Efficient Algorithms to Discover Flock Patterns in Trajectories

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

[1]Computer Science Department – Londrina State University

[2]Big Data Research Lab – Hitachi America, Ltd.

pedro.stanaka@gmail.com, marcos.vieira@hds.com, dskaster@uel.br

Abstract. *With the ubiquitous use of location enabled devices, pattern discovery in trajectories has been receiving increasing interest. Among such patterns, we have queries related to how groups of moving objects behave over time such as discovering flocks. A flock pattern is defined as a set of moving objects that move within a predefined distance to each other for a given continuous period of time. A typical application example is surveillance, where relies on discovering flocks on very large streaming spatiotemporal data efficiently. Previous work presented a polynomial solution to the problem of finding flocks with fixed time duration. And presented as well a set of algorithms based on this solution, which are the state-of-the-art algorithms regarding this problem. In this paper, we improve those algorithms by applying the plane sweeping technique in conjunction to an inverted index. The plane sweeping accelerates the detection of groups of objects that are candidates to be a flock in a time instant and the inverted index is used to compare candidate disks across time instants quickly. Using an assortment of real-world trajectory datasets, we show that our proposed methods are very efficient. When compared with the baseline flock algorithm, our proposed methods achieved up to 46x speedup reducing the elapsed time from thousands of seconds to milliseconds.*

1. Introduction

The massive usage of GPS enabled devices and location services such as Swarm [1] are two between many reasons of the growth of spatiotemporal datasets in the last decade. The fast paced growth of this kind of data demands new structures and algorithms to analyze and discovery knowledge within this large amount of data available.

Another interesting aspect of spatiotemporal data is that not only the historical datasets grows, but it also continuously grow in an on-line fashion since services like AccuTracking [2] which has clients like the United States Postal Service (USPS) that has one of the largest civilian vehicle fleets of the word [Service 2015], and Foursquare, which as of the end of 2013 passed the 45 million users mark with more than 5 billion check-ins [Crowley 2013]. When using these streams one interesting to analyze them is using on-line algorithms.

One of the goals when analyzing spatiotemporal data is to query for interactions between objects that may reveal some kind of common behavior. Interested in discovering these behaviors the research community proposed a large number of

[1]http://swarmapp.com/

[2]http://www.accutracking.com/

spatiotemporal patterns each describing one kind of behavior. Examples include the group [Wang et al. 2006, Li et al. 2013], swarm [Li et al. 2010, Hai et al. 2014] and flock patterns [Benkert et al. 2008, Gudmundsson and van Kreveld 2006, Vieira et al. 2009, Romero 2011] which is the subject of this work.

The flock pattern is defined by a set of objects that travel together for a specified time duration. Recent papers on flock detection have focused either on algorithms using the mappings of data domain [Romero 2011] or finding maximal duration flocks [Arimura and Takagi 2014]. Even though these algorithms present overall better results than previously proposed algorithms such as [Vieira et al. 2009, Benkert et al. 2008], they cannot process data in an on-line fashion, either because they need to preprocess data before analyzing it or because the nature of the problem being solved.

In [Vieira et al. 2009] a set of algorithms were proposed to address the problem of finding flocks with fixed time duration in polynomial time. These algorithms share a common base algorithm called Basic Flock Evaluation (BFE), therefore they all share in a certain degree some steps in the data processing. In this context, this paper proposes the application of the plane sweeping technique in addition to the concept of binary signature and inverted list structure to improve the algorithms proposed on [Vieira et al. 2009].

The rest of the paper is organized in the following manner: in Section 2 we present the basic theory about the flock pattern and explain the plane sweeping technique along with its usages; in Section 3 we show how we combined the plane sweeping technique and inverted index to create new improved algorithms to detect flock patterns; in Section 4 we evaluate the proposed algorithms via extensive tests against real datasets and show their efficiency; lastly in Section 5 we expose our final thoughts and considerations about the results obtained in this paper.

2. Background

2.1. Flock Pattern

In this work we consider the discovery of flock patterns, that is, the problem of identifying all sets of trajectories that stay "close together" for a time interval of fixed size. The notion of closeness here is depicted by a disk of a given radius that covers all the objects belonging to the pattern (see Figure 1). There are a number of flock definitions on the literature, for instance, in [Gudmundsson et al. 2004] the pattern is defined by a set of trajectories that are moving in the same direction, but only in one time stamp. In other works [Benkert et al. 2006, Vieira et al. 2009, Romero 2011] the definition evaluate not only one time stamp but a window that is either of maximal size or fixed size. In this work we use that latter definition where the lifespan of the pattern is fixed. In 1 is presented the formal definition of the Flock pattern [Vieira et al. 2009].

Definition 1 (Flock pattern) *Given a set of trajectories $\mathcal{T}$, a minimum number of trajectories $\mu > 1 (\mu \in \mathbb{N})$, a minimum distance $\epsilon > 0$ defined by a distance metric d and minimum duration $\delta > 1$ time instances, where $(\delta \in \mathbb{N})$. A **Flock**(μ, ϵ, δ) pattern reports a set $\mathcal{F}$ containing all the flocks $f_k f$, which are sets of maximal size such that: for each $f_k \in \mathcal{F}$, the number of trajectories is greater or equal to μ and there exist δ consecutive time stamps $t_j, \ldots, t_{j+\delta-1}$ 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, $j \leq i \leq j + \delta$.*

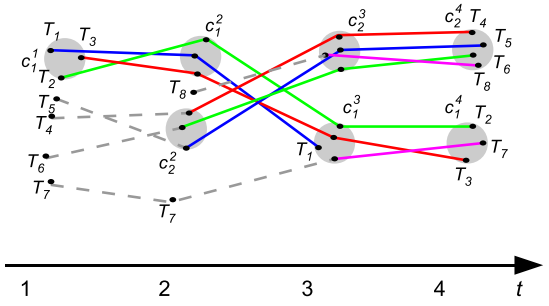


Figure 1. Example of flock pattern: $\{T_1, T_2, T_3\}$ and $\{T_4, T_5, T_6\}$ with timestamps 2–4 adn 3–5, respectively.

Figure 1 shows an example of flock pattern with two instances involving three moving objects. The flocks are: first one formed by trajectories with identifiers $\{T_1, T_2, T_3\}$ covered by the disks with center $\{c_1^1, c_1^2, c_1^3\}$ and the flock composed by $\{T_4, T_5, T_6\}$ which are covered by $\{c_2^2, c_2^3, c_2^4\}$. It is important to note that the center of the disks can move freely in order to cover all the trajectories an the center not necessarily coincide with any object location. This is a complicating factor on the task of discovering the flock patterns since there may be infinite possibilities to place the center of the disk at each time instance [Vieira et al. 2009].

2.2. Related Work

Since the first proposal of the flock pattern by Laube et al. [Laube and Imfeld 2002] several papers have been published addressing the problem of finding flock patterns. However, that definition only considered one time instant and was later modified [Benkert et al. 2006] to support a fixed size time window. Following this definition several papers were published presenting solutions using the mapping of polylines to points in a high dimensional space [Benkert et al. 2006] in addition to dimensionality reduction techniques [Al-Naymat et al. 2007], and later using pure spatiotemporal search [Vieira et al. 2009].

The algorithm proposed in [Al-Naymat et al. 2007] only reports approximations and though it improved the results from previous articles it has a exponential dependency on the size of the flock. After that, in [Vieira et al. 2009] was proposed the first algorithm that could detect flocks in polynomial time. In that article was proposed a set of algorithms all based in a common algorithm called Basic Flock Evaluation. These algorithms are state-of-art with respect to the reporting of flock with fixed time duration.

2.2.1. Basic Flock Evaluation Algorithm

As stated before, there may be infinite possibilities to place the disk centers in a time instant. This makes the tasks of finding candidate disks very complicated. In order to address this problem in [Vieira et al. 2009] was proposed and proved a theorem that reduces the search space of the task of finding candidate disks that might generate a new flock. Based on the Definition 1 the theorem is as follows:

Theorem 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:

$$\forall T_j \in f_k, d(p_j^{t_i}, c_k'^{t_i}) \leqslant \epsilon/2$$

and there are at least trajectories $T_a \in f_k$ and $T_b \in f_k$ such that:

$$\forall T_j \in \{T_a, T_b\}, d(p_j^{t_i}, c_k'^{t_i}) = \epsilon/2$$

This theorem states that if there is a disk with center $c_k^{t_i}$ with diameter ϵ that covers all trajectories in a flock f, then there another disk with different center $c_k'^{t_i}$ that also covers all the trajectories of f. This theorem affects considerably the search for flock patterns since it limits the locations inside the spatial domain where we there is a need to search for flocks. For a database with $|\mathcal{T}|$ trajectories, there are $|\mathcal{T}|^2$ possible combination of point pairs in a time instant. Fore each pair there are two disks with radius of $\epsilon/2$ that have these points in its circumference.

Using this theorem to reduce the search space the author proposed an algorithm called Basic Flock Evaluation (BFE). The BFE algorithm is the simplest amongst those presented in [Vieira et al. 2009], since it does not have any kind of heuristic in it. This algorithm utilizes a grid based index to help the task of finding disks. This index contain square cells with ϵ in side. Each position of a time instant is inserted in one cell of this index depending on the value of x and y of the location, therefore the size of the index depends on the distribution of locations in the space. After the index being constructed, the search for disks is done in the following manner:

1. for each cell of the index $g_{x,y}$, try to find points in the ϵ range, in the nine adjacent cells in relation to the point being processed.
2. for point pairs that have not been processed and are within ϵ range are used to compute two disks with centers c_1 and c_2.
3. those disks that have at least μ entities are reported as candidates.

After finding the candidate disks the algorithm perform the following steps to find flocks:

1. for each time instant, find those pair of points that might generate disks of radius $\epsilon/2$.
2. count the points inside the disks and discard those disks with less than μ entities.
3. for the rest disks that has at least μ objects, search for subsets and eliminate them.
4. finally, for all the disks that have been found in the previous time instant they should be merged with the ones of the current time instant.

Amongst these steps the more time consuming are ones with number 1 and 3, the first one involves the construction and querying of the grid based index and the third requires several set operations between disks which are also expensive.

2.2.2. Maximal Length Flocks

Another class of flock patterns is the *Maximal Length (or Duration) Flocks*. These patterns are very similar to the original flock, but there is no defined time window. Therefore the patterns are only defined by a diameter ϵ and a minimum number of objects μ. In [Romero 2011, Turdukulov et al. 2014, Rosero and Romero 2014] was proposed an algorithm to report maximal size flocks, that algorithm uses a frequent pattern mining [Zhang and Zhang 2002] approach, however it is not possible to apply such data mining technique directly in to spatiotemporal databases, for that reason the algorithm depends on the transformation of the input trajectorial database to a correspondent transactional database and this is the real input for the underlying algorithm called Linear time Closed itemset Miner, LCM for short [Uno et al. 2005]. Although the results from those works are satisfactory they can not be directly compared with the results produced by the algorithms introduced in this paper since the problem is different and mainly due to the preprocessing step present on the algorithms.

In [Arimura and Takagi 2014, Geng et al. 2014] was proposed the Flock Pattern Miner (FPM) algorithm family, these algorithms uses a depth first search (DFS) approach to find all maximal duration flocks in a input trajectorial database. The articles [Arimura and Takagi 2014, Geng et al. 2014] were the first address the problem of enumerating the maximal-size flocks in polynomial space. The FPM algorithm family share some similarities between its members, the main aspect that it is crucial to explain here is that since the algorithms uses DFS to check all the time instants for a particular object it would not be practicable to use such methods to process data in an on-line manner.

2.3. Plane Sweeping Technique

The plane sweep technique was first used in [Shamos and Hoey 1976] to detect intersection between line segments in the plane. Since then, this technique has proven to be very important to reduce computational complexity in a large number of problems in computational geometry. Its applications range from searching for closest pair of points, through convex hull computation [Graham 1972], to finding minimum spanning trees [Guibas and Stolfi 1983].

The main idea of the algorithms that use plane sweeping is to sweep a line (generally vertical) throughout the plane, stopping when some condition is met. When this condition is met we say an event occurred, then geometric operations are performed on the points that intersect this line. This way, the operations are limited to these points near the line, therefore reducing the number of data to be processed. The solution of the problem is generally found when all the points of the dataset were processed [Nievergelt and Hinrichs 1999].

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).

Suppose that the algorithm already processed $N-1$ points of the input (ordered

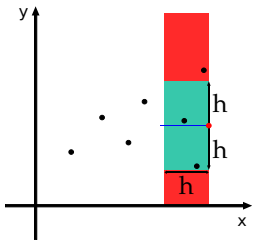


Figure 2. Application of plane sweeping technique to find closest pair.

by x-axis) and the shortest distance between consecutive points found so far is h (see Figure 2). The algorithm processes the last point N and try to find a point that is closer to it than h. We maintain a set of all the points processed that are within a range of h in x-axis of the point N (red rectangle in Figure 2). 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.

In order to check if a point is at a distance less than h from the point N, we just check those points that are elements of the balanced tree, furthermore the algorithm only consider the points in the range $y_N - h$ to $y_N + h$ (those inside the blue rectangle in Figure 2). This range can be recovered from the ordered set in $\log N$. 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 $O(N.\log(N))$.

It is possible to note through this example that the plane sweeping technique can drastically alter the complexity of algorithms. That is one of the reasons we use plane sweep in a very similar way to this algorithm to find the pair of points that are at a given distance ϵ and might generate new disks of flocks.

3. Proposed Methods

We now describe our proposed methods that employ plane sweeping technique and binary signatures to improve the search for disks on a specific time stamp. We then detail how to further improve our proposed methods using inverted indexes during the spatiotemporal join phase. This step improves the performance of the proposed methods by pruning set intersection operations.

3.1. Finding Disks Using Plane Sweep

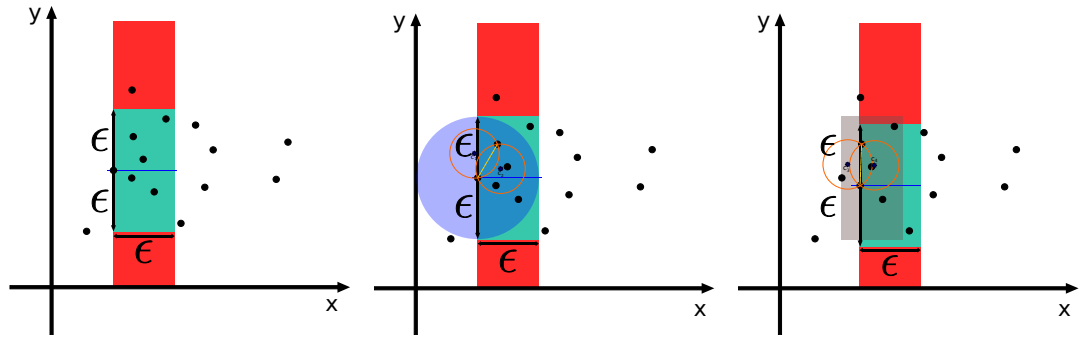
As already stated in Subsection 2.2.1 for the algorithms, that use BFE somehow, the search for disks can be very expensive since for each time instance there is a need to build an index before actually To eliminate this need we use plane sweeping to search for disks, with the direct application of this powerful technique we can find the disks directly with no preprocessing (index building). The process to find disks is depicted in Algorithm 1.

The algorithm first sweeps the plane, using a band of size ϵ along the x-axis (red box on Figure 3(a)). When a position $\mathbf{p}$ is about to get out of this band it selects the points that are in the range beginning in $\mathbf{p}.y - \epsilon$ and ending in $\mathbf{p}.y + \epsilon$ (blue box on Figure 3(a)) these steps are presented in the lines 2-9.

Algorithm 1: Finding candidate disks with plane sweeping technique

Input: $\mathcal{T}[t_i]$: Positions of time stamp t_i ordered by x-axis, ϵ: flock diameter, μ: Minimum size of flock
Output: $\mathcal{C}$: candidate disks for the time stamp t_i

1 $\mathcal{C} \leftarrow \emptyset$
2 **foreach** $p_r \in \mathcal{T}[t_i]$ **do**
3 $\quad \mathcal{P} \leftarrow \emptyset$ // list of neighbors of current box
4 $\quad$ **foreach** $p_s \in \mathcal{T}[t_i]$ **do**
5 $\quad\quad$ **if** $p_s.x \leqslant p_r.x + \epsilon$ **then**
6 $\quad\quad\quad$ **if** $(p_s.y \leqslant p_r.y + \epsilon)$ **and** $(p_s.y \geqslant p_r.y - \epsilon)$ **then**
7 $\quad\quad\quad\quad \mathcal{P} \leftarrow \mathcal{P} \cup p_s$
8 $\quad\quad$ **else** // We can stop, since outside box (x-axis).
9 $\quad\quad\quad$ **break**
10 $\quad$ **foreach** $p_s \in \mathcal{P}$ **do**
11 $\quad\quad$ **if** p_s *wasn't computed* **then**
12 $\quad\quad\quad$ **if** $d(p_r, p_s) \leqslant \epsilon$ **then** // Range-search
13 $\quad\quad\quad\quad$ calculate disks c_1, c_2 defined by $\{p_r, p_s\}$ with radius $\epsilon/2$
14 $\quad\quad\quad\quad$ **foreach** *disk* $c \in c_1, c_2$ **do**
15 $\quad\quad\quad\quad\quad$ **if** $|c \cap \mathcal{P}| \geqslant m$ **then**
16 $\quad\quad\quad\quad\quad\quad \mathcal{C} \leftarrow \mathcal{C} \cup c$
17 **return** $\mathcal{C}$



(a) Sweeping the plane across x-axis.

(b) Filter points in y-axis.

(c) Check for subsets in the boxes.

Figure 3. Steps needed to find disks in one time stamp.

After selecting the points that are near the reference point, we must check the pair of points that might generate new disks (refer to Theorem 1). In order to do this we must generate disks defined by the pair of points that are inside the box and are at most ϵ distant from each other (Figure 3(b)). If the disks contain at least μ entities, then they are reported as a candidate disk. These last steps are illustrated in the Figure 3(b) and represented by the lines 10-16 of Algorithm 1.

3.1.1. Filtering Final Candidate Disks

After we find the candidate disks for the current time stamps we must check for disks that are subsets or supersets, since in the definition used in this paper the flock pattern return always maximal sets of trajectories. A naive approach to detect supersets can be achieved through the comparison of all the disk one by one, however this is approach is quadratic in the number of disks for a time stamp and involve set intersection operations which are very expensive. In order to perform this filtering step in a more productive way, we use binary signatures and spatial properties of the boxes (generated in Algorithm 1) to prune disks that surely are not sub/supersets.

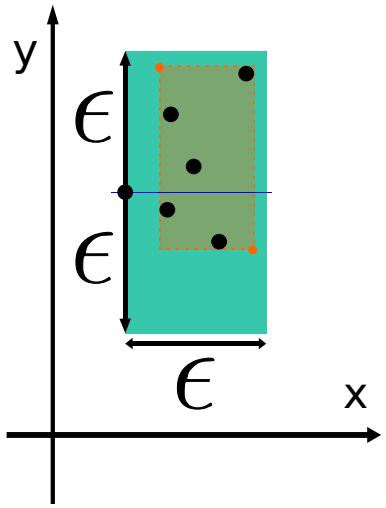


Figure 4. MBR of the boxes

These spatial relations between boxes are given by minimum bounding rectangles (MBR) which are calculated when the box are being constructed (Figure 4). Note that although the box have dimensions of $(2.\epsilon$ x $\epsilon)$ the MBR is way smaller. Each box has information about all the clusters that were formed inside themselves as well. The process of filtering the valid candidate disks is shown on Algorithm 2, where $|c|$ stands for the quantity of objects inside a disk.

Algorithm 2: Filter disks that are subsets

```
1  Algorithm FilterCandidates()
       Input: B_{t_i}: live boxes of time stamp t_i ordered by x-axis
       Output: C: final set of disks for time stamp t_i
2
3      C ← ∅
4      for j ← 0 to j ⩽ |B_{t_i}| do
5          for k ← j + 1 to k ⩽ |B_{t_i}| do
6              if IntersectsWith(B_{t_i}[j], B_{t_i}[k]) then
7                  foreach c ∈ B_{t_i}[j].disks do
8                      C ← InsertDisk(C, c)
9              else // No intersection.
10                 break
11 Procedure IntersectsWith(a, b)
12     return (a.x_1 < b.x_2 and a.x_2 > b.x_1 and a.y_1 > b.y_2 and a.y_2 < b.y_1)
13 Procedure InsertDisk(C, c)
       Input: C: set of disks, c: new disk
14
15     if |C| > 0 then
16         foreach d ∈ C do
17             if d.sign ∧ c.sign = c.sign then // c can be subset of d
18                 if |d ∩ c| = |c| then // Remove chance of false-positive
19                     return // No need to insert c
20             else
21                 if c.sign ∧ d.sign = c.sign then // d is subset of c
22                     if |c ∩ d| = |d| then // Remove chance of false-positive
23                         C ← C \ d // Remove d
24     return C ∪ c
```

The step of filtering or pruning just comprise the verification of the boxes that hold at least one cluster. The Algorithm 2 begins iterating all the live box (as we call a box that

has a cluster) in the current time stamp checking if there is intersection between the MBR of the box and the next ones; when this happens it is necessary to check whether there is duplicate or subset clusters between these boxes (Figure 3(c)).

To better explain how the process of checking set intersection through binary signatures it is better to explain how these signatures are generated. As the positions get inserted in the disks a set of hash functions are used to generate a signature. Each hash function maps an object identifier to a position inside the signature (*bucket*).

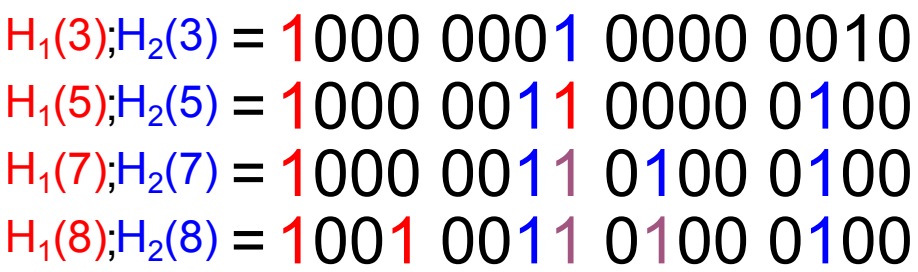


Figure 5. Process of generating binary signatures

In Figure 5 is possible to see the process of signature generation. Suppose that a disk contains the objects identified by $\{3, 5, 7\}$ then it signature will be identical to the third line of the figure. In the figure H_1 and H_2 refers respectively to the Spooky-Hash [Jenkins 2012] and MurMurHash [Appleby 2008] which are fast hashes and with good non-linearity (measured by avalanche criterion - refer to [Moon 2014]). Note that since the signature size is very small occurred some collisions between the hash functions (bits in pink).

Now, suppose we want to determine if a disk containing the objects identified by $\{3, 5, 7, 8\}$ is subset/superset of the disk containing $\{3, 5, 7\}$ without having to perform a set intersection operation and thus having to access all the objects of one of the sets. In order to achieve this, we apply a logical *and* operation between the two signatures from the disks. If the result of the operation is equal to the second operator, then the first one may be a subset of the second. For instance, the result signature of $\{3, 5, 7\}.binSignature \wedge \{3, 5, 7, 8\}.binSignature$ is $1001\ \ 0011\ \ 0100\ \ 0100$ which is exactly the signature for $\{3, 5, 7, 8\}$.

The primitives of performing subset queries using binary signatures comes from the article [Goel and Gupta 2010]. In this article it is used a set of Bloom filters to represent subsets of a universe and then these filters, that essentially are binary vectors, were used to check for subsets amongst the sets. It is also shown that this method can provoke false-positives, but not false-negatives. This is the reason why in Algorithm 2 after we check that one disk is subset of another using binary signatures we still have to check via set intersection if that is really true.

3.2. Joining Disks Between Two Consecutive Time Stamps

After we find all the disks for a given time instant we must join the disks from the current time instant with the previous one. In this paper we propose the application of inverted lists or inverted indexes to accelerate the process the process of joining disks across time instants.

One simple way to join the disks is to process all the disks from one of the time instants and check for the join condition which is if two disks have at least μ objects in common. However this process is very expensive, thus the need of some index structure in order to reduce the complexity of this process.

Inverted index is a well-known structure to index documents in order to speed up queries of terms [Zobel and Moffat 2006, Knuth 1998]. An inverted list is composed by a list of keys which are the common terms that appear in all the documents. Each item, in turn, has a set of document identifiers where the term represented by that item appeared.

Using the inverted index we can eliminate the need to search all the disks from previous time instant (t_{i-1}) to check which ones might have at least μ object in common with the disk being processed of the current time instant (t_i). When we get a new disk from t_i we pass the set of OID's that belongs to that disk as the query elements Q_s to the inverted index. The query condition is that a document (or disk) is returned iff it has at least μ terms (or objects) in common with query set. Now suppose that 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 $l; l \in \mathbb{N}, l < \mu$. If we compared all the disks from t_i with the ones in t_{i-1} then we would have a complexity of $n.m.l$, but introducing the use of inverted indexes we can reduce this drastically.

4. Experimental Evaluation

In order to verify the efficiency of the the methods proposed, we ran a large amount of experiments with several trajectory datasets and varied the parameters of the flock pattern as well. The datasets available to tests were four real datasets, namely *Trucks, Buses, Cars, and Caribous*. The dataset *Trucks* has 112,203 moving object locations generated by 276 moving trucks, while *Buses* has 66,096 locations generated by 145 buses, being that both of them were collected in the metropolitan area of Athens, Greece. The third dataset *Cars* contains 134,264 object locations collected from 183 private cars moving in Copenhagen, Denmark [Cho 2012]. The last dataset *Caribou* is generated from the migration movements of 43 caribous in northwest states of Canada, and it has a size of 15,796 locations.

The algorithms used to demonstrate the speed up brought by the techniques proposed in this paper are all based on BFE [Vieira et al. 2009]. Therefore, the five algorithms are:

BFE: The original implementation of the BFE algorithm.
BFI: BFE algorithm using the inverted index to join disks (**B**asic **F**lock Evaluation with **I**nverted index).
PSW: Raw plane swe eping technique directly applied to BFE replacing the grid index to search for disks (**P**lane **S**weep **R**aw).
PSB: Plane sweeping technique with binary signatures to accelerate the process of finding disks [same time instants] (**P**lane **S**weeping with **B**inary signature).
PSI: Plane sweeping technique with binary signatures and inverted index to help prune disks that may not form flocks (**P**lane **S**weeping with **I**nverted index).

All the methods and their variations were implemented in C++ and compiled with GCC v4.9. The tests were run in a computer with a Intel Core 2 Quad@2.83GHz CPU, 4 GB RAM @1333MHz and 500GB@3Gb/s HDD. In our experiments the parameters

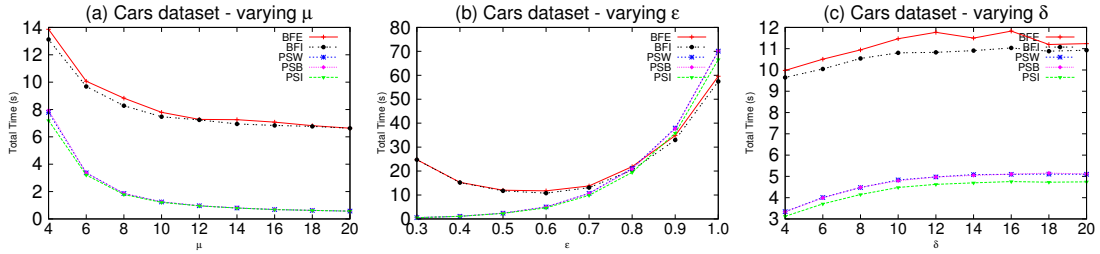


Figure 6. Experiments with *Cars* dataset when varying μ, ϵ and δ.

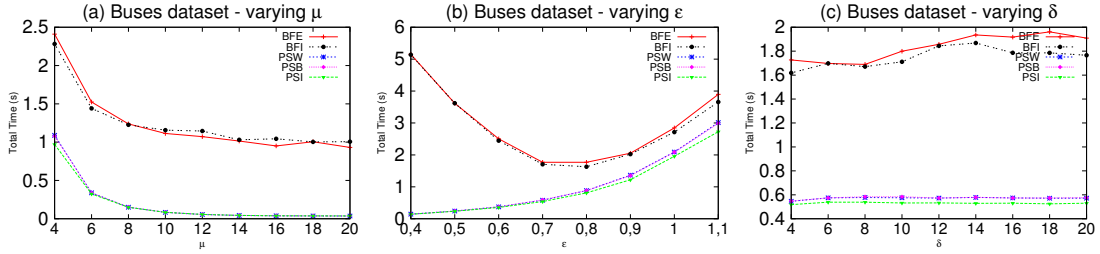


Figure 7. Experiments with *Buses* dataset when varying μ, ϵ and δ.

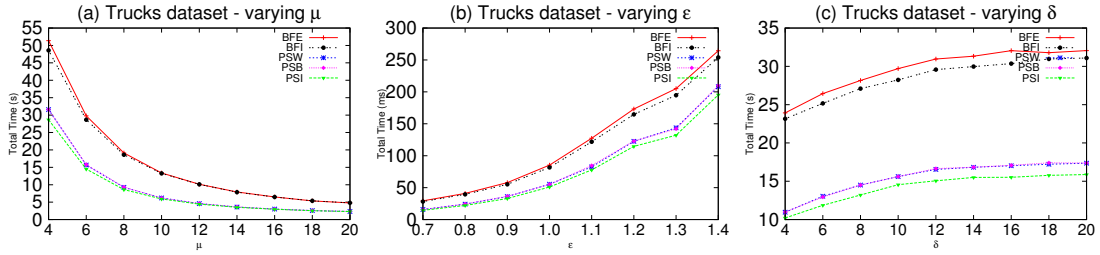


Figure 8. Experiments with *Trucks* dataset when varying μ, ϵ and δ.

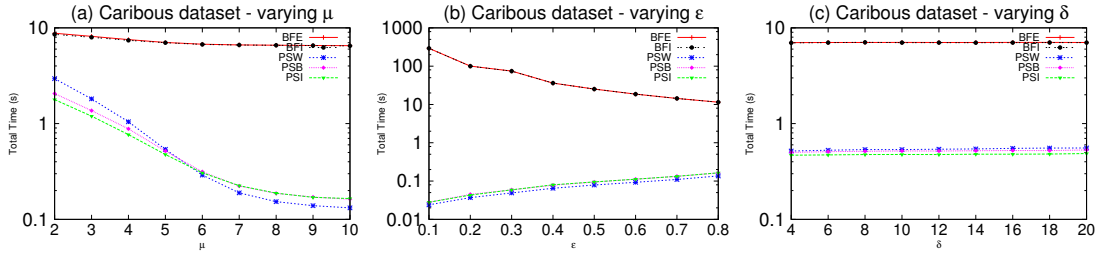


Figure 9. Experiments with *Caribous* dataset when varying μ, ϵ and δ.

of distance, size and duration of the flock patterns were used with many different values. Figures 4 – 4 show the results charts while varying the parameters μ, ϵ and δ, respectively in the first, second and third column. All plots show the time in seconds that was needed to process the whole database, only for the plots for the *Caribou* dataset was used the logarithmic scale to improve the visualization of the curves.

The first conclusion we can draw from the results is that as we decrease ϵ, increase μ, or decrease δ, the amount of time required to process the data is reduced, this is an expected behavior of such algorithms due the increase of selectivity in flock query and

thus the reduced number of candidate to maintain. It can be seen from the charts as well that the extension that offers the best gain is the plane sweeping technique. This result is explained by the fact that BFE algorithm needs to build an grid based index before actually begin to search for disks and this index is highly dependent on the data sparseness, e.g. if the data has a property of being more sparse not only the index itself will be bigger but it will take more time as well to construct it.

In a general manner the extensions proposed in this paper perform better than the previous algorithm (BFE). In the *Cars* dataset is the only dataset that the method presented a lose of performance, the reason for this is that when the disks diameter (ϵ) is increased, the number of cells in the grid index used by BFE is reduced, thus improving its efficiency. Further analyzing the results it is possible to conclude that neither inverted index nor binary signatures presented substantial gain. The best gain between these two techniques is the inverted index, and is better observed in the charts of the third column (varying δ). The behavior for the results of binary signatures is expected since the BFE algorithm in a detail level is optimized and uses spatial properties of the disks to avoid doing unnecessary set intersection operations [Vieira et al. 2009]. This is the reason why we preferred not to include the results for the application of binary signatures directly in BFE, because of the minor gain.

In the dataset *Caribou* we see that as we increase μ the use of the plane sweeping technique have a larger impact than in the others datasets. The explanation to this fact is that in this dataset although the data seems to be highly correlated the clusters are small, thus when you increase the cluster size in the query parameter it leads to reducing the number of box that are created since most of them will not have at least μ, not to mention the chance of generating any cluster. For the BFE algorithm the behavior is steady because it always will create the index and always will try to calculate all the possible disks.

5. 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, etc.). In particular, the flock pattern is characterized by a set of at least μ objects moving inside a circumference with diameter ϵ for at least δ time instants. The on-line detection of flock patterns is still important and has a wide range of applications from live monitoring suspicious activities in tracking streams to monitoring animal behavior. Recent related works have focused on the problem of finding maximal duration flocks, however due to the nature of the problem it is highly difficult to propose solutions that work in a on-line fashion. The last work on reporting flocks with fixed time proposed a base algorithm with polynomial time complexity called BFE and a series of heuristics based on this algorithm. In this work we proposed the application of the plane sweeping technique to help accelerate the process of finding disks in BFE. Later, we introduced the utilization of binary signatures and inverted index to reduce the amount set operations necessary to detect flocks. The experiments on real datasets show considerable speedups mainly with the use of the plane sweeping technique varying all the parameters of the pattern (μ, ϵ, δ). This has a great impact not only for the problem of reporting flocks with *fixed time* duration since the application of such techniques are primitives for every other algorithms like LCM_Flock [Romero 2011].

References

[Cho 2012] (2012). Chorochronos. `http://chorochronos.datastories.org/`. Accessed: 2015-04-12.

[Al-Naymat et al. 2007] Al-Naymat, G., Chawla, S., and Gudmundsson, J. (2007). Dimensionality reduction for long duration and complex spatio-temporal queries. In *Proc. of the ACM SAC*, page 393.

[Appleby 2008] Appleby, A. (2008). Murmurhash 2.0.

[Arimura and Takagi 2014] Arimura, H. and Takagi, T. (2014). Finding All Maximal Duration Flock Patterns in High-dimensional Trajectories. *Manuscript, DCS, IST, Hokkaido University, Apr*.

[Benkert et al. 2006] Benkert, M., Gudmundsson, J., Hübner, F., and Wolle, T. (2006). Reporting flock patterns. In *Algorithms – ESA 2006*, volume 4168, pages 660–671. Springer Berlin Heidelberg.

[Benkert et al. 2008] Benkert, M., Gudmundsson, J., Hübner, F., and Wolle, T. (2008). Reporting flock patterns. *Computational Geometry*, 41(3):111–125.

[Crowley 2013] Crowley, D. (2013). Ending the year on a great note (And with a huge thanks and happy holidays to our 45,000,000-strong community).

[Geng et al. 2014] Geng, X., Takagi, T., Arimura, H., and Uno, T. (2014). Enumeration of complete set of flock patterns in trajectories. In *Proc. of the ACM SIGSPATIAL Int'l Workshop on GeoStreaming*, pages 53–61.

[Goel and Gupta 2010] Goel, A. and Gupta, P. (2010). Small subset queries and bloom filters using ternary associative memories, with applications. In *ACM SIGMETRICS Performance Evaluation Review*, volume 38, pages 143–154. ACM.

[Graham 1972] Graham, R. L. (1972). An efficient algorith for determining the convex hull of a finite planar set. *Inf. Process. Lett.*, 1(4):132–133.

[Gudmundsson and van Kreveld 2006] Gudmundsson, J. and van Kreveld, M. (2006). Computing longest duration flocks in trajectory data. In *Proc. of the ACM GIS*, page 35.

[Gudmundsson et al. 2004] Gudmundsson, J., van Kreveld, M., and Speckmann, B. (2004). Efficient Detection of Motion Patterns in Spatio-temporal Data Sets. In *Proc. of the ACM GIS*, pages 250–257.

[Guibas and Stolfi 1983] Guibas, L. and Stolfi, J. (1983). On computing all north-east nearest neighbors in the L1 metric. *Inf. Process. Lett.*, 17(4):219–223.

[Hai et al. 2014] Hai, P. N., Poncelet, P., and Teisseire, M. (2014). All in One: Mining Multiple Movement Patterns. *Int'l Journal of Information Technology & Decision Making*, 13(8).

[Hinrichs et al. 1988] Hinrichs, K., Nievergelt, J., and Schorn, P. (1988). Plane-sweep solves the closest pair problem elegantly. *Inf. Process. Lett.*, 26(5):255–261.

[Jenkins 2012] Jenkins, B. (2012). Spookyhash: a 128-bit noncryptographic hash.

[Knuth 1998] Knuth, D. E. (1998). *The art of computer programming: sorting and searching*, volume 3. Pearson Education.

[Laube and Imfeld 2002] Laube, P. and Imfeld, S. (2002). Analyzing relative motion within groups oftrackable moving point objects. *Geographic information science*.

[Li et al. 2013] Li, X., Ceikute, V., Jensen, C. S., and Tan, K.-L. (2013). Effective Online Group Discovery in Trajectory Databases. *IEEE Trans. Knowl. Data Eng.*, 25(12):2752–2766.

[Li et al. 2010] Li, Z., Ding, B., Han, J., and Kays, R. (2010). Swarm: Mining relaxed temporal moving object clusters. *Proc. of the VLDB Endowment*, 3(1-2):723–734.

[Moon 2014] Moon, A. (2014). The non-cryptographic hash function zoo. `http://floodyberry.com/noncryptohashzoo/`. Accessed: 2015-04-10.

[Nievergelt and Hinrichs 1999] Nievergelt, J. and Hinrichs, K. H. (1999). *Algorithms and data structures - with applications to graphics and geometry*. vdf Lehrbuch. vdf.

[Romero 2011] Romero, A. (2011). *Mining moving flock patterns in large spatio-temporal datasets using a frequent pattern mining approach*. Mestrado, University of Twente.

[Rosero and Romero 2014] Rosero, O. E. C. and Romero, A. O. C. (2014). Performance analysis of flock pattern algorithms in spatio-temporal databases. In *Latin American Computing Conf.*, pages 1–6.

[Service 2015] Service, U. S. P. (2015). Size and Scope - Postal Facts.

[Shamos and Hoey 1976] Shamos, M. I. and Hoey, D. (1976). Geometric intersection problems. In *Proc. of the IEEE FOCS*, pages 208–215.

[Turdukulov et al. 2014] Turdukulov, U., Calderon Romero, A. O., Huisman, O., and Retsios, V. (2014). Visual mining of moving flock patterns in large spatio-temporal data sets using a frequent pattern approach. *Int'l J. of Geographical Information Science*, 28(10):2013–2029.

[Uno et al. 2005] Uno, T., Kiyomi, M., and Arimura, H. (2005). Lcm ver. 3: Collaboration of array, bitmap and prefix tree for frequent itemset mining. In *Proc. of the Int'l Workshop on Open Source Data Mining: frequent pattern mining implementations (OSDM)*, pages 77–86.

[Vieira et al. 2009] Vieira, M. R., Bakalov, P., and Tsotras, V. J. (2009). On-line discovery of flock patterns in spatio-temporal data. In *Proc. of the ACM SIGSPATIAL*, page 286.

[Wang et al. 2006] Wang, Y., Lim, E. P., and Hwang, S. Y. (2006). Efficient mining of group patterns from user movement data. *Data Knowl. Eng.*, 57(3):240–282.

[Zhang and Zhang 2002] Zhang, C. and Zhang, S. (2002). *Association rule mining: models and algorithms*. Springer-Verlag.

[Zobel and Moffat 2006] Zobel, J. and Moffat, A. (2006). Inverted files for text search engines. *ACM computing surveys (CSUR)*, 38(2):6.