UNIVERSITY OF CALIFORNIA
RIVERSIDE

Scalable Spatial Operations and Pattern Detection Using Distributed Systems

A Dissertation submitted in partial satisfaction
of the requirements for the degree of

Doctor of Philosophy

in

Computer Science

by

Andres Oswaldo Calderon Romero

November 2024

Dissertation Committee:

Dr. Vassilis Tsotras, Chairperson
Dr. Amr Magdy
Dr. Petko Bakalov
Dr. Ahmed Eldawy
Dr. Vagelis Hristidis

The Dissertation of Andres Oswaldo Calderon Romero is approved:

Committee Chairperson

University of California, Riverside

ABSTRACT OF THE DISSERTATION

Scalable Spatial Operations and Pattern Detection Using Distributed Systems

by

Andres Oswaldo Calderon Romero

Doctor of Philosophy, Graduate Program in Computer Science
University of California, Riverside, November 2024
Dr. Vassilis Tsotras, Chairperson

The Doubly Connected Edge List (DCEL) is an edge-list structure that has been widely utilized in spatial applications for planar topological computations. An important operation is the *overlay* which combines the DCELs of two input layers and can easily support spatial queries like the intersection, union and difference between these layers. Here we propose a distributed and scalable way to compute the overlay operation and its related supported queries. Previous sequential implementations do not scale and fail to complete for large datasets (for example the US census tracks). We address the issues involved in distributing the overlay operator and offer various optimizations. Using real datasets, our scalable solution can compute the overlay of very large datasets (32M edges) in few minutes.

Contents

List of Figures

List of Tables

Chapter 1

Background & Motivation

1.1 Background

Spatial data structures, crucial in fields like Geographic Information Systems (GIS), computational geometry, and spatial databases, enable efficient management and querying of geospatial information. Among these structures, the Doubly Connected Edge List (DCEL) stands out for its utility in topological computations on planar subdivisions. The DCEL data structure is a prominent choice for spatial applications that need to represent complex geospatial information due to its capacity to capture the relationships between vertices, edges, and faces. Its structure supports various geospatial operations, including intersection, union, and difference, making it suitable for spatial overlays.

DCEL has been widely applied in both practical and theoretical domains. Its applications span tasks in surveillance, such as the Art Gallery Problem, as well as in path finding and collision avoidance in robotics. Moreover, DCEL-based overlay methods provide an efficient means for integrating and analyzing thematic layers of geographic data, offering valuable insights for environmental studies, urban planning, and other geospatial analyses. Traditional implementations of DCEL-based overlays, however, operate sequentially and struggle to handle large-scale datasets, as is common in today's data-rich environments.

Parallel and distributed computing offer promising avenues for enhancing the scalability of DCEL-based overlay operations, especially with frameworks like Apache Spark that allow efficient partitioning and distributed processing. Leveraging these frameworks, spatial data scientists can potentially apply DCEL overlays to massive datasets, such as those derived from national census data or large ecological surveys, which may contain mil-

lions of polygons and edges. This scalability is crucial to ensure DCEL's continued relevance in increasingly data-intensive applications.

In a similar fashion, the last few decades have witnessed a transformative increase in the collection of spatio-temporal data, driven largely by the widespread use of GPS-enabled devices, smartphones, and the Internet of Things (IoT). This data proliferation has enabled novel insights into movement patterns across various domains, such as ecology, transportation, and urban planning. For instance, spatio-temporal data analysis is instrumental in identifying traffic congestion patterns in urban settings, monitoring migratory behavior in wildlife, and tracking the progression of weather events like hurricanes. More advanced analyses often focus on detecting not just isolated movement behaviors but group behaviors, where multiple entities move in close proximity over time.

Such group movement patterns —referred to as moving flocks, swarms, convoys, and clusters— capture complex collective behaviors. Detecting these patterns involves identifying groups that stay within a predefined distance over a set period, yielding insights into social dynamics, ecological phenomena, and urban mobility trends. The moving flock pattern, in particular, has garnered significant interest for its relevance across a broad spectrum of applications. From understanding migratory routes in animal populations to studying crowd dynamics in public spaces, the ability to detect and analyze moving flock patterns has proven invaluable. However, efficiently mining these patterns at scale remains a challenge due to the computational intensity of analyzing large, dense spatio-temporal datasets.

1.2 Motivation

The rise of big geospatial data demands scalable and efficient techniques to handle the complex overlay operations required for analyses in ecology, economics, and urban planning. Sequential implementations of DCEL-based overlays have proved inadequate when confronted with large datasets, often leading to performance bottlenecks and memory overflows. This limitation not only hampers the performance of geospatial analyses but also restricts the scope of investigations possible with current data.

The need for scalability in DCEL overlays is further amplified by the availability of spatial datasets that are inherently complex, such as road networks represented as individual line segments or census tracts with intricate boundaries. Processing such datasets requires

more than just traditional polygon overlay techniques, as these often need preprocessing steps like polygonization for line-based inputs. A distributed approach to DCEL overlays that can handle both large polygon layers and scattered line segment data would significantly expand the types of analyses available to spatial data scientists.

This research addresses these challenges by developing a distributed and scalable approach to compute DCEL overlays, capable of supporting various overlay operations and accommodating complex input data. By introducing novel partitioning strategies and optimizations tailored for DCEL overlays, this study aims to extend the utility of DCEL in spatial analysis, enabling rapid and scalable processing of large-scale geospatial datasets in a parallel computing environment.

Similarly, traditional approaches to identifying moving flock patterns, such as the Basic Flock Evaluation (BFE) algorithm, have established foundations for detecting group movement but are limited in scalability. These methods typically involve exhaustive spatial and temporal comparisons to track group cohesiveness, resulting in high computational costs. With the continued growth of data and the increasing density of spatial datasets, there is a pressing need for scalable, efficient solutions that can detect moving flocks in large, complex datasets.

This research is motivated by the limitations of current algorithms in processing large-scale dense datasets with high efficiency. Recognizing the potential of distributed computing frameworks and advanced partitioning strategies, this work proposes a novel, partition-based approach that enables scalable processing of moving flock patterns. By leveraging partitioning, replication, and parallel processing, the proposed methodology aims to overcome the bottlenecks of traditional methods. Additionally, integrating temporal joining strategies —such as the Cube-based approach— ensures efficient tracking of flock continuity across time while reducing computational overhead. The ultimate goal is to create a system that can handle extensive spatio-temporal datasets while maintaining accuracy in flock detection, enabling applications to extend to even larger and denser data environments.

Chapter 2

Scalable Processing of Moving Flock Patterns

2.1 Introduction

Technological advances in the past few decades have triggered an explosion in the collection of spatio-temporal data. The increasing popularity of GPS devices and smartphones, along with the emergence of new disciplines such as the Internet of Things (IoT) and high-resolution Satellite/UAS imagery, has made it possible to collect vast amounts of data with spatial and temporal components.

In tandem, interest in extracting valuable information from such large databases has also grown. Spatio-temporal queries about popular places or frequent events remain useful, but there has been growing interest in more complex patterns. In particular, patterns that describe the group behavior of moving objects over significant periods. Moving cluster [40], convoys [38], flocks [29] and swarm patterns [45] reveal how entities move together over a minimum time interval.

Applications for this type of information are both diverse and intriguing, particularly when dealing with trajectory datasets [37, 35]. Case studies span various domains, including transportation system management and urban planning [18], as well as ecology [42]. For example, [61] explores the identification of complex motion patterns to discover similarities in tropical cyclone paths. Similarly, [3] investigates eye movement trajectories to understand the strategies people use during visual searches. Additionally, [33] tracks

the behavior of tiger sharks along the coasts of Hawaii to gain insight into their migration patterns.

One particular pattern of interest is the moving flock pattern, which captures how objects move within close proximity for a given time period. Closeness is defined by a disk of a specified radius within which the entities must remain. Since this disk can be positioned anywhere, detecting such patterns is a non-trivial problem. In fact, [29] highlights that finding flock patterns where the same entities stay together over time is an NP-hard problem. To address this, [63] proposed the BFE algorithm, the first approach capable of detecting flock patterns in polynomial time.

Despite the increasing availability of data, current state-of-the-art techniques for mining complex movement patterns still struggle with the performance demands of large-scale spatial data. This work introduces a scalable approach designed to detect moving flock patterns in very large trajectory databases. By leveraging emerging trends in distributed frameworks for spatial operations we aim to significantly improve the speed and efficiency of detecting these patterns.

2.2 Related work

The recent increased use of location-aware devices (such as GPS, smartphones, and RFID tags) has enabled the collection of vast amounts of data with spatial and temporal components. Several studies have focused on discovering and analyzing these types of datasets [43, 48]. In this area, trajectory datasets have emerged as an interesting field where diverse kind of patterns can be identified [70, 64]. For instance, researchers have proposed techniques to discover spatial motion patterns such as moving clusters [40], convoys [38] and flocks [7, 29]. Specifically, [63] introduced BFE (Basic Flock Evaluation), an innovative algorithm designed to efficiently identify moving flock patterns in polynomial time across large spatio-temporal datasets.

A flock pattern is defined as a group of entities that move together over a specified time period [7]. The applications of such patterns are broad and diverse. For instance, [13] identifies moving flock patterns in iceberg trajectories to analyze their movement behavior and their relationship with changes in ocean currents.

The BFE algorithm provides an initial approach for detecting flock patterns. It begins by identifying disks with a predefined diameter (ε) where moving entities are suffi-

ciently close at specific time instants. This operation is computationally expensive due to the large number of points and time instances to be analyzed, with a complexity of $\mathcal{O}(2n^2)$ per time. Although the algorithm leverages a grid-based index and a stencil to accelerate this process, the overall complexity remains high.

Both [13] and [61] adopt a frequent pattern mining approach to enhance performance when combining disks across time instants. Similarly, [59] utilize plane sweeping techniques, binary signatures, and inverted indexes to further accelerate this process. However, these methods retain the core strategy of BFE for detecting disks at each time instant.

In contrast, [4] and [27] employ depth-first algorithms to analyze the time intervals of individual trajectories and report maximal duration flocks. However, these methods are less effective for dense datasets or those that involve large numbers of entities per time step, as they struggle to scale efficiently in such conditions.

Given the high computational demands of flock pattern detection, it is not surprising that parallelism has been employed to improve performance. For example, [24] use extreme and intersection sets to report maximal, longest, and largest flocks on GPUs, albeit with limitations imposed by the GPU's memory model.

Despite the increasing adoption of cluster computing frameworks, particularly those with spatial data capabilities [19, 68, 36, 66], significant advancements in this area remain limited. To the best of our knowledge, this work is the first to explore the detection of moving flock patterns in a scalable approach.

2.3 Background

Before discussing the details of our contributions, we first provide an overview of the current state-of-the-art. This will help highlighting their challenges and limitations in handling large spatio-temporal datasets, as described in the next Section.

2.3.1 The BFE sequential algorithm

The alternative approach we will discuss closely follows the steps outlined in [63]. In that work, the authors introduced the Basic Flock Evaluation (BFE) algorithm, designed to identify flock patterns in trajectory databases. While the full details of the algorithm can be found in the source, we will provide a general overview of the key aspects. It is important to note that the BFE algorithm operates in two phases: first, it identifies maximal disks

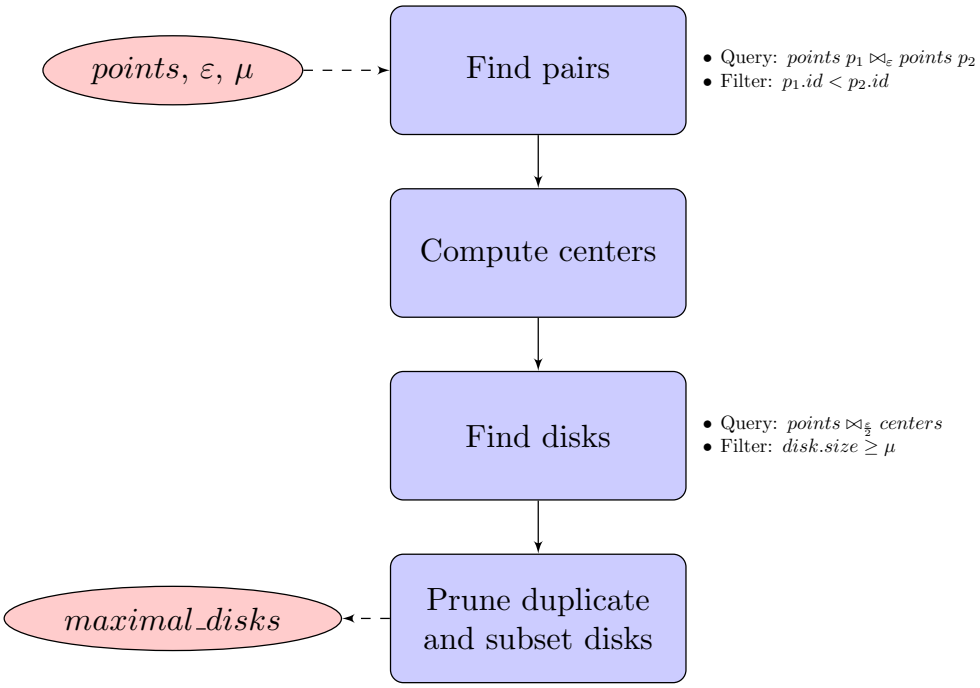

Figure 2.1: General steps in phase 1 of the sequential algorithm.

at the current time step; second, it extends and reports previous flocks by combining them with the newly discovered disks.

The input for the BFE algorithm consists of a set of points, a minimum distance ε (which defines the diameter of the disks where the moving entities must lie), a minimum number of entities μ per disk, and a minimum duration δ, representing the required time units that entities must remain together to be considered a flock. Based on this input, Figure 2.1 illustrates the workflow of the process in four general steps. The primary goal of this phase is to identify a set of disks at each time step, enabling the combination with future disks to form flocks.

The main steps in phase 1 follow:

1. Pair finding: The algorithm uses the parameter ε to identify pairs of points that are within a maximum distance of ε units from each other. This is achieved through a distance self-join operation on the set of points, using ε as the distance threshold. To avoid redundancy, duplicate pairs are eliminated; for example, the pair (p_1, p_2) is considered identical to the pair (p_2, p_1), so only one instance is retained. Point IDs

are used to filter out these duplicates efficiently.

2. Center computation: From the set of pairs obtained, each pair is used to compute the centers of two circles, each with a radius of $\frac{\varepsilon}{2}$, whose circumferences pass through the two points in the pair. The pseudo-code for this procedure is provided in Appendix 1.

3. Disk finding: Once the centers have been identified, a query is executed to gather the points within a distance of ε units from each center. This is accomplished by performing a distance join between the set of points and the set of centers, using $\frac{\varepsilon}{2}$ as the distance parameter. As a result, each disk is defined by its center and the IDs of the surrounding points. At this stage, a filter is applied to discard any disks that contain fewer than μ entities.

4. Disk pruning: It is possible for a disk to contain the same set, or a subset, of points as another disk. In such cases, the algorithm reports only that one disk which contains the other(s), referred to as the *maximal* disk. The procedure for identifying maximal disks is explained in Appendix 2.

It is important to note that BFE also employs a grid structure in this phase to optimize spatial operations. The algorithm divides the space into a grid, where each cell has a side length of ε (see Figure 2.2 from [63]). This structure allows BFE to limit its processing to each grid cell and its eight neighboring cells. There is no need to query cells beyond this neighborhood, as points in more distant cells are too far away to influence the results. Figure 2.3 shows an example of the Phase 1 steps using a sample dataset.

Figure 2.4 explains schematically phase 2. This phase performs a recursion using the set of disks found at time i and the set of partial flocks computed at the previous time instant $i-1$. As we do not know where and how far a group of points can move in the next time instant, this step performs a (temporal) join between both sets (partial flocks computed at time $i-1$ and maximal disks found in time i). When a join is performed, we check that the number of common points remains greater than μ, in which case the partial flock extends in time. A flock is reported in the answer if its duration has reached the minimum duration δ; otherwise, it remains as partial flock and it will be further evaluated

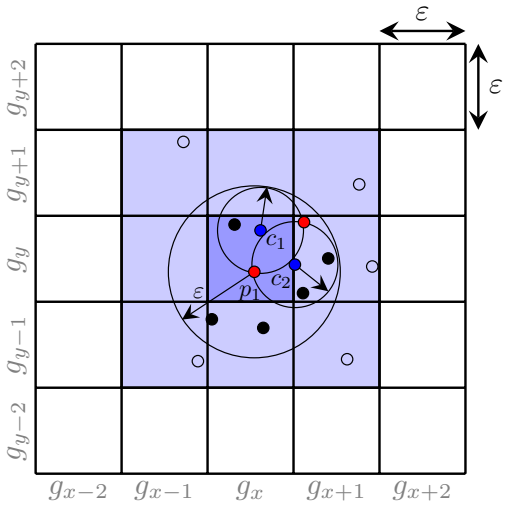

Figure 2.2: The grid-based structure proposed in [63].

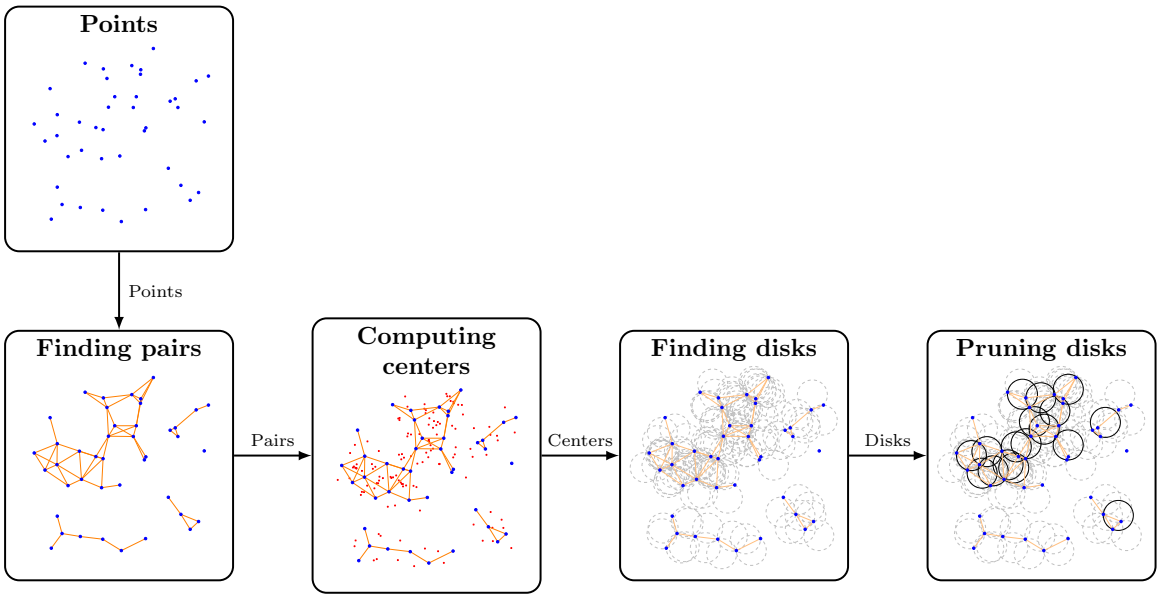

Figure 2.3: BFE Phase 1 example execution on a sample dataset.

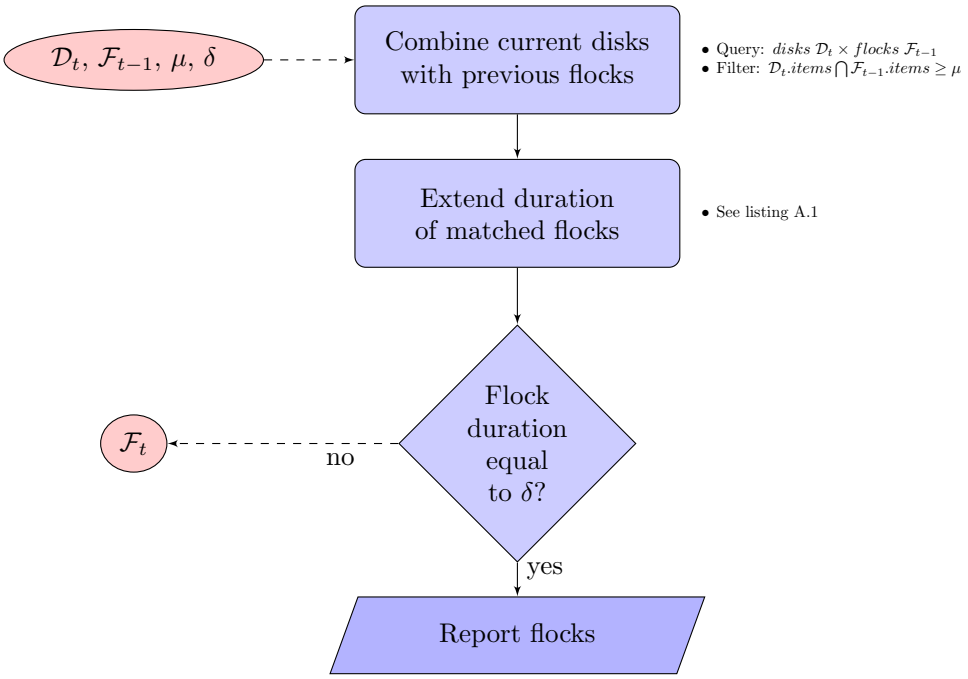

Figure 2.4: Steps in BFE phase two. Combination, extension and reporting of flocks.

during the next iteration at the next time instant.

Similarly, Figure 2.5 illustrates the recursive process and how the set of partial flocks from previous time instants feeds into the next iteration. The example assumes a δ value of 3, meaning flocks start being reported from time instant t_2. Note that time instants t_0 and t_1 are considered the initial conditions. At the start of the algorithm, maximal disks are identified at t_0, which are immediately transformed into partial flocks with a duration of 1 and then passed on to the next time instant. At t_1, a new set of maximal disks $\mathcal{D}_1$ is found and joined with the partial flocks from t_0, denoted as $\mathcal{F}_0$. The information for each partial flock is updated accordingly, including its duration and the points it contains. From this point onward, subsequent time instants follow the exact steps outlined in Figure 2.4.

2.3.2 The PSI sequential algorithm

The PSI algorithm, proposed by [59], follows a similar process to the BFE algorithm. However, instead of using a grid structure to index points within the area, PSI employs a sweep-line approach that processes points in order of their x-coordinates. For each visited point p, the algorithm considers a square of side length 2ε centered at p. It

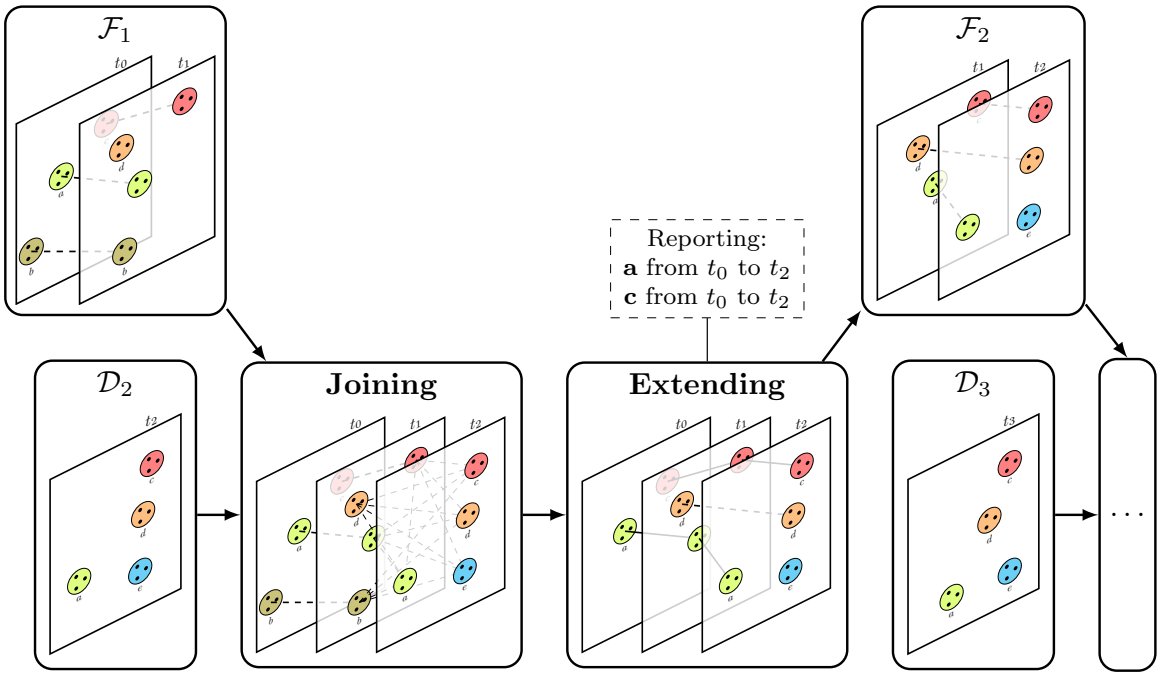

Figure 2.5: BFE Phase 2 example explaining the stages along time instants and the initial conditions.

only examines the points to the right of p that lie within two half-squares of side length ε, as illustrated by the shaded regions in Figure 2.6.

While BFE processes points inside a grid cell of side length ε along with its eight neighboring cells, PSI focuses on the points in these two half-squares. As a result, PSI more efficiently identifies the points relevant for detecting candidate pairs, centers, and disks. This indexing method has been shown to outperform BFE in most cases, with BFE offering similar or better performance only when ε values are relatively small. In such cases, the number of points to consider is smaller, and PSI still requires sorting the points for the sweep-line approach. Therefore, both approaches are considered in the following sections.

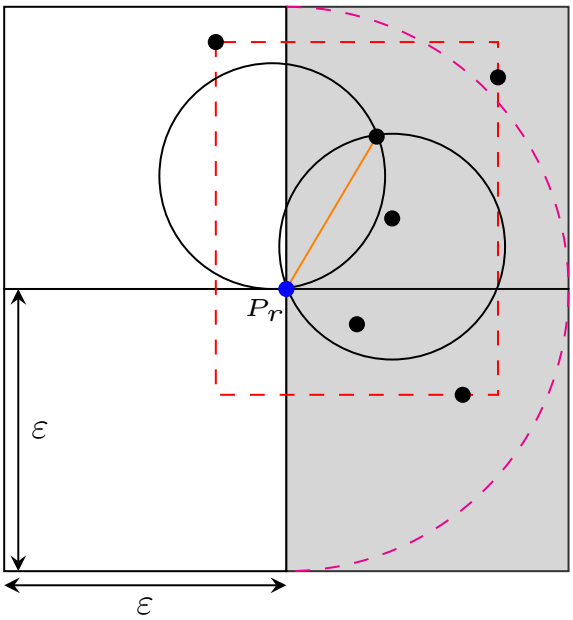

Figure 2.6: An example of the two half squares used in PSI algorithm.

Appendix

A Center computation.

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

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

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.

```
1: function PruneDisks(D)
2:     E ← ∅
3:     for all disk d_i in D do
4:         N ← d_i ∩ D
5:         for all disk n_j in N do
6:             if d_i contains all the elements of n_j then
7:                 E ← E ∪ n_j
8:             end if
9:         end for
10:    end for
11:    D' ← D \ E
12:    return D'
13: end function
```

C Clique and MBC approach.

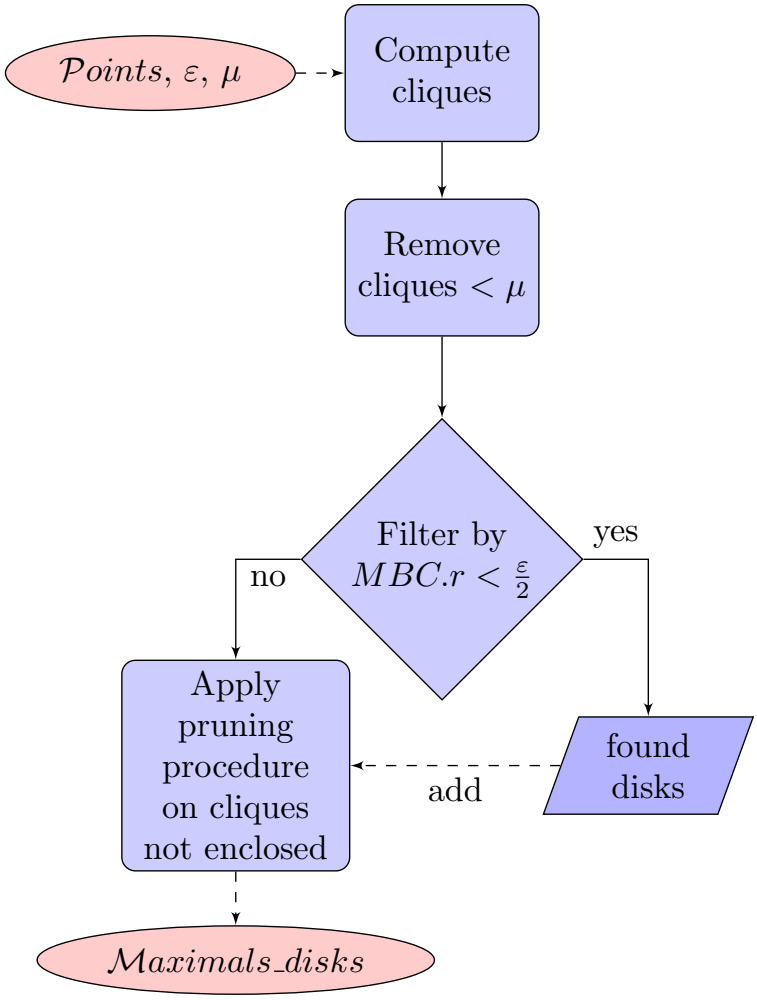

Figure .1: Schematic description of the Clique and MBC approach.

Bibliography

[1] Laila Abdelhafeez, Amr Magdy, and Vassilis Tsotras. DDCEL: Efficient Distributed Doubly Connected Edge List for Large Spatial Networks. In *2023 24th IEEE International Conference on Mobile Data Management (MDM)*, pages 122–131, 2023.

[2] Ablimit Aji, Fusheng Wang, Hoang Vo, Rubao Lee, Qiaoling Liu, Xiaodong Zhang, and Joel Saltz. Hadoop-GIS: A High Performance Spatial Data Warehousing System Over Map-Reduce. In *VLDB Journal*, 2013.

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

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

[5] G. Barequet. DCEL - A Polyhedral Database and Programming Environment. *Ijcga*, 08(05n06):619–636, 1998.

[6] N. Beckmann, H. Kriegel, R. Schneider, and B. Seeger. The R*-Tree: An Efficient and Robust Access Method for Points and Rectangles. In *Acm Sigmod Pods*, pages 322–331, New York, NY, USA, 1990. Association for Computing Machinery.

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

[8] E. Berberich, E. Fogel, D. Halperin, M. Kerber, and O. Setter. Arrangements on Parametric Surfaces. *Mathematics in Computer Science*, 4(1):67–91, 2010.

[9] M. Berg, O. Cheong, M. Kreveld, and M. Overmars. *Computational Geometry: Algorithms and Applications*. Springer, TU Eindhoven, P.O. Box 513, 2008.

[10] P. Boguslawski, C. Gold, and H. Ledoux. Modelling and analysing 3D buildings with a primal/dual data structure. *Isprs*, 66(2):188–197, 2011.

[11] D. Boltcheva, J. Basselin, C. Poull, H. Barthélemy, and D. Sokolov. Topological-based roof modeling from 3D point clouds. In *Wscg*, volume 28, pages 137–146, CZ 301 00 Plzen, 2020. Union Agency, Science Press.

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

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

[14] Andres Calderon, Vassilis Tsotras, and Amr Magdy. Scalable Overlay Operations over DCEL Polygon Layers. In *Proceedings of the 18th International Symposium on Spatial and Temporal Data*, SSTD '23, pages 85–95, New York, NY, USA, 2023. Association for Computing Machinery.

[15] J. Challa, P. Goyal, S. Nikhil, A. Mangla, S. Balasubramaniam, and N. Goyal. DD-Rtree: A dynamic distributed data structure for efficient data distribution among cluster nodes for spatial data mining algorithms. In *IEEE Big Data*, pages 27–36, 222 Rosewood Drive, Danvers, MA 01923., 2016. Ieee.

[16] L. Chew and K. Kedem. A convex polygon among polygonal obstacles. *Computational Geometry*, 3(2):59–89, 1993.

[17] V. Chvátal. A combinatorial theorem in plane geometry. *Combinatorial Theory*, 18(1):39–41, 1975.

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

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

[20] Ahmed Eldawy and Mohamed F Mokbel. Spatialhadoop: A Map-Reduce Framework For Spatial Data. In *Proceedings of the IEEE International Conference on Data Engineering, ICDE*, 2015.

[21] Martin Ester, Hans-Peter Kriegel, Jörg Sander, and Xiaowei Xu. A density-based algorithm for discovering clusters in large spatial databases with noise. In *Proceedings of the Second International Conference on Knowledge Discovery and Data Mining*, KDD'96, page 226–231. AAAI Press, 1996.

[22] Raphael Finkel and Jon Bentley. Quadtrees: A Data Structure for Retrieval on Composite Keys. *Acta Inf.*, 4:1–9, March 1974.

[23] E. Fogel, D. Halperin, and R. Wein. *CGAL Arrangements and Their Applications*. Springer Berlin, Heidelberg, 2012.

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

[25] W. Franklin, S. Magalhães, and M. Andrade. Data Structures for Parallel Spatial Algorithms on Large Datasets. In *ACM BigSpatial*, pages 16–19, Seattle, WA, USA, 2018. Acm.

[26] W. Freiseisen. Colored DCEL for boolean operations in 2D, 1998.

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

[28] GEOS Polygonizer Implementation. https://github.com/libgeos/geos.

[29] J. Gudmundsson and M. van Kreveld. Computing Longest Duration Flocks in Trajectory Data. In *Acm Sigspatial*, pages 35–42, 2006.

[30] A. Guttman. R-Trees: A Dynamic Index Structure for Spatial Searching. In *Acm Sigmod Icmd*, pages 47–57, New York, NY, United States, 1984. Association for Computing Machinery.

[31] Mordechai Haklay and Patrick Weber. Openstreetmap: User-generated street maps. *IEEE Pervasive computing*, 7(4):12–18, 2008.

[32] Erik G Hoel and Hanan Samet. Data-parallel polygonization. *Parallel Computing*, 2003.

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

[34] R. Holmes. The DCEL Data Structure for 3D Graphics, 2021.

[35] P. Huang and B. Yuan. Mining Massive-Scale Spatiotemporal Trajectories in Parallel: A Survey. In *Takddm*, volume 9441. Springer, 2015.

[36] J. Hughes, A. Annex, C. Eichelberger, A. Fox, A. Hulbert, and M. Ronquest. Geomesa: a distributed architecture for spatio-temporal fusion. In *Defense + Security Symposium*, 2015.

[37] H. Jeung, M. Yiu, and C. Jensen. Trajectory Pattern Mining. In *Computing with Spatial Trajectories*, pages 143–177. Springer, 2011.

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

[39] JTS Polygonizer Implementation. https://github.com/locationtech/jts.

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

[41] Daniel Krajzewicz, Jakob Erdmann, Michael Behrisch, and Laura Bieker. Recent Development and Applications of SUMO - Simulation of Urban MObility. *International Journal On Advances in Systems and Measurements*, 5(3&4):128–138, December 2012.

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

[43] Y. Leung. *Knowledge Discovery in Spatial Data*. Springer, 2010.

[44] Y. Li, A. Eldawy, J. Xue, N. Knorozova, M. Mokbel, and R. Janardan. Scalable computational geometry in Map-Reduce. *VLDB*, 28(1):523–548, 2019.

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

[46] S. Magalhães, M. Andrade, W. Franklin, and W. Li. Fast exact parallel map overlay using a two-level uniform grid. In *ACM BigSpatial*, pages 45–54, New York, NY, USA, 2015. Association for Computing Machinery.

[47] K. Mehlhorn and S. Näher. LEDA: a platform for combinatorial and geometric computing. *Communications of the ACM*, 38(1):96–102, 1995.

[48] H. Miller and J. Han. *Geographic Data Mining and Knowledge Discovery*. Taylor & Francis, Inc., 2001.

[49] D. Muller and F. Preparata. Finding the intersection of two convex polyhedra. *Theoretical Computer Science*, 7(2):217–236, 1978.

[50] J. Nievergelt, H. Hinterberger, and K. Sevcik. The Grid File: An Adaptable, Symmetric Multikey File Structure. *ACM Trans. Database Syst.*, 9(1):38–71, 1984.

[51] J. O'Rourke. *Art Gallery Theorems and Algorithms*. Oxford University Press, United States, 1987.

[52] OpenStreetMap Data Extracts. http://download.geofabrik.de/.

[53] Varun Pandey, Alexander van Renen, Andreas Kipf, and Alfons Kemper. How Good Are Modern Spatial Libraries? *Data Science and Engineering*, 2021.

[54] F. Preparata and M. Shamos. *Computational Geometry: An Introduction*. Springer, New York, NY, 1985.

[55] S. Puri, D. Agarwal, X. He, and S. Prasad. MapReduce Algorithms for GIS Polygonal Overlay Processing. In *Ieee Ipdps*, pages 1009–1016, Cambridge, MA, USA, 2013. Ieee.

[56] S. Puri and S. Prasad. Efficient Parallel and Distributed Algorithms for GIS Polygonal Overlay Processing. In *Ieee Ipdps*, pages 2238–2241, Usa, 2013. IEEE Computer Society.

[57] I. Sabek and M. Mokbel. On Spatial Joins in MapReduce. In *Acm Sigspatial*, pages 1–10, New York, NY, USA, 2017. Association for Computing Machinery.

[58] H. Samet. *The Design and Analysis of Spatial Data Structures*. Wesley, 75 Arlington Street, Suite 300 Boston, MA, United States, 1990.

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

[60] Etsuji Tomita, Yoichi Sutani, Takanori Higashi, and Mitsuo Wakatsuki. A Simple and Faster Branch-and-Bound Algorithm for Finding a Maximum Clique with Computational Experiments. *IEICE Transactions on Information and Systems*, E96.D(6):1286–1298, 2013.

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

[62] U.S. Street Network Shapefiles, Node/Edge Lists, and GraphML Files. https://doi.org/10.7910/DVN/CUWWYJ.

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

[64] M. Vieira and V. Tsotras. *Spatio-Temporal Databases: Complex Motion Pattern Queries*. Springer, 2013.

[65] E. Welzl. Smallest enclosing disks (balls and ellipsoids). In *New Results and New Trends in Computer Science*, pages 359–370. Springer, 1991.

[66] D. Xie, F. Li, B. Yao, G. Li, L. Zhou, and M. Guo. Simba: Efficient In-Memory Spatial Analytics. In *Icmd*, pages 1071–1085, 2016.

[67] Simin You, Jianting Zhang, and Le Gruenwald. Large-scale Spatial Join Query Processing In Cloud. In *Proceedings of the IEEE International Conference on Data Engineering, ICDE*, 2015.

[68] J. Yu, J. Wu, and M. Sarwat. A Demonstration of GeoSpark: A Cluster Computing Framework for Processing Big Spatial Data. In *Icde*, pages 1410–1413, 2016.

[69] Jia Yu, Zongsi Zhang, and Mohamed Sarwat. Spatial Data Management in Apache Spark: the GeoSpark Perspective and Beyond. *GeoInformatica*, 2018.

[70] Y. Zheng and X. Zhou. *Computing with Spatial Trajectories*. Springer, 2011.