I. 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 work load but, before to explain the details of our contributions, we will explains the in a genaral 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.

A. The BFE algorithm

The alternatives we will discuss later follows closely the steps explained at [1]. 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 explaing 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 minumun 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 subsequents set of disks coming in the future.

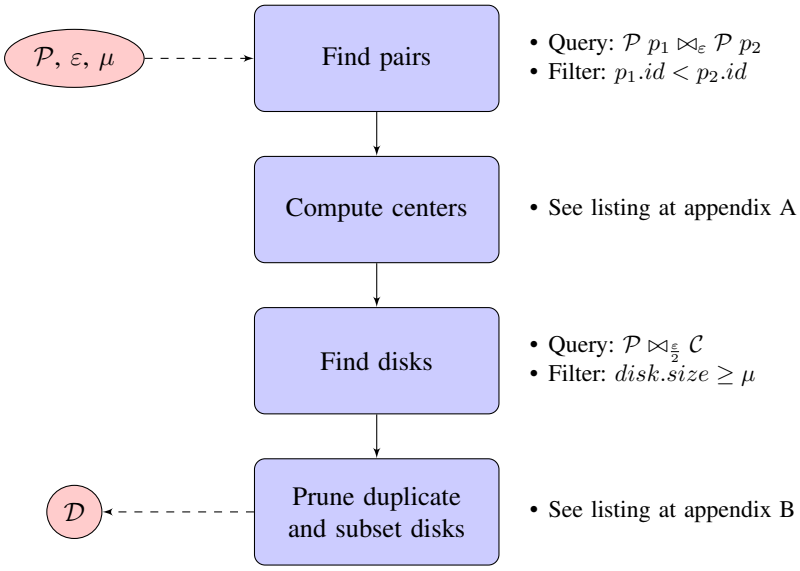


Fig. 1. General steps in phase 1 of the BFE algorithm.

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).
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 A.
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 B.

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 [1]). In this way, BFE just processes each grid and its

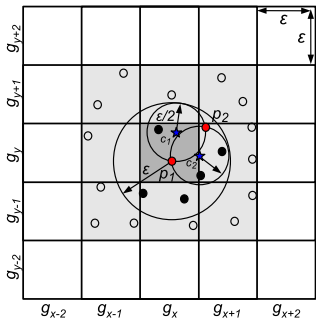


Fig. 2. The grid-based index structure proposed at [1].

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 straighforward. 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 requiered.

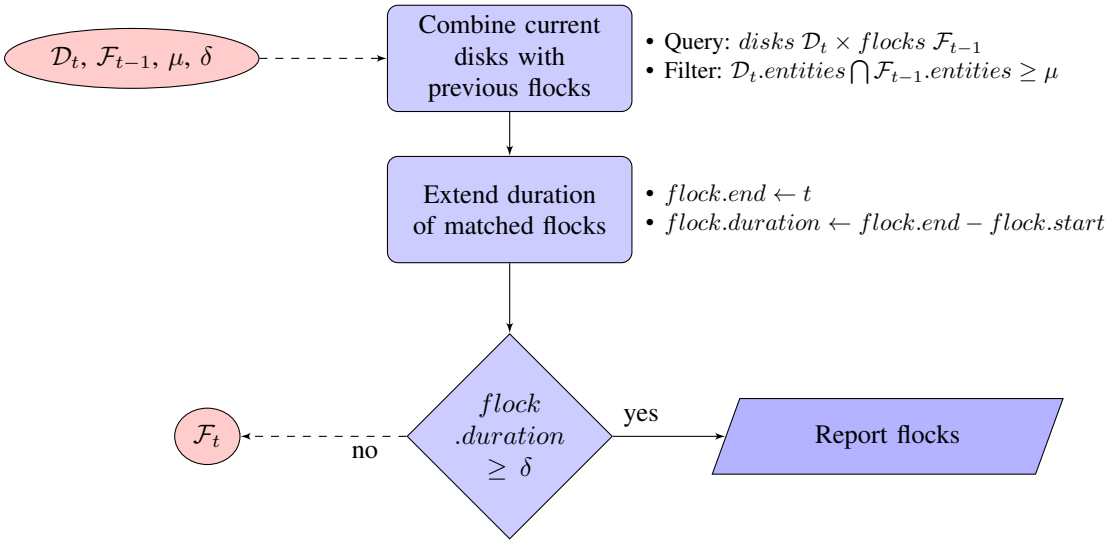


Fig. 3. Steps in BFE phase two. Combination, extension and reporting of flocks.

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 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 asumes 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 begining of the execution, we just can find valid disks at t_0 which inmediately are tranformed 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.

B. 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 paramenter).

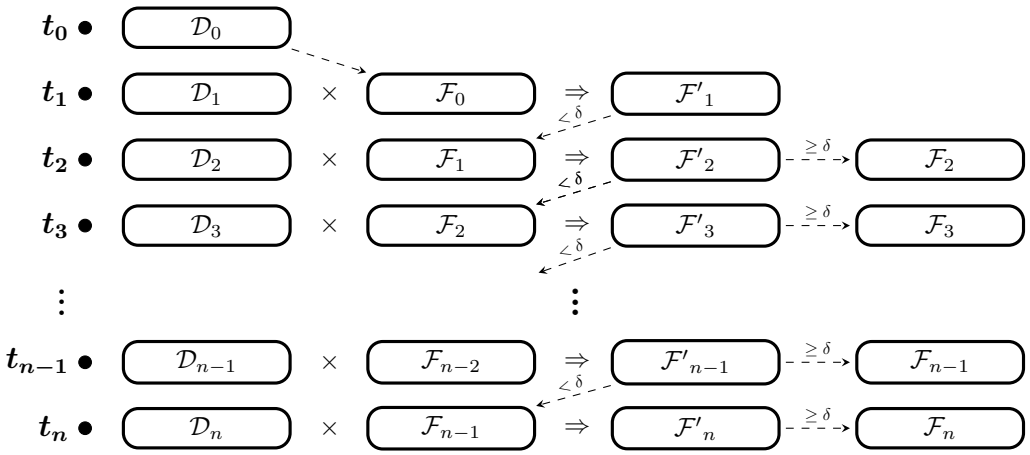


Fig. 4. BFE phase 2 example explaining the recursion of the set of flocks along time instants and the initial conditions.

[1] 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.

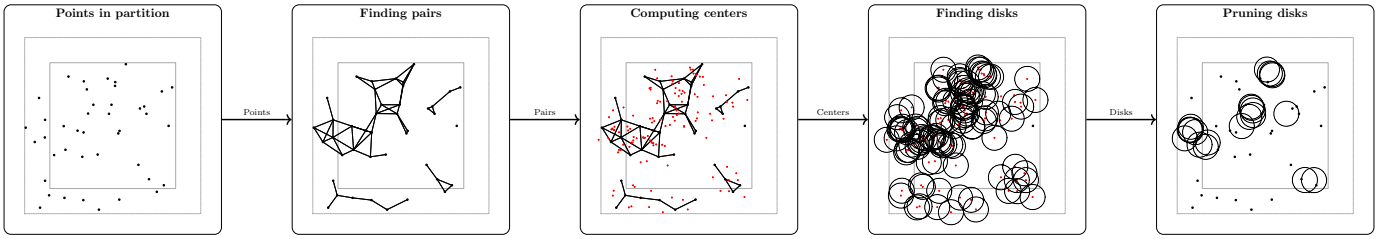


Fig. 5. Example of BFE execution on a sample dataset.

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.

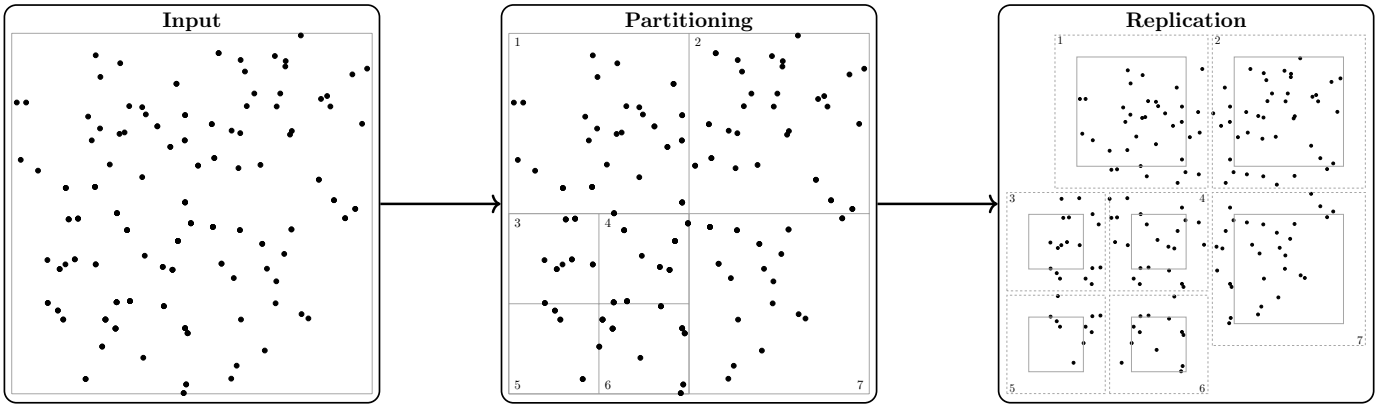
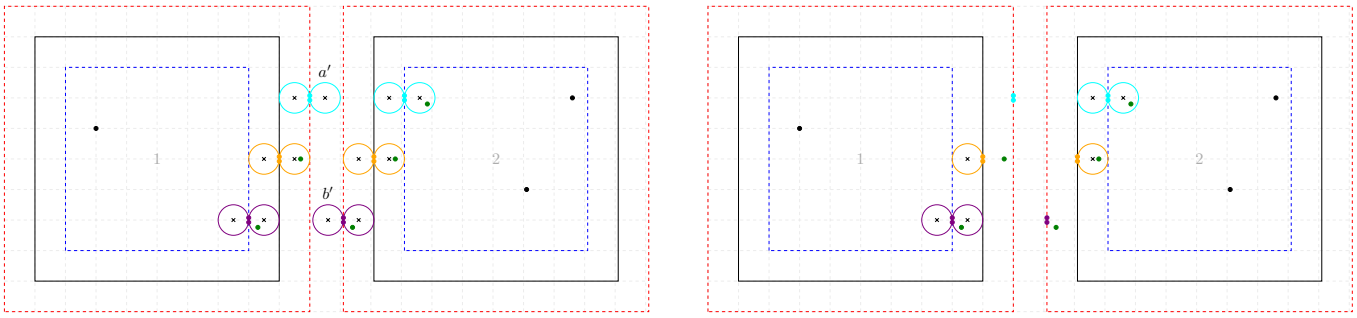


Fig. 6. An example of partitioning and replication on a sample dataset.

- 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, however both are located outside of their expansion zone boundaries and we should to avoid to report them twice. We present lemma 1 to show that we can safely remove those kind of disks.

  Lemma 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). □



(a) Disks in current and replicated partitions.

(b) Disks in current and replicated partitions after pruning.

Fig. 7. Merging strategy.

APPENDIX A
CENTER COMPUTATION.

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

Input: Radius $\frac{\varepsilon}{2}$ and points p_1 and p_2.

Output: Centers c_1 and c_2.

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

APPENDIX B
DISK PRUNING.

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

Input: Set of disks D.

Output: Set of disks D' without duplicate or subsets.

function PRUNEDISKS(D)
  $E \leftarrow \varnothing$
  For Each disk d_i in D **do**
    $N \leftarrow d_i \cap D$
    For Each 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

REFERENCES

[1] M. R. Vieira, P. Bakalov, and V. J. Tsotras, "On-line Discovery of Flock Patterns in Spatio-temporal Data," in *Proceedings of the 17th ACM SIGSPATIAL International Conference on Advances in Geographic Information Systems*. ACM, 2009, pp. 286–295.