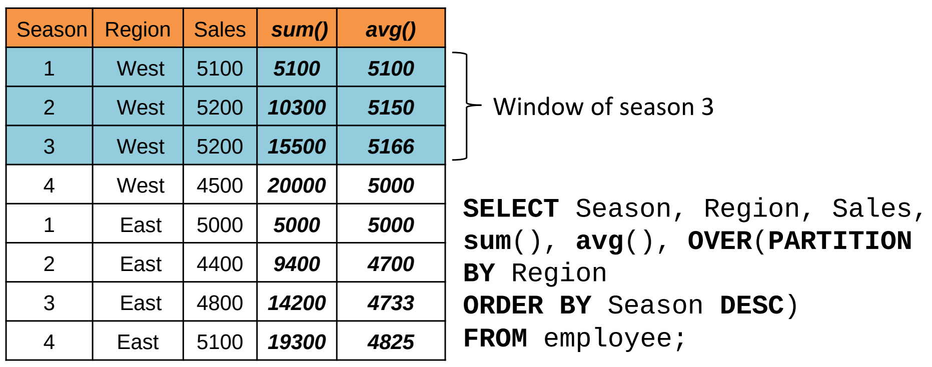
Chapter 1

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***

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

Window of season 3

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 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 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 very 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 neighborhood 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.

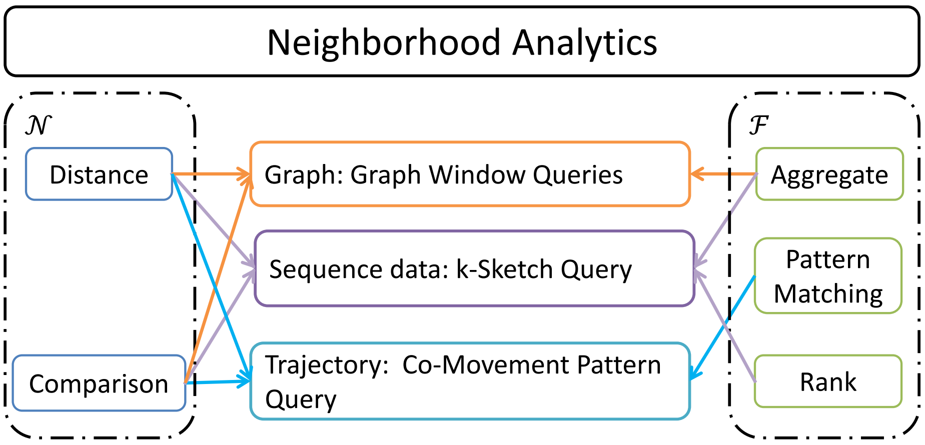


Figure 1.2: The road map of this thesis. There are three major contributions as highlighted in the center. Each contribution is a neighborhood analytic based on different $\mathcal{N}$ and $\mathcal{F}$ as indicated by arrows.

In a nutshell, we propose the neighborhood queries in three data domains. In *graph*, we identify two types of queries: *k-hop window query* is based on the *distance* neighborhood and *topological window query* is based on the *comparison* neighborhood. In *sequence data*, we propose a nested *distant* and *comparison* neighborhoods query named k-Sketch to finding striking rank-aware streaks. In *trajectory*, we analyze existing movement patterns based on *distance* and *comparison* neighborhoods and unify the existing works with a general pattern discovery query. Next, we present our contributions in detail for each of the data domains.

1.3.1 Window Queries for Graph Data

The first piece of the thesis deals with neighborhood query on graph data. We identify that there are a wealth information for many applications lying inside each vertex's neighborhoods. Thus, we are motivated to propose the Graph Window Queries to support neighborhoods based analytics in graphs. Two quick examples of the Graph Windows that we have identified are as follows:

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 LinkedIn, it is of great interest to summarize the potential connections of each user for the purpose of recommendation. These information are embedded in the 2-hop neighborhoods of each user.

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. In biological study, it is of great interest to study the types of enzymes influencing each proteins. These information are contained in the upstream of each molecule's pathways.

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 further 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 4.1.

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 $n = \langle o_i, w, t, \overline{v} \rangle$.

[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

(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

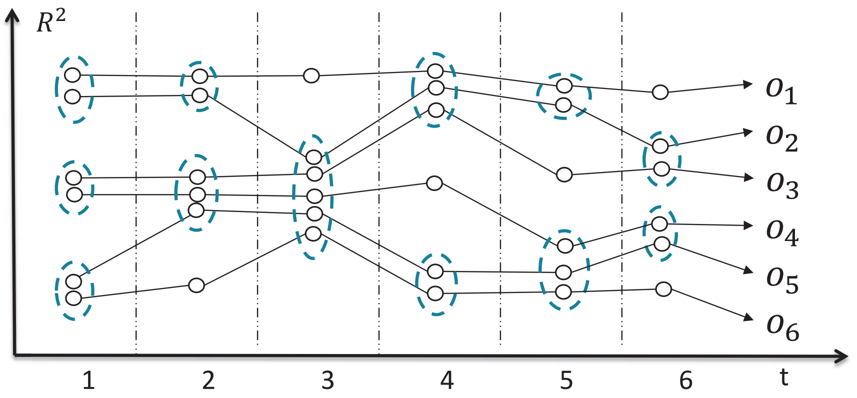
Besides proposing the rank-aware streak, we also technically study how to efficiently support the *k-Sketch* query in both offline and online scenarios. We propose various window-level pruning techniques to find striking candidate steaks. Among those candidates, we then develop approximation methods, 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].

1.3.3 Co-Movement Pattern Query in Trajectory Data

The third piece of the thesis studies the neighborhood query on trajectory data. In trajectory domain, the neighborhoods of objects across multiple time snapshots collectively form a movement pattern. Due to its wide spectrum of applications, mining the movement patterns have attracted many research attention. 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 to-

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

gether 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 5.1 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.

As shown, there are various types of co-movement patterns facilitating different application needs and it is cumbersome to deploy and optimize each of the pattern discovery algorithms. Therefore, it calls for a general framework to provide versatile and efficient support of all these pattern discoveries. We observe that 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.

Enlightened by 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.

On the technical side, we study how to efficiently processing GCMP in a MapReduce platform to gain scalability for large-scale trajectory databases. 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 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.

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] 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.

[3] 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.

[4] 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.

[5] 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.

[6] 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.

[7] 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.

[8] 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.

[9] 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.

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

[11] 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.

[12] 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.

[13] 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.

[14] 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.

[15] 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.

[16] 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.

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

[18] 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.

[19] 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.

[20] 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.

[21] 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.

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

[23] 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.

[24] 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.

[25] 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.

[26] 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.

[27] 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.

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

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

[30] 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.

[31] 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.

[32] 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.

[33] 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.