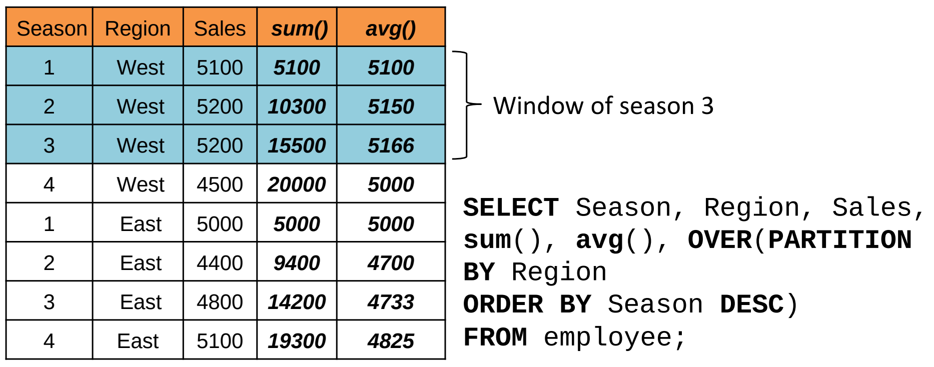
Chapter 1

Introduction

With the maturity of database technologies, nowadays applications collect data in all domains at an unprecedented scale. For example, billions of social network users and their activities are collected in the form of *graphs*; Thousand sensor reports are collected per second in the form of *time series*; Hundreds of millions of temporal-locations are collected as *trajectories*, to name just a few. Flooded by the tremendous amount of data, it is emerging to provide useful and efficient analytics for various data domains. Traditional SQL analytics which comprises of operations (such as, partition, sorting and aggregation) in the relational domain become limited in non-structured domains. In SQL context, interesting analytics such as graph traversal and pattern detection often involve complex joins which are very hard to optimize without domain knowledge. In this thesis, we explore the neighborhood data analytic, which in SQL is expressed by the window function, on different domains and demonstrate how to efficiently deploy neighborhood analytic to gain useful insights.

1.1 Neighborhood Analytic

By its self-describing name, neighborhood analytic aims to provide summaries of each object over its vicinity. In contrast to the global analytics which aggregates the entire collection of data as a whole, neighborhood analytic provides a personalized view on each object per se. Neighborhood data analytics originates from the window function defined in SQL which is illustrated in Figure 1.1.



Season	Region	Sales	***sum()***	***avg()***
1	West	5100	***5100***	***5100***
2	West	5200	***10300***	***5150***
3	West	5200	***15500***	***5166***
4	West	4500	***20000***	***5000***
1	East	5000	***5000***	***5000***
2	East	4400	***9400***	***4700***
3	East	4800	***14200***	***4733***
4	East	5100	***19300***	***4825***

Window of season 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 season-3 is highlighted.

As shown in the figure, the sales report contains five attributes: "Season", "Region" and "Sales" are the original fields, "`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 other tuples o_j such that o_i and o_j are in the same "region" and o_j's "season" is prior to o_i's. The window of season-3 for region-"West" is highlighted. Apart from this example, there are many other usages of the window function in the relational context. Being aware of the success of the window function, SQL 11 standard incorporates "`LEAD`" and "`LAG`" keywords which offer fine-grained specifications on a tuple's window.

Despite the usefulness, there are few works reporting the window analytics in the non-relational domain. This may dues to the usage of *sorting* in relational windows. For example, in Figure 1.1, objects need to be sorted according to "Season", and then the window of each objects is implicitly formed. However, in non-relational context, sorting may be ambiguous and even undefined.

To generalize the window function to other domains, we propose the neighborhood analytics in a broader context. Given a set of objects (such as tuples in relational domain or vertexes in graph domain), the neighborhood analytic is a composite function ($\mathcal{F} \circ \mathcal{N}$) applied on every object. $\mathcal{N}$ is the *neighborhood function*, which contains the related objects of an object; $\mathcal{F}$ is an *analytic function*, which could be aggregate, rank, pattern matching, etc. Apparently, the relational 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 $\mathcal{F} = \texttt{avg}$. Since the *sorting* requirement is relaxed, our neighborhood analytics is able to enrich the semantic of relational window notations and can be applied on many other domains.

1.2 Scope of the Thesis

In this thesis, we explore the neighborhood analytics in three prevalent data domains, namely **attributed graph**, **sequence data** and **trajectory**. Since the neighborhood analytics could be very broad, this thesis only focus on the following two intuitive neighborhood functions:

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.

There could be other types of neighborhood functions with different level of granularity. However, despite the simpleness of these two neighborhood functions, they are indeed versatile in representing many useful analytics.

1.3 Contributions

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

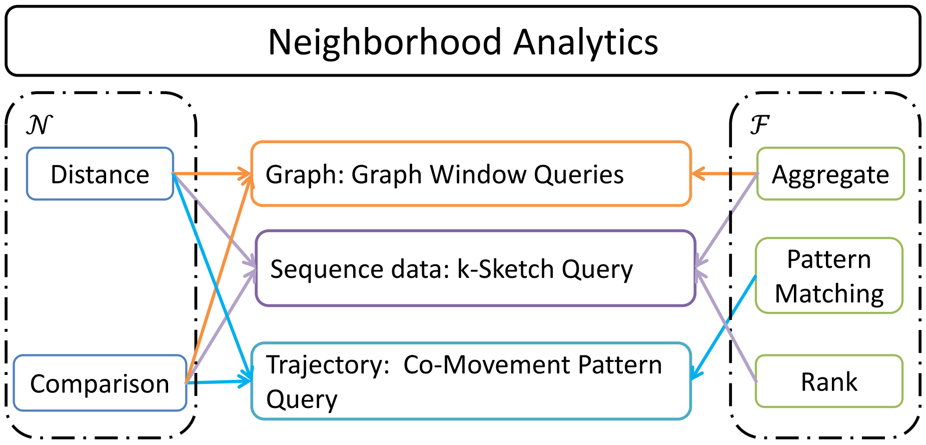


Figure 1.2: The road map of this thesis. There are three major contributions as highlighted in the center. Each contribution is a neighborhood analytic based on different $\mathcal{N}$ and $\mathcal{F}$ as indicated by arrows.

In a nutshell, we propose and enrich the neighborhood analytics in three data domains. In *graph*, we identify two types of window queries: *k-hop* window is based on the *distance* neighborhood and *topological* window is based on the *comparison* neighborhood. In *sequence data*, we leverage the nested *distant* and *comparison* neighborhoods to design the k-sketch query in finding prominent rank-aware streaks. In *trajectory*, we analyze existing movement patterns based on *distance* and *comparison* neighborhoods. Then we propose a general query to capture movement patterns built on top of the *comparison* neighborhoods.

1.3.1 Window Queries for Graph Data

The first piece of the thesis deals with neighborhood query 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:

Example 1.3.1 (k-hop window)**.** In a social network (such as Linked-In and Facebook etc.), users are normally modeled as vertexes and connectivity relationships are modeled as edges. In the social network scenario, it is of great interest to summarize the most relevant connections to each user such as the neighbors within 2-hops. Some analytic queries such as summarizing the related connections' distribution among different companies, and computing age distribution of the related friends can be useful. In order to answer these queries, collecting data from every user's neighborhoods within 2-hop is necessary.

Example 1.3.2 (Topological window)**.** In biological networks (such as Argocyc, Ecocyc etc.), genes, enzymes and proteins are vertexes and their dependencies in a pathway are edges. Because these networks are directed and acyclic, in order to study the protein regulating process, one may be interested to find out the statistics of molecules in each protein production pathway. For each protein,we can traverse the graph to find every other molecules that are in the upstream of its pathway. Then we can group and count the number of genes and enzymes among those molecules.

The two *windows* shown in the 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 on supporting efficient window processing. 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 indexing time as compared to the state-of-the-art competitor and upto 10x speedup in query processing.

1.3.2 k-Sketch Query for Sequence Data

The second piece of the thesis explores the neighborhood query for sequence data. In sequence data analysis, an important and revenue-generating task is to detect phenomenal patterns. An outstanding neighborhood based pattern is the *streak* which are constructed by aggregating temporally nearby data points for each subject. Streak has been found useful in many time series applications, such as network monitoring, stock analysis and computational journalism. However, the amount of streaks is too huge (i.e., $O\binom{N}{2}$ for a subject with N data points) to conduct effective analytics. Observed that many streaks share almost identical information, we are motivated to propose a k-Sketch query to effectively select k most representative streaks.

The k-sketch query is built on a core concept named *rank-aware streak*, out of which k representatives are chosen based on a novel scoring function that balances the strikingness and the diversity. A *rank-aware streak* enriches traditional streak by including the relative position of a streak among its cohort. Such a concept is not rare in real-life. For example, journalists often adopt the rank-aware streak to promote the attractiveness of their news:

1. (Feb 26, 2003) With 32 points, Kobe Bryant saw his 40+ scoring streak end at **nine** games, tied with Michael Jordan for **fourth** place on the all-time list[1].
2. (April 14, 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**)[2].
3. (May 28, 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[3].
4. (Jun 9, 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[4].
5. (Jul 22, 2011) Pelican Point recorded a maximum rainfall of 0.32 inches for **12 months**, making it the **9th driest** places on earth[5].

In the above examples, each news is a rank-aware streak consisting of five elements: a subject (e.g., Kobe Bryant, Stocks, Delhi), an event window (e.g., nine straight games, seventh consecutive days, past month), an aggregate function on an attribute (e.g., minimum points, count of gains, average of degrees), a rank (e.g., fourth, first time, highest), and a historical dataset (e.g., all time list, seven years, since 1952). These indicators are summarized in Table 1.1.

[1]`http://www.nba.com/features/kobe_40plus_030221.html`
[2]`http://www.cbssports.com/nba/eye-on-basketball/24525914/stephen-curry-makes-history-with-consecutive-seasons-of-250-3s`
[3]`http://www.zacks.com/stock/news/176469/china-stock-roundup-ctrip-buys-elong-stake-trina-sol`
[4]`http://www.dnaindia.com/delhi/report-delhi-records-highest-temperature-in-62-years-1994332`
[5]`http://www.livescience.com/30627-10-driest-places-on-earth.html`

The rank-aware streak can be formally described using a joint neighborhood function. Let $e_s(t)$ denote the event of subject s at time t. Then the rank-aware streaks are generated using neighborhood analytics 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 *event windows* 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 event window among all other event windows with the same window size. The result of the this step is a tuple $\langle o_i, w, t, r \rangle$, where r is the *rank*.

E.g.	**Subject**	**Aggregate function**	**Event window**	**Rank**
1	Kobe Bryant	`min`(points)	9 straight games	4
2	Stephen Curry	`sum`(shot attempts)	82 games	10
3	Stocks	`count`(gains)	7 consecutive days	1
4	Delhi	`avg`(degree)	past months (30 days)	1
5	Pelican Point	`max`(raindrops)	12 months	9

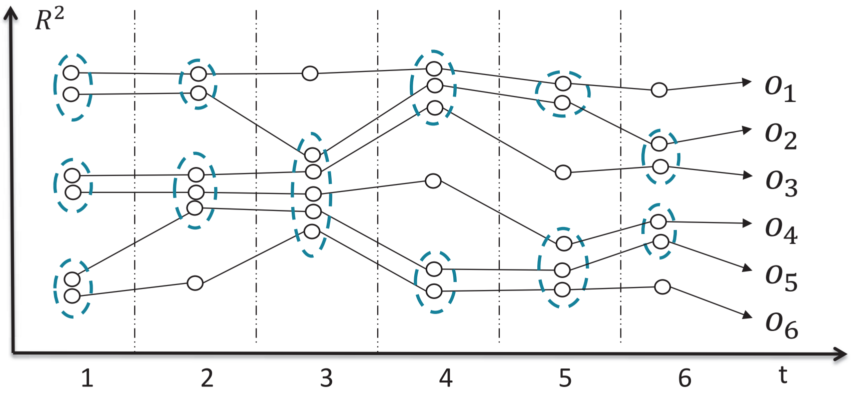
Table 1.1: News theme summary

Besides proposing the rank-aware streak, we also technically study how to efficiently support the *k-Sketch* query in both offline and online scenarios. We propose various window-level pruning techniques to find striking candidate steaks. Among those candidates, we then develop approximation methods, with theoretical bounds, to discover k-sketches for each subject. We conduct experiments on 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 [6].

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

1.3.3 Co-Movement Pattern Query in Trajectory Data

The third piece of the thesis studies the neighborhood query on trajectory data. An important mining task in trajectory domain is to discover the movement patterns of objects which is useful in a wide spectrum of applications. Existing studies have identified several types of interesting movement patterns called *co-movement* patterns. A co-movement pattern refers to a group of moving objects traveling together for a certain period of time and the group of objects is normally determined by their spatial proximity. A pattern is prominent if the group size exceeds M and the length of duration exceeds K. Inspired by the basic definition and driven by different mining applications, there are a bunch of variant co-movement patterns that have been developed with more advanced constraints.



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 1.3: 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 1.3 is an example to demonstrate the concepts of 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 black-box 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 $\langle o_5, o_6 : 3, 4, 5\rangle$ remains 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.

These co-movement patterns can be uniformly described as a two-step neighborhood query as follows: (1) $\mathcal{N}_1$ is a *distance neighborhood* used to determine the spatial proximity of objects. For example, flock and group 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, swarm and platoon 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 the temporal constraints, i.e., $\mathcal{N}_2(o_i) = \{o_j, T | \forall t \in T, C_t(o_i) = C_t(o_j)\}$, where $C_t(\cdot)$ returns the cluster ID of an object at time t.

Based on the neighborhood unification, we propose a *General Co-Movement Pattern* (GCMP) query to capture all existing co-movement patterns. 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., $\mathcal{N}_2$). It is notable that the GCMP query is able to detect any of the existing co-movement patterns by adopting different analytic functions.

Technical wise, we study how to efficiently processing GCMP in a MapReduce platform to gain scalability for large-scale trajectory databases. 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 then show the efficiency of both our methods in the Apache Spark platform with three real trajectory datasets upto 170 million points. The results show that SPARE achieves upto 14 times efficiency as compared to TRPM, and 112 times speedup as compared to the state-of-the-art centralized schemes.

1.4 Thesis Organization

The remaining part of the thesis are organized as follows: in Chapter 2, we summarize related literature on our proposed neighborhood based queries in different data domains. In Chapter 3, we present the window function on graph data. In Chapter 4, we present the k-sketch query on sequence data. In Chapter 5, we present a pattern mining framework on trajectory data. Chapter 6 summarizes this thesis and highlights future directions.