Parallel finding of moving flock patterns

Andres Calderon-Romero
acald013@ucr.edu
University of California, Riverside

Petko Balakov
petko@
Esri

Marcos Vieira
marcos@
Google

Vassilis J. Tsotras
tsotras@cs.ucr.edu
University of California, Riverside

ABSTRACT

CCS CONCEPTS

• **Computing methodologies** → **Parallel algorithms**; **MapReduce algorithms**; • **Information systems** → **Data structures**.

KEYWORDS

Mobile patterns

ACM Reference Format:
Andres Calderon-Romero, Petko Balakov, Marcos Vieira, and Vassilis J. Tsotras. 2023. Parallel finding of moving flock patterns. In . ACM, New York, NY, USA, 7 pages. https://doi.org/XXXXXXX.XXXXXXX

1 INTRODUCTION

Technology advances in last past decades have triggered an explosion in the capture of spatio-temporal data. The increase popularity of GPS devices and smart-phones together with the emerging of new disciplines such as the Internet of Things and Satellite/UAS high-resolution imagery have made possible to collect vast amount of data with a spatial and temporal component attached to them.

Together with this, the interest to extract valuable information from such large databases has also appeared. Spatio-temporal queries about most popular places or frequent events are still useful, but more complex patterns are recently shown an increase of interest. In particular, those what describe group behaviour of moving objects through significant periods of time. Moving cluster [16], convoys [14], flocks [11] and swarm patterns [19] are new movement patterns which unveil how entities move together during a minimum time interval.

Applications for this kind of information are diverse and interesting, in particular if they come in the way of trajectory datasets [13, 15]. Case of studies range from transportation system management and urban planning [6] to Ecology [17]. For instance, [23] explore the finding of complex motion patterns to discover similarities between tropical cyclone paths. Similarly, [1] use eye trajectories to understand which strategies people use during a visual search. Also, [12] tracks the behavior of tiger sharks in the coasts of Hawaii in order to understand their migration patterns.

In particular, a moving flock pattern show how objects move close enough during a given time period. Closeness is defined by a disk of a given radius where the entities must keep inside. Given that the disk can be situated at any location, it is not a trivial problem. In deed, [11] claims that find flock patterns where the same entities stay together during their duration is a NP-hard problem. [24] proposed the BFE algorithm which the first approach to be able to detect flock patterns in polynomial time.

Despite the fact that much more data become available, state-of-the-art techniques to mine complex movement patterns still depict poor performance for big spatial data problems. This work presents a parallel algorithm to discover moving flock patterns in large trajectory databases. It is thought that new trends in distributed in-memory frameworks for spatial operations could help to speed up the detection of this kind of patterns.

2 RELATED WORK

Recently increase use of location-aware devices (such as GPS, Smart phones and RFID tags) has allowed the collection of a vast amount of data with a spatial and temporal component linked to them. Different studies have focused in discovering and analyzing this kind of collections [18, 20]. In this area, trajectory datasets have emerged as an interesting field where diverse kind of patterns can be identified [25, 29]. For instance, authors have proposed techniques to discover motion spatial patterns such as moving clusters [16], convoys [14] and flocks [3, 11]. In particular, [24] proposed BFE (Basic Flock Evaluation), a novel algorithm to find moving flock patterns in polynomial time over large spatio-temporal datasets.

A flock pattern is defined as a group of entities which move together for a defined lapse of time [3]. Applications to this kind of patterns are rich and diverse. For example, [5] finds moving flock patterns in iceberg trajectories to understand their movement behavior and how they related to changes in ocean's currents.

The BFE algorithm presents an initial strategy in order to detect flock patterns. In that, first it finds disks with a predefined diameter (ε) where moving entities could be close enough at a given time interval. This is a costly operation due to the large number of points and intervals to be analyzed ($O(2n^2)$ per time interval). The technique uses a grid-based index and a stencil to speed up the process, but the complexity is still high.

[5] and [23] use a frequent pattern mining approach to improve performance during the combination of disks between time intervals. Similarly, [22] introduce the use of plane sweeping along with

Conf '23, ,

ACM ISBN 978-1-4503-XXXX-X/18/06…$15.00
https://doi.org/XXXXXXX.XXXXXXX

binary signatures and inverted indexes to speedup the same process. However, the above-mentioned methods still keep the same strategy as BFE to find the disks at each interval.

[2] and [10] use depth-first algorithms to analyze the time intervals of each trajectory to report maximal duration flocks. However, these techniques are not suitable to find patterns in an on-line fashion.

Given the high complexity of the task, it should not be surprising the use of parallelism to increase performance. [9] use extreme and intersection sets to report maximal, longest and largest flocks on the GPU with the limitations of its memory model.

Indeed, despite the popularity of cluster computing frameworks (in particular whose supporting spatial capabilities [7, 21, 27, 28]) there are not significant advances in this area. At the best of our knowledge, this work is the first to explore in-memory distributed systems towards the detection of moving flock patterns.

3 METHODS

This section explains two alternative methodologies to find moving flock patterns in large spatio-temporal datasets using modern distributed frameworks to divide and parallelize the workload. Before to explain the details of our contributions we will explains the in a general manner the details of the current state-of-the-art to highlight the challenges and drawbacks at the moment to dealt with very large spatio-temporal datasets.

3.1 The BFE algorithm

The alternatives we will discuss later follows closely the steps explained at [24]. In this work, the authors proposed the Basic Flock Evaluation (BFE) algorithm to find flock patterns on trajectory databases. The details of the algorithm can be accessed at the source but we will explain the main aspects in a general view. It is important to clarify that BFE runs in two phases: firstly, it finds valid disks in the current time instant; secondly, it combines previous flocks with the recently discovered disks to extend them and report them.

The main inputs of the BFE algorithm are a set of points, a minimum distance ε which will define the diameter of the disks where the moving entities should lay, a minimum number of entities μ at each disk and a minimum duration δ which is the minimum number of time units the entities should be keep together to be considered a flock. Based on these inputs, figure 1 breaks down schematically the work flow of this phase where we can identify 4 general steps. The main goal of this phase is to find a set of valid disks at each time instant to allow further combinations with subsequent sets of disks coming in the future.

The main steps in phase one can are explained as follows:

(1) Pair finding: Using the ε parameter, the algorithm query the set of points to get the set of pairs which laid at a maximum distance of ε units. Usually, it is a distance self-join operation over the set of points using ε as the distance parameter. The query also pays attention to do not return pair duplicates. For instance, the pair between point p_1 and p_2 is the same that pair between p_2 and p_1 and just one of them should be reported (the id of each point is used to filter duplicates).

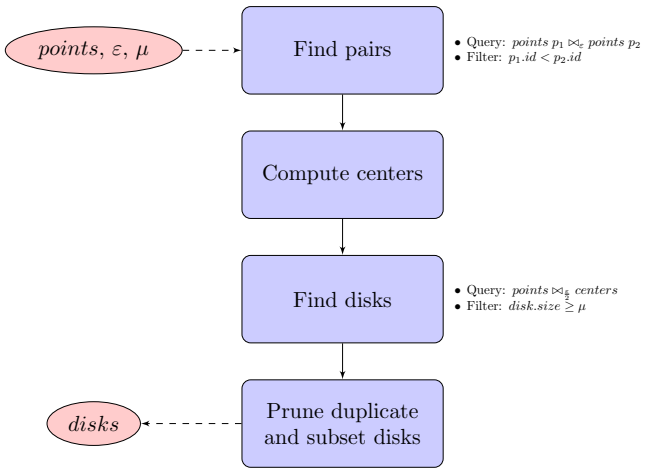


Figure 1: General steps in phase 1 of the BFE algorithm.

(2) Center computation: From the previous set of pairs, each tuple is the input of a simple computation to locate the centers of the two circles of radius $\frac{\varepsilon}{2}$ which circumference laid on the input points. The pseudocode of the procedure can be seen in appendix 1.

(3) Disk finding: Once the centers have been identified, a query to collect the points around those centers is needed in order to group the set of points which laid ε distance units each other. This is done by running a distance join query between the set of points and the set of centers using $\frac{\varepsilon}{2}$ as the distance parameter. Therefore, a disk will be defined by its center and the IDs of the points around it. At this stage, a filter is applied to remove those disks which collect less than μ entities around it.

(4) Disk pruning: It is possible than a disk collects the same set of points, or a subset, of the set of points of another disk. In such cases the algorithm should report just that one which contains the others. An explanation of the procedure can be seen in appendix 2.

It is important to note that BFE also proposes a grid index structure in this phase to speed up spatial operations. The algorithm divides the space area in a grid of ε side (see figure 2 from [24]). In this way, BFE just processes each grid and its 8 neighbor grids. It does not need to query grids outside of its neighborhood given that points in other grids are far away to affect the results.

The second phase is more straightforward. Figure 3 explains schematically what is done once the set of current disks (as explained in figure 1) is found at every time instant. This phase performs a recursion using the current set of disks and the previous set of flocks which comes from the previous time instant. Due to we do not know where and how far a group of entities can move in the next time instant, a cross product between both sets is required.

However, just disks which match the entities of previous flocks are kept. Indeed, only when the size of common items is greater than μ we keep the pair. Then, it updates the end time and duration of the filtered flocks. This information is used to decide if it is time to report a flock or keep it for further analysis. If the duration has reached the minimum duration δ, a flock is reported and removed

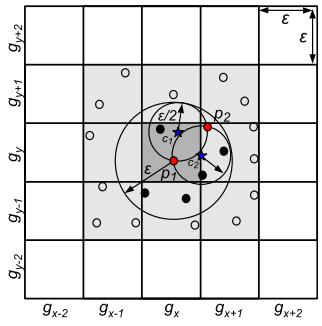


Figure 2: The grid-based index structure proposed at [24].

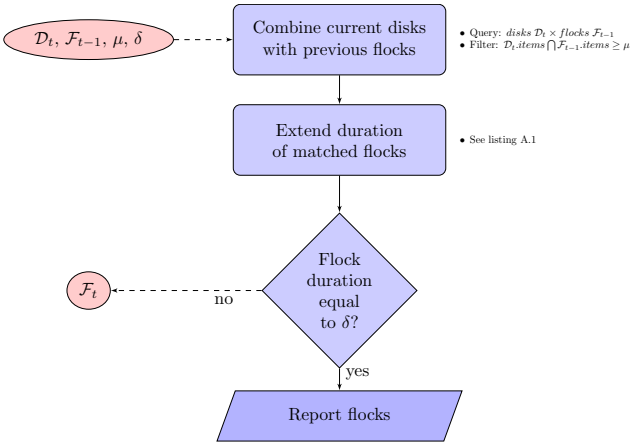


Figure 3: Steps in BFE phase two. Combination, extension and reporting of flocks.

from the set. The remaining flocks are sent to the next iteration for further evaluation in the next time instant.

Similarly, figure 4 illustrates the recursion and how the set of flocks from previous time instants feeds the next iteration. The example assumes a δ value of 3, so it starts reporting flock since time instant t_2. Note that time instants t_0 and t_1 are initial conditions. At the very beginning of the execution, we just can find valid disks at t_0 which immediately are transformed to flocks of duration 1 and feed the next time instant. At t_1 we can find a new set of disks $\mathcal{D}_1$ which combines with the set of previous flocks $\mathcal{F}_0$. It updates the information of each flock accordingly but it does not report any flock yet. From now on, subsequent time instants follows strictly the steps summarized on figure 3.

3.2 Bottlenecks in BFE and possible solutions

There are some steps during the execution of BFE which are particularly affected when it deals with very large datasets. Firstly, we will focus on phase 1 of BFE. In figure 5, the steps of this phase are illustrated for a sample dataset. You can see that the number of centers and disks found is considerable large in comparison with the final set of valid disks. Indeed, the finding of centers and the following operations grows quadratic depending on the number of points and possible pairs (which itself depend on the ε parameter).

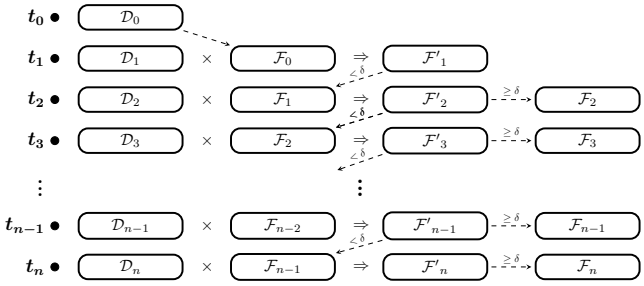


Figure 4: BFE phase 2 example explaining the recursion of the set of flocks along time instants and the initial conditions.

[24] claims that the number of centers, and consequent disks, to be evaluated is equal to $2|\tau|^2$ where τ is the number of trajectories. However, our experience show that there are a large number of duplicate and subset disks which are later pruned in the final stage. This behaviour is exacerbated no just in very large datasets but also in those with areas with high density of moving entities.

As solution for this issue, we proposed a partition strategy to divide the study area in smaller sections which can be evaluated in parallel. The strategy has three steps: first, a partition and replication stage, then the flock discovery in each local partition and finally the merge stage where we collect and unify the results. Let's explain each stage in more detail:

- Partition and Replication: Figure 6 shows a brief example of the partition and replication stage. Note that it is possible to use different types of spatial indexes (grids, r-tree, quadtree, etc.) to create spatial partitions over the input dataset. In the case of the example we use a quadtree which creates 7 partitions. Now, we need to ensure that each partition has access to all the required data to complete the finding of flocks locally. To accomplish this, all the points laying at ε distance of the border of its partition are replicated to adjacent partitions. At the right of figure 6, it can be seen each partition surrounded by a dotted area with the points which need to be copied from its neighbor partitions. At this point, each partition is ready to be submitted to different nodes for local processing.
- Local discovery: Once we have all the data we need at each partition we can run the steps of the phase 1 of the BFE algorithm locally (as were explained on figure 1). Actually, you can see that the example of figure 5 describes the execution using the partition 2's points of figure 6 as sample data.
- Merging: In order to merge back the results we will have to pay special attention to disks laying close to the border of each partition. We will show that if the position of disk centers lay inside of the current partition they will be safe to operate but those located in the expansion zone or outside of it will require to be treated to avoid duplicate reports. Disks with centers in the expansion zone will be repeated in contiguous partitions and, therefore, they will lead to duplication. In addition, it is possible that pairs of points generates disks with centers outside of the expansion zone. For example, Fig. 7a illustrates the case. Disks a' and b' are generated for points in partitions 1 and 2 respectively,

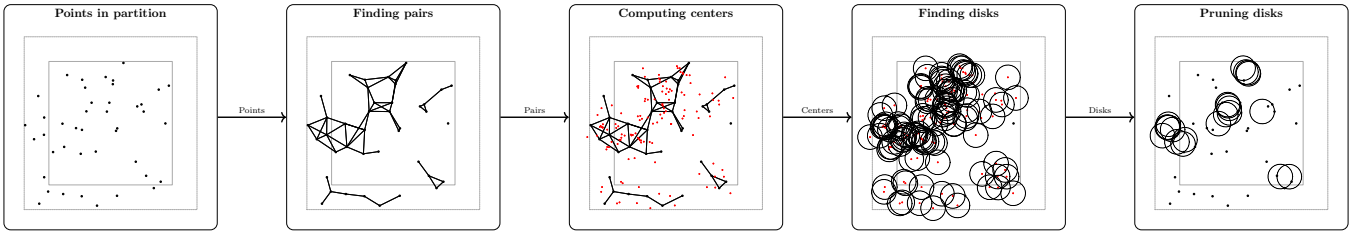


Figure 5: Example of BFE execution on a sample dataset.

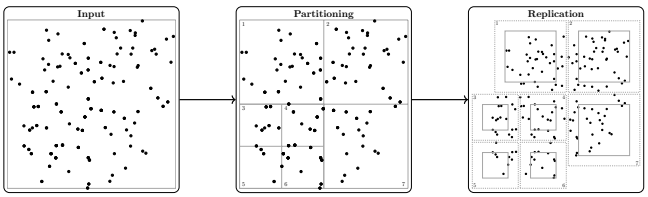


Figure 6: An example of partitioning and replication on a sample dataset.

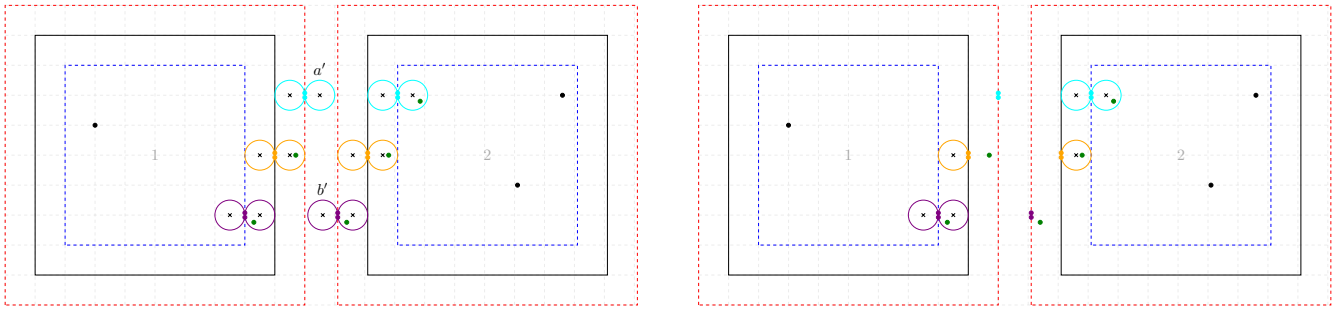
however both are located outside of their expansion zone boundaries and we should to avoid to report them twice. We present lemma 3.1 to show that we can safely remove those kind of disks.

Lemma 3.1. *A disk with its center laying in the expansion zone or outside of it can be discarded as they will be correctly evaluated by one of the partitions in its neighborhood.*

Proof. In order to support our proof we will define some concepts: First, we will divide the area of a partition in three zones to clarify our assumptions: we already talked about the *expansion zone* as the area beyond the border of a partition (between black line and dotted red line in figure 7a) and a width equal to ε. The *border zone* is a strip of width equal to ε touching the interior border of a partition. In figure 7a, it is compromised by the dotted blue and the black lines. The *safe zone* will be the remaining internal area in the partition which is not covered by the border zone. Second, we will call the contiguous partition which replicate a particular disk with the current one as the *replicated partition*. In figure 7a, the partition 2 is the replicated partition of partition 1 and vice versa.
From here, it will be clear that there is a symmetric relation between the disks in the border and expansion zones in the current partition and the disks in its replicated partition. We can certainly said that if a disk is located in the border zone of the current partition, it will be located in the expansion zone of its replicated partition. Similarly, any disk with a center laying outside of the expansion zone of the current partition will be located in the safe zone of its replicated partition. Keeping just the disks with centers laying in the current partition (border or safe zone) will be enough to ensure no lost of information (Fig. 7b). □

3.3 Additional improvements during local execution

Even the partition strategy could reduce the size of the points to be processed at each local partition , it is possible to introduce some filter techniques which allows to reduce the impact in the creation of centers and disks and subsequent pruning. This section explains the use of maximal cliques and minimum bounding circles (MBC) techniques as heuristics to improve the finding of disk at local level.
In graph theory, a clique is a subset of vertices forming a subgraph where all of their nodes share connections among them. It is said that the induced subgraph is complete. Similarly, a maximal clique happens when no additional vertex can be included in the current clique, that is, there is not a superset of vertices which could include the current clique [4]. Although it is know that finding all maximal cliques is a problem of high complexity, studies claims that working on relatively small real-worlds graphs, they can be found in near-optimal time [8].
Now, in our approach, it can be noted that after obtaining the set of pairs at the beginning of the BFE algorithm, we can see the result as a graph where the points are the vertices and the connections between nearby points are the edges. So it is possible to run an algorithm (i.e. Bron-Kerbosch) over that graph to get a set of maximal cliques. The advantage to find a set of cliques is that we will have access to set of points inside of each clique that by sure must generate one or more disks. Given the condition of the maximal cliques, all the points inside them are ε distance each other, and we can know that no other point must be included because that contradicts the definition of a maximal clique. That constitutes a kind of sub-partitioning technique where we could do additional processing. For example, a simple filter we can apply is to remove those cliques which number of points are less than the *mu* parameter given that those cliques will be unable to generate disks that fulfill that condition.
To introduce the next filter we need define the concept of minimum bounding circle (MBC). Algorithms for MBC finding aims to return the center and minimum radius possible of a circle that encloses all the items of a set of points in the plane. One of the most representative algorithm is proposed at [26] which is able to solve the problem in linear time.



(a) (b)

Figure 7: Merging strategy.

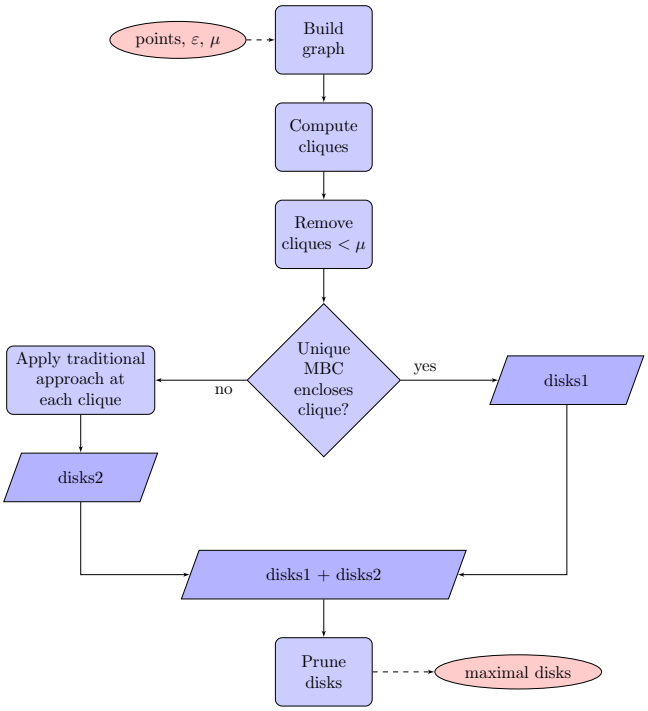


Figure 8: Flow chart for the maximal clique and MBC filters.

The intention to use MBC algorithms at this stage is to detect maximal cliques where all their points can be enclosed in one single disk. If that is the case, we could report directly the MBC result and save the costly computation of finding centers and pruning duplicates. This can bring important improvements especially in datasets where the density of points is high.

However, we have to clarify that when an unique MBC to cover all the points is not possible, the traditional steps (BFE algorithm) should be applied over them. Finally, disks from both branches are unified and a final pruning is done to remove possible duplication. This final step should not be costly because most of the pruning is done at maximal clique level.

Figure 8 shows a schematic flow chart of the steps for this alternative.

REFERENCES

[1] T. Amor, S. Reis, D. Campos, H. Herrmann, and J. Andrade. 2016. Persistence in Eye Movement during Visual Search. *Scientific Reports* 6 (2016), 20815.

[2] H. Arimura, T. Takagi, X. Geng, and T. Uno. 2014. Finding All Maximal Duration Flock Patterns in High-Dimensional Trajectories. (2014).

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

[4] C. Bron and J. Kerbosch. 1973. Algorithm 457: finding all cliques of an undirected graph. *Commun. ACM* 16, 9 (1973), 575–577.

[5] A. Calderon. 2011. *Mining Moving Flock Patterns in Large Spatio-Temporal Datasets Using a Frequent Pattern Mining Approach.* Master's thesis. University of Twente.

[6] G. Di Lorenzo, M. Sbodio, F. Calabrese, M. Berlingerio, F. Pinelli, and R. Nair. 2016. AllAboard: Visual Exploration of Cellphone Mobility Data to Optimise Public Transport. *IEEE TVCG* 22, 2 (2016), 1036–1050.

[7] A. Eldawy. 2014. SpatialHadoop: Towards Flexible and Scalable Spatial Processing Using Mapreduce. In *SIGMOD PhD Symposium*. 46–50.

[8] D. Eppstein, M. Löffler, and D. Strash. 2010. Listing All Maximal Cliques in Sparse Graphs in Near-Optimal Time. In *Algorithms and Computation*. 403–414.

[9] M. Fort, J. Antoni, and N. Valladares. 2014. A Parallel GPU-Based Approach for Reporting Flock Patterns. *IJGIS* 28, 9 (2014), 1877–1903.

[10] X. Geng, T. Takagi, H. Arimura, and T. Uno. 2014. Enumeration of Complete Set of Flock Patterns in Trajectories. In *IWGS*. 53–61.

[11] J. Gudmundsson and M. van Kreveld. 2006. Computing Longest Duration Flocks in Trajectory Data. In *ACM SIGSPATIAL*. 35–42.

[12] K. Holland, B. Wetherbee, C. Lowe, and C. Meyer. 1999. Movements of Tiger Sharks (Galeocerdo Cuvier) in Coastal Hawaiian Waters. *Marine Biology* 134, 4 (1999), 665–673.

[13] P. Huang and B. Yuan. 2015. Mining Massive-Scale Spatiotemporal Trajectories in Parallel: A Survey. In *TAKDDM*. Vol. 9441. Springer.

[14] H. Jeung, M. Yiu, X. Zhou, C. Jensen, and H. Shen. 2008. Discovery of Convoys in Trajectory Databases. *VLDB* 1, 1 (2008), 1068–1080.

[15] Hoyoung Jeung, Man Lung Yiu, and Christian S. Jensen. 2011. Trajectory Pattern Mining. In *Computing with Spatial Trajectories*. Springer, 143–177. 00038.

[16] P. Kalnis, N. Mamoulis, and S. Bakiras. 2005. On Discovering Moving Clusters in Spatio-Temporal Data. In *ASTD*. Springer, 364–381.

[17] F. La Sorte, D. Fink, W. Hochachka, and S. Kelling. 2016. Convergence of Broad-Scale Migration Strategies in Terrestrial Birds. *Royal Society: Biological Sciences* 283, 1823 (2016), 2588.

[18] Yee Leung. 2010. *Knowledge Discovery in Spatial Data.* Springer Science & Business Media. 00032.

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

[20] Harvey J. Miller and Jiawei Han. 2001. *Geographic Data Mining and Knowledge Discovery.* Taylor & Francis, Inc., Bristol, PA, USA. 00000.

[21] James N., Andrew A., Christopher N., Anthony F., Andrew H., and Michael R. 2015. GeoMesa: a distributed architecture for spatio-temporal fusion, Vol. 9473. SPIE, 128 – 140.

[22] P. Tanaka, M.. Vieira, and D. Kaster. 2016. An Improved Base Algorithm for Online Discovery of Flock Patterns in Trajectories. *JIDM* 7, 1 (2016).

[23] U. Turdukulov, A. Calderon, O. Huisman, and V. Retsios. 2014. Visual Mining of Moving Flock Patterns in Large Spatio-Temporal Data Sets Using a Frequent Pattern Approach. *IJGIS* 28, 10 (2014), 2013–2029.

[24] M. Vieira, P. Bakalov, and V. Tsotras. 2009. On-Line Discovery of Flock Patterns in Spatio-Temporal Data. In *ACM SIGSPATIAL*. 286–295.

[25] M. Vieira and V. Tsotras. 2013. *Spatio-Temporal Databases: Complex Motion Pattern Queries.* Springer.
[26] E. Welzl. 1991. Smallest enclosing disks (balls and ellipsoids). In *New Results and New Trends in Computer Science.* Springer, 359–370.
[27] D. Xie, F. Li, B. Yao, G. Li, L. Zhou, and M. Guo. 2016. Simba: Efficient In-Memory Spatial Analytics. In *ICMD.* 1071–1085.
[28] J. Yu, J. Wu, and M. Sarwat. 2016. A Demonstration of GeoSpark: A Cluster Computing Framework for Processing Big Spatial Data. In *ICDE.* 1410–1413.
[29] Y. Zheng and X. Zhou. 2011. *Computing with Spatial Trajectories.* Springer.

A CENTER COMPUTATION.

Algorithm 1 Find the centers of given radius which circumference laid on the two input points.

Require: Radius $\frac{\epsilon}{2}$ and points p_1 and p_2.
Ensure: Centers c_1 and c_2.

function FindCenters(p_1, p_2, $\frac{\epsilon}{2}$)
  $r^2 \leftarrow (\frac{\epsilon}{2})^2$
  $X \leftarrow p_1.x - p_2.x$
  $Y \leftarrow p_1.y - p_2.y$
  $d^2 \leftarrow X^2 + Y^2$
  $R \leftarrow \sqrt{|4 \times \frac{r^2}{d^2} - 1|}$
  $c_1.x \leftarrow X + \frac{Y \times R}{2} + p_2.x$
  $c_1.y \leftarrow Y - \frac{X \times R}{2} + p_2.y$
  $c_2.x \leftarrow X - \frac{Y \times R}{2} + p_2.x$
  $c_2.y \leftarrow Y + \frac{X \times R}{2} + p_2.y$
  return c_1 and c_2
end function

B DISK PRUNING.

Algorithm 2 Prune disks which are duplicate or subset of others.

Require: Set of disks D.
Ensure: Set of disks D' without duplicate or subsets.

function PruneDisks(D)
  $E \leftarrow \emptyset$
  for all disk d_i in D **do**
    $N \leftarrow d_i \cap D$
    for all disk n_j in N **do**
      if d_i contains all the elements of n_j **then**
        $E \leftarrow E \cup n_j$
      end if
    end for
  end for
  $D' \leftarrow D \setminus E$
  return D'
end function