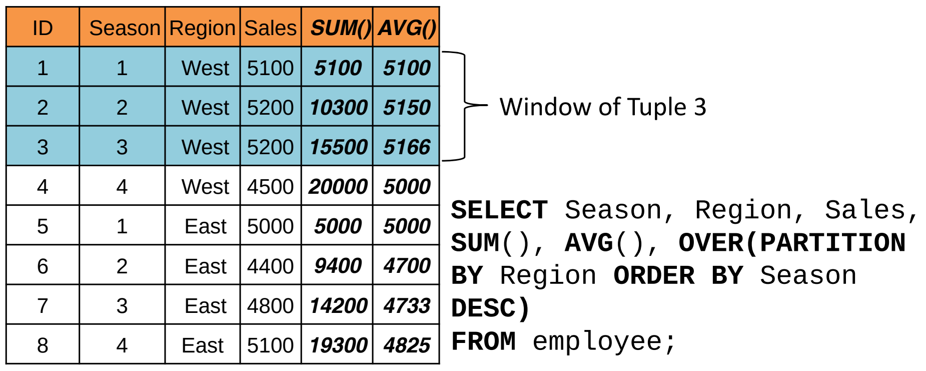
Chapter 1

Introduction

With the maturity of database technologies, many applications today collect and store data from all domains at unprecedented scale. For example, billions of social network users and their activities are collected in the form of *graphs*; Thousands of sensor ticks are collected every second in the form of *time series*; Hundreds of millions of spatiotemporal points are collected as *trajectories*. Flooded by the tremendous amount of data, it is becoming increasingly critical to efficiently conduct analytics to discover useful insights. However, traditional SQL analytics, which comprises primary operations (e.g., partition, sorting and aggregation), is unable to cope with domain-specific analytics such as graph traversal and pattern detection. In practice, expressing these domain-specific analytics using SQL query often involves complex joins which are hard to optimize. In this thesis, we introduce the concept of *neighborhood analytics* which originates from the recognized SQL window functions. We study the usage of *neighborhood analytics* in different data domains and design efficient algorithms to cope with today's big data.

1.1 Neighborhood Analytics

Neighborhood analytics aims to provide summaries of each object over its vicinity. In contrast to aggregating the entire collection of data as a whole, neighborhood analytic provides a personalized view for each object from its own perspective. Neighborhood data analytics originates from the window function defined in SQL which is illustrated in Figure 1.1.



ID	Season	Region	Sales	***SUM()***	***AVG()***
1	1	West	5100	***5100***	***5100***
2	2	West	5200	***10300***	***5150***
3	3	West	5200	***15500***	***5166***
4	4	West	4500	***20000***	***5000***
5	1	East	5000	***5000***	***5000***
6	2	East	4400	***9400***	***4700***
7	3	East	4800	***14200***	***4733***
8	4	East	5100	***19300***	***4825***

Window of Tuple 3

```
SELECT Season, Region, Sales,
SUM(), AVG(), OVER(PARTITION
BY Region ORDER BY Season
DESC)
FROM employee;
```

Figure 1.1: A SQL window function computing running sum and average of sales. The window of tuple 3 is highlighted.

As shown in the figure, the sales report contains six columns: "ID", "Season", "Region" and "Sales" are the *facts*, "`sum`()" and "`avg`()" are the *analytics* representing the running sum and average. A window function is represented by the `over` keyword. In this context, the window of a tuple o_i contains another tuple o_j if o_i and o_j are in the same "region" and the"season" of o_j is prior to the season of o_i. The window of tuple-3 is highlighted. Apart from this example, there are also many other usages of the window functions in the relational context [15]. Being aware of the success of the window functions, SQL 11 [74] standard incorporates "`LEAD`" and "`LAG`" keywords to offer fine-grained specifications on a tuple's window.

Despite the usefulness, there are few works reporting the usage of window functions in data domains such as graphs, sequence data and trajectories. This may be due to the requirement of *sorting* in the window functions. For example, in Figure 1.1, objects need to be sorted according to "Season", and then the window of each object is implicitly formed based on the sorted order. However, in domains like graphs and time series data, sorting may be ambiguous and even undefined.

To broaden the usages of the window functions in other data domains, we propose the *neighborhood analytics* in a more general sense. Given a set of objects (such as tuples in relational tables, vertexes in graphs, moving objects in trajectories), the neighborhood analytics is a composite function ($\mathcal{F} \circ \mathcal{N}$) applied on every object. Here, $\mathcal{N}$ is the *neighborhood function*, which contains the related objects (i.e., vicinity) of an object; $\mathcal{F}$ is the *analytic function*, which could be aggregation, ranking, pattern matching, etc. Apparently, the SQL window function is a special case of the neighborhood analytics. For example, the window function in Figure 1.1 can be represented as $\mathcal{N}(o_i) = \{o_j | o_i.season > o_j.season \wedge o_i.region = o_j.region\}$ and $\mathcal{F} = \texttt{avg}$. By relaxing the sorting constraint, neighborhood analytics gains an enriched semantic and can be applied on many other data domains.

1.2 Thesis Scope

In this thesis, we explore the neighborhood analytics in three prevalent data domains, namely **attributed graph**, **sequence data** and **trajectory**. To provide useful analytics, we define the following two intuitive instances of the neighborhood function:

Distance Neighborhood: the neighborhood is defined based on numeric distance, that is $\mathcal{N}(o_i, K) = \{o_j | \texttt{dist}(o_i, o_j) \leq K\}$, where `dist` is a distance function and K is a distance threshold.

Comparison Neighborhood: the neighborhood is defined based on the comparison of objects, that is $\mathcal{N}(o) = \{o_i | o.a_m \ \texttt{cmp} \ o_i.a_m\}$, where a_m is an attribute of object and `cmp` is a binary comparator (i.e., $=, <, >, \leq, \neq, \geq$).

Despite the simplicity of these two neighborhood functions, they can weave many useful analytics as we shall see in the remaining part of the thesis.

1.3 Thesis Contributions

In brief, the contribution of this thesis is twofold. First, by sewing different $\mathcal{N}$s and $\mathcal{F}$s, several novel neighborhood based queries are proposed for *graph*, *sequence data* and *trajectory* respectively. Second, this thesis deals with the efficiency issues in deploying the corresponding analytic queries to handle data of large scale data. The roadmap of this thesis is shown in Figure 1.2.

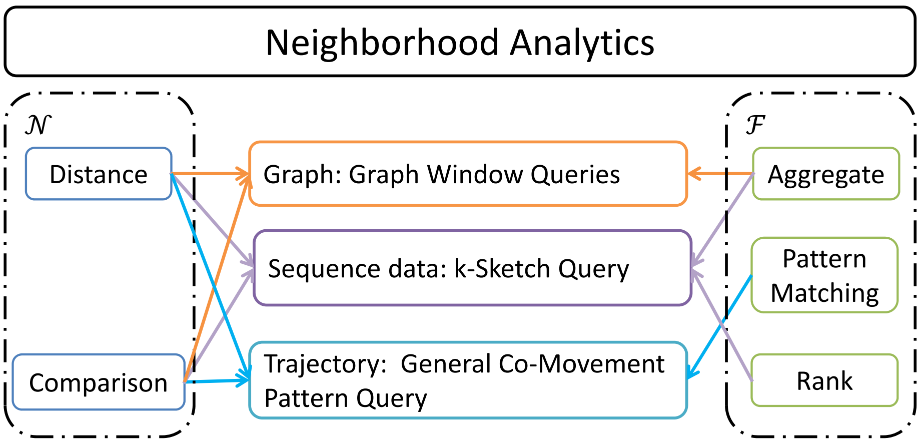

Neighborhood Analytics
$\mathcal{N}$
Distance
Comparison
Graph: Graph Window Queries
Sequence data: k-Sketch Query
Trajectory: General Co-Movement Pattern Query
$\mathcal{F}$
Aggregate
Pattern Matching
Rank

Figure 1.2: The road map of this thesis. There are three major contributions as highlighted in the center. Each contribution is a neighborhood based analytics with different $\mathcal{N}$ and $\mathcal{F}$ as indicated by arrows.

In a nutshell, we propose three neighborhood based queries in respective data domains. In *graph*, we define the Graph Window Query which summarizes the vicinity of each vertex. The query utilizes both distance neighborhood and comparison neighborhood to facilitate both general graphs and direct acyclic graphs. In *sequenced data*, we propose a k-Sketch Query to summarize a subject's history. The k-Sketch query builds on a nested *distant* and *comparison* neighborhood based pattern called *rank-aware streak*. In *trajectory*, we propose a General Co-movement Pattern (GCMP) Query to discover co-moving behaviors among moving objects. The GCMP query leverage the neighborhood notion to unify existing co-moving patterns. In the following parts of this section, we present our contributions in detail.

1.3.1 Graph Window Queries

The first piece of the thesis deals with neighborhood analytics on graph data. Nowadays information network are typically modeled as attributed graphs where the vertexes correspond to objects and the edges capture the relationships between these objects. As vertexes embed a wealth of information (e.g., user profiles in social networks), there are emerging demands on analyzing these data to extract useful insights. We propose the concept of *window analytics* for attributed graph and identify two types of such analytics as shown in the following examples:

k-hop Window: The k-hop neighbors of a vertex form its k-hop window. Since the k-hop neighbors are the most structurally relevant vertexes to the vertex, analytics on the information from the k-hop window would be beneficial. Typical analytic queries include summarizing the related connections' distribution among different companies, and computing age distribution of the related friends can be useful.

Topological Window: The topological neighbors are defined in the context of Directed Acyclic Graph (DAG). In DAGs, topological neighbors are composed of all the ascendant vertexes of a vertex. The topological neighbors represent the most influential vertexes of a given vertex. Since DAGs are often found in biological networks, topological window would be helpful to analyze the the statistics of molecules of each biological proteins' pathway.

The two *windows* shown in the above examples are essentially neighborhood functions defined for each vertex. Specifically, let $G = (V, E, A)$ be an attributed graph, where V is the set of vertexes, E is the set of edges, and each vertex v is associated as a multidimensional points $a_v \in A$ called attributes. The *k-hop* window is a *distance* neighborhood function, i.e., $\mathcal{N}_1(v, k) = \{u | \texttt{dist}(v, u) \leq k\}$, which captures the vertexes that are k-hop nearby. The *topological* window, $\mathcal{N}_2(v) = \{u|u \in v.ancestor\}$, is a *comparison* neighborhood function that captures the ancestors of a vertex in a directly acyclic graph. The analytic function $\mathcal{F}$ is an aggregate function (sum, avg, etc.) on A.

Apart from demonstrating the useful use-cases on these two windows, we also investigate how Graph Window Query processing can be efficiently supported. We propose two different types of indexes: Dense Block Index (DBIndex) and Inheritance Index (I-Index). The DBIndex and I-Index are specially optimized to support k-hop window and topological window processing. These indexes integrate the aggregation process with partial work sharing techniques to achieve efficient computation. In addition, we develop space-and-performance efficient techniques for the index construction. Notably, DBIndex saves upto 80% of the index construction time as compared to the state-of-the-art competitor and upto 10x speedup in query processing.

1.3.2 k-Sketch Query on Sequence Data

The second piece of this thesis explores the neighborhood analytics on sequence data. As part of the sequence data analysis, summarizing a subject's history with sensational patterns is an important and revenue-generating task in a plethora of applications such as computational journalism [21, 75], automatic fact checking [33, 62], and perturbation analysis [63]. An outstanding example of such patterns is the *streak* [75], which is commonly found in stock and sports reports. For instance:

1. [STOCK]:"Apple Inc. has an *average* price of USD 115.5 in the *last week*"
2. [SPORTS]: "Kobe has scored *at least* 60 points in *three straight games*"

In general, a streak is constructed from two concepts: an *aggregate function* (e.g., average, min) applied on *consecutive events* (e.g., seven days, three games). However, the streak itself does not embed the strikingness information, which is limited to represent a sensational pattern. For example, *Streak 2* would not be striking if all NBA players were able to score over 60 points. On the contrary, knowing that most NBA players only score 20 points in a game, *Streak 2* is indeed quite striking. Therefore, the strikingness of a streak should be measured by comparing with other streaks. Based on this observation, we propose a rank-aware pattern named *ranked-streak* which measures the strikingness of a streak by comparing among all streaks under same condition (i.e., streak length).

Technically, the ranked-streak can be viewed as a joint neighborhood function. Let $e_s(t)$ denote the event of subject s at time t. Then the ranked-streaks are generated using neighborhood functions in a two-step manner:

1. a *distance neighborhood* $\mathcal{N}_1(o_i, w) = \{o_j | o_i.t - o_j.t \leq w\}$ groups a consecutive w events for each event. Let $\overline{v}$ be the aggregate value associated with $\mathcal{N}_1$, then the output of this step is a set of *streak*s of the form $n = \langle o_i, w, t, \overline{v} \rangle$.

2. a *comparison neighborhood* $\mathcal{N}_2(n_i) = \{n_j | n_j.w = n_i.w \wedge n_i.\overline{v} \geq n_j.\overline{v}\}$ ranks a subject's streak among all other streaks with the same streak length. The result of this step is a tuple $\langle o_i, w, t, r \rangle$, where r is the *rank*.

As the ranked-streak contains the relative position of the streak among its cohort, it provides a quantitative measure of the strikingness. For example, the rank-aware version of *Streak 2* would be "Kobe has scored at least 60 points in three straight games, which is best in the league". This clearly suggests that *Streak 2* is striking.

On the basis of the ranked-streaks, we study the problem of effectively summarizing a subject history. We notice that, for a subject with n events, there are $O\binom{n}{2}$ streaks. In real life, "Kobe" has played $1,000$ games which may produce near half million streaks. Such a large number of streaks is too overwhelming to represent a subject's history. Hence, we are motivated to propose a k-Sketch query that selects k ranked-streaks which best represent a subject's history. To find the qualified streaks, we design a novel scoring which considers both the events covered of the streaks and the ranks of the streaks.

In this thesis, we extensively study the technical issues in processing k-Sketch queries in both online and offline scenarios. In the offline scenario, we design two pruning techniques which largely reduces the streaks enumerated in generating ranked-streaks. Then, we adopt a $(1 - 1/e)$-approximate sketch selection algorithm by utilizing the submodularity of the k-Sketch query. In the online scenario, we design an *online-streak bound* to avoid evaluating many unnecessary streaks. Furthermore, we propose a 1/8-approximate algorithm to facilitate efficient sketch maintenance. In the experimental study, we compare our solutions with baselines using four real datasets, and the results demonstrate the efficiency and effectiveness of our proposed algorithms: the running time achieves up to 500x speedup as compared to baseline and the quality of the detected sketch is endorsed by the anonymous users from Amazon Mechanical Turk[1].

1.3.3 Co-Movement Pattern Query on Trajectory Data

The third piece of the thesis studies the neighborhood analytics in the trajectory domain. In trajectory analysis, an important mining task is to discover traveling patterns among moving objects. A traveling pattern is often determined by the neighborhoods of moving objects. One of the prominent examples is the *co-movement pattern* [42,78]. A co-movement pattern refers to a group of moving objects traveling together for a certain period of time. A pattern is significant if the group size exceeds M and the length of duration exceeds K. The group is defined based on the spatial proximity of objects, which can be seen as the spatial neighbors of each object.

Rooted from the basic movement definition and driven by different mining applications, there are several instances of co-movement patterns that have been developed

[1]`https://requester.mturk.com`

with more advanced constraints, namely *flock* [29], *convoy* [35], *swarm* [46], *group* [65] and *platoon* [45]. However, these solutions are tailored for each individual pattern and it is cumbersome to deploy and optimize each of the algorithm in real applications. Therefore, there calls for a general framework which provides versatile and efficient support of all these pattern discoveries.

We observe that these co-movement patterns can be uniformly represented as a two-step neighborhood analytics as follows:

(1) $\mathcal{N}_1$ is a *distance neighborhood* used to determine the spatial proximity of objects. For example, flock [29] and group [65] patterns uses the *disk-based* clustering, which is equivalent to $\mathcal{N}_1(o_i) = \{o_j | \texttt{dist}(o_i, o_j) < r\}$ for each object. Convoy [35], swarm [46] and platoon [45] patterns uses the *density-based* clustering, which is equivalent to $\mathcal{N}_1(o_i) = \{o_j | \texttt{dist}(o_j, o_k) \leq \epsilon \wedge o_k \in \mathcal{N}_1(o_i)\}$.

(2) $\mathcal{N}_2$ is a *comparison neighborhood* used to determine co-moving behavior, i.e., $\mathcal{N}_2(o_i) = \{o_j, T | \forall t \in T, o_j \in C_t(o_i)\}$, where $C_t(\cdot)$ returns the neighborhoods (i.e., N_1) of an object at time t.

Enlightened by the neighborhood based unification, we propose a *General Co-Movement Pattern* (GCMP) query to capture all existing co-movement patterns in one shot. In GCMP, we treat the proximity detection (i.e., $\mathcal{N}_1$) as a black box and only focus on the pattern detection (i.e., $\mathcal{N}_2$). By tuning different parameters (as explained in later sections), GCMP query is able to detect any of the existing patterns.

On the technical side, we study how to efficiently process GCMP query on the modern parallel platform (i.e., Apache Spark) to gain scalability over large-scale trajectories. In particular, we propose two parallel frameworks: (1) TRPM, which partitions trajectories by replicating snapshots in the temporal domain. Within each partitions, a line-sweep method is developed to find all patterns. (2) SPARE, which partitions trajectories based on object's neighborhood. Within each partitions, a variant of Apriori enumerator is applied to generate all patterns. We deploy the two solutions in our in-house cluster with 11 machines. The experiments on three real trajectory datasets upto 170 million data points confirms the scalability and efficiency of our methods.

1.4 Thesis Organization

The rest of the thesis is organized as follows: in Chapter 2, we review the literature related to our proposed queries in different data domains. In Chapter 3, we present the Graph Window Query on graph data. In Chapter 4, we present the k-Sketch Query on sequence data. In Chapter 5, we present the General Co-Movement Pattern Query on trajectory data. Chapter 6 summarizes this thesis and highlights future directions.

1.5 Published Material

The research in this thesis has led to numerous publications.

- The overview of the thesis has been published in SIGMOD Ph.D. Symposium.
  Qi Fan, Kian-Lee Tan. Towards Neighborhood Analytics. Proceedings of the ACM SIGMOD on PhD Symposium, 2015

- The work in Chapter 3 appears in DASFAA 2016.
  Qi Fan, Zhengkui Wang, Chee-Yong Chan, Kian-Lee Tan. Towards Window Analytics over Large-Scale Graphs. International Conference on Database Systems for Advanced Applications, 2016

- The finding in Chapter 4 have been submitted for publication.
  Qi Fan, Yuchen Li, Dongxiang Zhang, Kian-Lee Tan. Discovering Newsworthy Themes From Sequenced Data: A Step Towards Computational Journalism.

- The work in Chapter 5 has been accepted in VLDB 2017.

  Qi Fan, Dongxiang Zhang, Huayu Wu, Kian-Lee Tan. A General and Parallel Platform for Mining Co-Movement Patterns over Large-scale Trajectories. Proceedings of the VLDB Endowment, 2017

Chapter 3

Graph Window Query: Neighborhood Analytics on Attributed Graphs

3.1 Introduction

In this chapter, we study the neighborhood analytics on large-scale attributed graphs. Attributed graphs are prevalently adopted to model real life information networks such social networks, biological networks and phone-call networks. In the attributed graph model, the vertexes correspond to objects and the edges capture the relationships between these objects. For instance, in social networks, every user is represented by a vertex and the friendship between two users is reflected by an edge connecting the vertexes. Besides, a user's profile is maintained as the vertex's attributes. Such graphs contain a wealth of valuable information which can be analyzed to discover interesting patterns [16, 59, 66, 76]. With the increasingly larger network sizes, it is becoming challenging to query, analyze and process these graph data. Therefore, there is an urgent call for effective and efficient mechanisms to draw out information over graph data resources.

Recent advances in graph analytics such as graph aggregation [66, 76] and summarization [16, 59] focus on analyzing the entire graph as a whole. In fact, it is often useful to perform *neighborhood analytics* on graph data to analyze the vicinity of vertexes. That is for each vertex, the analytics is conducted over its *neighborhoods.* For instance, in a social network, it is important to detect a person's social influence among his/her social community. The "social community" of the person is essentially his/her neighborhood vertexes representing his/her friends.

Similar neighborhood concept has been supported by window functions [8, 15] in relational data analytics. Instead of performing analysis (e.g., ranking and aggregate) over the entire data set, a window function returns for each tuple a value derived from its neighboring tuples. For instance, when finding each employee's salary ranking within every department, each tuple's neighbors are basically the tuples from the same department. However, the window definition in the relational context ignores graph structures which makes it unsuitable in the graph context. Therefore, we seek to utilize the general *neighborhood analytics* to formulate the notion of *graph windows.*

We have derived two graph windows from the perspective of neighborhood functions, which are referred to as *k-hop window* and *topological window.* The semantic of these windows are first demonstrated in the following two examples.

Example 3.1.1. *(k-hop window)* In a social network (such as LinkedIn and Facebook), users are normally modeled as vertexes and connectivity relationships are modeled as edges. In this scenario, a **distance neighborhood** function, such as 2-hop neighbors, represents the most relevant connections to each user. Some analytic queries such as summarizing related connections' distribution among different companies, and computing age distribution of the related friends can be useful. In order to answer these queries, computing the distance neighborhood is necessary.

Example 3.1.2. *(Topological window)* In biological networks (such as Argocyc, Ecocyc etc. [38]), genes, enzymes and proteins are vertexes and their dependencies in a pathway are edges. Because these networks are directed and acyclic, **comparison neighborhood** based on the ancestry relationship helps to reveal the influences among molecules. For instance, to find out the statistics of molecule in a protein production pathway, we can traverse the graph to find every molecule that is in its upstream. Then we summarize the number of genes and enzymes among those molecules. To answer such queries, computing the ancestry based comparison neighborhood is necessary.

To support these analytics in the above examples, we propose the Graph Window Query (GWQ in short) on attributed graphs. GWQ is a neighborhood analytics which aims to facilitate vertex-centric analysis. It supports two graph windows namely *k-hop window* and *topological window*. The k-hop window of a vertex is defined by its k-hop distance (e.g., friends-of-friends in Example 3.1.1). Thus it is essentially a **distance neighborhood**. On the other hand, the topological window of a vertex contains its ancestors (e.g., upstream molecules in Example 3.1.2). Hence, it is a **comparison neighborhood** based on the ancestry relationship.

To the best of our knowledge, existing graph databases or graph query languages do not directly support our proposed GWQ. There are two major challenges in processing GWQ. First, we need an efficient scheme to calculate the window of each vertex. Second, we need efficient solutions to process the aggregation over a large number of windows that may overlap. This offers opportunities to share the computation. However, it is non-trivial to address these two challenges.

For k-hop window query, the latest processing algorithm can be adopted from literature is EAGR [53]. EAGR leverages an overlay graph to represent the shared components among different windows. It incrementally construct the overlay graph through multiple iterations. In each iteration, it builds a Frequent-Pattern Tree [32] to discover the largest shared component among vertex windows. However, to achieve efficient shared component detection, EAGR requires all vertex's k-hop neighbors to be pre-computed and resided in memory; otherwise EAGR incurs high performance overheads due to secondary storage accesses (e.g., disk I/Os). This limits the usage of EAGR in large-scale graphs. For instance, a LiveJournal social network graph[1] (4.8M vertexes, 69M edges) generates over 100GB neighborhood information for $k = 2$ in adjacency list representation. In addition, the overlay graph construction is not a one-time task, but periodically performed after a certain number of structural updates in order to maintain the quality. The high memory consumption renders the scheme impractical when k and the graph size increases.

In this chapter, we propose the *Dense Block Index (DBIndex)* to process the two graph window queries efficiently. Like EAGR, DBIndex seeks to exploit common components among different windows to salvage partial work done. However, different from EAGR, we identify the window similarity by utilizing a hash-based clustering technique. This ensures efficient memory usage, as the window information of each vertex can be computed on-the-fly. On the basis of the clusters, we develop different optimization techniques to extract the shared components which result in an efficient index construction. Moreover, we provide another *Inheritance Index (I-Index)* tailored to topological window query. I-Index differentiates itself from DBIndex by integrating additional ancestry relationships to reduce repetitive computations. This results in more efficient index construction and query processing. Our contributions of this chapter are summarized as follows:

- We study the neighborhood analytics in the graph domain and propose the *Graph Window Query* which instantiates two neighborhood functions. We formally define two graph windows: k-hop window and topological window, and illustrate how these window queries would help users better query and under-

[1]Available at http://snap.stanford.edu/data/index.html, which is used in [53]

stand the graphs under these different semantics.

- To support efficient query processing, we further propose two different types of indexes: *Dense Block Index* (DBIndex) and *Inheritance Index* (I-Index). The DBIndex and I-Index are specially optimized to support k-hop window and topological window query processing. We develop the indexes by integrating the window aggregation sharing techniques to salvage partial work done for efficient computation. In addition, we develop space and performance efficient techniques for index construction.

- We perform extensive experiments over both real and synthetic datasets with hundreds of millions of vertexes and edges on a single machine. Our experiments indicate that our proposed index-based algorithms outperform the naive non-index algorithm by upto four orders of magnitude. In addition, our experiments also show that DBIndex is superior over the state-of-the-art baseline (i.e., EAGR) in terms of both scalability and efficiency. In particular, DBIndx saves up to 80% of index constructing time as compared to EAGR, and performs well even when EAGR fails due to memory limitations.

The rest of the chapter is organized as follows. In Section **??**, the graph window query is formulated. In Section **??**, Dense Block Index is presented to process general window queries. A specialized index to handle topological query is presented in Section **??**. Section **??** demonstrates our experimental findings and Section 3.2 concludes this chapter.

3.2 Summary

In this chapter, we studied the neighborhood analytics in attributed graphs. We leverage the distance and comparison neighborhoods to propose two graph windows: k-hop window and topological window. Based on the two window definitions, we proposed a new type of graph analytic query, *Graph Window Query* (GWQ). Then, we studied GWQ processing for large-scale graphs. In particular, we developed the Dense Block Index (DBIndex) to facilitate efficient processing of both types of graph windows. Moreover, we proposed the Inheritance Index (I-Index) that exploits a containment property of DAG to enhance the query performance of topological window queries. Last, we conducted extensive experimental evaluations over both large-scale real and synthetic datasets. The experimental results showed the efficiency and scalability of our proposed indexes.

Chapter 4

k-Sketch Query: Neighborhood Analytics in Sequence Data

4.1 Introduction

Next, we explore the opportunity of neighborhood analytics in the sequence data. Sequence data is widely adopted to model the history of *subjects* which consists of temporally ordered *events*. For example, in sports application, the subject could be a player and his history would be the games he participated. Likewise, in stock application, the subject could be a stock and its history would be its daily closing prices. An important analytic task for these sequence data is to summarize the subject histories. Such a task is useful in a variety of applications [33, 57, 62, 68] and is also the cornerstone for the emerging computational journalism [21].

Leveraging the temporal ordering of events, neighborhood analytics in sequence data are expressed as *streaks* [75]. A streak refers to an aggregated period in a subject's history, such as "points scored by a player in the last ten straight games". Technically, a streak is a tuple $sk = \langle s, L, t, \overline{v} \rangle$ which represents the last L events of subject s at time t. $\overline{v}$ is the result of an analytic function (e.g., min, average) on all the events in the streak. A streak is essentially built on a **distance neighborhood** function. Formally, let $t_s(e)$ be the sequence number of the event e in the history of subject s. Then, a length-L streak of event e is defined by the following distance neighborhood function: $\mathcal{N}(s, L, e) = \{e_i | t_s(e) - t_s(e_i) \leq L\}$.

Based on the streaks, Zhang et. al. [75] studied the problem of summarizing a subject's history with *prominent streaks*. Prominent streaks are the skylines among all streaks of a subject in terms of streak size (i.e., L) and analytic value (i.e., $\overline{v}$). Hence, they are outstanding to represent the subject's history. However, there are two major drawbacks that limit the usability of prominent streaks in real applications. First, the prominent streaks generated by [75] may not be striking enough because they are derived from the historical data of a *single* subject without comparing to other subjects. For example, "Steve Nash has scored over 15 points in consecutive 10 games" is a prominent streak for "Steve Nash", but it is not striking given the fact that there are more than 90 players with better performance[1]. Second, the number of the prominent streaks can be overwhelming. Since prominent streaks in [75] are defined as skylines, a subject with n historical events may generate upto n streaks that are not dominated (i.e., prominent streaks). Therefore, there calls for a new method to automatically select a limited number of striking streaks which best summarize a subject's history.

In this chapter, we tackle the problem of effectively and efficiently summarizing a subject's history by applying a novel **comparison neighborhood** function to transform streaks to the *ranked-streaks*. Given a streak $sk = (s, L, t, \overline{v})$, a comparison neighborhood is defined as $\mathcal{N}(sk) = \{sk_i | sk_i.L = sk.L \wedge sk_i.\overline{v} >= sk.\overline{v}\}$. Then a *count* function would generate the rank of this streak based on its neighborhood. In other words, the neighborhood function groups the streaks with the same length and rank them based on their aggregated values. Compared with streak, ranked-streak is

[1]`http://www.sporcle.com/games/nbadarinh/nba-all-players -with-10-consecutive-20-point-games-90-11`

able to capture the strikingness of a streak which is very newsworthy as evidenced in the following news excerpts:

1. (26 Feb 2003) With 32 points, Kobe Bryant saw his 40+ scoring streak end at **nine** games, tied with Michael Jordan for the **fourth** place on the all-time list[2].
2. (14 April 2014) Stephen Curry has made 602 3-pointer attempts from beyond the arc,... are the **10th** most in NBA history in a season (**82 games**)[3].
3. (28 May 2015) Stocks gained for the **seventh consecutive day** on Wednesday as the benchmark moved close to the 5,000 mark for **the first** time in seven years[4].
4. (9 Jun 2014) Delhi has been witnessing a spell of hot weather over the **past month**, with temperature hovering around 45 degrees Celsius, ... **highest** ever since 1952[5].
5. (22 Jul 2011) Pelican Point recorded a maximum rainfall of 0.32 inches for **12 months**, making it the **9th driest** places on earth[6].

In the above examples, each news theme is a *ranked-streak* which consists of five indicators: (I) a subject (e.g., Kobe Bryant, Stocks, Delhi), (II) a streak length (e.g., nine straight games, seventh consecutive days, past month), (III) an aggregate function on an attribute (e.g., minimum points, count of gains, average of degrees), (IV) a rank (e.g., fourth, first time, highest), and (V) a historical dataset (e.g., all time list, seven years, since 1952). The indicators (I)-(IV) are summarized in Table 4.1.

[2]`http://www.nba.com/features/kobe_40plus_030221.html`

[3]`http://www.cbssports.com/nba/eye-on-basketball/24525914/stephen-curry-makes-history-with-consecutive-seasons-of-250-3s`

[4]`http://www.zacks.com/stock/news/176469/china-stock-roundup-ctrip-buys-elong-stake-trina-solar-beats-estimates`

[5]`http://www.dnaindia.com/delhi/report-delhi-records-highest-temperature-in-62-years-1994332`

[6]`http://www.livescience.com/30627-10-driest-places-on-earth.html`

Table 4.1: Indicators of ranked-streaks.

E.g.	**Subject**	**Aggregate Function**	**Streak Length**	**Rank**
1	Kobe	min(points)	9 straight games	4
2	Stephen	sum(shot attempts)	82 games	10
3	Stock Index	count(gains)	7 consecutive days	1
4	Delhi	average(degree)	past months (30 days)	1
5	Pelican Point	max(raindrops)	12 months	9

Based on the ranked-streaks, we propose a novel k-Sketch query to effectively summarize a subject's history by leveraging a novel scoring function that chooses the best k ranked-streaks. Our scoring function considers two aspects: (1) we prefer the ranked-streaks that cover as many events as possible to represent a subject's history, and (2) we prefer the ranked-streaks that have better ranks[7] as they indicate more strikingness. Our objective is then to process the k-Sketch query for each subject in the domain.

We study the k-Sketch query processing under both offline and online scenarios. In the offline scenario, our objective is to efficiently discover the sketch for each subject from historical data. The major challenge lies in generating the rank information of streaks. Since the number of streaks is quadratic with respect to the number of events, enumerating all of them is not scalable. By leveraging the subadditivity among the upper bounds of streaks, we design two effective pruning techniques to facilitate efficient ranked-streak generation. Furthermore, we notice that generating exact sketches from ranked-streaks is computationally expensive. Thus, we design an efficient $(1 - 1/e)$-approximate algorithm by exploiting the submodularity of the k-Sketch query.

In the online scenario, fresh events are continuously fed into the system and our goal is to maintain the sketches for each subject uptodate. When a new event about subject s arrives, many ranked-streaks of various lengths can be derived. For each

[7]We consider a ranked-streak with rank i to be more attractive than j if $i < j$ and the other fields are the same.

derived ranked-streak, not only the sketch of s but also the sketches of other subjects may be affected. Dealing with such a complex updating pattern is non-trivial. To efficiently support the update while maintaining the quality of sketches, we propose a 1/8-approximate algorithm which only examines $2k$ ranked-streaks for each subject whose sketch is affected.

Our contributions of this chapter are hereby summarized as follows:

- We study the neighborhood analytics in sequence data to tackle the problem of automatic summarization of a subject's history. We use both the distance and comparison neighborhood functions to model the ranked-streak, which is a common news theme in real-life reports but has not been addressed in previous works. We formulate the summarization problem as a k-Sketch query under a novel scoring function that considers both strikingness and coverage.

- We study the k-Sketch query processing in both offline and online scenarios. In the offline scenario, we propose two novel pruning techniques to efficiently generate ranked-streaks. Then we design a $(1 - 1/e)$-approximate algorithm to compute the sketches for each subject. In the online scenario, we propose a 1/8-approximate algorithm to efficiently support the complex updating patterns as new event arrives.

- We conduct extensive experiments with four real datasets to evaluate the effectiveness and the efficiency of our proposed algorithms. In the offline scenario, our solution is three orders of magnitude faster than baseline algorithms. While in the online scenario, our solution achieves up to 500x speedup. In addition, we also perform an anonymous user study via Amazon Mechanical Turk[8] platform, which validates the effectiveness of our k-Sketch query.

The rest of this chapter is organized as follows. In Section **??**, we formulate the

[8]https://requester.mturk.com

k-Sketch query. Section **??** presents the algorithms for processing the k-Sketch query in the offline scenario. Section **??** describes the algorithms for maintaining k-Sketches in the online scenario. In Section **??**, comprehensive experimental studies on both the efficiency and the effectiveness of our algorithms are conducted. Section A.1 discusses the extension of our methods. Finally, Section 4.2 concludes our paper.

4.2 Summary

In this chapter, we looked at the neighborhood analytics in sequence data. We leverage the joint distance and comparison neighborhood functions to design the novel *ranked-streak* which quantifies the strikingness of a streak. We then formulated the *k-Sketch* query which aims to best summarize a subject's history using k ranked-streaks. We studied the k-Sketch query processing in both offline and online scenarios, and propose efficient solutions to cope each scenario. In particular, we designed novel streak-level pruning techniques and a $(1 - 1/e)$-approximate algorithm to achieve efficient processing in offline. Moreover, we designed a 1/8-approximate algorithm for the online sketch maintenance. Our comprehensive experiments demonstrated the efficiency of our solutions and a human study confirmed the effectiveness of the k-Sketch query.

Chapter 5

GCMP Query: Neighborhood Analytics in Trajectories

5.1 Introduction

Trajectory analysis is another emerging field in data analytics. A trajectory is the spatial trace of a moving object which contains a sequence of spatial-temporal records. Data analysis on trajectories benefits a wide range of applications and services, including traffic planning [80], animal analysis [47], location-aware advertising [30], and social recommendations [7], to name just a few.

An important analytics on top of trajectories is to discover co-moving objects. A *co-movement* pattern [42, 79] refers to a group of objects traveling together for a certain period of time. Such a pattern can be concisely expressed by two neighborhood functions in the spatial and the temporal dimensions respectively. Specifically, in the spatial dimension, the groups of objects at each time point are determined by a *distance neighborhood function.* Let $o(t)$ be the object o's location at time t, then the objects in the same group with o at time t are: $\mathcal{N}_1(o,t) = \{o_i | \texttt{dist}(o(t), o_i(t)) \leq r\}$[1].

[1]This refers to as the *disk-based* clustering. Density-based clustering can be expressed similarly as: $\mathcal{N}_1(o,t) = \{o_i | \texttt{dist}(o_j(t), o_i(t)) \leq \epsilon \wedge o_j \in \mathcal{N}_1(o,t)\}$

Next, in the temporal dimension, the objects co-moving with an object o for a time period T are determined by: $\mathcal{N}_2(o, T) = \{o_i | \forall t \in T, o_i \in \mathcal{N}_1(o, t)\}$. A movement pattern is prominent if the size of the group exceeds M (i.e., $|\mathcal{N}_2| \geq M$) and the length of the duration exceeds K (i.e., $|T| \geq K$), where M and K are parameters specified by users. Rooted from such a basic definition and driven by different mining applications, there are many variants of co-movement patterns that have been developed with additional constraints.

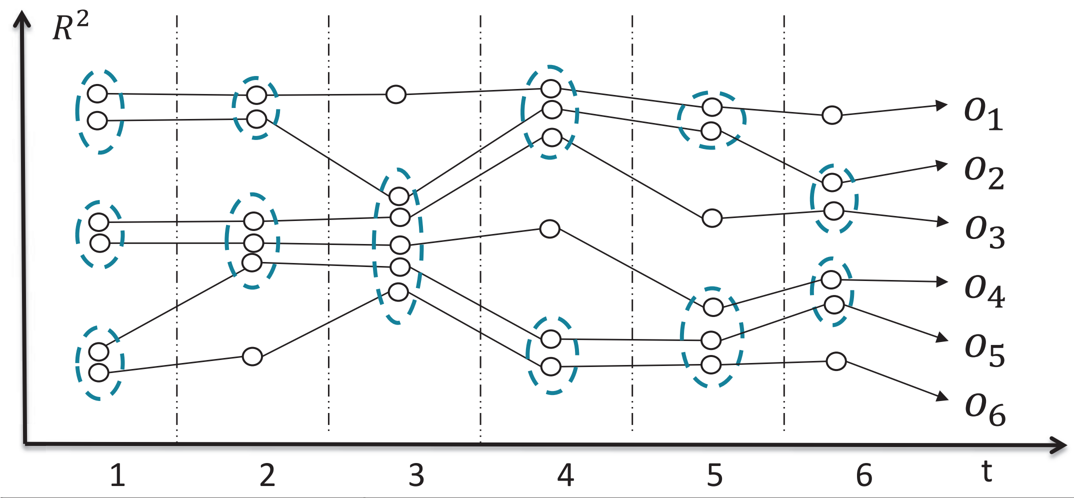
Table 5.1 summarizes several popular co-movement patterns with different constraints with respect to spatial neighborhood, temporal constraints in consecutiveness and computational complexity. In terms of spatial neighborhood, the *flock* [29] and the *group* [65] patterns adopt disk-based clustering which requires all the objects in a group to be enclosed by a disk with radius r[2]; whereas the *convoy* [35], the *swarm* [46] and the *platoon* [45] patterns resort to density-based spatial clustering[3]. In terms of temporal constraints, the *flock* and the *convoy* require all the timestamps of each detected spatial group to be consecutive, which is referred to as *global consecutiveness*; whereas the *swarm* does not impose any restriction. The *group* and the *platoon* adopt a compromised approach by allowing arbitrary gaps between consecutive segments, which is called *local consecutiveness*. They introduce a parameter L to control the minimum length of each local consecutive segment.

Table 5.1: Constraints and complexities of co-movement patterns. The time complexity indicates the performance wrt. $|\mathbb{O}|$, $|\mathbb{T}|$ in the worst case, where $|\mathbb{O}|$ is the number of objects, and $|\mathbb{T}|$ is the number of discretized timestamps.

Pattern	**Spatial Neighborhood**	**Temporal Constraint**	**Time Complexity**
flock [29]	disk based	global consecutive	$O(\lvert\mathbb{O}\rvert\lvert\mathbb{T}\rvert \log(\lvert\mathbb{O}\rvert))$
convoy [35]	density based	global consecutive	$O(\lvert\mathbb{O}\rvert^2 + \lvert\mathbb{O}\rvert\lvert\mathbb{T}\rvert)$
swarm [46]	density based	-	$O(2^{\lvert\mathbb{O}\rvert}\lvert\mathbb{O}\rvert\lvert\mathbb{T}\rvert)$
group [65]	disk based	local consecutive	$O(\lvert\mathbb{O}\rvert^2\lvert\mathbb{T}\rvert)$
platoon [45]	density based	local consecutive	$O(2^{\lvert\mathbb{O}\rvert}\lvert\mathbb{O}\rvert\lvert\mathbb{T}\rvert)$

[2]Disk-based clustering is equivalent to $\mathcal{N}(o_i, t) = \{o_j | \mathtt{dist}(o_i(t), o_j(t)) < r\}$.

[3]Density-based clustering is equivalent to $\mathcal{N}(o_i, t) = \{o_j | \mathtt{dist}(o_j(t), o_k(t)) \leq \epsilon \wedge o_k \in \mathcal{N}(o_i, t)\}$.



Flock (M=2, K=3)	$\langle o_3, o_4$: 1,2,3$\rangle$ $\langle o_5, o_6$: 3,4,5$\rangle$
Convoy (M=2, K=3)	$\langle o_3, o_4$: 1,2,3$\rangle$ $\langle o_5, o_6$: 3,4,5$\rangle$
Group (M=2, L=2)	$\langle o_1, o_2$: 1,2,4,5$\rangle$ $\langle o_3, o_4, o_5$: 2,3$\rangle$ $\langle o_5, o_6$: 3,4,5$\rangle$
Platoon (M=2,K=3,L=2)	$\langle o_1, o_2$: 1,2,4,5$\rangle$ $\langle o_3, o_4$: 1,2,3$\rangle$ $\langle o_4, o_5$: 2,3,5,6$\rangle$ $\langle o_5, o_6$: 3,4,5$\rangle$
Swarm (M=2,K=3)	$\langle o_1, o_2$: 1,2,4,5$\rangle$ $\langle o_3, o_4$: 1,2,3$\rangle$ $\langle o_4, o_5$: 2,3,5,6$\rangle$ $\langle o_5, o_6$: 1,3,4,5$\rangle$ $\langle o_2, o_3$: 3,4,6$\rangle$

Figure 5.1: Trajectories and co-movement patterns. The example consists of six trajectories across six snapshots. Objects in spatial clusters are enclosed by dotted circles. M is the minimum cluster cardinality; K denotes the minimum number of snapshots for the occurrence of a spatial cluster; and L denotes the minimum length for local consecutiveness.

Figure 5.1 is an example to demonstrate the concepts of the various co-movement patterns. The trajectory database consists of six moving objects and the temporal dimension is discretized into six snapshots. In each snapshot, we treat the clustering method as a blackbox and assume that they generate the same clusters. Objects in proximity are grouped in the dotted circles. As aforementioned, there are three parameters to determine the co-movement patterns and the default settings in this example are $M = 2$, $K = 3$ and $L = 2$. Both the *flock* and the *convoy* require the spatial clusters to last for at least K consecutive timestamps. Hence, $\langle o_3, o_4 : 1, 2, 3\rangle$ and $\langle o_5, o_6 : 3, 4, 5\rangle$ are the only two candidates matching the patterns. The *swarm* relaxes the pattern matching by discarding the temporal consecutiveness constraint. Thus, it generates many more candidates than the *flock* and the *convoy*. The *group* and the *platoon* add another constraint on local consecutiveness to retain meaningful patterns. For instance, $\langle o_1, o_2 : 1, 2, 4, 5\rangle$ is a pattern matching local consecutiveness because timestamps $(1, 2)$ and $(4, 5)$ are two segments with length no smaller than $L = 2$. The difference between the *group* and the *platoon* is that the *platoon* has an additional parameter K to specify the minimum number of snapshots for the spatial clusters. This explains why $\langle o_3, o_4, o_5 : 2, 3\rangle$ is a *group* pattern but not a *platoon* pattern.

As can be seen, there are various co-movement patterns requested by different applications and it is cumbersome to design a tailored solution for each type. In addition, despite the generality of the *platoon* (i.e., it can be reduced to other types of patterns via proper parameter settings), it suffers from the so-called *loose-connection* anomaly. We use two objects o_1 and o_2 in Figure 5.2 to illustrate the scenario. These two objects form a *platoon* pattern in timestamps $(1, 2, 3, 102, 103, 104)$. However, the two consecutive segments are 98 timestamps apart, resulting in a false positive co-movement pattern. In reality, such an anomaly may be caused by the periodic movements of unrelated objects, such as vehicles stopping at the same petrol station or animals pausing at the same water source. Unfortunately, none of the existing patterns have directly addressed this anomaly.

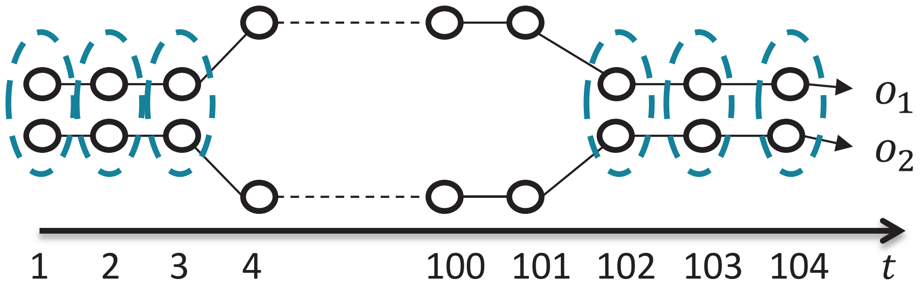
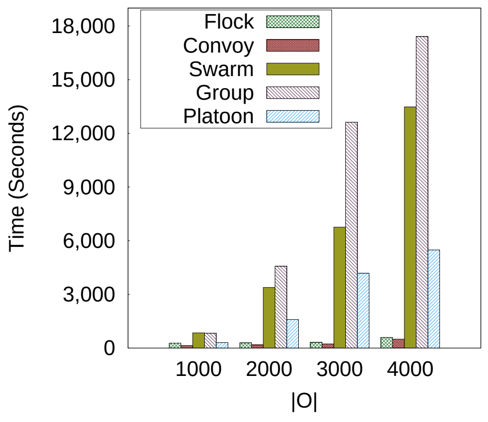
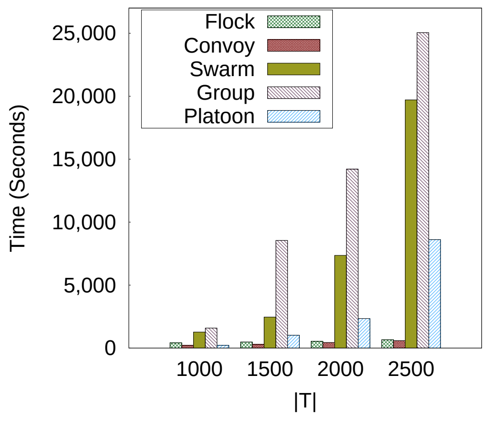


Figure 5.2: *Loose-connection* anomaly. Even though $\langle o_1, o_2\rangle$ is considered as a valid *platoon* pattern, it is highly probable that these two objects are not related as the two consecutive segments are 98 timestamps apart.

The other issue with existing methods is that they are built on top of centralized indexes. Thus, they may not be scalable to handle real large-scale trajectories collected by today's positioning technologies. Table 5.1 shows their theoretical complexities in the worst cases and the largest real dataset ever evaluated in previous studies is up to million-scale points collected from hundreds of moving objects. In practice, the dataset is of much higher scale and the scalability of existing methods is left unknown. Thus, we conduct an experimental evaluation with 4000 objects moving for 2500 timestamps to examine the scalability. Results in Figure 5.3 show that their performances degrade dramatically as the dataset scales up. For instance, the detection time of *group* drops twenty times as the number of objects grows from *1k* to *4k*. Similarly, the performance of *swarm* drops over fifteen times as the number of snapshots grows from *1k* to *2.5k*. These observations imply that existing methods are not scalable to support large-scale trajectory databases.



(a) Varying No. of objects



(b) Varying No. of timestamps

Figure 5.3: Performance measures on existing co-movement patterns. A sampled GeoLife dataset is used with upto 2.4 million data points. Default parameters are $M = 15$, $K = 180$, $L = 30$.

In this chapter, we close these two gaps by making the following contributions. First, we propose the *general co-movement pattern* (GCMP) which models various co-moment patterns in a unified way and can avoid the *loose-connection* anomaly. In GCMP, we introduce a new gap parameter G to pose a constraint on the temporal gap between two consecutive segments. By setting a feasible G, the loose-connection anomaly can be effectively controlled. In addition, our GCMP is also general. It can be reduced to any of the previous pattern by customizing its parameters.

Second, we investigate deploying our GCMP detector on the modern MapReduce platform (i.e., Apache Spark) to tackle the scalability issue. Our technical contributions are threefold. First, we design a baseline solution by replicating the snapshots to support effective parallel mining. Second, we devise a novel *Star Partitioning and ApRiori Enumerator* (SPARE) framework to resolve limitations of the baseline. SPARE achieves workload balance by partitioning objects into fine granular stars. For each partition, an Apriori Enumerator is adopted to mine the co-movement patterns. Third, we leverage the *temporal monotonicity* property of GCMP to design several optimization techniques including *sequence simplification*, *monotonicity pruning* and *forward closure check* to further reduce the number of candidates enumerated in SPARE.

We conduct a set of extensive experiments on three large-scale real datasets with hundreds of millions of temporal points. The results show that both our parallel schemes efficiently support GCMP mining in large datasets. In particular, with over 170 million trajectory points, SPARE achieves upto 112 times speedup using 162 cores as compared to the state-of-the-art centralized schemes. Moreover, SPARE further achieves almost linear scalability with upto 14 times efficiency as compared to the baseline algorithm.

The rest of this chapter is organized as follows: Section **??** states the problem of general co-movement pattern mining. Section **??** provides a baseline solution. An advanced solution named *Star Partitioning and ApRiori Enumerator* (SPARE) is presented in Section **??**. Section **??** reports our experimental evaluation. Finally, Section 5.2 summarizes this chapter.

5.2 Summary

In this chapter, we studied one of the neighborhood analytics, namely the co-movement pattern discovery, on trajectory data. We proposed a generalized co-movement pattern query to unify those proposed in the past literature. We then devised two types of parallel frameworks on Apache Spark that can scale to support pattern detection in trajectory databases with hundreds of millions of points. The efficiency and scalability were verified by extensive experiments on three real datasets.

Bibliography

[1] Rakesh Agrawal, Sreenivas Gollapudi, Alan Halverson, and Samuel Ieong. Diversifying search results. In *Proceedings of the Second ACM International Conference on Web Search and Data Mining*, pages 5–14. ACM, 2009.

[2] Rakesh Agrawal, Ramakrishnan Srikant, et al. Fast algorithms for mining association rules. In *Proc. 20th int. conf. very large data bases, VLDB*, volume 1215, pages 487–499, 1994.

[3] James Allan, Ron Papka, and Victor Lavrenko. On-line new event detection and tracking. In *Proceedings of the 21st annual international ACM SIGIR conference on Research and development in information retrieval*, pages 37–45. ACM, 1998.

[4] Noga Alon, Baruch Awerbuch, and Yossi Azar. The online set cover problem. In *Proceedings of the thirty-fifth annual ACM symposium on Theory of computing*, pages 100–105. ACM, 2003.

[5] Htoo Htet Aung and Kian-Lee Tan. Discovery of evolving convoys. In *International Conference on Scientific and Statistical Database Management*, pages 196–213. Springer, 2010.

[6] Baruch Awerbuch, Yossi Azar, Amos Fiat, and Tom Leighton. Making commitments in the face of uncertainty: How to pick a winner almost every time. In *Proceedings of the twenty-eighth annual ACM symposium on Theory of computing*, pages 519–530. ACM, 1996.

[7] Jie Bao, Yu Zheng, David Wilkie, and Mohamed F Mokbel. A survey on recommendations in location-based social networks. *ACM Transaction on Intelligent Systems and Technology*, 2013.

[8] Srikanth Bellamkonda, Hua-Gang Li, Unmesh Jagtap, Yali Zhu, Vince Liang, and Thierry Cruanes. Adaptive and big data scale parallel execution in oracle. *Proceedings of the VLDB Endowment*, 6(11):1102–1113, 2013.

[9] Allan Borodin, Hyun Chul Lee, and Yuli Ye. Max-sum diversification, monotone submodular functions and dynamic updates. In *Proceedings of the 31st ACM SIGMOD-SIGACT-SIGAI symposium on Principles of Database Systems*, pages 155–166. ACM, 2012.

[10] Thorsten Brants, Francine Chen, and Ayman Farahat. A system for new event detection. In *Proceedings of the 26th Annual International ACM SIGIR Conference on Research and Development in Informaion Retrieval*, SIGIR '03, pages 330–337, New York, NY, USA, 2003. ACM.

[11] Erica J Briscoe, D Scott Appling, Rudolph L Mappus IV, and Heather Hayes. Determining credibility from social network structure. In *Proceedings of the 2013 IEEE/ACM International Conference on Advances in Social Networks Analysis and Mining*, pages 1418–1424. ACM, 2013.

[12] Andrei Z Broder, Steven C Glassman, Mark S Manasse, and Geoffrey Zweig. Syntactic clustering of the web. *Computer Networks and ISDN Systems*, 29(8):1157–1166, 1997.

[13] Ronald S Burt. *Structural holes: The social structure of competition.* Harvard university press, 2009.

[14] Juan Miguel Campanario. Empirical study of journal impact factors obtained using the classical two-year citation window versus a five-year citation window. *Scientometrics*, 87(1):189–204, 2011.

[15] Yu Cao, Chee-Yong Chan, Jie Li, and Kian-Lee Tan. Optimization of analytic window functions. *Proceedings of the VLDB Endowment*, 5(11):1244–1255, 2012.

[16] Chen Chen, Xifeng Yan, Feida Zhu, Jiawei Han, and S Yu Philip. Graph olap: Towards online analytical processing on graphs. In *2008 Eighth IEEE International Conference on Data Mining*, pages 103–112. IEEE, 2008.

[17] Lisi Chen and Gao Cong. Diversity-aware top-k publish/subscribe for text stream. In *Proceedings of the 2015 ACM SIGMOD International Conference on Management of Data*, pages 347–362. ACM, 2015.

[18] James Cheng, Silu Huang, Huanhuan Wu, and Ada Wai-Chee Fu. Tf-label: a topological-folding labeling scheme for reachability querying in a large graph. In *Proceedings of the 2013 ACM SIGMOD International Conference on Management of Data*, pages 193–204. ACM, 2013.

[19] James Cheng, Zechao Shang, Hong Cheng, Haixun Wang, and Jeffrey Xu Yu. K-reach: who is in your small world. *Proceedings of the VLDB Endowment*, 5(11):1292–1303, 2012.

[20] Tom Choe, Alexander Skabardonis, and Pravin Varaiya. Freeway performance measurement system: operational analysis tool. *Transportation Research Record: Journal of the Transportation Research Board*, (1811):67–75, 2002.

[21] Sarah Cohen, Chengkai Li, Jun Yang, and Cong Yu. Computational journalism: A call to arms to database researchers. In *CIDR*, volume 2011, pages 148–151, 2011.

[22] Lianghao Dai, Jar-der Luo, Xiaoming Fu, and Zhichao Li. Predicting offline behaviors from online features: an ego-centric dynamical network approach. In *Proceedings of the First ACM International Workshop on Hot Topics on Interdisciplinary Social Networks Research*, pages 17–24. ACM, 2012.

[23] Mark De Berg, Marc Van Kreveld, Mark Overmars, and Otfried Cheong Schwarzkopf. Computational geometry. In *Computational geometry*, pages 1–17. Springer, 2000.

[24] David H Douglas and Thomas K Peucker. Algorithms for the reduction of the number of points required to represent a digitized line or its caricature. *Cartographica: The International Journal for Geographic Information and Geovisualization*, 10(2):112–122, 1973.

[25] Marina Drosou and Evaggelia Pitoura. Diverse set selection over dynamic data. *IEEE Transactions on Knowledge and Data Engineering*, 26(5):1102–1116, 2014.

[26] Katherine Edwards, Simon Griffiths, and William Sean Kennedy. Partial interval set cover–trade-offs between scalability and optimality. In *Approximation, Randomization, and Combinatorial Optimization. Algorithms and Techniques*, pages 110–125. Springer, 2013.

[27] Martin Ester, Hans-Peter Kriegel, Jörg Sander, Xiaowei Xu, et al. A density-based algorithm for discovering clusters in large spatial databases with noise. In *Kdd*, volume 96, pages 226–231, 1996.

[28] Lukasz Golab, Howard Karloff, Flip Korn, Avishek Saha, and Divesh Srivastava. Sequential dependencies. *Proceedings of the VLDB Endowment*, 2(1):574–585, 2009.

[29] Joachim Gudmundsson and Marc van Kreveld. Computing longest duration flocks in trajectory data. In *Proceedings of the 14th annual ACM international*

symposium on Advances in geographic information systems, pages 35–42. ACM, 2006.

[30] Long Guo, Dongxiang Zhang, Gao Cong, Wei Wu, and Kian-Lee Tan. Influence maximization in trajectory databases. In *TKDE*, page 1, 2016.

[31] Jiawei Han, Jian Pei, and Yiwen Yin. Mining frequent patterns without candidate generation. In *ACM Sigmod Record*, volume 29, pages 1–12. ACM, 2000.

[32] Jiawei Han, Jian Pei, Yiwen Yin, and Runying Mao. Mining frequent patterns without candidate generation: A frequent-pattern tree approach. *Data mining and knowledge discovery*, 8(1):53–87, 2004.

[33] Naeemul Hassan, Afroza Sultana, You Wu, Gensheng Zhang, Chengkai Li, Jun Yang, and Cong Yu. Data in, fact out: automated monitoring of facts by factwatcher. *Proceedings of the VLDB Endowment*, 7(13):1557–1560, 2014.

[34] Clyde W Holsapple and Wenhong Luo. A citation analysis of influences on collaborative computing research. *Computer Supported Cooperative Work (CSCW)*, 12(3):351–366, 2003.

[35] Hoyoung Jeung, Man Lung Yiu, Xiaofang Zhou, Christian S Jensen, and Heng Tao Shen. Discovery of convoys in trajectory databases. *Proceedings of the VLDB Endowment*, 1(1):1068–1080, 2008.

[36] Ryota Jinno, Kazuhiro Seki, and Kuniaki Uehara. Parallel distributed trajectory pattern mining using mapreduce. In *Cloud Computing Technology and Science (CloudCom), 2012 IEEE 4th International Conference on*, pages 269–273. IEEE, 2012.

[37] Panos Kalnis, Nikos Mamoulis, and Spiridon Bakiras. On discovering moving clusters in spatio-temporal data. In *International Symposium on Spatial and Temporal Databases*, pages 364–381. Springer, 2005.

[38] Ingrid M Keseler, Julio Collado-Vides, Socorro Gama-Castro, John Ingraham, Suzanne Paley, Ian T Paulsen, Martín Peralta-Gil, and Peter D Karp. Ecocyc: a comprehensive database resource for escherichia coli. *Nucleic acids research*, 33(suppl 1):D334–D337, 2005.

[39] YongChul Kwon, Magdalena Balazinska, Bill Howe, and Jerome Rolia. Skewtune: mitigating skew in mapreduce applications. In *Proceedings of the 2012 ACM SIGMOD International Conference on Management of Data*, pages 25–36. ACM, 2012.

[40] Patrick Laube, Marc van Kreveld, and Stephan Imfeld. Finding remodetecting relative motion patterns in geospatial lifelines. In *Developments in spatial data handling*, pages 201–215. Springer, 2005.

[41] Srivatsan Laxman, PS Sastry, and KP Unnikrishnan. A fast algorithm for finding frequent episodes in event streams. In *Proceedings of the 13th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 410–419. ACM, 2007.

[42] Xiaohui Li, Vaida Ceikute, Christian S Jensen, and Kian-Lee Tan. Effective online group discovery in trajectory databases. *IEEE Transactions on Knowledge and Data Engineering*, 25(12):2752–2766, 2013.

[43] Xiaohui Li, Vaida Ceikute, Christian S Jensen, and Kian-Lee Tan. Effective online group discovery in trajectory databases. *IEEE Transactions on Knowledge and Data Engineering*, 25(12):2752–2766, 2013.

[44] Xuefei Li, Hongyun Cai, Zi Huang, Yang Yang, and Xiaofang Zhou. Social event identification and ranking on flickr. *World Wide Web*, 18(5):1219–1245, 2015.

[45] Yuxuan Li, James Bailey, and Lars Kulik. Efficient mining of platoon patterns in trajectory databases. *Data & Knowledge Engineering*, 100:167–187, 2015.

[46] Zhenhui Li, Bolin Ding, Jiawei Han, and Roland Kays. Swarm: Mining relaxed temporal moving object clusters. *Proceedings of the VLDB Endowment*, 3(1-2):723–734, 2010.

[47] Zhenhui Li, Bolin Ding, Jiawei Han, Roland Kays, and Peter Nye. Mining periodic behaviors for moving objects. In *Proceedings of the 16th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 1099–1108. ACM, 2010.

[48] M. Lichman. UCI machine learning repository, 2013.

[49] Huaiyu Harry Ma, Steven Gustafson, Abha Moitra, and David Bracewell. Ego-centric network sampling in viral marketing applications. In *Mining and Analyzing Social Networks*, pages 35–51. Springer, 2010.

[50] Nan Ma, Jiancheng Guan, and Yi Zhao. Bringing pagerank to the citation analysis. *Information Processing & Management*, 44(2):800–810, 2008.

[51] Heikki Mannila, Hannu Toivonen, and A Inkeri Verkamo. Discovery of frequent episodes in event sequences. *Data mining and knowledge discovery*, 1(3):259–289, 1997.

[52] Peter V Marsden. Egocentric and sociocentric measures of network centrality. *Social networks*, 24(4):407–422, 2002.

[53] Jayanta Mondal and Amol Deshpande. Eagr: Supporting continuous ego-centric aggregate queries over large dynamic graphs. In *Proceedings of the 2014 ACM*

SIGMOD International Conference on Management of Data, pages 1335–1346. ACM, 2014.

[54] George L Nemhauser, Laurence A Wolsey, and Marshall L Fisher. An analysis of approximations for maximizing submodular set functionsi. *Mathematical Programming*, 14(1):265–294, 1978.

[55] Jian Pei, Jiawei Han, Runying Mao, et al. Closet: An efficient algorithm for mining frequent closed itemsets. In *ACM SIGMOD workshop on research issues in data mining and knowledge discovery*, volume 4, pages 21–30, 2000.

[56] Barna Saha and Lise Getoor. On maximum coverage in the streaming model & application to multi-topic blog-watch. 9:697–708, 2009.

[57] Afroza Sultana, Naeemul Hassan, Chengkai Li, Jun Yang, and Cong Yu. Incremental discovery of prominent situational facts. In *2014 IEEE 30th International Conference on Data Engineering*, pages 112–123. IEEE, 2014.

[58] Nikolaj Tatti and Boris Cule. Mining closed strict episodes. *Data Mining and Knowledge Discovery*, 25(1):34–66, 2012.

[59] Yuanyuan Tian, Richard A Hankins, and Jignesh M Patel. Efficient aggregation for graph summarization. In *Proceedings of the 2008 ACM SIGMOD international conference on Management of data*, pages 567–580. ACM, 2008.

[60] Virginia Vassilevska and Ali Pinar. Finding nonoverlapping dense blocks of a sparse matrix. *Lawrence Berkeley National Laboratory*, 2004.

[61] Jeroen B.P. Vuurens, Arjen P. de Vries, Roi Blanco, and Peter Mika. Online news tracking for ad-hoc information needs. In *Proceedings of the 2015 International Conference on The Theory of Information Retrieval*, ICTIR '15, pages 221–230, New York, NY, USA, 2015. ACM.

[62] Brett Walenz, You Will Wu, Seokhyun Alex Song, Emre Sonmez, Eric Wu, Kevin Wu, Pankaj K Agarwal, Jun Yang, Naeemul Hassan, Afroza Sultana, et al. Finding, monitoring, and checking claims computationally based on structured data. In *Computation+ Journalism Symposium*, 2014.

[63] Brett Walenz and Jun Yang. Perturbation analysis of database queries. *Proc. VLDB Endow.*, 9(14):1635–1646, October 2016.

[64] Jianyong Wang, Jiawei Han, and Jian Pei. Closet+: Searching for the best strategies for mining frequent closed itemsets. In *Proceedings of the ninth ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 236–245. ACM, 2003.

[65] Yida Wang, Ee-Peng Lim, and San-Yih Hwang. Efficient mining of group patterns from user movement data. *Data & Knowledge Engineering*, 57(3):240–282, 2006.

[66] Zhengkui Wang, Qi Fan, Huiju Wang, Kian-Lee Tan, Divyakant Agrawal, and Amr El Abbadi. Pagrol: parallel graph olap over large-scale attributed graphs. In *2014 IEEE 30th International Conference on Data Engineering*, pages 496–507. IEEE, 2014.

[67] Hao Wei, Jeffrey Xu Yu, Can Lu, and Ruoming Jin. Reachability querying: An independent permutation labeling approach. *Proceedings of the VLDB Endowment*, 7(12):1191–1202, 2014.

[68] You Wu, Pankaj K Agarwal, Chengkai Li, Jun Yang, and Cong Yu. On one of the few objects. In *Proceedings of the 18th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 1487–1495. ACM, 2012.

[69] Xifeng Yan, Bin He, Feida Zhu, and Jiawei Han. Top-k aggregation queries over large networks. In *2010 IEEE 26th International Conference on Data Engineering (ICDE 2010)*, pages 377–380. IEEE, 2010.

[70] Hilmi Yildirim, Vineet Chaoji, and Mohammed J Zaki. Dagger: A scalable index for reachability queries in large dynamic graphs. *arXiv preprint arXiv:1301.0977*, 2013.

[71] Jin Soung Yoo and Shashi Shekhar. A joinless approach for mining spatial colocation patterns. *IEEE Transactions on Knowledge and Data Engineering*, 18(10):1323–1337, 2006.

[72] Jeffrey Xu Yu and Jiefeng Cheng. Graph reachability queries: A survey. In *Managing and Mining Graph Data*, pages 181–215. Springer, 2010.

[73] Matei Zaharia, Mosharaf Chowdhury, Tathagata Das, Ankur Dave, Justin Ma, Murphy McCauley, Michael J Franklin, Scott Shenker, and Ion Stoica. Resilient distributed datasets: A fault-tolerant abstraction for in-memory cluster computing. In *Proceedings of the 9th USENIX conference on Networked Systems Design and Implementation*, pages 2–2. USENIX Association, 2012.

[74] Fred Zemke. What. s new in sql: 2011. *ACM SIGMOD Record*, 41(1):67–73, 2012.

[75] Gensheng Zhang, Xiao Jiang, Ping Luo, Min Wang, and Chengkai Li. Discovering general prominent streaks in sequence data. *ACM Transactions on Knowledge Discovery from Data (TKDD)*, 8(2):9, 2014.

[76] Peixiang Zhao, Xiaolei Li, Dong Xin, and Jiawei Han. Graph cube: on warehousing and olap multidimensional networks. In *Proceedings of the 2011 ACM SIGMOD International Conference on Management of data*, pages 853–864. ACM, 2011.

[77] Kai Zheng, Yu Zheng, Nicholas Jing Yuan, and Shuo Shang. On discovery of gathering patterns from trajectories. In *Data Engineering (ICDE), 2013 IEEE 29th International Conference on*, pages 242–253. IEEE, 2013.

[78] Yu Zheng. Trajectory data mining: an overview. *ACM Transactions on Intelligent Systems and Technology (TIST)*, 6(3):29, 2015.

[79] Yu Zheng. Trajectory data mining: an overview. *ACM Transactions on Intelligent Systems and Technology (TIST)*, 6(3):29, 2015.

[80] Yu Zheng, Yanchi Liu, Jing Yuan, and Xing Xie. Urban computing with taxicabs. In *Proceedings of the 13th international conference on Ubiquitous computing*, pages 89–98. ACM, 2011.

[81] Wenzhi Zhou, Hongyan Liu, and Hong Cheng. Mining closed episodes from event sequences efficiently. In *Pacific-Asia Conference on Knowledge Discovery and Data Mining*, pages 310–318. Springer, 2010.