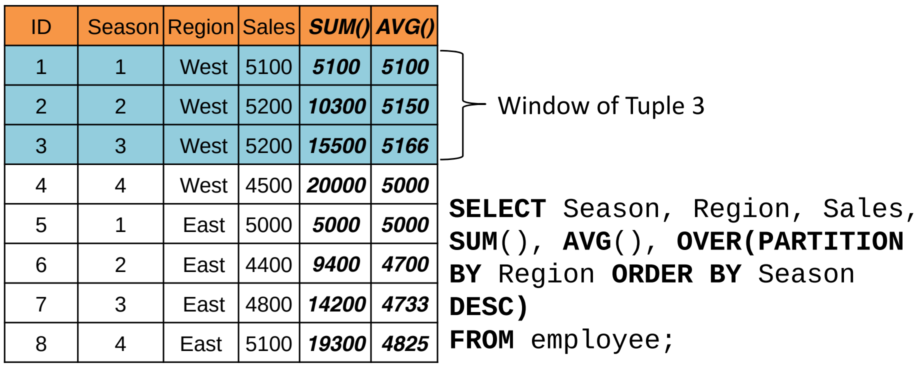
Chapter 1

Introduction

With the maturity of database technologies, nowadays applications 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 emerging to efficiently conduct analytics to discover useful insights. However, traditional SQL analytics which comprises of primary operations (e.g., partition, sorting and aggregation) becomes limited in coping 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 bring to light the *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 nowadays data sizes.

1.1 Neighborhood Analytics

Followed by its self-descriptive name, 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 other 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 [68] 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 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 objects is implicitly formed based on the sorted order. However, in other data domains, 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 Scope of the Thesis

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

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 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 nowadays scale. The roadmap of this thesis is shown in Figure 1.2.

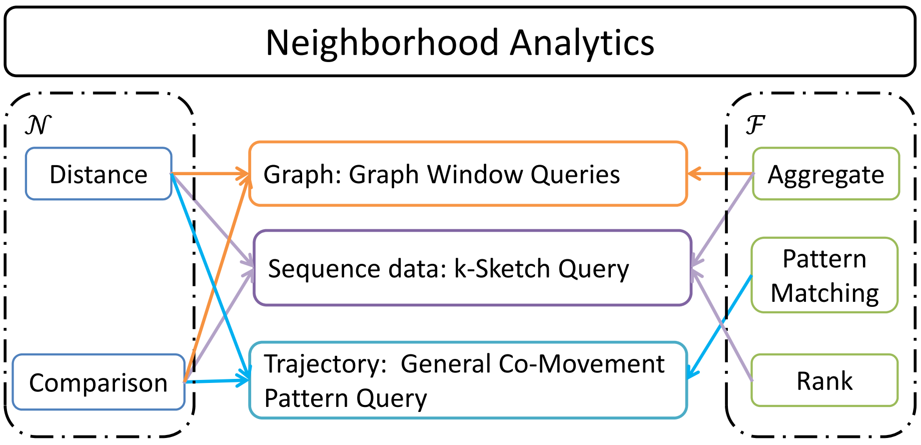

Neighborhood Analytics
$\mathcal{N}$
Distance
Comparison
Graph: Graph Window Queries
Sequence data: k-Sketch Query
Trajectory: General Co-Movement Pattern Query
$\mathcal{F}$
Aggregate
Pattern Matching
Rank

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 summarize the vicinity of each vertex. The query utilize the 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 is on the basis of a nested *distant* and *comparison* neighborhoods 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 on supporting efficient Graph Window Query 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 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, 69], automatic fact checking [30, 56], and perturbation analysis [57]. An outstanding example of such patterns is the *streak* [69], 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 the 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* [40, 72]. 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* [28], *convoy* [32], *swarm* [43], *group* [59] and *platoon* [42]. 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 [28] and group [59] patterns uses the *disk-based* clustering, which is equivalent to $\mathcal{N}_1(o_i) = \{o_j | \mathtt{dist}(o_i, o_j) < r\}$ for each object. Convoy [32], swarm [43] and platoon [42] patterns uses the *density-based* clustering, which is equivalent to $\mathcal{N}_1(o_i) = \{o_j | \mathtt{dist}(o_j, o_k) \leq \epsilon \wedge o_k \in \mathcal{N}_1(o_i)\}$.

(2) $\mathcal{N}_2$ is a comparison neighborhood used to determine the temporal constraints, i.e., $\mathcal{N}_2(o_i) = \{o_j, T | \forall t \in T, C_t(o_i) \equiv C_t(o_j)\}$, 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 adopting different analytic functions, GCMP query is able to detect any of the existing patterns.

On the technical side, we study how to efficiently processing 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.

Chapter 2

Literature Review

Our proposed neighborhood based queries are inspired by the usefulness of window functions in SQL analytics [68]. A window function in SQL specifies a set of partitioning attributes A and an aggregate function f. Its evaluation first sorts input records based on A to form overlapped partitions for each record. And then, f is evaluated for every partition and the aggregate result is associated with the corresponding records. Several optimization techniques [15, 7] have been developed to evaluate complex SQL queries involving multiple window functions.

However, the semantic and evaluation of the window function are restricted. In SQL window functions, tuples need to be sorted in order to form individual partitions (i.e., windows). In fact, such a need is hard to meet in other data domains. Therefore, optimization techniques that are developed for the relational model become inapplicable in other data domains. Nevertheless, there are quite a few works that related to our proposed neighborhood based queries and we review them in this section.

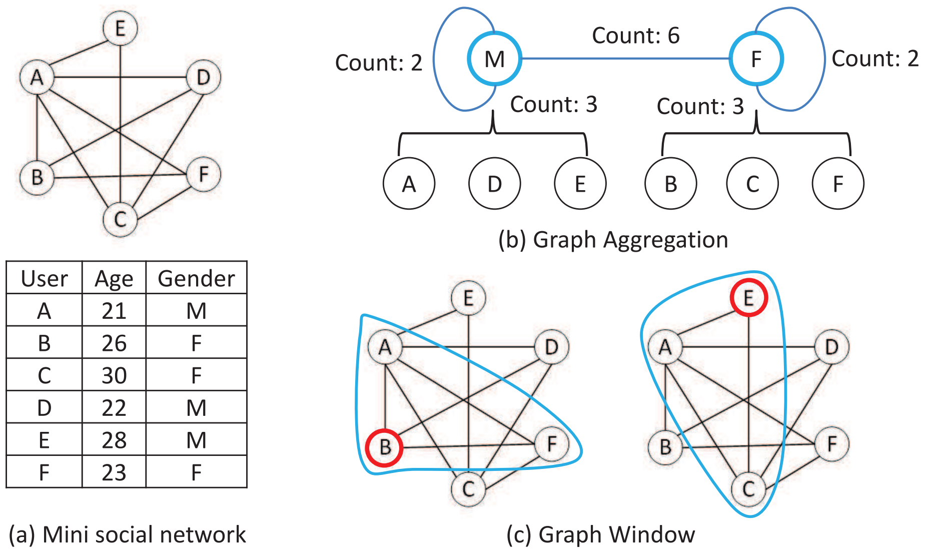
2.1 Graph Window Queries

2.1.1 Graph Aggregation

Previous graph data analytics focus on graph aggregation [70, 60, 16, 53], which are different from Graph Window Queries (GWQ). In a general model, graph aggregation comprises three steps: (1) partition graph based on attributes of vertex (and/or edges), (2) aggregate each partition to form Aggregated Nodes, and (3) link each aggregate node to form one Aggregated Graph. An illustration of the Graph Aggregation is shown in Figure 2.1 (b). In the first step, the input graph is partitioned on the“Gender” attribute of vertex which results in two partitions. In the second step, two aggregated nodes are formed, i.e., M (stands for Male) containing nodes A, D, E and F (stands for female) containing nodes B, C, F. In the third step, the links between M and F are added, with the "count" attached on each links. Differently, Graph Window Queries perform graph analytics from the vertex-centric perspective. In GWQ, the neighborhood structure of each vertex form overlapping partitions. Then, analytics are computed over each neighborhood structure. In Figure 2.1 (c), the neighborhood structure of B and E are highlighted. Clearly, the GWQ is different from graph aggregation and they could not model each other.

2.1.2 Reachability Queries and Indexes

Classic reachability queries, which answer whether two vertexes are connected, have been studied extensively in literature. To facilitate fast query processing, many indexes are proposed [18, 19, 61, 66]. Although our graph window queries can be built on top of the reachability queries, directly using these techniques is inefficient. For example, the most related reachability query to our k-hop window query is the k-reach query [19] which test if an input pair of vertexes is within a k-hop distance. In order to compute the k-hop window query for n vertexes, there would be $\theta(n^2)$



User	Age	Gender
A	21	M
B	26	F
C	30	F
D	22	M
E	28	M
F	23	F

Figure 2.1: Illustration of Graph Aggregation and Graph Window Queries. (a) is an example social network, (b) is graph aggregation, (c) are the windows of vertexes B and E.

reachability tests. This would be inefficient on graphs with over millions of vertexes.

2.1.3 Top-k Neighborhoods

In [63], the authors investigated the problem of finding the vertexes that have top-k highest aggregate values over their h-hop neighbors. This is similar to our k-hop query, while the difference is that they focus on providing pruning techniques to select the k best vertexes and our graph window query aims to compute the analytics for each vertex. Therefore, in our setting, the pruning techniques in [63] does not take effect and would be equivalent to the non-indexed approach.

2.1.4 Egocentric Networks

Egocentric networks [45, 48] have been playing an important role in network study. The egocentric networks refer to the neighborhood structure of each vertex in a graph. Although there are many works have studied on egocentric networks in structural analysis, they do not consider efficient processing of data analytics (e.g., aggregation) within each egocentric networks. Recently, Jayanta et.al. [49] proposed an EAGR system to summarize attribute information among each vertex's neighborhoods. Their technique is to build an overlay graph to leverage the shared components among vertexes' neighborhoods structures to boost query processing. Technically, EAGR runs in iterations and starts with the vertex-neighborhood mapping as the initial overlay graph. During each iteration, it sorts vertexes in an overlay graph according to their neighborhood information. Then an FP-Tree [29] is built to mining the largest shared components based on the sorted vertexes. As the algorithm iterates, the overlay graph evolves to be sparser.

The main drawback of EAGR is its high demands of resources on the overlay construction. In terms of memory cost, EAGR assume the initial vertex-neighbor mapping can be stored in memory. However, the assumption does not scale well for computing higher hop windows (such as k $\geq$ 2). For instance, a LiveJournal social network graph [1] (4.8M nodes, 69M edges) generates over 100GB mapping information for k=2 in adjacency list representation. If the neighborhood information is resided in disk, the performance of EAGR will largely reduced. In terms of computational cost, EAGR requires to sort all vertexes in a graph and build an FP-Tree in each iteration. When the graph has millions of vertexes, the indexing is largely slow down.

We tackle these drawbacks by adopting a hashing based approach that clusters each vertex according to its neighborhood similarity. During the hashing, a vertex's neighborhood information is computed on-the-fly. As compared to the sorting based approach, we do not require vertex's neighborhood to be resided in memory. In order to reduce the repetitive computation, we adopt a Dense Block heuristic to leverage the shared components among vertexes' neighborhoods. We then propose an estimation scheme that further reduces the number of neighborhood accesses. Experiments show that our schemes outperform EARG in both query processing and

[1] Available at http://snap.stanford.edu/data/index.html, which is used [49]

memory usage. Our methods are able to perform well even when EAGR algorithm fails when neighborhood information overwhelms system's memory, and our methods takes much shorter indexing time.

2.2 k-Sketch Query

Our proposed k-Sketch query is closely related to the following four areas: *news discovery in computational journalism*, *frequent episode mining on sequence data*, *top-k diversity query* and *event detection in information retrieval.*

2.2.1 Computational Journalism

An important aspect of computational journalism is to leverage computational technology to discover striking news themes. Previous works on automatic news theme generation belong to two categories: *dimension-oriented* approach and *subject-oriented* approach. Dimension-oriented approaches aim to select appropriate dimensions to make an event interesting. Representative works include the *situational facts* [51] and the *one-of-the-few facts* [62]. On the other hand, subject-oriented approaches aim to summarize subjects' histories from their historical events, such as the *prominent streaks* [69]. Our proposed solution falls into the subject-oriented category.

2.2.1.1 Situational facts and One-of-the-few facts

Situational Facts [51]: it finds for a given event, the best constraint-measure pair that makes the event unique (i.e., not dominated by others). An example of the *situational fact* is listed in Table 2.1 and is extracted from the NBA game events where the measure dimensions are "points, assists, and rebounds" and the constraint dimensions are "team, result, and date". The *situational fact* of the event in Table 2.1 is the constraint-measure pair $\langle$team=Blazer, points$\rangle$, since under this situation, this event is a skyline among all events (i.e., no other events matching the constraint "team=Blazer" contain an even higher "point").

One-of-the-few Facts [62]: it finds the dimensions under which no more than τ events are in the k-skyband (i.e., not dominated by k events and k is as small as possible). An example of the *one-of-the-few fact* is listed in Table 2.1. In the example, when $\tau = 5$, two dimensions are selected (i.e., points and rebounds). Under these two dimensions, five players are the skylines (i.e., 1-skyband), thus each of them is a *one-of-the-5* player.

As demonstrated, "situational facts" and "one-of-the-few facts" are dimension-oriented since they attempt to generate news themes by selecting dimensions.

Table 2.1: Examples of different news themes

Method	**Example news theme**
Situational facts [51]	Damon Stoudamire scored 54 points on January 14, 2005. It is the highest score in history made by any Trail Blazer.
One-of-the-τ facts [62]	Jordan, Chamberlain, James, Baylor and Pettit are the five players with highest points and rebounds in NBA history.
Prominent streaks [69]	1.Kobe scored 40+ in 9 straight games, first in his career! 2.Kobe scored 50+ in 4 straight games, first in his career! 3...
k-Sketch	1.Kobe scored 40+ in 9 straight games, ranked 4th in NBA history! 2.Kobe scored 50+ in 4 straight games, ranked 1st in NBA history. 3....

2.2.1.2 Prominent streaks

Zhang et al. [69] proposed a subject-oriented approach to generate news themes by discovering *prominent streaks*. In [69], a streak is modeled as a pair of the streak duration and the minimum value of all events in the streak. For example, as shown in Table 2.1, a streak of "Kobe" may be $\langle$9 consecutive games, minimum points of 40$\rangle$. The objective of [69] is to discover all the non-dominated streaks (i.e., prominent streaks) where the dominance is defined among streaks of the same subject. Despite being the same subject-oriented approach, our k-Sketch query differs from [69] in two aspects. First, we look at the *global prominence among all* subjects (i.e., rank in NBA history) rather than local prominence within one subject (i.e., non-dominance in one's career). Second, our model provides the best k ranked-streaks for each subject, whereas [69] returns a set of skylines which could be potentially large.

2.2.2 Frequent Episode Mining

In sequence data mining, an episode [47, 75, 52, 38] is defined as a collection of time sequenced events which occur together within a time window. The uniqueness of an episode is determined by the containing events. The objective of frequent episode mining is to discover episodes whose occurrences exceed a threshold. Our k-Sketch query differs from the episode mining in two major aspects. First, episodes are associated with categorical values thus they can be grouped to count the occurrences. On the other hand, our ranked-streaks are defined with numerical values, making it inappropriate to be grouped. Second, the episodes are selected based on the occurrences which do not contain the *rank* information, whereas our k-Sketch query explicitly provides the rank of selected streaks. As such, episode mining techniques cannot support k-Sketch query.

2.2.3 Top-k Diversity Query

Top-k diversity queries [1, 9, 25, 17] aim to find a subset of objects to maximize a scoring function. The scoring function normally penalizes subsets with similar elements. Our k-Sketch query has two important distinctions. First, the inputs to top-k diversity queries are known in advance, whereas in k-Sketch query, the ranks of streaks need to be derived. Second, existing methods for online diversity queries [9, 25, 17] only study the update on a single result set when a new event arrives. However, our online sketch maintenance incurs the problem of multiple sketch updates for each new event. Such a complex update pattern has not been studied yet.

2.2.4 Event Detection and Tracking

In information retrieval, event detection and tracking aim to extract and organize new events from various media sources such as text streams [3, 10], social media streams [41] and web articles [55]. Despite the usefulness of these works, they differ from our k-Sketch query as they focuses on the detection of a single event, whereas k-Sketch aims to summarize a subject's history. Therefore, the abovementioned techniques cannot be directly applied.

2.3 Co-Movement Pattern Discovery

Related work can be grouped into three categories: *co-movement patterns*, *dynamic movement patterns* and *trajectory mining frameworks.*

2.3.1 Co-Movement Patterns

2.3.1.1 Flock and convoy

The difference between *flock* and *convoy* lies in the object clustering methods. In *flock*, objects are clustered based on their distances. Specifically, the objects in the same cluster need to have a pairwise distance less than *min_dist*. This essentially requires the objects to be within a disk-region of delimiter less than *min_dist*. In contrast, *convoy* clusters objects using density-based spatial clustering [27]. Technically, *flock* utilizes a m^{th}-order Voronoi diagram [37] to detect whether a subset of n ($n \geq m$) objects stay in a disk region. *Convoy* employs a trajectory simplification [24] technique to boost pairwise distance computations in the density-based clustering. After clustering, both *flock* and *convoy* use a sequential scanning method to examine each snapshot. During the scan, the object groups that do not appear in consecutive snapshots are pruned. However, such a method faces high complexity when supporting other patterns. For instance, in *swarm*, the candidate set during the sequential scanning grows exponentially, and many candidates can only be pruned after the entire dataset are scanned.

2.3.1.2 Group, swarm and platoon

Different from *flock* and *convoy*, all the *group*,*swarm* and *platoon* patterns have more relaxed constraints on the pattern duration. Therefore, their techniques of mining are of the same skeleton. The main idea of mining is to grow an object set from an empty set in a depth-first manner. During the growth, various pruning techniques are provided to prune unnecessary branches. *Group* pattern uses a VG-graph to guide the pruning of false candidates [59]. *Swarm* designs two more pruning rules called backward pruning and forward pruning [43]. *Platoon* [42] leverages a prefix table structure to steer the depth-first search, which shows efficiency as compared to the other two methods. However, the pruning rules adopted by the three patterns heavily rely on depth-first search which loses efficiency in a parallel scenario.

2.3.2 Other Related Trajectory Patterns

A closely related literature to co-movement patterns is the *dynamic movement* patterns. Instead of requiring the same set of object traveling together, *dynamic movement* patterns allow objects to temporally join or leave a group. Typical works include *moving clusters* [34], *evolving convoy* [4], *gathering* [71] etc. These works cannot model GCMP since they enforce global consecutiveness on the temporal domain.

2.3.3 Trajectory Mining Frameworks

Jinno et al. in [33] designed a MapReduce based algorithm to efficiently support T-pattern discovery, where a T-pattern is a set of objects visiting the same place at simliar time. Li et al. proposed a framework of processing online *evolving group* pattern [39], which focuses on supporting efficient updates of arriving objects. As these works essentially differ from co-movement pattern, their techniques cannot be directly applied to discover GCMPs.

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] 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.

[5] 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.

[6] 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.

[7] 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.

[8] Michael A Bender, Martín Farach-Colton, Giridhar Pemmasani, Steven Skiena, and Pavel Sumazin. Lowest common ancestors in trees and directed acyclic graphs. *Journal of Algorithms*, 57(2):75–94, 2005.

[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] Artur Czumaj, Mirosław Kowaluk, and Andrzej Lingas. Faster algorithms for finding lowest common ancestors in directed acyclic graphs. *Theoretical Computer Science*, 380(1):37–46, 2007.

[23] 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.

[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] 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.

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

[30] 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.

[31] 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.

[32] 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.

[33] 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.

[34] 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.

[35] 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.

[36] 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.

[37] 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.

[38] 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.

[39] 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.

[40] Xiaohui Li and Tan Kian-Lee. *Managing moving objects and their trajectories.* PhD thesis, National University of Singapore, 2013.

[41] 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.

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

[43] 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.

[44] 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.

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

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

[47] 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.

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

[49] 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.

[50] 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.

[51] 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.

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

[53] 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.

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

[55] 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.

[56] 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.

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

[58] 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.

[59] 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.

[60] 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.

[61] 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.

[62] 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.

[63] 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.

[64] 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.

[65] 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.

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

[67] 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.

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

[69] 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.

[70] 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.

[71] 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.

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

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

[74] 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.

[75] 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.