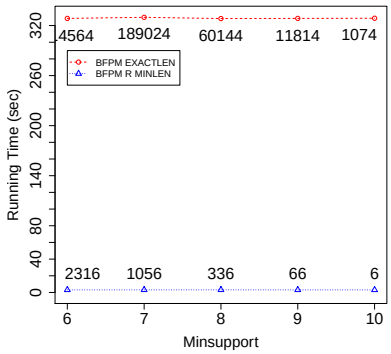

Figure 5: Exp 4a: The running time (and the number of patterns by mark) by algorithm FPM (BFPM) for FPs and RFPM (BFPM R) for RFPs by varying the min-sup m from 7 to 10.

Exp 1a: In Fig. 2, we show the running time and the number of patterns of by varying the number of points of input size n from 60 to 100 trajectories.

Exp 2a: In Fig. 3, we show the running time by varying the length of input trajectory database from 100 to 200 trajectories.

From Exp 1b and Exp 2 above, we see that the algorithm RFPM exactly detect the number $K = 6$ of true patterns, while FPM detects the larger numbers depending on n.

Exp 3a: In Fig. 4, we show the running time and the number of patterns by varying the minlen k of mining parameter from 20 to 100.

For example, in the case of minlen is $k = 20$ points, the running times for FPM and RFPM are 72.92 (sec) and 1.44 (sec), respectively, resulting around 50 times speed-up, while the numbers of solutions are 594 and 6 patterns, resulting around 100 times reduction.

Exp 4a: In Fig. 5, we show the running time and the number of patterns by varying the minsup m of mining parameter from 6 to 10. For this experiment, we generate patterns with the support of 10.

Exp 5a: In Fig. 10, we show the running time and the number of patterns by varying the maxwidth r of mining parameter from 1 to 5. For this experiment, we fix $a \times a$ area $\mathbb{A}$ to $a = 500.0$.

Summary of Results A: Overall, RFPM with RFPs is around 50 times faster than FPM with FPs as well as the number of RFPs is around 100 times smaller than that of FPs at maximum in our experiments. Specifically, we obtain the larger speed-up by RFP, the longer the input trajectories, or the smaller the minlen of flock patterns, as expected by theory (Arimura et al., 2013).

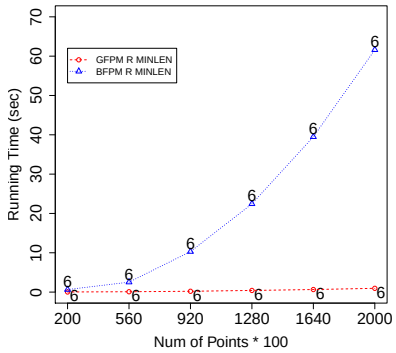

Figure 6: Exp 1b: The running time (and the number of patterns by mark) by algorithms RFPM (BFPM R) and G-RFPM (GFPM R) for RFPs by varying the the total number n of input points from 20K to 200K points.

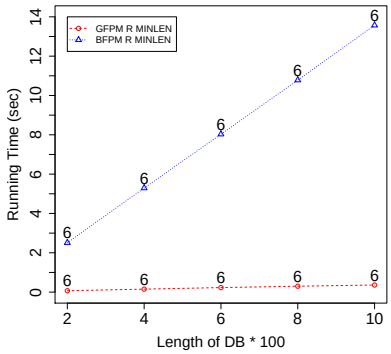

Figure 7: Exp 2b: The running time (and the number of patterns by mark) by algorithms RFPM (BFPM R) and G-RFPM (GFPM R) for RFPs by varying the length T of input trajectories from 200 to 1000 points.

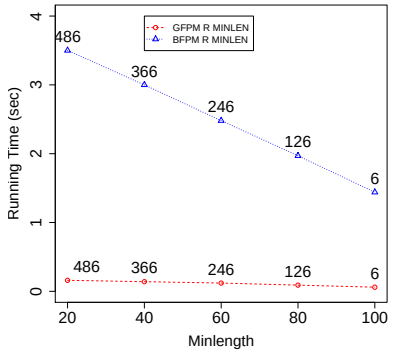

Figure 8: Exp 3b: The running time (and the number of patterns by mark) by algorithms RFPM (BFPM R) and G-RFPM (GFPM R) for RFPs by varying the min-len k from 20 to 100 points.

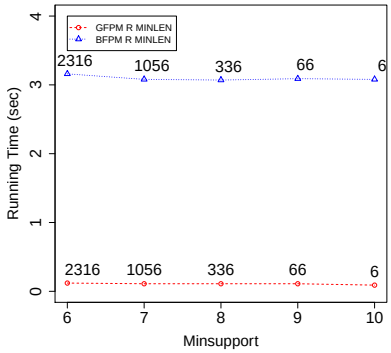

Figure 9: Exp 4b: The running time (and the number of patterns by mark) by algorithms RFPM (BFPM R) and G-RFPM (GFPM R) for RFPs by varying the min-sup m from 7 to 10.

4.4 Results B: the speed-up by geometric database reduction

In this subsection, we examine the speed-up by geometric database reduction technique introduced in Sec. 3.3. The task is mining all RFPs in a database. We compared two algorithms RFPM (BFPM R, in plots) of Sec. 3.1 and G-RFPM (GFPM R, in plots) of Sec. 3.3, without and with geometric database reduction, respectively. Note that the numbers of solutions are same between two algorithms since they solve the same task.

Exp 1b: In Fig. 6, we show the running time and the number of patterns by varying the number n of input points from 20K to 200K points, where $T = 200$. For example, in the case of the input with 200K points, the running times for RFPM and G-RFPM are 61.61 (sec) and 0.96 (sec), respectively, resulting around 70 times speed-up.

Exp 2b: In Fig. 7, we show the running time and the number of patterns by varying the length T of input trajectory database from 0.2K to 1K points, where $n = 200$.

Exp 3b: In Fig. 8, we show the running time and the number of patterns by varying the minlen k of mining parameter from 20 to 100.

Exp 4b: In Fig. 9, we show the running time and the number of patterns by varying the minsup m of mining parameter from 6 to 10.

Exp 5b: In Fig. 11, we show the running time and the number of patterns by varying the maxwidth r of mining parameter from 1 to 5.

Summary of Results B: Overall, in the task of mining RFPs, the modified algorithm G-RFPM with geometric database reduction improves the performance of RFPM more than ten to 70 times on the basic algorithm FPM. In actual running time, G-FPM found all RFPs in less than a second on a PC from an in-

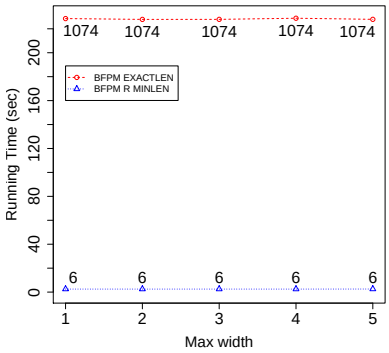

Figure 10: Exp 5a: The running time (and the number of patterns by mark) by algorithm FPM (BFPM) for FPs and RFPM (BFPM R) for RFPs by varying the number of max-width r from 1 to 5.

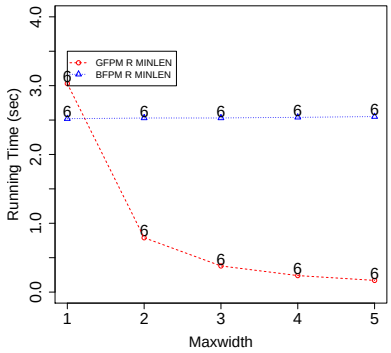

Figure 11: Exp 5b: The running time (and the number of patterns by mark) by algorithm RFPM (BFPM R) and G-RFPM (GFPM R) for RFPs by varying the number of max-width r from 1 to 5.

put database consisting of totally 0.2 million points, which seems enough for actual applications.

Summary of Results A and B: From Fig. 2 of Exp 1a and Fig. 6 of Exp 1b, in the case of the input with 200K points (small dataset), the running times for FPM, RFPM, and G-RFPM are 61.61, 0.96, 0.03 (sec), respectively. From these timing, the speed-ups from FPM to RFPM and RFPM to G-RFPM were around 64 times and 32 times. Overall, we obtained the total speed-up of around 2,000 times from the basic FPM to most advanced G-RFPM.

5 Conclusion

In this paper, we showed empirical study of trajectory mining algorithms, called FPM, from trajectory data. We implemented two of recent theoretical progress of depth-first algorithms for mining flock patterns (Arimura et al., 2013) based on the *pattern-growth approach* (Pei et al., 2004). The experimental results demonstrated that both of extensions, RFPs and geometric database reduction, improve on the speed of FPM by orders of magnitude.

To Scale out flock pattern mining to bigdata in cloud environment, it will be interesting future research to developt efficient implementation of our FPM in massively parallel environment, such as *map-reduce* or *hadoop*, on cloud environments

REFERENCES

Arimura, H., Geng, X., and Uno, T. (2013). Efficient mining of length-maximal flock patterns from large trajectory data. Manuscript, DCS, IST, Hokkaido University. http://www-ikn.ist.hokudai.ac.jp/~arim/papers/flockpattern201303.pdf.

Avis, D. and Fukuda, K. (1993). Reverse search for enumeration. *Discrete Applied Math.*, 65:21–46.

Benkert, M., Gudmundsson, J., Hubner, F., and Wolle, T. (2008). Reporting flock patterns. *Computational Geometry*, 41:111–125.

Giannotti, F., Nanni, M., Pinelli, F., and Pedreschi, D. (2007). Trajectory pattern mining. In *Proc. KDD'07*, pages 330–339. ACM.

Gudmundsson, J. and van Kreveld, M. (2006). Computing longest duration flocks in trajectory data. In *Proc. ACM GIS '06*, pages 35–42. ACM.

Laube, P., van Kreveld, M., and Imfeld, S. (2005). Finding REMO — detecting relative motion patterns in geospatial lifelines. In *Spatial Data Handling*, pages 201–215. Springer.

Pei, J., Han, J., Mortazavi-Asl, B., Wang, J., Pinto, H., Chen, Q., Dayal, U., and Hsu, M.-C. (2004). Mining sequential patterns by pattern-growth: The prefixspan approach. *IEEE TKDE*, 16(11):1424–1440.

Romero, A. O. C. (March 2011). Mining moving flock patterns in large spatio-temporal datasets using a frequent pattern mining approach. Master's thesis, University of Twente.

Uno, T., Asai, T., Uchida, Y., and Arimura, H. (2004). An efficient algorithm for enumerating closed patterns in transaction databases. In *Proc. the 7th Discovery Science (DS'04)*, volume 3245 of *LNCS*, pages 16–31. Springer.

Vieira, M. R., Bakalov, P., and Tsotras, V. J. (2009). On-line discovery of flock patterns in spatio-temporal data. In *Proc. GIS'09*, pages 286–295. ACM.

Zaki, M. J. (2000). Scalable algorithms for association mining. *IEEE Transactions on Knowledge and Data Engineering*, 12(3):372–390.

Zaki, M. J. and Hsiao, C.-J. (2005). Efficient algorithms for mining closed itemsets and their lattice structure. *IEEE Transactions on Knowledge and Data Engineering*, 17(4):462–478.