An Improved Base Algorithm for Online Discovery of Flock Patterns in Trajectories

Pedro Sena Tanaka[1], Marcos R. Vieira[2], Daniel S. Kaster[1]

[1] Department of Computer Science – University of Londrina, Brazil
pedro.stanaka@gmail.com, dskaster@uel.br
[2] Big Data Lab – Hitachi America, Ltd. – R&D, USA
marcos.vieira@hal.hitachi.com

Abstract.

The high availability, cost, and usage of location-aware devices have increased the interest in the research of spatiotemporal patterns. The main goal in studying such patterns is to discover spatial relationships over time between moving objects. Recent articles have proposed a wide variety of such patterns, among them is the flock pattern. This pattern is defined as a set of moving objects with minimum size that stay together within a maximum distance for a continuous period of time. Typical application examples are monitoring and surveillance, which rely on efficiently identifying groups of suspicious people/vehicles in large spatiotemporal streaming data. Previous works proposed polynomial-time algorithms to the flock pattern problem with fixed time duration. In this article, we propose a new online method, called PSI, which is an improved base method to discover flock patterns that applies computational geometry techniques (e.g., plane sweeping) along with binary signatures and inverted indexes. In summary, plane sweeping speeds up the detection of candidate groups in a particular timestamp, binary signatures allow saving costly set intersection operations, and inverted indexes are employed to quickly compare candidate disks across timestamps. Using a variety of real-world datasets and a large synthetic one we show that our proposed methods are efficient compared to state-of-the-art solution. In our experimental evaluation our proposed method achieved up to 69 times speedup compared to the previous solution.

Categories and Subject Descriptors: H.2 [**Database applications**]: Data mining; Spatial databases and GIS

Keywords: moving objects, spatio-temporal patterns, flock pattern, plane sweep

1. INTRODUCTION

In the last decade, location-aware devices, such as GPS, RFID tags and smartphones, have become ubiquitous as we observe large technology improvements as well as drops in prices over these last years. This popularization in association with the ubiquitous use of location services, like Swarm[1] and Waze[2], are generating a continuous and fast paced growth of datasets in form of trajectories. A trajectory is a sequence of recorded locations over time for a moving object. An interesting aspect of trajectories is not only that the availability of large historical spatiotemporal data is increasing in recent years, but also the rapid expansion of online services providing spatiotemporal streaming data. One example of such services is AccuTracking[3], which helps large retailer and shipping companies (e.g., United States Postal Service (USPS)) to online track large vehicle fleets around the world. Other examples are applications that provide location-based services to end-users, like Foursquare[4]

[1] www.swarmapp.com
[2] www.waze.com
[3] www.accutracking.com
[4] www.foursquare.com

This work has been supported by a CAPES scholarship.

and Waze, which both have millions of users reporting their location activities over time. Therefore, there is an increasing need to develop efficient algorithms that can analyze those large repositories of information.

Discovering patterns in large volumes of spatiotemporal streaming data is a very challenging task. This is due to the fact that spatiotemporal patterns are generally defined as how the spatial relationships among moving objects evolve over time. Although this analysis is computationally expensive, it may reveal common, but important, behaviors involving the observed moving objects in a period of time (e.g., migration patterns of wild animals, traffic patterns in road networks and suspicious activities in urban areas).

Over the last few years, several spatiotemporal patterns were proposed, each of which describing a different kind of behavior. Examples of such patterns include density-based patterns, such as group [Wang et al. 2006; Li et al. 2013], swarm [Li et al. 2010] and convoy [Jeung et al. 2008] patterns, and distance-based patterns, such as flock pattern [Benkert et al. 2008; Gudmundsson and van Kreveld 2006; Vieira et al. 2009; Romero 2011; Arimura et al. 2014], which is the subject of our work. A flock is defined by a set of minimum number of objects that are spatially "close together" for a time duration. A few proposals on flock pattern focus on discovering *maximal length* (or *duration*) flocks (e.g., [Gudmundsson and van Kreveld 2006; Arimura et al. 2014]), which are the minimum sets of μ objects moving enclosed by a disk with diameter ϵ for the *longest timespan*. Other works provide solutions to find flocks with *fixed time duration*, i.e., same as above but for *at least* δ time instants. Methods to discover flocks can be classified as: **(a) offline methods**, e.g., [Al-Naymat et al. 2007; Romero 2011], which require the entire data set be available beforehand in order to map/compute statistics; and **(b) online methods** that report results as soon as they are discovered, thus they can deal with spatiotemporal streaming data. The *online* detection of flock pattern has a wide range of applications in different domains, ranging from real-time monitoring of suspicious activities to observing animal migration and behavior. In order to fast act in response to observed activities in a monitored environment, we need algorithms that quickly detect flock patterns from applications that continuously consume large volumes of data from location-aware sensors. For example, a highway patrol can assemble an emergency task force to intercept and punish the participants of a street race after detecting a flock of vehicles moving in high speed. The state-of-the-art work on reporting flock patterns with fixed time duration is [Vieira et al. 2009], which presents a baseline algorithm, called Basic Flock Evaluation (BFE), and also several heuristics to improve the baseline algorithm.

In this article we present a novel method named PSI – ***P****lane sweeping,* ***S****ignatures and* ***I****ndexes* – for online discovery of fixed time duration flock patterns. PSI significantly extends the BFE algorithm by employing three new techniques: **(1) plane sweeping** to quickly detect candidate flock groups in a particular time instant; **(2) binary signatures** to allow reducing the number of set comparisons by pruning subsets of candidates in a given time instance; and **(3) inverted index** to quickly check when a candidate disk in the current timestamp follows any partial flock in a previous time instant. To the best of our knowledge, our work is pioneer in applying the above techniques to the online flock pattern problem.

This article extends the previous version [Tanaka et al. 2015] by including further details regarding the main aspects of our proposal and an improved evaluation of its behavior. This new evaluation is enriched with additional experiments to test our method against a large synthetic dataset to check its scalability and with new analysis of how the spatial distribution of the data sets affects its performance. Such analysis in light of details in implementation level of our method allows a deeper understanding of our results as well as it helps to elucidate in which situations our method's underlying techniques may be successfully employed to discover other spatiotemporal patterns. We show in our experiments that our proposed PSI algorithm consistently outperformed the baseline method BFE, regarding both real-world and synthetic data sets: it achieved up to **69x speedup** in the experiments. An important observation is that our method could leverage the heuristics used to extend BFE [Vieira et al. 2009]

to potentially achieve even higher performance improvements. We are exploring these extensions as part of our future work.

The remainder of this article is organized as follows: Section 2 introduces the flock pattern problem and discusses related work. Section 3 presents the BFE algorithm and the plane sweeping, which are foundations of our work. Section 4 describes our new proposed PSI algorithm to find flock patterns, which combines plane sweeping technique, binary signatures and inverted index; Section 5 presents extensive performance evaluation of our proposed algorithms and previous work along with a thorough analysis of data skewness impact. Section 6 concludes the article and summarizes future work.

2. PROBLEM STATEMENT AND RELATED WORK

2.1 Flock Pattern Problem

Common to all variations in the literature, the flock pattern is the problem of identifying all sets of trajectories that stay "close together" during a time period. This pattern enforces that there must be no pair of elements in a flock which are "farther" from each other than a given distance threshold during the flock's lifespan. Therefore, it is known as a disk-based spatiotemporal pattern. In this article we employ the definition for the flock pattern introduced by Vieira et al. [2009], in which the lifespan of the pattern is fixed. The property of closeness can be depicted by a disk of a given diameter ϵ that covers all objects belonging to a flock in all timestamps during a period. The pattern is formally defined as follows.

Definition 2.1 *Flock pattern.* Let $\mathcal{T}$ be a set of trajectories, $\mu > 1$ be a minimum number of trajectories ($\mu \in \mathbb{N}$), $\epsilon > 0$ be a distance threshold regarding a distance metric d ($\epsilon \in \mathbb{R}^+$) and $\delta > 1$ be a minimum number of time instances ($\delta \in \mathbb{N}$). A **Flock**(μ, ϵ, δ) pattern reports a set $\mathcal{F}$ containing all the flocks f_k, which are sets of maximal size such that: for each $f_k \in \mathcal{F}$, the number of trajectories is greater than or equal to μ and there exist δ consecutive timestamps $t_j, \ldots, t_{j+\delta-1}$ in which there is a disk of center $c_k^{t_i}$ and diameter ϵ that covers all trajectories of $f_k^{t_i}$, which is the flock f_k in time t_i, $j \leq i \leq j + \delta - 1$.

Figure 1 shows an example of flock pattern with two flocks, each containing three moving objects. One flock is formed by trajectories $\{T_1, T_2, T_3\}$ covered by disks labeled according to their centers $\{c_1^1, c_1^2, c_1^3\}$, where each c_i^j is the center of a disk of the flock i in the timestamp j. The second flock is composed by trajectories $\{T_4, T_5, T_6\}$ covered by disks centered on $\{c_2^2, c_2^3, c_2^4\}$. It is important to note two properties in flock pattern: (1) in order to cover all trajectories, the center of the disks can freely move in the spatial domain; and (2) centers do not necessarily coincide with an object location in a specific time instant. These two properties make the problem of discovering flock patterns very challenging, as there exist an infinite number of spatial positions to place the disk center at each time instance.

2.2 Related Work

Since the work that first presented the flock pattern problem [Laube and Imfeld 2002], several others have followed addressing variations of the problem. Gudmundsson et al. [2004] define the flock pattern problem by a set of trajectories moving in the same direction for a *single timestamp*. Clearly, this definition cannot find more complex and interesting flock patterns that evolve over time. Other variations have followed by introducing a pattern that happen during a *time window*, which can be either of *fixed size* (as in Definition 2.1) or *maximal size*. Existing algorithms can also be classified in *offline* methods, which require the entire historical data set to be loaded to work, and *online* methods that are able to work both with historical and streaming data.

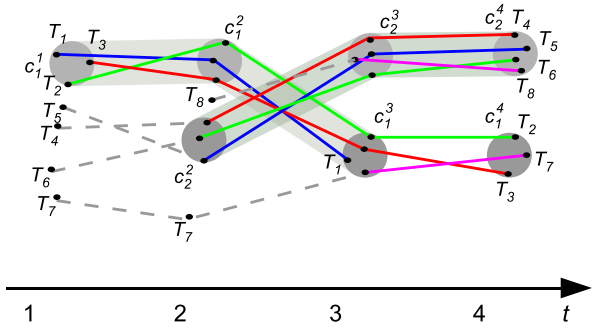


Fig. 1. Example of flock pattern: flocks $\{T_1, T_2, T_3\}$ and $\{T_4, T_5, T_6\}$ from timestamps 2 to 4 and 3 to 5, respectively.

Several different solutions have been proposed to discover flock patterns. The proposed solutions range from mapping of trajectories to a high dimensional space (e.g. [Benkert et al. 2008]), passing through pure spatiotemporal approaches (e.g. [Vieira et al. 2009]), to mapping of trajectories to transactions([Romero 2011]). Regarding mapping-based approaches, Benkert et al. [2008] proposed to first map polylines representing trajectories to points in a high dimensional space, and then to search for fixed size flocks in the mapped space. Similarly, Al-Naymat et al. [2007] employ dimensionality reduction technique (Random Projection [Bingham and Mannila 2001]) to report flocks. The main reason for the mapping is to avoid the need to keep track of all candidate disks in different time instances. Nonetheless, due to the high dimensional space mapping the data suffers from degeneration making the answers reported by the methods to be approximations. Other methods that use this kind of technique also have this limitation (e.g., [Gudmundsson et al. 2008; Gudmundsson and van Kreveld 2006; Benkert et al. 2006; 2008]).

A problem slightly different from Definition 2.1 is the *Maximal Duration (or Length) Flock Pattern*, which does not have a defined time window (i.e., no definition of δ parameter). In [Turdukulov et al. 2014; Romero 2011; Rosero and Romero 2014] it was proposed the LCM_Flock algorithm to report maximal duration flocks based on a frequent pattern mining approach. This approach transforms the input historical trajectory data set to a transactional one, which is then used in the LCM algorithm (Linear time Closed itemset Miner) [Uno et al. 2005]. A second class of algorithm is Flock Pattern Miner (FPM) [Arimura et al. 2014; Geng et al. 2014], which was the first work to address the problem of enumerating the maximal duration flocks in polynomial delay. This approach uses a depth-first search (DFS) to check all time instants for a particular trajectory to find all-maximal duration flocks.

All aforementioned methods were designed to work offline (i.e. they are limited to historical data sets). The landmark online approach was presented by Vieira et al. [2009] as it was the first to propose algorithms for detecting flock patterns in streaming data using only spatiotemporal search techniques. Vieira et al. proposed a set of online algorithms, which are considered the state-of-art solution for reporting flock patterns with fixed time duration. All these algorithms are based in a common one, called BFE (Basic Flock Evaluation). In this article we propose a new online base algorithm to report fixed size flock patterns that employ the premises of BFE improved with the plane sweeping technique, binary signatures and inverted indexes. Our proposed method consistently outperforms BFE, as shown in Section 5, while it can handle both historical and streaming spatiotemporal data.

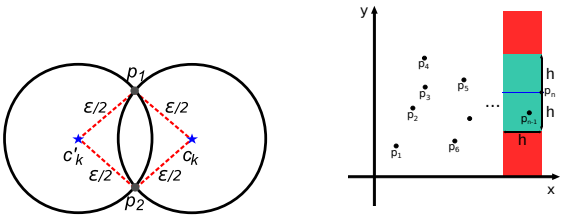


Fig. 2. (a) Disks for $\{p_1, p_2\}$ such that $d(p_1, p_2) \leq \epsilon$. (b) Application of plane sweeping technique to find closest pair.

3. FUNDAMENTAL CONCEPTS

3.1 Basic Flock Evaluation Algorithm

When searching for flock patterns there may have infinite possible spatial locations to place disk centers in a time instant. This makes the task of finding candidate disks very challenging. In order to overcome this problem, Vieira et al. [2009] introduced a theorem that reduces the search space to finding candidate flock disks as follows.

THEOREM 3.1. *If for a given time instance t_i there exists a point in the space $c_k^{t_i}$ such that, $\forall T_j \in f_k$, $d(p_j^{t_i}, c_k^{t_i}) \leqslant \epsilon/2$, then there exists another point in the space $c_k'^{t_i}$ such that $d(p_j^{t_i}, c_k'^{t_i}) \leqslant \epsilon/2$ and there are at least trajectories $T_a \in f_k$ and $T_b \in f_k$ such that $\forall T_j \in \{T_a, T_b\}, d(p_j^{t_i}, c_k'^{t_i}) = \epsilon/2$.*

Theorem 3.1 states that if there is a disk with center $c_k^{t_i}$ and diameter ϵ that covers all trajectories in a flock f_k, then there is another disk with different center $c_k'^{t_i}$ that also covers all the trajectories of f_k. This theorem affects considerably the search space for flock patterns as it limits the locations inside the spatial domain where it is necessary to search for flocks. For a data set of $|\mathcal{T}|$ trajectories, there are $|\mathcal{T}|^2$ possible combinations of pairs of points in a time instant. For each pair there are (at most) two disks with radius $\epsilon/2$ that have these two points in their circumferences (Figure 2(a)).

Based on Theorem 3.1, Vieira et al. [2009] proposed the Basic Flock Evaluation (BFE) algorithm. This algorithm is the simplest among all five presented in that paper. The other four algorithms enhance the BFE algorithm with different heuristics to, potentially, speed up their running time. In summary, the BFE algorithm operates in the following three steps:

(1) Find flock disks: this step generates a set of candidate disks, where each disk defines a flock pattern. For each time instant, this step searches for pairs of points that qualify to generate disks. Since this step has a running time of $O(n^2)$ (i.e., all possible pairs of point combinations at any given time instance), BFE employs a grid-based index to speed up the candidate disks and flock detection. The grid-based index is based on fixed-size cells of ϵ distance. Each object in the data set is mapped to one cell in the index based on the objects' locations, thus the index size and performance depend on the spatial distribution of the data set. In the search phase, each cell $g_{x,y}$ is evaluated to find pairs of points that are within ϵ distant to each other. To find pairs of points, each point in cell $g_{x,y}$ is matched to every other point located in cell $g_{x,y}$, as well as points in the nine adjacent cells to cell $g_{x,y}$ (i.e., cells that may have points within ϵ distance to the point being evaluated). Each pair of points within ϵ distance is used to compute (at most) two disks whose circumferences intersect exactly in the pair of points. Afterwards, the algorithm simply counts the number of points within the disks, and then filtering out disks with less than μ entities;

(2) Filter candidate disks: since the first step finds *all* flock disks in a particular time instant, the second step is to keep only disks containing *maximal* sets of trajectories. A naive approach to select *maximal* sets is to compare every possible pair of disks. However, this approach has running

time quadratic to the number of disks, without considering computational expensive set intersection operations. In order to avoid set intersection operations, BFE only compares pair of disks that intersects each other in the space domain;

(3) Set join between consecutive timestamps: the result from the second step is a set of flocks for one time instant. Thus, in order to find pattern flock with a time duration, the result set has to be "merged" with results from previous time instant. Different from the previous step, joining sets between consecutive timestamps cannot employ topological relations to reduce the search space. Therefore, this last step is computationally expensive when joining very large sets. After the joining phase, the results that could not be joined are discarded, and the ones with lifespan of δ are reported as flocks. The current "active" flocks (valid until the current timestamp) are maintained and further used in the next iteration (step 1 for the next timestamp).

3.2 Plane Sweeping Technique

The plane sweep technique was first introduced to detect intersection between line segments in the plane [Shamos and Hoey 1976]. Since then, this technique has proven to be important to reduce computational complexity in a large variety of problems, mainly in the computational geometry area. The main idea of plane sweeping is to use a "sweeping line" (generally vertical) throughout the plane, i.e., scanning the plane from left-to-right in x-axis. The "sweeping" process continues until a condition is met (e.g., line intersects with a point). Whenever this event happens, then geometric operations are performed on the points that prompted the event. Note that the operations are performed on a reduced set of points closer to the sweeping line. This process of sweeping ends when all data set points are swept by the line.

In order to further understand the technique consider the problem of finding the pair of points that is closest. This problem can be solved with a naive approach in $O(N^2)$, but with the help of plane sweeping it can be solved in $O(N.\log(N))$. The following algorithm was taken from [Hinrichs et al. 1988]. To identify the closest pair it is used a band instead of a line (see Figure 2(b)). Suppose that the algorithm already processed $N-1$ points of the input (ordered by x-axis) and the shortest distance between consecutive points found so far is h. The algorithm processes the last point p_n and try to find a point that is closer to it than h. It maintains a set of all the points processed that are within a range of h in x-axis of the point p_n (red rectangle in the Figure 2(a)). Every time a new point is processed, it is added to the set, and the set is destroyed every time we jump to other point or the value of h changes. This set is ordered by the y coordinate and a balanced binary tree can be used to maintain this set, and accounts for the $\log(N)$ factor. The algorithm only considers the points in the range $p_n.y - h$ to $p_n.y + h$ (those inside the blue rectangle in Figure 2(b)). In order to check if a point is at a distance less than h from the point p_n, it just checks those points that are elements of the balanced tree. It follows that the search for each of the N points the complexity of the search is $\log(N)$, hence the algorithm has the total complexity of $O(N.\log(N))$. In our work we use plane sweeping technique in a similar fashion to the closest-pair problem, as detailed in the next section.

4. METHOD PSI: PLANE SWEEPING, SIGNATURES AND INDEXES

This section presents our proposed method PSI, which stands for ***P**lane sweeping, **S**ignatures and **I**ndexes* for online discovery of fixed time duration flock patterns. It employs the plane sweeping technique and binary signatures to improve the search for disks on a specific timestamp. PSI also relies on inverted indexes in the spatiotemporal join phase to reduce the number of set intersection operations.

Algorithm 1: Find candidate disks with plane sweeping technique

Input: $\mathcal{T}[t_i]$: positions in timestamp t_i, sorted by x-axis values, ϵ: flock diameter, μ: minimum size of flock
Output: $\mathcal{C}$: candidate disks for timestamp t_i, $\mathcal{B}$: active boxes in timestamp t_i

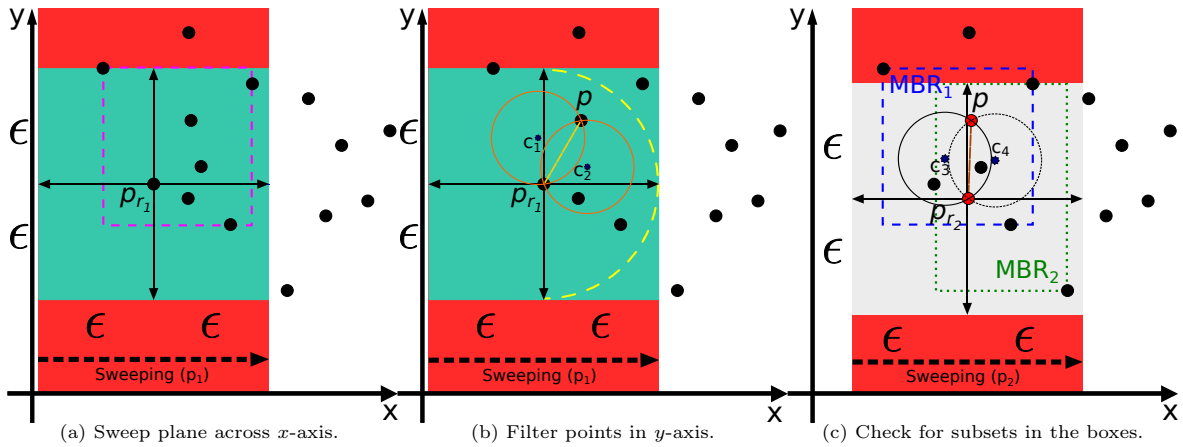
1 $\mathcal{C} \leftarrow \emptyset$, $\mathcal{B} \leftarrow \emptyset$
2 **foreach** $p_r \in \mathcal{T}[t_i]$ **do** // analyze elements in increasing x-values
3 $\quad \mathcal{P} \leftarrow \emptyset$ // list of elements of current box defined by p_r
4 $\quad$ **foreach** $p_s \in \mathcal{T}[t_i] : \mid p_s.x - p_r.x \mid \leqslant \epsilon$ **do** // test only elements inside 2ϵ x-band
5 $\quad\quad$ **if** $|p_s.y - p_r.y| \leqslant \epsilon$ **then** // check if p_s is inside 2ϵ y-band
6 $\quad\quad\quad \mathcal{P} \leftarrow \mathcal{P} \cup p_s$ // add element p_s to box
7 $\quad$ **foreach** $p \in \mathcal{P} : p.x \geqslant p_r.x$ **do** (// elements inside right half of box
8 $\quad\quad$ **if** $dist(p_r, p) \leqslant \epsilon$ **then** // calculate pair distance
9 $\quad\quad\quad$ let $\{c_1, c_2\}$ be disks defined by $\{p_r, p\}$ and radius $\epsilon/2$
10 $\quad\quad\quad$ **foreach** $c \in \{c_1, c_2\}$ **do**
11 $\quad\quad\quad\quad$ **if** $|c \cap \mathcal{P}| \geqslant \mu$ **then** // check the number of entries in disk
12 $\quad\quad\quad\quad\quad \mathcal{C} \leftarrow \mathcal{C} \cup c$ // add c to candidate disks
13 $\quad\quad\quad\quad\quad \mathcal{B} \leftarrow \mathcal{B} \cup box(p_r)$ // add box to active boxes
14 **return** $\mathcal{C}, \mathcal{B}$

4.1 Plane Sweep-based Disk Detection

As previously described, the BFE algorithm (and its extensions) first constructs a grid-based index and then generates candidate disks for each timestamp. This process of building and searching the index can be time consuming. Thus, to reduce this cost we propose a new approach based on plane sweeping to find flock disks without index construction. Our proposed approach is described in Algorithm 1.

Algorithm 1 first sweeps the plane (from left to right in x-axis) using a band of size 2ϵ along the x-axis centered at a point p_r (red box in Figure 3(a)). The algorithm selects all the points inside the band that are in the range $[p_r.y - \epsilon, p_r.y + \epsilon]$ (blue box in Figure 3(a)). These steps are presented in lines 4–6 of Algorithm 1.

After selecting the points in the $2\epsilon \times 2\epsilon$ box defined by p_r, we then check for pairs of points that qualify for new flock disks (refer to Theorem 3.1). Thus, we generate disks defined by p_r and any point p inside the right half of box such that the distance between p_r and p is at most ϵ (yellow-dashed semicircle in Figure 3(b)). Points in the left half of box were checked in previous steps. If a candidate disk contains at least μ entities inside it, then the underlying entity set is reported as a candidate set and the box is set active in the timestamp. Every active box is represented through the Minimum Bounding Rectangle (MBR) enclosing its elements (dashed/dotted rectangles in figures). These last steps are represented by lines 7–13 of Algorithm 1.



(a) Sweep plane across x-axis. (b) Filter points in y-axis. (c) Check for subsets in the boxes.

Fig. 3. Steps needed to find disks in one timestamp.

Algorithm 2: Filter out disks which are subsets

```
1  Algorithm FilterCandidates(B)
       Input: B: active boxes of timestamp t_i, sorted by x-axis values
       Output: C: final set of disks for timestamp t_i
2      C ← ∅
3      for j ← 0 to j ⩽ |B| do
4          for k ← j + 1 to k ⩽ |B| do
5              if IntersectsWith(B[j], B[k]) then
6                  foreach c ∈ B[j].disks do
7                      C ← InsertDisk (C, c)
8              else // No intersection.
9                  break
10 Procedure InsertDisk(C, c)
       Input: C: set of disks, c: new disk
11     foreach d ∈ C do
12         if c.sign ∧ d.sign = c.sign && dist(c, d) ⩽ ε then // c can be a subset of d
13             if d ∩ c = c then // Remove chance of false-positive
14                 return C // No need to insert c
15         else if c.sign ∧ d.sign = d.sign then // d can be a subset of c
16             if c ∩ d = d then // Remove chance of false-positive
17                 C ← C \ d // Remove d
18     return C ∪ c
```

4.2 Signature-based Candidate Set Filtering

The next step after detecting the candidate sets of a particular timestamp is to check for disks that are subsets or supersets. Similar to BFE algorithm, we are interested in finding only *maximal* sets of flocks. The BFE algorithm uses only spatial properties of disks to accelerate the maximal set detection. Here we employ a different approach: we use spatial properties of boxes (before using properties of disks as in BFE) and binary signatures to prune subsets without executing (expensive) set intersection operations.

The process of filtering out candidate sets is shown in Algorithm 2. The spatial relationship between boxes is given by their MBRs. Note that although the box has dimensions of $2\epsilon \times 2\epsilon$, the MBR of the set of objects is usually smaller (see Figure 3). Each box, as well, has information about all candidate sets that belong to it. The step of filtering checks boxes that hold at least one candidate set. Algorithm 2 begins by iterating on each active box (i.e., a box that has a candidate set) in the *current* timestamp, and checks if there is intersection between the MBR of a box and the MBRs of boxes near it. In this case, it is necessary to check whether there is duplicate or subsets between these boxes (Figure 3(c)).

Before performing the set intersection operation, we propose to apply a second filtering step using binary signatures (lines 10–18 of Algorithm 2). As entities get inserted in a candidate set, a set of hash functions are used to generate a signature for it. The idea of using binary signatures to accelerate subset operations is not new. In fact, it has been used to solve the substring problem [Harrison 1971] and is common in information retrieval [Faloutsos and Christodoulakis 1984] to the fast retrieval of documents (known as the technique of signature files). In this work a set of Bloom filters is used to represent subsets of a universe, and then these filters, which essentially are binary vectors, are used to check for subsets amongst the sets. It is well-known that Bloom filters can generate false-positives, but most importantly, no false-negatives. This is the reason that, after we check one disk is subset of another using binary signatures, we still need to verify using set intersection whether the result is a false-negative (lines 13 and 16 of Algorithm 2).

Set signatures are generated as follows: Starting from a zero signature (e.g., 0000 0000 0000 0000), the resulting signature is computed by executing the hash functions in sequence for every identifier in

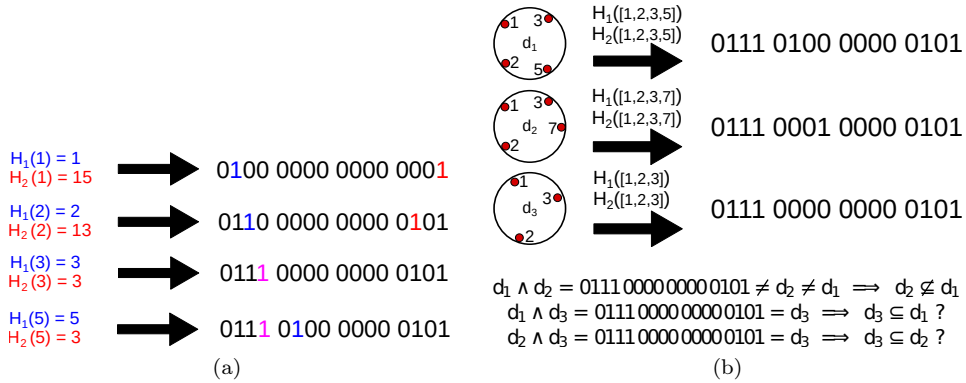


Fig. 4. (a) Process of generating binary signatures. (b) Subset checking using signatures.

the set. Each hash function maps an object identifier to a position inside the signature (bucket) and sets a number of 1-bit according to the given identifier. Figure 4(a) shows the step-by-step signature generation for the set $\{1, 2, 3, 5\}$ highlighting the bits set by each hash function regarding every object identifier. In the figure, H_1 and H_2 refer, respectively, to the SpookyHash[5] and MurMurHash[6], which are fast hashes implementations and with good non-linearity (measured by avalanche criterion[7]). Note that some collisions may happen between the hash functions (bits represented in purple in Figure 4(a)). Therefore, the ideal size of the signatures depends on the cardinality of the sets. However, we recommend to be up to 64 bits, since we have run tests (using a 64-bit architecture, which is the currently the "standard") that showed that **AND** bitwise operations of signatures greater than this size may suffer a performance drop.

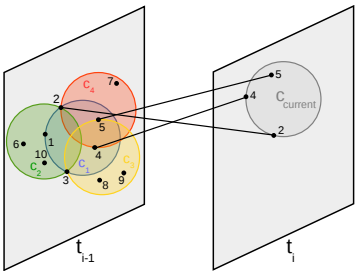
Figure 4(b) illustrates the process of filtering through binary signatures. The idea is to avoid performing a set intersection operation to determine if a disk is subset/superset of another disk in case this is surely false. In order to achieve this, we apply an **AND** bitwise operation between the two signatures from the disks. If the result of the operation is equal to one of the operands, and then this operand may be a subset of the other. Otherwise, we can surely say that no disk is a superset of the other. In Figure 4(b), the set intersection between d_1 and d_2 is avoided as none of these sets is a subset of the other according to the signature checking. However, as this approach is subject to false positives, it is necessary to perform a set intersection operation as a post-processing step regarding d_1, d_3 and d_2, d_3. Nevertheless, this step should eliminate many false-negatives depending on the chosen hash functions. The primitives of performing subset queries using binary signatures are described in details in [Goel and Gupta 2010].

4.3 Inverted Index-based Join Between Sets of Consecutive Timestamps

After finding all disks for a given time instant, we then need to join them with others from the previous timestamp. A straightforward way to join disks is to process all disks from one timestamp with the other timestamp, and then check for the joining condition (i.e., if two disks have at least μ objects in common). However, this process is computationally costly since we have to perform, for each timestamp, set intersection operations for all candidates of a timestamp with all the maintained flocks of the previous one. Instead, here we propose the use of inverted indexes to speed up the step of joining disks across two consecutive timestamps.

Inverted index is a well-known method employed to index documents and then efficiently search for terms in the index [Zobel and Moffat 2006]. Usually, an inverted index has a list of keys that are the

[5] `burtleburtle.net/bob/hash/spooky.html`
[6] Murmurhash 2.0: `sites.google.com/site/murmurhash`
[7] `floodyberry.com/noncryptohashzoo`



Inverted index of disks of t_{i-1}	
OID	Disks
1	{c1, c2}
2	{c1, c2, c4}
3	{c1, c2, c3}
4	{c1, c3, c4}
5	{c1, c3, c4}
...	

Current disk: {2,4,5}
List2 ⋂ List4 ⋂ List5 = {c1,c4}

Fig. 5. Joining disks/sets between consecutive timestamps using an inverted index ($\mu = 3$).

Dataset	Positions	Objects	μ	ϵ	δ
Cars	134,264	183	4, 6, ..., 20 **[5]**	0.4, 0.5, ..., 1.1 **[0.6]**	4, 6, ..., 20 **[10]**
Buses	66,096	145	4, 6, ..., 20 **[5]**	0.3, 0.4, ..., 1.0 **[0.7]**	4, 6, ..., 20 **[10]**
Trucks	112,203	276	4, 6, ..., 20 **[5]**	0.7, 0.8, ..., 1.4 **[0.9]**	4, 6, ..., 20 **[10]**
Caribous	15,796	43	2, 3, ..., 10 **[5]**	0.1, 0.2, ..., 0.8 **[0,7]**	4, 6, ..., 20 **[10]**
SG	2,548,084	50,000	4, 6, ..., 20 **[5]**	2.2, 2.6, ..., 5.0 **[3.0]**	4, 6, ..., 20 **[10]**

Table I. Number of objects, amount of positions and tested values for each parameter of the pattern.

common terms appearing in all the documents. Each item, in turn, has a set of document identifiers where the term represented by that item appeared. In our particular problem, an inverted index is employed to search disks from previous timestamp (t_{i-1}) that have at least μ object in common with the disk being processed of current timestamp (t_i) (see Figure 5). When a current disk from t_i is processed, we use the set of OIDs that belongs to the disk as the query elements to the inverted index. The query condition is that a document (or disk) is returned if it has at least μ terms (or objects) in common with the query set. In the figure, disks c_1 and c_4 compose the query answer.

Now, suppose we have $n \in \mathbb{N}$ disks in t_{i-1} and $m \in \mathbb{N}$ in t_i, and the average number of objects in each disk is $l; l \in \mathbb{N}, l \geqslant \mu$. If we have to compare all disks from t_i with the ones in t_{i-1}, then we have a time complexity of $O(n.m.l)$. However, our approach can drastically reduce this time complexity in most cases, as shown in the experimental evaluation.

5. EXPERIMENTAL EVALUATION

In order to verify the efficiency of our proposed PSI method, we performed an extensive experimental evaluation with several real-world spatiotemporal datasets and one synthetic dataset. The datasets employed in our experiments are: *Trucks*, *Buses*, *Cars*, *Caribous* and SG. *Trucks* dataset has 112,203 GPS locations generated by 276 moving trucks, while *Buses* has 66,096 locations generated by 145 buses, both of them were collected in the metropolitan area of Athens, Greece[8]. *Cars* contains 134,264 locations collected from 183 private cars moving in Copenhagen, Denmark[9]. *Caribous* is generated from the migration movements of 43 caribous in northwest states of Canada, with a total of 15,796 locations. Lastly, to test the scalability of the methods in this article we used the synthetic dataset *SG* which has 2,548,084 positions generated by the movement simulation of 50 thousand vehicles in Singapore road network. All datasets are as provided by their owners, except by some conversions between coordinate systems. Some of the datasets have missing data, i.e., one entity appears in timestamp t disappears in t+1 and reappears in t+2. Whenever this situation happens flock patterns will decompose. Table I presents a summary of the datasets and the tested ranges for the flock pattern parameters regarding each one (defaults inside brackets).

[8] chorochronos.datastories.org/
[9] www.daisy.aau.dk

We used the BFE as baseline approach, which is the base implementation described in [Vieira et al. 2009]. As PSI improves BFE using three techniques, we tested different versions of BFE and PSI to evaluate which technique works better in which conditions. We also compare our proposed approach with an online version based on the LCM_Flock algorithm [Romero 2011]. The online LCM_Flock algorithm partitions the data domain in the time dimension in δ-window slices. Each of these slices is then handled as the input dataset for the original (offline) LCM_Flock algorithm, which reports the answers found in each window. In summary, we performed experiments with the following six methods:

(1) **BFE:** The original BFE algorithm;
(2) **LCM:** The online version of the LCM_Flock algorithm;
(3) **BFI:** BFE with inverted index to join disks;
(4) **PSW:** BFE with plane sweeping method (grid-based index is absented);
(5) **PSB:** PSW (as described above) with binary signatures to accelerate the step of finding subsets in a timestamp;
(6) **PSI:** PSB with inverted index to prune candidates across subsequent timestamps.

We implemented and tested all methods in C++ with GCC v4.9, using an Intel Core 2 Quad@2.83GHz CPU, 4GB@1333MHz RAM and 500GB@3Gb/s HDD.

5.1 Overall Analysis

Figure 5.1 shows the experimental results regarding all datasets varying values for the parameters μ, ϵ and δ (according to Table I). All plots show the total running time (in seconds) needed to evaluate the *entire* dataset. Plots for *Caribous* and *SG* are in logarithmic scale. Additionally, the charts for the *SG* dataset do not include the online LCM_Flock algorithm due to disk thrashing. This algorithm keeps in memory all candidates generated in the whole δ-window, which made it unfeasible to be executed using the dataset *SG* in our test machine as the number of candidate disks generated in a timestamp of this dataset is up to 1.26×10^6. The algorithm demanded excessive swap usage, therefore we did not make a direct wall clock time comparison to avoid being biased by the test machine.

In general, we achieved better results with the techniques employed by the PSI method. This is due the fact the BFE method needs to build a grid-based index before actually starting the search for disks. Another observation from the results is that as we increase μ, or decrease δ, the running time required to report flock patterns decreases. This behavior is expected, since the parameters of flock patterns become more selective and, thus, the bookkeeping costs related to maintain candidate sets reduce. However, as ϵ decreases, the running time required by PSI variants always decrease while the running time of BFE and BFI decreases until an optimal value and regrows below it (see figures 5.1(b) and 5.1(e)). This behavior is explained by the fact the grid-based index is highly dependent on the spatial data distribution. BFE is less efficient when the data is skewed concerning the given ϵ. In most situations, the online LCM_Flock algorithm was significantly slower than our method. It was the best only when the data is dense regarding a value of ϵ (see figures 5.1(b) and 5.1(h)). This occurs due to the way LCM_Flock performs the join of candidate disks in consecutive timestamps. In this algorithm the set intersection operations, which are very expensive, are not necessary. Hence, when the number of disks begin to increase excessively in each timestamp the underlying itemset miner algorithm LCM [Uno et al. 2005] begins to counterbalance the combined costs of translation of the trajectories to transactions and of bookkeeping all the candidate disks that it requires.

In particular, in the datasets *Caribous* and *SG* we see that when increasing μ values, the use of the plane sweeping technique has a bigger impact on performance than in the other datasets, being up to more than two orders of magnitude faster than using the grid index. This is due to the relatively reduced numbers of flocks that were found in those datasets. *Caribous* has only 43 objects whose are

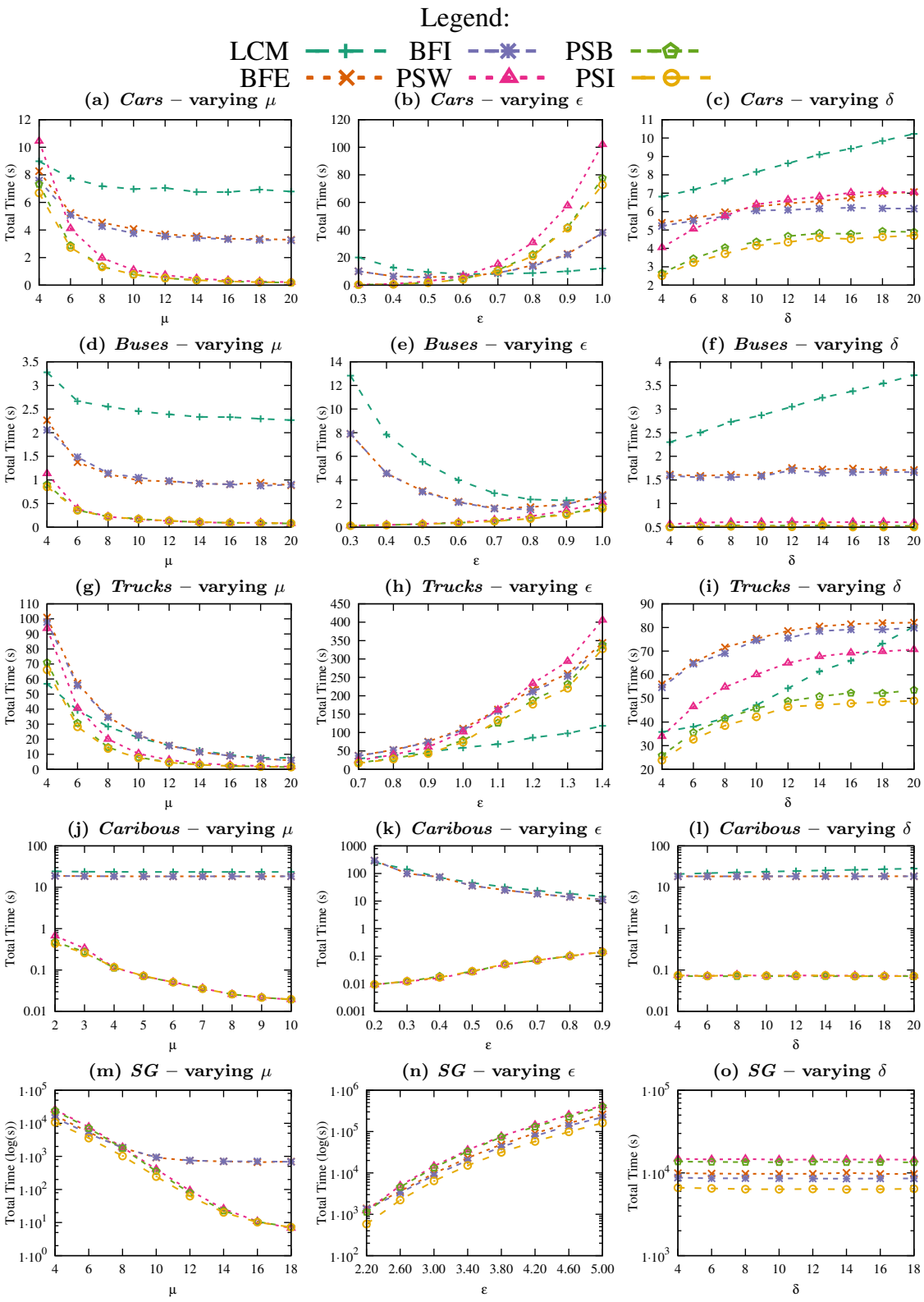


Fig. 6. Experiments with *Cars* (1st row), *Buses* (2nd row), *Trucks* (3rd row), *Caribous* (4th row) and *SG* (5th row) datasets when varying μ (cardinality - 1st column), ϵ (distance diameter - 2nd column) and δ (time duration - 3rd column).

dataset	Defaults	μ		ϵ		δ	
	(μ, ϵ, δ)	**Min**	**Max**	**Min**	**Max**	**Min**	**Max**
Cars	1295	2598	0	203	5940	3906	319
Buses	319	787	20	36	1666	1247	34
Trucks	4926	5708	104	3741	15607	9934	756
Caribous	0	5063	0	0	150	20	0
SG	144	975	0	53	741	270	69

Table II. Total number of answers for each dataset regarding parameter's defaults and limits.

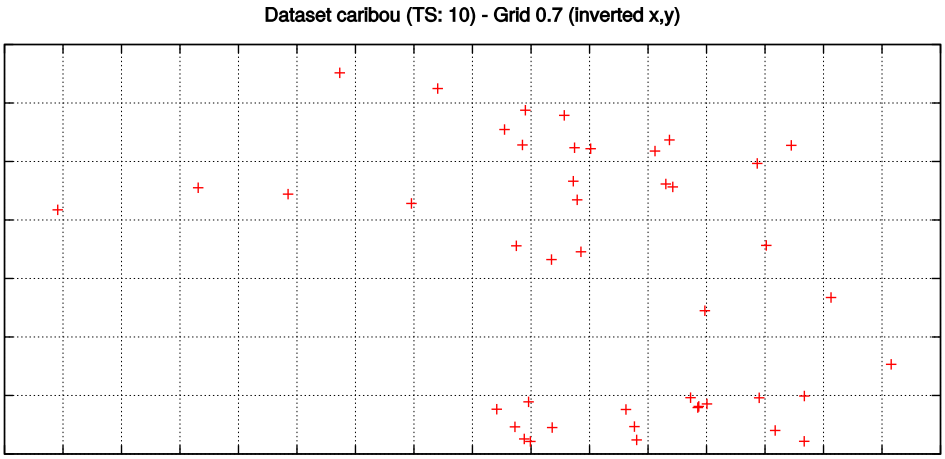
spatially disperse, thus only a few flocks are found. Conversely, the *SG* dataset has 50,000 objects however only few locations per object (51 in the average), resulting in few flocks as well. This can be verified looking at the number of answers found for those datasets (see Table II).

Further analyzing the results, we conclude that the PSI method achieved the best performance. The impact of binary signatures in execution time was not expressive, since both BFE and PSI are optimized to use spatial properties of disks/boxes, and thus they avoid performing unnecessary set intersection operations. This is the main reason we did not include the results for using binary signatures with BFE as they promote almost no gain in performance. On the other hand, while employing an inverted index yielded a reasonable improvement regarding real datasets, especially when varying δ (see plots in the third column of Figure 5.1); it was crucial for PSI in the synthetic dataset, where PSW and PSB performed worse than BFE and BFI for low values for μ including the default. The main reason PSI outperformed the other methods in *SG* dataset is the high amount of disks pruned using inverted index (up to 1.8×10^{11} out of 3.4×10^{11}), as there are many flock candidates per timestamp due to the number of objects in the dataset but only a few of them remain alive across timestamps.

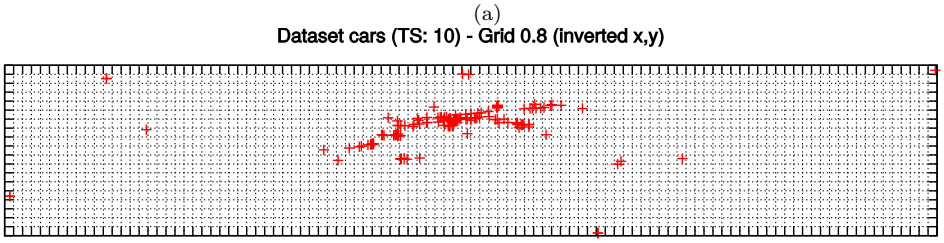
5.2 Impact of Data Skewness

One of the hurdles of searching flock patterns is that as we "relax" the parameters the number of candidate disks increase very fast as well. That is, when the value of ϵ is excessively high or the value of μ is very low the number of candidates becomes large and the performance of the methods starts to worsen (see first and second columns of Figure 5.1). The reason is that those disks have to be maintained over time and are subject of costly set operations as well. This scenario is valid for both algorithms using grid index and those based on plane sweeping. Nevertheless, as the disk diameter (ϵ) decreases the methods using grid index and LCM_Flock demand growing time to execute while the plane sweeping-based algorithms became faster and faster. When data is skewed concerning the given ϵ, BFE is less efficient for two main reasons. First, the index is big, taking more time to be constructed and traversed. Second, less computations can be saved during the process of sliding through grid cells as the amount of overlap between subsequent cells is smaller than when data is dense. The methods using plane sweeping do not suffer from data skewness as they first check if the boxes are capable of generating disks and only then the disks are calculated. Regarding LCM_Flock, the performance degrades due to the high number of transactions.

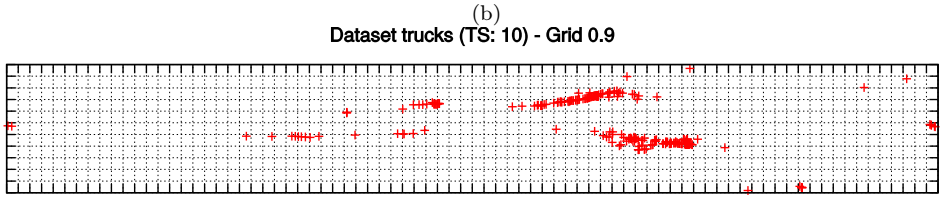
However, for large values of ϵ PSI becomes slower, being even worse than concurrents. This could be noticed in *Cars* and *Trucks* datasets (figures 5.1(b) and 5.1(h), respectively), however it may occur in other datasets for values larger than we tested. The reason for this result is that in LCM_Flock the frequent pattern mining underlying method is faster than the spatial-based approaches employed by the other algorithms. Regarding BFE, when the disks diameter increases the number of cells in the grid-based index used is reduced, which makes its performance to be more efficient than the point-by-point approach of the plane sweeping technique. We limit the topmost ϵ to a value having spatial semantics, as flocks having entities excessively apart from one another are meaningless in practice. For instance, in *Caribous* the value $\epsilon = 0.02$ is approximately equivalent to 2km.



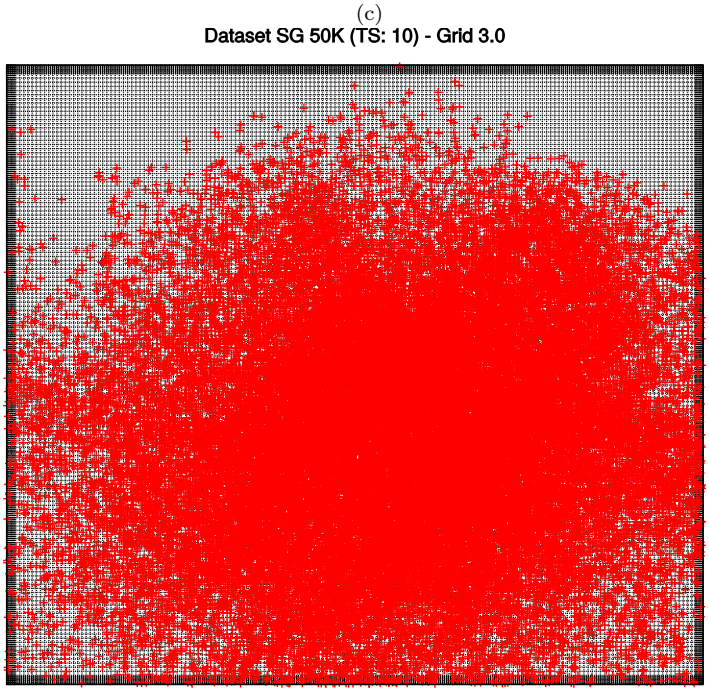
(a)



(b)



(c)



(d)

Fig. 7. Positions of the datasets Caribous, Cars, Trucks and SG 50K for a single time instance.

From the results presented it is possible to conclude that when the number of candidate flocks per timestamp decreases the total number of boxes is drastically reduced by PSI, improving its performance. This is an interesting achievement because in many situations flock patterns are really outliers when analyzing moving objects. Therefore, regarding data distribution, PSI and its variants perform better in sparse data. Figure 7 presents datasets' distribution in a given timestamp. Figure 7(a) shows a time instance of the dataset *Caribou* with a grid superimposing it. This grid has square cells with 0.2 of side. One can note that the data distribution in this data collection is diffuse, which lead to BFE underperform in relation to the plane sweeping algorithms. For this dataset, when the distance is too small it is necessary to create a large number of cells in the grid index. On the other hand, algorithms that use plane sweeping create boxes which are only kept if there are at least μ entities, this way the actual search space is considerably reduced when compared with the methods using the grid index. Figures 7(b) and 7(c) show the distribution of datasets *Cars* and *Trucks*, respectively. These datasets are relatively dense and when we apply a grid with a reasonable large value of ϵ over the data is possible to see entities grouped inside a few grid cells. When data follows this kind of packed distribution (along the y-axis, in *Cars*) there is a considerably high chance of the candidate boxes in the plane sweep to generate candidate disks. Therefore, the box generation step ends up being an overhead while searching for candidate disks degrading the overall performance, as showed afore in the figures 5.1(b) and 5.1(h). Finally, in the *SG* dataset we can observe that the behavior of the methods PSW and PSB changed considerably, being worse than BFE and BFI. This is due the fact that this dataset has a high density of objects (much more dense than *Cars*) and the radius of the parameter is set to a quite high value in order to return answers. The combination of these situations causes the method to underperform in relation to the methods using the grid index for the same reasons presented before. Nonetheless, PSI was still the best as it pruned a lot of candidate disks that did not live across the pattern time window.

6. CONCLUSION

The broad usage of location devices has aroused the interest in studying patterns that portray collaborative behaviors in spatiotemporal data (e.g., groups, swarm, flocks). Related works have focused on the problem of finding maximal duration flocks or providing off-line solutions to report flock patterns with fixed time duration. Nevertheless, several real-world applications demand online solutions to this problem. In this work we proposed the application of plane sweeping technique to accelerate the process of finding candidate disks in a timestamp. Next, we employed the use of binary signatures and inverted index to reduce the amount of set operations necessary to detect flocks. Our extensive experimental evaluation showed considerable speedups compared to existing approaches. We also discussed how data skewness impacts the performance of different flock discovery algorithms.

Future work include applying all the techniques/improvements of this article in the heuristics proposed by Vieira et al. [2009] to verify the gain in performance. After the employment of the techniques over the four heuristics there will be at least nine algorithm variations. This is a huge number of algorithms for the user to select the most adequate one. In order to make this selection process easier we want to develop a algorithm/method chooser based on the data distribution and performance in partials of the input data set. Other possible future extension could be related to handling missing data. Currently, whenever an object helping to form a flock disappear for even *one* timestamp the flock is dismantled, therefore not reported. The base algorithm to merge candidate disks could be modified in order to keep disks from more previous timestamps instead of only one, and then check the new flocks against those disks.

REFERENCES

Al-Naymat, G., Chawla, S., and Gudmundsson, J. Dimensionality Reduction for Long Duration and Complex Spatio-temporal Queries. In *Proceedings of the ACM Symposium on Applied Computing*. Seoul, Korea, pp. 393–397, 2007.

Arimura, H., Takagi, T., Geng, X., and Uno, T. Finding All Maximal Duration Flock Patterns in High-dimensional Trajectories. http://www-ikn.ist.hokudai.ac.jp/~arim/papers/maxlenflock201404r4.pdf, 2014.

Benkert, M., Gudmundsson, J., Hübner, F., and Wolle, T. Reporting Flock Patterns. In Y. Azar and T. Erlebach (Eds.), *Algorithms - ESA 2006*. Lecture Notes in Computer Science, vol. 4168. Springer, pp. 660–671, 2006.

Benkert, M., Gudmundsson, J., Hübner, F., and Wolle, T. Reporting Flock Patterns. *Computational Geometry* 41 (3): 111–125, 2008.

Bingham, E. and Mannila, H. Random Projection in Dimensionality Reduction: applications to image and text data. In *Proceedings of the ACM SIGKDD Conference on Knowledge Discovery and Data Mining*. San Francisco, USA, pp. 245–250, 2001.

Faloutsos, C. and Christodoulakis, S. Signature Files: An Access Method for Documents and Its Analytical Performance Evaluation. *ACM Transactions on Information Systems* 2 (4): 267–288, 1984.

Geng, X., Takagi, T., Arimura, H., and Uno, T. Enumeration of Complete Set of Flock Patterns in Trajectories. In *Proceedings of the ACM SIGSPATIAL International Conference on Advances in Geographic Information Systems*. Dallas, USA, pp. 53–61, 2014.

Goel, A. and Gupta, P. Small Subset Queries and Bloom Filters using Ternary Associative Memories, with Applications. In *Proceedings of the ACM International Conference on Measurement and Modeling of Computer Systems*. New York, USA, pp. 143–154, 2010.

Gudmundsson, J., Laube, P., and Wolle, T. Movement Patterns in Spatio-temporal Data. In *Encyclopedia of GIS*, Shekhar, Shashi and Xiong, Hui (Ed.). Springer US, Boston, USA, pp. 726–732, 2008.

Gudmundsson, J. and van Kreveld, M. Computing Longest Duration Flocks in Trajectory Data. In *Proceedings of the ACM International Symposium on Advances in Geographic Information Systems*. Arlington, USA, pp. 35–42, 2006.

Gudmundsson, J., van Kreveld, M., and Speckmann, B. Efficient Detection of Motion Patterns in Spatio-temporal Data Sets. In *Proceedings of the ACM International Workshop on Geographic Information Systems*. Washington, USA, pp. 250–257, 2004.

Harrison, M. C. Implementation of the Substring Test by Hashing. *Communications of the ACM* 14 (12): 777–779, 1971.

Hinrichs, K., Nievergelt, J., and Schorn, P. Plane-sweep Solves the Closest Pair Problem Elegantly. *Information Processing Letters* 26 (5): 255–261, 1988.

Jeung, H., Yiu, M. L., Zhou, X., Jensen, C. S., and Shen, H. T. Discovery of Convoys in Trajectory Databases. *Proceedings of the VLDB Endowment* 1 (1): 1068–1080, 2008.

Laube, P. and Imfeld, S. Analyzing Relative Motion within Groups of Trackable Moving Point Objects. In M. J. Egenhofer and D. M. Mark (Eds.), *Geographic Information Science*. Lecture Notes in Computer Science, vol. 2478. Springer, pp. 132–144, 2002.

Li, X., Ceikute, V., Jensen, C. S., and Tan, K. Effective Online Group Discovery in Trajectory Databases. *IEEE Transactions on Knowledge and Data Engineering* 25 (12): 2752–2766, 2013.

Li, Z., Ding, B., Han, J., and Kays, R. Swarm: Mining Relaxed Temporal Moving Object Clusters. *Proceedings of the VLDB Endowment* 3 (1): 723–734, 2010.

Romero, A. O. C. *Mining Moving Flock Patterns in Large Spatio-temporal Datasets using a Frequent Pattern Mining Approach*. Ph.D. thesis, University of Twente, The Netherlands, 2011.

Rosero, O. E. C. and Romero, A. O. C. Performance Analysis of Flock Pattern Algorithms in Spatio-temporal Databases. In *Proceedings of the Conferencia Latinoamericana en Informática*. Montevideo, Uruguay, pp. 1–6, 2014.

Shamos, M. I. and Hoey, D. Geometric Intersection Problems. In *Proceedings of the Annual Symposium on Foundations of Computer Science*. Houston, USA, pp. 208–215, 1976.

Tanaka, P. S., Vieira, M. R., and Kaster, D. S. Efficient Algorithms to Discover Flock Patterns in Trajectories. In *Proceedings of the Brazilian Symposium on Geoinformatics*. Campos do Jordão, Brazil, pp. 56–67, 2015.

Turdukulov, U., Calderon Romero, A. O., Huisman, O., and Retsios, V. Visual Mining of Moving Flock Patterns in Large Spatio-temporal Data Sets using a Frequent Pattern Approach. *International Journal of Geographical Information Science* 28 (10): 2013–2029, 2014.

Uno, T., Kiyomi, M., and Arimura, H. LCM Ver.3: Collaboration of Array, Bitmap and Prefix Tree for Frequent Itemset Mining. In *Proceedings of the International Workshop on Open Source Data Mining: frequent pattern mining implementations*. New York, USA, pp. 77–86, 2005.

Vieira, M. R., Bakalov, P., and Tsotras, V. J. On-line Discovery of Flock Patterns in Spatio-temporal Data. In *Proceedings of the ACM SIGSPATIAL International Conference on Advances in Geographic Information Systems*. Seattle, USA, pp. 286–295, 2009.

Wang, Y., Lim, E., and Hwang, S. Efficient Mining of Group Patterns from User Movement Data. *Data & Knowledge Engineering* 57 (3): 240–282, 2006.

Zobel, J. and Moffat, A. Inverted Files for Text Search Engines. *ACM Computing Surveys* 38 (2): 1–56, 2006.