Mining Significant Trend Sequences in Dynamic Attributed Graphs

Philippe Fournier-Viger[a,*], Chao Cheng[b], Zhi Cheng[c], Jerry Chun-Wei Lin[d], Nazha Selmaoui-Folcher[e]

[a] *School of Humanities and Social Sciences, Harbin Institute of Technology (Shenzhen), China*
[b] *School of Computer Science and Technology, Harbin Institute of Technology (Shenzhen), China*
[c] *University of New Caledonia, ISEA, BP R4, F-98851 Noumea, New Caledonia*
[d] *Department of Computing, Mathematics, and Physics, Western Norway University of Applied Sciences (HVL), Bergen, Norway*
[e] *University of New Caledonia, ISEA, BP R4, F-98851 Noumea, New Caledonia*

Abstract

Discovering patterns in graphs has many applications such as social network, biological and chemistry data analysis. Although many algorithms were proposed to identify interesting patterns in graphs, most of them consider simple types of graphs such as static graphs or graphs having a single attribute per vertice. Recently, studies have considered discovering frequent patterns in dynamic attributed graphs. These graphs can represent not only relationships between entities but also how they evolve over time, and describe entities with multiple attributes. Algorithms for mining frequent patterns in dynamic attributed graphs select patterns based on their occurrence frequency. But a major drawback of this approach is that many frequent patterns may contain entities that are weakly correlated. Thus, numerous frequent but spurious patterns may be shown to the user. To allows discovering strongly correlated patterns in dynamic attributed graphs, this paper proposes a novel significance measure named *Sequence Virtual Growth Rate*. It allows evaluating if a pattern represents entities that are correlated in terms of their proximity in a graph over time. Based on this measure a novel type of graph patterns is defined called *Significant Trend Sequence*. To efficiently mine these patterns, two algorithms named TSeqMiner$_{dfs-bfs}$ and TSeqMiner$_{dfs-dfs}$ are proposed. They rely on a novel upper bound and pruning strategy to reduce the search space. Experimental results show that the proposed algorithms are efficient and can identify interesting patterns in real-world social network and flight data.

Keywords: graph mining, dynamic attributed graph, significance measure, sequence
2018 MSC: 00-01, 99-00

1. Introduction

In the last decades, analyzing graphs has received an increasing amount of attention from the data mining community. The reason is that graphs naturally capture the structure of data in many domains [1]. In particular, graph data is collected in many emerging applications such as social network [2], wireless sensor network [3] and biological network [4, 5] analysis. To analyze graphs, various techniques have been proposed such as to detect communities [6], outliers [7] and patterns in graphs. Traditional techniques to mine patterns in graphs discover structures such as subgraphs or trees that frequently appear in a graph or multiple graphs [8, 9]. Discovered patterns can then be used to understand the structure of graphs, and for decision-making [10].

*Corresponding author
Email addresses: philfv@hit.edu.cn (Philippe Fournier-Viger), tidescheng@gmail.com (Chao Cheng), zhi.cheng@univ-nc.nc (Zhi Cheng), jerrylin@ieee.org (Jerry Chun-Wei Lin), nazha.selmaoui@univ-nc.nc (Nazha Selmaoui-Folcher)

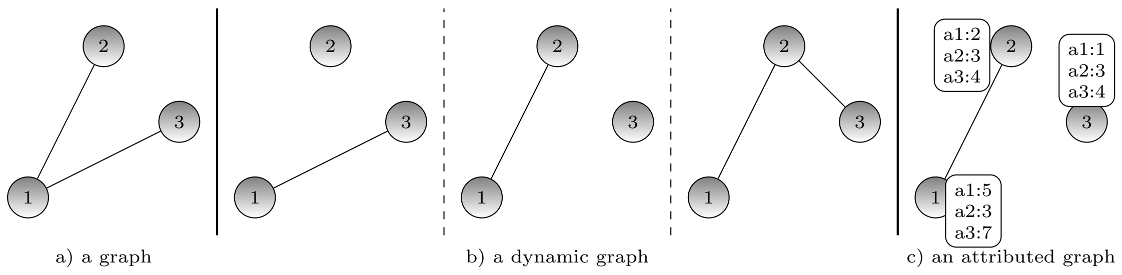

Figure 1: Examples of a) a graph, b) a dynamic graph, c) an attributed graph.

Several algorithms have been proposed for mining interesting subgraphs. However, many algorithms can only analyze simple graph structures such as static graphs (graphs that do not change over time) [11, 12, 13, 14], and graphs where vertices and edges are described using a single attribute [15, 16]. For instance, Fig. 1 a) presents a small graph representing relationships between people in a social network, where nodes represent people, uniquely identified using positive numbers, and edges represent friendship links. Not considering the time dimension allows to design very efficient algorithms to analyze graphs but fails to capture the dynamics of the graph, that is how the graph has evolved over time (e.g. how friendship relationships are formed).

To consider the time dimension in graph pattern mining, recent studies have considered discovering patterns in dynamic graphs [17, 18, 19]. A *dynamic graph* is a time-ordered sequence of graph snapshots where edges and nodes can be inserted, removed, and attribute values may change at each timestamp. Dynamic graphs are widely used to model social networks because relationships (edges) between persons (nodes) evolve over time. For instance, consider the dynamic graph depicted in Fig. 1 b). It shows how a graph has evolved during three consecutive timestamps. In particular, in this dynamic graph, a new edge has appeared and one has been removed at the second timestamp. Studying how such dynamic graph evolves over time can provide various useful insights such as how communities are formed, and how persons influences others. Besides social networks, dynamic graphs can be used to model data in many other applications. such as changes in ontologies [20], processes in complex systems [21], computer networks [22], spatio-temporal changes observed in satellite images [23] and movements of vehicles [24].

Although discovering patterns in dynamic graphs is useful, several work consider that no more than a single attribute can be used to describe vertices and edges. But in several domains such as social networks, edges and vertices can have multiple attributes. For example, a person in a social graph may be described using numerous attributes such as age, country and gender. Ignoring some of these attributes or combining them in a single attribute results in not discovering some interesting patterns involving multiple attributes or a subset of these attributes. To address this issue, some recent papers have considered mining patterns in *attributed graphs*, that is graphs where multiple attributes are used to describe vertices and edges [25, 26, 27]. To illustrate the concept of attributed graph, Fig. 1 c) shows a social graph where three numeric attributes denoted as $a1$, $a2$ and $a3$ are used to describe persons (nodes). For instance, the node 1 has values 5, 3 and 7 for the attributes $a1$, $a2$ and $a3$, respectively. Discovering patterns in an attributed graph can be used to find relationships between nodes and edges that involves attribute values. For example, patterns could be found indicating that people having similar values for a given attribute tend to be friends.

Discovering patterns in dynamic graphs and in attributed graphs is useful as they can respectively capture how the relationships and characteristics of graph entities evolve over time. But for many applications, it is necessary to jointly consider both of these aspects. For example, in social network analysis, it is useful to consider not only attributes of persons but also how the social graph (nodes, edges and attributes) evolves over time, to analyze the complex interactions between nodes. Hence, several studies have recently proposed algorithms to find interesting patterns in *dynamic attributed graphs* [28, 23], that is graphs that are both dynamic and attributed. Several of these algorithms find patterns indicating trends for sets of nodes [28, 23].

A trend means that the value of a given attribute has increased or decreased for some nodes over consecutive timestamps.For example, consider a social network graph such as DBLP where each node (person) is described using attributes indicating the number of papers published in some top conferences such as KDD.

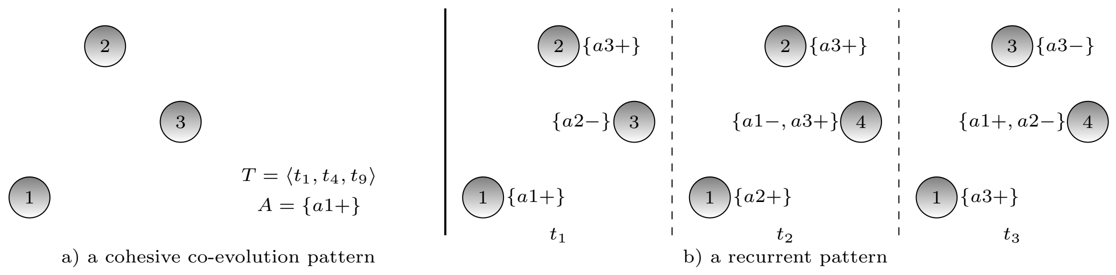

Figure 2: Illustration of a) a cohesive co-evolution pattern, represented as a triple (V=$\{1, 2, 3\}$, T=$\langle t_1, t_4, t_9\rangle$, A=$\{a1+\}$) and b) a recurrent pattern, ($\langle$(1: $\{a1+\}$|2: $\{a3+\}$|3: $\{a2-\}$)(1: $\{a2+\}$|2: $\{a3+\}$|4: $\{a1-, a3+\}$)(1: $\{a3+\}$|3: $\{a3-\}$|4: $\{a1+, a2-\}$)$\rangle$, t_1, t_4, t_9).

Algorithms such as [28, 23] can find a type of patterns called *co-evolution patterns* indicating that several nodes display a similar trend of an increased/decreased number of conference papers from one year (timestamp) to the next. Such pattern is illustrated in Fig. 2 a). It indicates that the value of the attribute $a1$ has increased for three nodes labelled as 1, 2 and 3 and that this pattern appeared in the first time interval t_1 (from the first to the second timestamp), the fourth time interval t_4 (from the fourth to the fifth timestamp) and the ninth time interval t_9 (from the ninth to the tenth timestamp). Recently, Cheng et al. [29] extended these studies by considering the evolution of not only trends but also vertices in a dynamic attributed graph, to discover a type of patterns called *recurrent pattern* representing the evolution of subgraphs in a dynamic attributed graph. The designed algorithm was used to analyze the evolution of co-authorship relationships between researchers in an academic social network and the impact of a hurricane on US flight delays [30]. An example of recurrent pattern is shown in Fig. 2 b). It indicates that during a time interval t_1, the attribute $a1$ of vertex 1 increased, the attribute $a3$ of vertex 2 increased, and the attribute $a2$ of vertex 3 decreased. Then, it was followed by a time interval t_2 where the attribute $a2$ of vertex 1 increased, the attribute $a3$ of vertex 2 increased, the attribute $a1$ of vertex 4 decreased, and the attribute $a3$ of vertex 4 increased. Then, it was followed by another time interval t_3 as depicted in Fig. 2 b). Such pattern can be seen as a generalization of the concept of co-evolution patterns.

Although current algorithms for mining patterns in dynamic attributed graphs such as those above were shown to be useful, most algorithms evaluate patterns based on their occurrence frequency, and aim to find frequent patterns. Using the frequency as main criterion to select patterns has the advantage of filtering some noise (patterns that are insignificant because they seldomly appear in a graph). But using the frequency as main selection criterion can result in discovering many weakly correlated frequent patterns. For example, it could be found that values of two attributes tend to increase together for some nodes of a dynamic attributed graph over consecutive timestamps. However, it is possible that this pattern exists just because these two attributes generally often increase in the database, rather than because of a correlation between these attributes for these nodes. Thus, generally, frequent patterns can be misleading as they do not ensure that values appearing in a frequent pattern have a strong correlation between them. A frequent pattern mining algorithm may thus find many frequent but weakly correlated patterns and ignore many not so frequent but strongly correlated patterns.

The importance of finding correlated patterns is illustrated with an example. Consider Fig. 3, which will be used as running example. It depicts the evolution of a dynamic attributed graph over six timestamps $(1, 2, \ldots 6)$ representing consecutive years. Consider that this is an academic social graph where nodes represent researchers labelled using numbers from 1 to 5, and attributes $a1$, $a2$ and $a3$ respectively denote the number of papers published in the KDD, ICDE and ICDM conferences. To study how attribute values change over time in such dynamic attributed graph, several studies propose to convert it into a *trend graph* [23, 31]. The trend graph corresponding to Fig. 3 is shown in Fig. 4. Fig. 4 a), b), c), d) and e) repectively depict the evolution of the graph from the 1st to 2nd, 2nd to 3rd, 3rd to 4th, 4th to 5th, and 5th to 6th timestamp. Attribute variations (trends) are represented using the “+”, or “-” symbols to indicate whether each attribute value has increased or decreased since the previous timestamp (year), respectively.

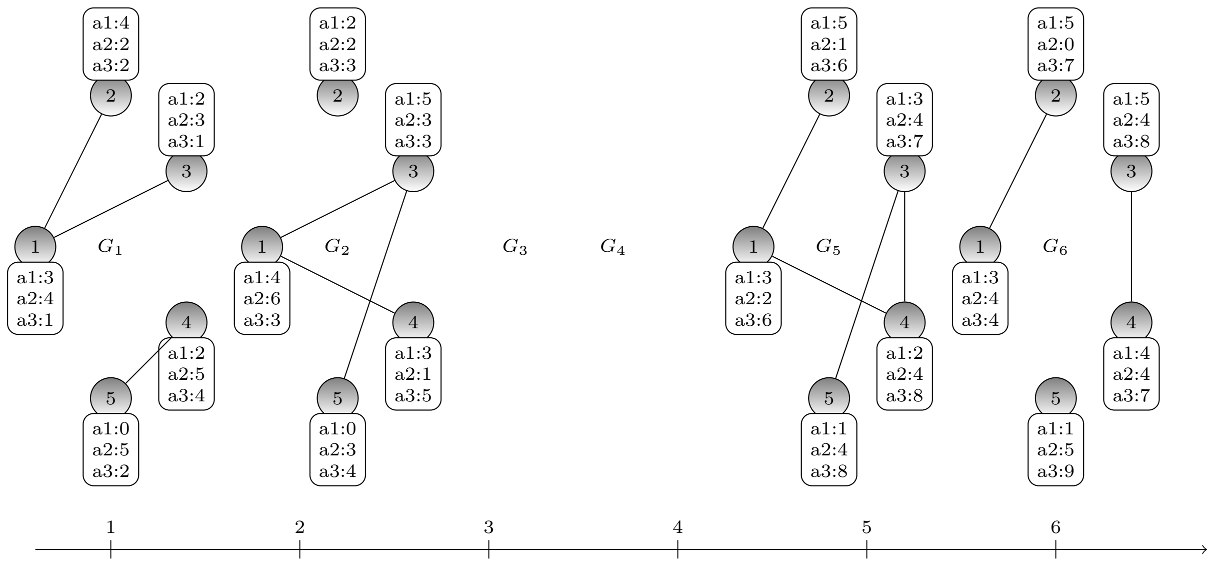

Figure 3: A dynamic attributed graph with 6 timestamps (G_3 and G_4 are not shown), from which the trend graph of Fig. 4 is derived.

Note that the symbol "=" could also be used to represent no significant change, but it is not considered in the examples of this paper. Now consider that one wants to find patterns indicating that a vertex influences the trends of its neighbors for the following timestamp, and in particular whether an increase in the number of KDD publications ($a1+$) of a researcher influences the number of publications of one of its neighbor the following year for KDD ($a1$), ICDE ($a2$) and ICDM ($a3$). By observing Fig. 4, it is found that the trend $\{a1+\}$ appear for several vertices such as vertex 1 and 3 at t_1, and vertex 4 at timestamp t_2. Consider the occurrence of $\{a1+\}$ for vertex 1 at timestamp t_1. That vertex is connected with vertices 2 and 3 at t_1. Consider the influence of $a1+$ of vertex 1 on vertices 2 and 3 at t_2. If only the frequency is considered to find patterns in that graph, patterns such as $\langle\{a1+\},\{a3+\}\rangle$ can be found, which indicates that an increase of attribute $a1$ for a vertex is followed by an increase of $a3$ for a neighbor. However, since {a3+} appears almost everywhere in the graph, the pattern $\langle\{a1+\},\{a3+\}\rangle$ is weakly correlated (it may simply appear by chance), and is uninteresting. Now consider the pattern $\langle\{a1+,a2+\},\{a3-\}\rangle$, which indicates that if the number of papers of a researcher in KDD and ICDE increases ($\{a1+,a2+\}$), then the number of ICDM papers of one of its collaborator will decrease ($\{a3-\}$) at the next timestamp. This pattern is interesting because although it has a relatively low support, there is a strong correlation between $\{a1+,a2+\}$ and $\{a3-\}$. In fact, the trend {a3-} does not frequently appear globally but it often locally appears following the trends {a1+, a2+}. Thus this pattern may represent a significant trend providing insights about how attributes of nodes evolve in that dynamic attributed graph.

As shown in the above example, to reveal meaningful patterns, it is desirable to measure not only the frequency but how significant or correlated values in a pattern are. Therefore, to filter the many spurious weakly correlated patterns and present meaningful patterns to the user, a novel significance measure is needed. Such measure would allow to find that {a3-} is more likely to appear after the observation {a1+, a2+} than {a3+} is likely to follow {a1+}.

In the field of pattern mining several measures have been proposed to assess whether values are correlated in a database. For example, in frequent itemset mining, measures such as the *bond*, *affinity*, *coherence* and *all-confidence* are used [32]. But these measures are not designed for graphs or spatial data. Huang et al. proposed to mine sequential patterns [33] in spatio-temporal data using a significance measure [34]. Although this framework is useful, it is not designed for graphs and cannot be easily extended for it. Besides, considering the time dimension to evaluate the significance of patterns in graphs is an important challenge.

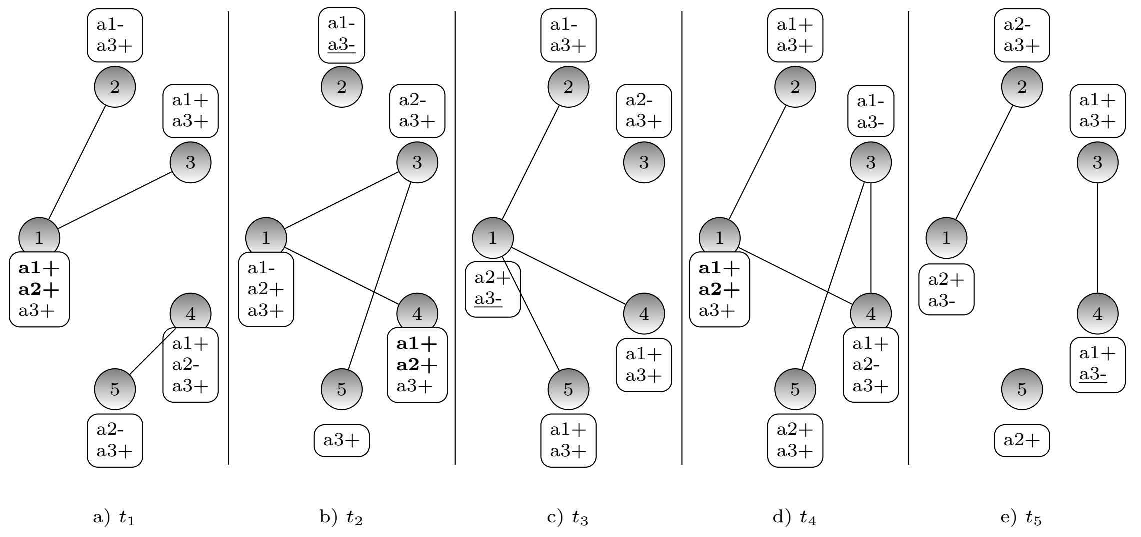

Figure 4: A sequence of graphs derived from preprocessing a dynamic attributed graph with 6 timestamps, which indicate significant pattern $< \{a1+, a2+\}, \{c-\} >$. Subgraph a) depict trend during time section 1. Each vertex is associated with a set of attribute trends with not significant trends ignored.

This paper addresses the above issues of previous work by proposing the novel problem of mining significant trend sequences in dynamic attributed graphs. The contributions of this paper are as follows:

- A novel significance measure is introduced, named *Sequence Virtual Growth Rate(SVGR)*, to evaluate if a trend sequence represents entities that are strongly correlated when considering time and space. This measure was developped by drawing inspiration from the concept of growth rate in emerging pattern mining [35, 36, 31] and the concept of spatio-temporal correlation in spatial data mining [34].
- Based on the proposed measure, a new type of patterns is defined, called *Significant Trend Sequence*. A significant trend sequence in a dynamic attributed graph is a sequence of trends, in which each trend has a strong correlation with its neighbors from the previous time interval. The concept of neigborhood is defined based on the proximity of vertices in a graph and can be parameterized. The problem of mining significant trend sequences is formalized and its properties are studied.
- Two efficient algorithms are developed to mine the proposed patterns, named *TSeqMiner*$_{dfs-dfs}$ and *TSeqMiner*$_{dfs-bfs}$, respectively. They rely on a novel efficient pruning technique and constraints to reduce the search space.
- To evaluate the performance of the algorithms, a quantitative study is performed. Results show that the pruning techniques are effective and that the algorithms are efficient. Moreover, an analysis of patterns found confirm that insightful patterns are discovered in real data.

The rest of the paper is organized as follows. Section 2 reviews related work. Section 3 introduces preliminaries and defines the proposed problem. Section 4 presents the algorithms. Section 5 reports results of the quantitative experiments and the analysis of patterns found. Finally, section 6 concludes the paper.

2. Related work

This section gives an overview of relevant related work for mining patterns in dynamic attributed graphs, discovering emerging patterns, and spatio-temporal data mining. Moreover, a brief overview of other techniques for capturing changes in dynamic graphs is presented.

2.1. Mining trends in dynamic attributed graphs

Analyzing trends in social networks is an important task for both the commercial and scientific communities [37]. Unlike supervised learning techniques such as classification models [38], pattern mining techniques rarely require labeled data and can provide results that are interpretable by humans. Recently, much attention has been given to mining patterns representing trends in graphs, which capture the evolutions of both entity relationships and entity attributes. To discover patterns containing trends from a dynamic attributed graph having continuous attributes, a graph is first transformed into a trend graph (in a preprocessing step) to then mine patterns representing trends from the graph [23, 30, 39]. In a trend graph, each vertex is described by trends or attribute variations. For example, Fig. 4 shows a trend graph, where a trend is an increase $(+)$ or decrease $(-)$ of an attribute value during a time interval of consecutive timestamps. In the following paragraphs, the discussion assumes that dynamic attributed graphs are transformed into trend graphs.

Different from most studies on complex networks that focus mainly on network topology, Jin et al. [28] considered a dynamic network where each vertex has a weight, i.e. a dynamic attributed graph with a single attribute. They proposed an algorithm to discover connected subgraphs whose vertices show the same trend during a time interval of two consecutive timestamps. Such pattern can reveal important changes occurring in a dynamic system. However, an important limitation of this work is that it considers a single attribute with only two trend types, that is an increase (+) or decrease (-), which restricts its applications. A second limitation is that it enforces a strict constraint that timestamps must be consecutive for each considered time interval.

To address these two limitations, Desmier et al. [28] generalized this work by considering a novel type of patterns called *cohesive co-evolution patterns*. A cohesive co-evolution pattern is a set of vertices that are similar (as measured by a similarity measure) and display the same trends for some attributes during a time interval. But differently from the work of Jing et al., each node is described using multiple attributes, and time intervals consisting of non consecutive timestamps are also considered. An example co-evolution pattern is shown in Fig. 2 a). A co-evolution pattern may appear multiple times in a dynamic attributed graph, where each occurrence consists of time intervals formed by different pairs of timestamps. Desmier et al. also introduced additional constraints to filter uninteresting patterns by considering the graph structure.

Recently, Cheng et al. [30] generalized the concept of cohesive co-evolution patterns by proposing a new type of patterns called *recurrent patterns*. While a cohesive pattern describes how attribute values changed in a time interval for some vertices, a recurrent pattern describes how attribute values changed for a set of vertices over several time intervals. Thus, while a cohesive pattern is a subgraph, a recurrent pattern is a sequence of subgraphs, which can be each described using different trends. For instance, Fig. 2 b) shows a recurrent pattern indicating successive changes over three time intervals t_1, t_2 and t_3. Note that it is required that each recurrent pattern is frequent (occurs at least a minimum number of times), respects a continuity requirement, and other constraints such as non-redundancy, connectivity and volume size of the vertex set. Recurrent patterns allow to capture the frequent evolutions of trends for nodes in a dynamic attributed graph. However, an important drawback of this work is that the frequency is used as main measure to select patterns. As a result, recurring patterns may be graphs that represent weakly correlated. For example, a recurring pattern containing a sequence of two subgraphs may be output because the second subgraph appears very frequently rather than because of a strong correlation with the first subgraph. Hence, a significance measure is needed to find interesting sequences having a strong correlation. Such measure will be proposed in this paper.

2.2. Mining emerging patterns

Another related work is the discovery of emerging patterns in databases[40, 41, 42]. An *emerging pattern* or *contrast pattern* is a pattern that is drastically different in two datasets with respect to some interestingness measure [35]. A popular measure for discovering emerging patterns is the growth rate [43]. It is defined as the ratio of the frequency of a pattern in a database to its frequency in another database. The growth-rate was used by Kaytoue et al. to discover patterns in a dynamic attributed graph [23], called *triggering patterns*. A triggering pattern is a sequence of attribute variations, followed by a single topological change.

An example of triggering pattern is $\{a+, b+\}\{c-, d-, e-\} \rightarrow \{closeness-\}$, where a, b, c, d and e are node attributes, the symbol $+$ and $-$ indicate trends, and *closeness* is a topological attribute [31]. This pattern indicates that the trends on the left side of $\rightarrow$ will trigger the topological change "closeness-" on the right side of $\rightarrow$. The growth rate of a pattern is measured to ensure that attribute variations triggered the topological change. However, a limitation of this work is that the growth-rate is only calculated for a triggering pattern to assess the influence of the last attribute variation on the topological change. Thus, this approach may find patterns where attribute variations are weakly correlated with each other over time. The novel significance measure presented in this paper addresses this limitation by calculating a significance measure for all attribute variations.

Another limitation of the work of Kaytoue et al. is that relationships between entities (nodes) are summarized as topological properties that are then encoded as attribute values of nodes. Doing this makes it easier to mine triggering patterns. But the flip side is that some information is lost because if edges are added or deleted between nodes, it will in some cases not change to the topological attribute values. The type of patterns presented in this paper does not rely on such simplification and thus does not suffer from such information loss.

2.3. Spatio-temporal data mining

Another important related work is spatio-temporal (ST) data mining [44, 45, 46]. The relationship between ST data mining and graph mining is that (1) graphs are often used to model spatial information [2, 29], and (2) graphs are often interpreted from a spatial perspective (e.g. the shortest path between two vertices can be considered as a spatial distance)[47].

In ST data mining, a database is a set of events annotated with a time, location, and event type. A framework was proposed by Huang et al. for mining significant sequential patterns in ST event data sets [34]. In this work, a sequential pattern is a subsequence of events that frequently appears and is significant. Significance is assessed using a measure called *sequence index*, which is similar to the growth rate used in emerging pattern mining. The sequence index measures the significance of a sequence of events, or in other words how correlated events of a sequential patterns are. In that study, patterns revealing useful information were discovered such as "Low Evaporation $\rightarrow$ High Temperature $\rightarrow$ High Evaporation" [34].

Although the concept of pattern significance as defined by the sequence index is useful, it cannot be directly applied to mine patterns in a dynamic attributed graph. The main reason is that it considers that two event types cannot co-occur (cannot have the same timestamp). Thus, this approach can only find sequences of consecutive event types. But in dynamic attributed graphs, time and locations are discrete and multiple trend types often appear simultaneoulsy for different vertices and time intervals. For instance, in the running example of Fig. 3, the trend types $\{a1+, a2+\}$ appear simultaneously. Finding patterns containing simultaneous event types is desirable but greatly increases the size of the search space. For a time interval, instead of choosing an event type from M event types, 2^M combinations of event types must be considered. To address this problem, reasonable constraints and efficient pruning techniques must be designed to avoid exploring the whole search space. Another challenge for applying the sequence index in dynamic graph mining is that ST mining mainly considers the distance relationship between events as spatial information. But in a dynamic graph, vertices and edges provide much richer information as the topology of a graph may dynamically change over time. This enables users to adopt various neighborhood definitions suitable for specific applications.

After that, Mohan et al. [48] extended the topological structure of sequential patterns in ST data to discover partially ordered patterns. To mine the newly defined *cascading spatio-temporal patterns*, they introduced a novel significance measure, named *Cascade Participation Index.* Besides, to efficiently discover these patterns, several filtering strategies were adopted, including an upper bound filter. Although this work partially generalizes sequential pattern mining, the main focus is still on spatio-temporal data mining and the approach is not directly applicable to graph mining. In graph mining, additional challenges must be considered such as the multiple attribute variations, and a different concept of distance.

2.4. Other techniques for capturing changes in dynamic graphs

Table 1: The three most relevant studies and the key differences with this work

Previous work	**Limitations of previous work**	**Improvement in this paper**
Cheng et al. (2017) [29]	Subgraphs describing trends in sequence may be weakly correlated.	A novel significance measure is proposed to make sure that trend sequences are strongly correlated.
Kaytoue et al. (2014) [23]	1. Correlation is only evaluated for the last part of a pattern. 2. Topological data is encoded as attributes, which can result in information loss.	1. For a pattern, significance is evaluated by considering the whole sequence of subgraphs 2. Topological properties are not encoded as attributes, and thus no information is lost.
Huang et al. (2007) [34]	Does not consider co-occurrences of multiple event types.	Allow multiple trends to co-occur. Pruning strategies based on a novel upper-bound are proposed to avoid exploring the whole search space.

Other approaches have also been proposed to study or capture changes in dynamic graphs. Several visualization techniques have been designed to see how a graph changes using a directed graph, hierarchical graph or matrix representations [49]. Various metrics have also been proposed to characterize dynamic graphs such as flexibility, promiscuity and cohesion [50]. Graph matching approaches have also been used to track objects at different timestamps in a dynamic graph [51]. Shah et al. [52] analyzed dynamic graphs by finding substructures that summarize a graph while minimizing the encoding cost. Although this approach is efficient, it assumes that graph nodes are unlabelled. A data store has also been proposed to efficiently store, update and query dynamic graphs [53]. Algorithms have also been proposed to compute metrics describing a stream of graphs such as the number of triangles [54]. Clustering approaches have also been designed to find interesting clusters in dynamic graphs [55]. Tensor decomposition approaches have also been applied to summarize dynamic graphs [56]. Approaches have also been designed to identify outliers in dynamic attributed graphs [57]. In machine learning, various embedding methods have also been designed for learning representations of dynamic graphs [58]. But the approaches described in this paragraph do not aim to find patterns and/or are not designed to handle attributed graphs. Thus, they cannot be directly compared with the approach proposed in this paper.

The most relevant related work for this paper are the pattern mining based approaches of Cheng et al. (2017) [29], Kaytoue et al. (2014) [23], and Huang et al. (2007) [34]. For the convenience of the reader, Table 1 provides a summary of their limitations and explains how these limitations have been addressed in this paper.

By the above literature review, it was found that pattern significance is an important concept in data mining but has not been fully introduced in dynamic attributed graph mining. In some related work, the concept of significant sequential patterns was defined for spatio-temporal mining [34]. But it is not trivial to extend these concepts for pattern mining in a dynamic attributed graph. In the next section, we define a new problem to fill that important research gap and mine significant patterns in dynamic attributed graphs. Then, the following sections presents efficient algorithms for this new problem, and experimental results on real data.

Key characteristics of the proposed approach is that it is unsupervised, and that the aim is to find patterns representing trends that are interpretable by humans. The focus of this paper is not on performing tasks such as classification, prediction or community detection, but to identify insightful patterns that describe interesting significant changes in the data that can help to understand the data.

3. Preliminaries and problem definition

This section introduces definitions related to dynamic attributed graphs and then define the novel problem of mining significant trend sequences. Formally, the concept of graph, attributed graph and dynamic attributed graph are defined as follows.

Definition 1 (Graph). A graph is a tuple $G = (\mathcal{V}, E)$ where $\mathcal{V}$ is a set of vertices, and $E \subseteq \mathcal{V} \times \mathcal{V}$ is a set of edges.

Definition 2 (Attributed graph). An attributed graph is a tuple $G = (\mathcal{V}, \mathcal{A}, E, \lambda)$ where $\mathcal{V}$ is a set of vertices, $\mathcal{A}$ is a set of attributes, $E \subseteq \mathcal{V} \times \mathcal{V}$ is a set of edges, and $\lambda : \mathcal{V} \times \mathcal{A} \to \mathbb{R}$ is a function that associates a real value to each vertex-attribute pair.

Definition 3 (Dynamic attributed graph). Let there be a set of timestamps $\mathcal{T} = \{1, 2, \ldots, t_{max}\}$. A dynamic attributed graph is a sequence of attributed graphs $\mathcal{G} = \langle G_1, G_2, ..., G_{t_{max}} \rangle$ where $G_t = (\mathcal{V}_t, \mathcal{A}_t, E_t, \lambda_t)$, where $\mathcal{V}_t$ is a set of vertices, $\mathcal{A}_t$ is a set of attributes, $E_t \subseteq \mathcal{V}_t \times \mathcal{V}_t$ is a set of edges, and $\lambda_t : \mathcal{V}_t \times \mathcal{A}_t \to \mathbb{R}$ is a function that associates a real value to each vertex-attribute pair, for the timestamp t.

Example 1. Fig 3 shows a dynamic attributed graph for six timestamps $\mathcal{T} = \{1, 2, 3, 4, 5, 6\}$, where the attributed graphs $G_1, G_2 \ldots G_6$ have a set of vertices $\mathcal{V}_1 = \mathcal{V}_2 = \mathcal{V}_3 = \mathcal{V}_4 = \mathcal{V}_5 = \mathcal{V}_6 = \{1, 2, 3, 4, 5\}$ and a set of attributes $\mathcal{A}_1 = \mathcal{A}_2 = \mathcal{A}_3 = \mathcal{A}_4 = \mathcal{A}_5 = \mathcal{A}_6 = \{a1, a2, a3\}$. For timestamp 1, the attributed graph $G_1 = (\mathcal{V}_1, \mathcal{A}_1, E_1, \lambda_1)$ is defined as $E_1 = \{(1, 2),\ (1, 3),\ (4, 5)\}$ and $\lambda_1 = \{(1, a1) \to 3, (1, a2) \to 4, (1, a3) \to (1), ..., (5, a1) \to 0, (5, a2) \to 5, (5, a3) \to 2\}$.

Dynamic attributed graphs can be used to model various types of data from many domains. To discover interesting patterns in a dynamic attributed graph that represents trends, it is possible to apply a preprocessing step that converts numerical values into trends [23]. This conversion allows to extract more general patterns that are not influenced by the noise found in raw numerical data. Thus, in this paper, a dynamic attributed graph is first preprocessed to obtain trends for the attributes of each vertex. This is done as follows. If an attribute value a_x for a vertex increased significantly between two consecutive timestamps t_i and t_{i+1}, then a_x is replaced by a_x+. Similarly, if an attribute value a_x for a vertex decreased significantly between two consecutive timestamps t_i and t_{i+1}, then a_x is replaced by a_x-. A significant increase or decrease can be simply defined as an increase or decrease by a value that is greater than 0 [31]. However, this definition of significant variation may not be suitable for all applications. Thus, other definitions can be used such as using a percentage of the standard deviation. Another alternative is to define a scale with different types of increase/decrease such as a small increase a_x+, a large increase $a_x{++}$ or a very large increase $a_x{+++}$. For the sake of simplicity, in the rest of this paper we only consider increase or decrease by values greater than zero. But different definitions can be used for specific applications.

To illustrate the preprocessing step, consider the dynamic attributed graph of Fig. 3. From timestamp t_1 to t_2, the values of attributes $a1$, $a2$ and $a3$ of vertex 1 increases. Thus, these values are replaced by $\{a1+, a2+, a3+\}$. The set of graphs obtained by preprocessing the dynamic graph of Fig. 3 is shown in Fig. 4.

Note that in the following, it will be assumed that graphs of a dynamic attributed graph have been transformed into trend graphs by preprocessing. When explicitly refering to an original dynamic attributed graph such as that of Fig. 3, the term *timestamp* will refer to a specific time point, while for a transformed graph with trends such as Fig. 4, a timestamp t_i will refer to the i-th time interval.

To be able to refer to a vertex at a given timestamp, a concept of *point* is introduced. Furthermore, a concept of neighborhood around points is defined to consider their proximity to other points. The concept of proximity between two points is defined from a spatio-temporal perspective, that is by considering their distance in a graph and their occurrence time.

Definition 4 (Point, Neighboring space of a point). A point $p = (t, v)$ in a dynamic attributed graph is a vertex v at a timestamp t. Let Ws be the set of all points for all timestamps in the dynamic attributed graph. The neighboring space $Ns(p)$ of a point $p = (t, v)$ in a dynamic attributed graph is defined as a set of points that are neighbors of p, that is:

$$Ns(p) = \{\ p' \mid \mathit{neighborhood}(p', p) = \mathit{true} \wedge p' \in Ws\}$$

where *neighborhood* is a boolean function indicating if two points p' and p are neighbors.

The above definition depends on that of the *neighborhood* predicate, which can be parameterized for various applications. Since there may be rich relationships between entities in graphs, a variety of case-specific neighborhood relationships can be used. A simple setting is to consider $\mathit{neighborhood}(p', p) = \mathit{true}$ for two points p and p' if only if p' appears at the timestamp following p, and the vertex of p' is connected to that of p at that timestamp. This is the definition that will be used in the rest of this paper.

Example 2. Consider the dynamic attributed graph of Fig. 4 and the above neighborhood definition. The set of points at timestamp t_1 are $(t_1, 1)$, $(t_1, 2)$, $(t_1, 3)$, $(t_1, 4)$, and $(t_1, 5)$. Consider the point $(t_1, 1)$. This point is connected by edges to points $(t_1, 2)$ and $(t_1, 3)$ at t_1. These two points then appear as $(t_2, 2)$ and $(t_2, 3)$ at the following timestamp. Thus, the neighboring space of $(t_1, 1)$ is $\{(t_2, 2), (t_2, 3)\}$. Consider the point $(t_2, 4)$. This point is connected by an edge to point $(t_2, 1)$ at t_2. This point then appears as $(t_3, 1)$ at the following timestamp. Thus, the neighboring space of $(t_2, 4)$ is $\{(t_3, 1)\}$. Similarly, it can be found that the neighborhood space of point $(t_4, 1)$ is $Ns\left((t_4, 1)\right) = \{(t_5, 2), (t_5, 4)\}$. The above example is illustrated in Fig. 5. The points $(t_1, 1)$, $(t_2, 4)$ and $(t_4, 1)$ are represented as circles with a thick blue line, while points in their neighboring spaces are illustrated as dark gray filled circles. Dark edges indicates direct connections from $(t_1, 1)$, $(t_2, 4)$ and $(t_4, 1)$ to directly connected nodes, while dashed edges indicate a relationship from these latter nodes to the same nodes at the following timestamp.

The concept of neighboring space is also defined for a set of points.

Definition 5 (Neighboring space of a set of points). The neighboring space of a set of points ps is defined as the union of the neighboring spaces of each point in ps.

$$\begin{aligned} Ns(ps) &= \{\ p' \mid \exists p \in ps, \mathit{neighborhood}(p', p) = \mathit{true} \wedge p' \in Ws\} \\ &= \bigcup_{p \in ps} Ns(p) \end{aligned}$$

Example 3. Consider the points $(t_1, 1)$, $(t_2, 4)$ and $(t_4, 1)$. The neighboring space of these points is $Ns(\{(t_1, 1), (t_2, 4), (t_4, 1)\}) = \{(t_2, 2), (t_2, 3)\} \cup \{(t_3, 1)\} \cup \{(t_5, 2), (t_5, 4)\} = \{(t_2, 2), (t_2, 3), (t_3, 1), (t_5, 2), (t_5, 4)\}$.

To study attribute variations in a dynamic attributed graph, the concept of trend set is defined.

Definition 6 (Trend set). An attribute variation during a time interval (e.g. an increase or decrease) is called a *trend*. A set of trends is called a *trend set*. Let $tsp(p)$ denotes the trend set of a point p.

Example 4. Consider the timestamp t_1 of the dynamic attributed graph of Fig. 4. The trend set of point $(t_1, 1)$ is $tsp((t_1, 1)) = \{a1+, a2+, a3+\}$.

Definition 7 (Supporting points of a trend set). The *supporting points* of a trend set ts are the points that have a trend set that is a superset of ts. Formally, it is the set $SP(ts) = \{p | ts \subseteq tsp(p)\}$.

Example 5. The set $\{a1+, a3+\}$ is a trend set, supported by the vertices 1, 3 and 4 at timestamp t_1, that is $SP(\{a1+, a3+\}) = \{(t_1, 1), (t_1, 3), (t_1, 4)\}$. The trend set $\{a1+, a2+\}$ is supported by the points $SP(\{a1+, a2+\}) = \{(t_1, 1), (t_2, 4), (t_4, 1)\}$, which are represented using circles with thick blue lines in Fig. 5.

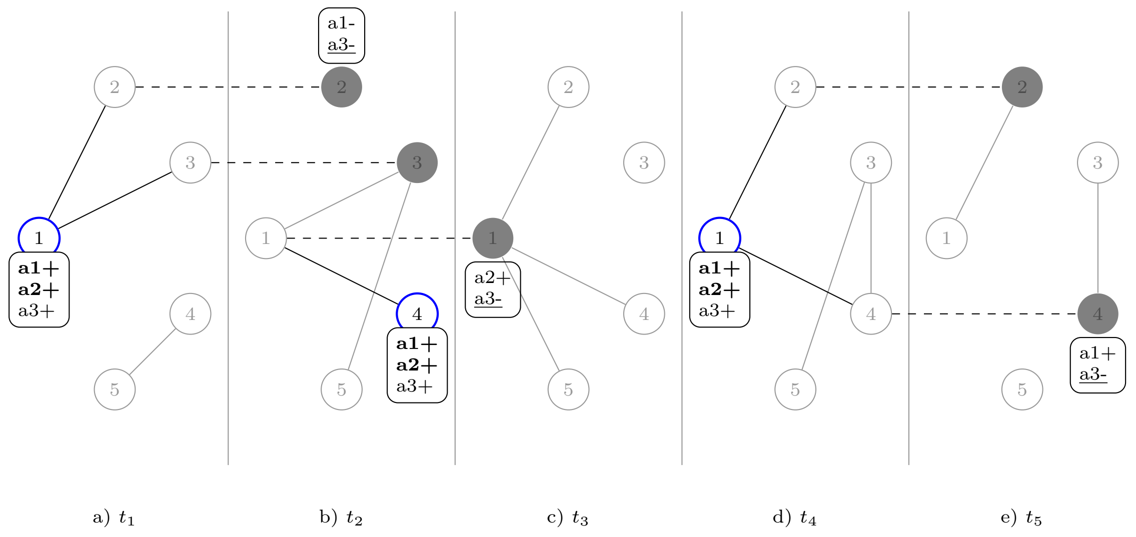

Figure 5: The supporting points of the trend set $\{a1+, a2+\}$ (represented as nodes with thick blue lines) and the corresponding neighboring space (dark gray filled nodes). The tail supporting points of the sequence $\langle\{a1+, a2+\}, \{a3-\}\rangle$ are the vertices that contain $\{a3-\}$ and are in the neighboring space of $\langle\{a1+, a2+\}$ (the dark gray filled nodes).

Analyzing the points that support a given trend set in a time interval can be interesting, especially if it is supported by many points. However, looking at each time interval individually does not allow to analyze more complex relationships between these trends over several time intervals. To find more interesting patterns, this paper does not only look at a single time interval but also at how trend sets are related to each other over succesive time intervals. The goal of this paper is to discover sequences of trend sets showing the evolution of vertice attributes over succesive time intervals. To capture this information, a novel type of patterns is defined, called *significant k-trend sequence*. The following paragraphs define the concept of k-trend sequence.

Definition 8 (k-trend-sequence TsS). A k-trend-sequence tss is an ordered list of k trend sets that have appeared at consecutive time intervals in a dynamic attributed graph. Let the notation $tss[i]$ denotes the i-th trend set of tss. Moreover, let $tss[i:j]$ denotes the subsequence consisting of the i-th trendset to the $(j-1)$-th trendset.

Example 6. The sequence $\langle\{a1+, a2+\}\rangle$ is a 1-trend-sequence containing a single trend set. The sequence $tss = \langle\{a1+, a2+\}, \{a3-\}, \{a2+, a3+\}\rangle$ is a 3-trend-sequence. The second trend set of that sequence is $tss[2] = \{a3-\}$. Moreover, $tss[2:4] = \langle\{a3-\}, \{a2+, a3+\}\rangle$.

The next paragraph explains how a k-trend sequence appears in a dynamic attributed graph. Briefly, a k-trend-sequence containing k trend sets is said to appear in a dynamic attributed graph if $tss[1]$ has some supporting points, and the supporting points of $tss[i]$ are in the neighboring space of the supporting points of $tss[i-1]$ for all $1 < i \leq k$. Thus, a k-trend sequence appears in a dynamic attribute graph, if its trend sets appear in points of consecutive time intervals that are spatio-temporally related. A formal definition is given based on a concept of tail supporting points:

Definition 9 (Tail supporting points of a k-trend-sequence). Let there be a k-trend sequence tss. If $k = 1$, the *tail supporting points* (TSP) of tss are the supporting point of $tss[1]$. If $k > 1$, the tail supporting points of tss are the points that support $tss[k]$ in the neighboring space of the tail supporting points of the sequence $tss[1:k-1]$.

$$TSP(tss) = \begin{cases} SP(tss[1]) & k = 1 \\ \{p \mid p \in Ns(TSP(tss[1:k-1])) \text{ and } tss[k] \subseteq ts(p)\} & k > 2 \end{cases}$$

If a k-trend sequence tss has a non empty set of tail supporting points ($TSP(tss) \neq \emptyset$) in a dynamic attributed graph, then tss is said to *appear* in that dynamic attributed graph.

Example 7. Consider the 2-trend-sequence $tss = \langle\{a1+, a2+\}, \{a3-\}\rangle$. Then, $TSP(tss[1:2]) = SP(tss[1])$ $= \{(t_1, 1), (t_2, 4), (t_4, 1)\}$ and $Ns(TSP(tss[1:2])) = Ns((t_1, 1)) \cup Ns((t_2, 4)) \cup Ns((t_4, 1)) = \{(t_2, 2), (t_2, 3)\} \cup$ $\{(t_3, 1)\} \cup \{(t_5, 2), (t_5, 4)\}$. Then, by intersecting $SP(\{a3-\})$ with $Ns(TSP(tss[1:2]))$, it is found that $TSP(tss) = \{(t_2, 2), (t_3, 1), (t_5, 2)\}$. Furthermore, $Ns(TSP(tss)) = Ns((t_2, 2)) \cup Ns((t_3, 1)) \cup Ns((t_5, 2)) =$ $\{\} \cup \{(t_4, 2), (t_4, 4), (t_4, 5)\} \cup \{\}$.

The above definition have explained how trend sequences appear in a dynamic attributed graph. Some important observations can be made. A trend set can be regarded as a 1-trend sequence, where the supporting points of the trend set are also its tail supporting points. Then, based on these supporting points and the neighborhood definition, a neighboring space can be calculated for that 1-trend sequence. In that neighboring space, a trend set can be found to extend the 1-trend sequence. This allows to define a 2-trend sequence, where the tail supporting points of that 2-trend sequences are the set of points that have the second trend set as subset in the neighboring space of the 1-trend sequence. Then, based on the tail supporting points of the 2-trend sequence, another neighboring space can be calculated and 3-trend sequences can be found, and so on for larger trend sequences.

In a dynamic attributed graph, many trend sequences may appear. Some trend sequences may be more relevant than others to the user. To find useful trend sequences, it is thus necessary to apply constraints to select interesting trend sequences and filter spurious ones. The next paragraphs introduces these concepts. In particular, they present a spatial correlation measure to find significant trend sequences.

Based on the concept of neighboring space, the novel *Virtual Growth Rate* significance measure is introduced. It assesses the significance of a trend set by dividing its number of occurrences (supporting points) in a neighboring space by its number of occurrence in the whole space (the whole dynamic attributed graph). The assumption is that if this ratio is small, it is highly significant because the trend set appears much more often locally than globally.

Definition 10 (Virtual Growth Rate (VGR)). The support of a trend set ts in a neighboring space Ns is defined as:

$$supp(ts, Ns) = \frac{|\{p \mid p \in Ns \;\; and \;\; ts \subseteq ts(p)\}|}{|Ns|}$$

The virtual growth rate of ts in Ns is defined as:

$$VGR(ts, Ns) = \frac{supp(ts, Ns)}{supp(ts, Ws)}$$

where Ws is the whole space, i.e. all points in $\mathcal{G}$. In the following, $supp(ts, Ws)$ is denoted as $supp(ts)$.

Example 8. Let $tss = \langle\{a1+, a2+\}, \{a3-\}\rangle$, $ts = \{a3-\}$, and $Ns = Ns(TSP(tss[1:2]))$. Then, $supp(ts) = \frac{|\{(t_2,2),(t_3,1),(t_4,3),(t_5,1),(t_5,4)\}|}{5\times5} = \frac{1}{5}$, $supp(ts, Ns) = \frac{\{(t_2,2),(t_3,1),(t_4,3),(t_5,1),(t_5,4)\} \cap Ns}{|Ns|}$ $= \frac{|(t_2,2),(t_3,1),(t_5,4)|}{|Ns|} = \frac{3}{5}$ and $VGR(ts, Ns) = \frac{\frac{3}{5}}{\frac{1}{5}} = 3$. Thus, the trend set $\{a3-\}$ is three times more likely to occur in the neighboring space of $\{a1+, a2+\}$ than in the whole space.

The above measure can be viewed as an adaptation of the growth-rate measure used in traditional emerging pattern mining. A key difference is that instead of comparing two datasets, the proposed VGR measure compares the density in a local space with that of the whole space. Moreover, the VGR measure calculates density by considering the spatial and temporal relationships between points. However, the VGR is only defined to evaluate the significance of a trend set. The next paragraph adapts the VGR measure to obtain a more general significance measure that can be applied for a k-trend sequence.

Definition 11 (Sequence Virtual Growth Rate (SVGR)). The Sequence Virtual Growth Rate of a k-trend-sequence ($k \geq 2$) tss is defined as:

$$SVGR(tss) = \min_{i \in [2,k]} VGR(tss[i], Ns(TSP(tss[1:i]))) \quad k \geq 2$$

An equivalent definition is:

$$SVGR(tss) = \begin{cases} \min\{SVGR(tss[1:k]), VGR(tss[k], Ns(TSP(tss[1:k])))\}, & if \quad k \geq 3 \\ VGR(tss[2], Ns(SP(tss[1]))), & if \quad k = 2 \end{cases}$$

In the field of pattern mining, anti-monotone measures are often used to reduce the search space and design efficient algorithms [59]. The following property shows that the SVGR is partially monotone.

Property 1 (Partial Anti-monotone). For any k-trend-sequence tss such that $k \geq 2$, it follows that $SVGR(tss[1:i]) \geq SVGR(tss)$ for $i \in [3, k+1]$.

Proof 1. *a*) For $k = 2$, it is trivial.
b) For $k \geq 3$, $SVGR(tss[1:k+1]) = min\{SVGR(tss[1:k]), VGR(tss[k], Ns(TSP(tss[1:k])))\} \leq SVGR(tss[1:k]) \leq SVGR(tss[1:k-1]) \leq \ldots \leq SVGR(tss[1:i])$.

It is important to note that the anti-monotonicity property of the SVGR does not hold for the general case where $SVGR(tss[i:j]) \geq SVGR(tss)\ \forall\ [i,j] \subset [1:k+1]$ and $i \neq j$. This can be seen in the following example.

Example 9. Let $tss = \langle\{a1+, a2+\}, \{a3-\}, \{a2+, a3+\}\rangle$ and $Ns = Ns(TSP(tss[1:3])) = \{(t_4, 2), (t_4, 4), (t_4, 5)\}$ as in previous examples. Then, $SVGR(tss[1:3]) = 3$. Furthermore, $SP(\{a2+, a3+\}) = \{(t_1, 1), (t_2, 1), (t_2, 4), (t_4, 1), (t_4, 5)\}$, $supp(\{a2+, a3+\}, Ws) = \frac{|SP(\{a2+,a3+\})|}{|Ws|} = \frac{5}{25} = \frac{1}{5}$, $supp(\{a2+, a3+\}, Ns) = \frac{|SP(\{a2+,a3\}) \cap Ns|}{|Ns|} = \frac{|\{t_4,5\}|}{3} = \frac{1}{3}$ and $VGR(\{a2+, a3+\}, Ns) = \frac{supp(\{a2+,a3+\},Ns(tss1:3))}{supp(\{a2+,a3+\},Ws)} = \frac{\frac{1}{3}}{\frac{1}{5}} = \frac{5}{3}$. Then, $SVGR(tss) = \min\{3, \frac{5}{3}\} = \frac{5}{3}$. Consider sequence $tss' = tss[2:4] = \langle\{a3-\}, \{a2+, a3+\}\rangle$ and $Ns' = Ns(SP(\{a3-\})) = \{(t_4, 2), (t_4, 4), (t_4, 5), (t_5, 4), (t_5, 5)\}$. Then, $supp(\{a2+, a3+\}, Ns') = \frac{SP(\{a2+,a3+\}) \cap Ns'}{|Ns'|} = \frac{|(t_4,5)|}{5} = \frac{1}{5}$, $GVGR(tss') = \frac{\frac{1}{5}}{\frac{5}{25}} = 1$. Here, $SVGR(tss[1:3]) > SVGR(ts[1:4])$ while $SVPR(tss[2:4]) < SVGR(tss[1:4])$.

To select interesting trend sequences for a user, it is also necessary to filter noise. For this purpose, frequency constraints are applied. They are used to eliminate patterns that have very low occurrence frequencies, and thus may appear by chance. The first frequency constraint is used by the proposed algorithms to generate all trend sets to be considered in the whole space. The second constraint is designed to avoid considering trend sets that are infrequent in neighboring spaces. Such constraints are simple but play an important role to eliminate spurious patterns and reduce the search space.

Definition 12 (Frequent k-trend-sequence). A k-trend-sequence tss such that $k \geq 2$ is said to be *frequent* if and only if

$$|SP(tss[1])| \geq minInitSup$$

and

$$|TSP(tss[1:i])| \geq minTailSup, \quad \forall i \in [3, k+1]$$

where $minInitSup$ and $minTailSup$ are user specified thresholds.

Property 2. A subsequence $tss[1:i] (i \in [3, k+1])$ of a frequent k-trend-sequence tss ($k \geq 2$) is also frequent.

Proof 2. The sequence $tss[1:i]$ has the same initial support as tss. If $tss[1]$ has an initial support that is larger than $minInitSup$, then it will be the same for $tss[1:i]$. For $tss[1:i]$ $(3 \leq i \leq k)$, if its tail support is less than $minTailSup$, tss cannot be frequent.

Example 10. Consider $tss = \langle\{a1+, a2+\}, \{a3-\}, \{a2+, a3+\}\rangle$, $tss' = \langle\{a3-\}, \{a2+, a3+\}\rangle$, $minInitSup = 3$ and $minTailSup = 1$. Since $|SP(tss[1])| = 5 \geq 3$, $|TSP(tss[1:3])| = 3 \geq 1$, $|TSP(tss[1:4]) \geq 1$, it is found that tss is frequent. Because $|SP(tss'[1])| = 5 \geq 3$, $|TSP(tss'[1:3])| = 1 \geq 1$, tss' is also frequent.

By combining the frequency and significance constraints defined by the support and SVGR measures, this paper aims to identity trend sequences that are spatio-temporally correlated and do not appear by chance. The k-trend sequences meeting these constraints are called significant k-trend-sequences.

Definition 13 (Significant k-trend-sequence). A k-trend-sequence is said to be significant if and only if tss is frequent and $SVGR(tss) \geq minSig$, where $minSig$ is a parameter specified by the user.

Example 11. Consider $tss = \langle\{a1+, a2+\}, \{a3-\}, \{a2+, a3+\}\rangle$, $tss' = \langle\{a3-\}, \{a2+, a3+\}\rangle$ and $minSig = 3$. While tss and tss' are frequent (see Example 10), $SVGR(tss) = \frac{5}{3} < 3$ and $SVGR(tss') = 1 < 3$ (see Example 10), and thus both are not significant. However, according to Property 2 and $SVGR(tss[1:3]) = 3$ (see Example 10), $tss[1:3]$ is a significant trend sequence.

Definition 14 (Problem of discovering all significant k-trend-sequences). Let there be a dynamic attributed graph, a significance threshold $minSig$, and two support thresholds $minInitSup$ and $minTailSup$. The problem of discovering all significant k-trend sequences is to find each significant trend sequence $SigTSeq$ such that $SVGR(SigTSeq) \geq minSig$, $|SP(SigTSeq[1])| \geq minSup$ and $|TSP(SigTSeq[1:i])| \geq minTailSup$ for $\forall i \in [3, k+1]$ (where k is the length of $SigTSeq$).

Example 12. Consider the dynamic attributed graph shown in Fig. 4, $minSig = 3$, $minInitSup = 3$ and $minTailSup = 1$. The set of all significant k-trend sequences is shown in Table 2. The pattern $\langle\{a1+, a2+\}, \{a3-\}\rangle$ is a significant k-trend sequence. Detailed calculations for that pattern have been shown in Examples 3, 5, 7, 8, 9, 10 and 11. Moreover, occurrences of that pattern have been highlighted in Fig. 4. Consider another pattern $\langle\{a1-\}, \{a3+\}\rangle$. It is found that $|SP(\{a1-\})| = |\{(t_1, 2), (t_2, 1), (t_2, 2), (t_3, 2), (t_4, 3)\}| = 5 > minInitSup$. Moreover, $Ns(\{a1-\}) = \{(t_2, 1), (t_3, 3), (t_3, 4), (t_4, 1), (t_5, 4), (t_5, 5)\}$ and $|TSP(\{a1-, a3+\})| = |\{(t_2, 1), (t_3, 3), (t_3, 4), (t_4, 3)\}| = 4 > minTailSup$. $|SP(\{a3+\})| = 19$. Thus, $\frac{\frac{|TSP(\{a1-, a3+\}}{|SP(\{a1-\})|}}{\frac{|SP(\{a3+\})|}{|Ws|}} = \frac{\frac{4}{6}}{\frac{19}{25}} < minSig$, and this pattern is not significant.

As it will be shown in the experiments, this proposed type of patterns is useful to find sequences of attribute changes (trends) that frequently appear in a dynamic attributed graph and provides insights about how a dynamic graph evolves. Differently from previous work that have focused on finding frequent patterns [30], this paper applies a significance measure to eliminate patterns that are not strongly correlated. Unlike the work of Kaytoue et al. [31], the proposed significance measure is applied to evaluate the correlation between all consecutive elements of a pattern rather than only the last ones. This ensure that all elements of each pattern are correlated.

From the perspective of pattern structure, the type of patterns proposed in this paper has some important difference with that of recurrent patterns [30]. A recurrent pattern, as shown in Fig. 2 b), is a sequence of trend changes that are associated to some vertices. Each occurrence of a recurrent pattern must always be associated to the same vertices. In this paper, that constraint is removed. The supporting points of each trend do not need to always be associated to the same vertices, for each occurrence of the pattern. The advantage of removing that constraint is that patterns applicable to many different vertices can be discovered. This is illustrated with an example. Consider an academic social graph where nodes are researchers, edges indicate collaborations, and attributes are the number of papers in various top conferences. Each recurrent pattern in such graph would always be associated to the same researchers (nodes), and thus may not be representative of the behavior of other researchers. On the other hand, this paper considers that a pattern may be associated to multiple sets of researchers, and thus provide information about trends followed by many researchers. The following section presents the proposed algorithms to efficiently extract k-trend sequences.

Table 2: Extracted patterns in Fig. 4, with $minSig = 3$, $minInitSup = 3$, $minTailSup = 1$

pattern	**initial support** ts	**tail support**	**significance**
$\langle\{a1+, a2+\}, \{a3-\}\rangle$	3	$\{3\}$	3
$\langle\{a1+, a2+, a3+\}, \{a3-\}\rangle$	3	$\{3\}$	3

4. Two Efficient Algorithms

This section presents the proposed algorithms for efficiently mining significant k-trend sequences in a dynamic attributed graph. Subsection 4.1 first give an overview of how the search space is explored. Subsection 4.2 presents effective search space pruning strategies. Subsection 4.3 describes data structures used for exploring the search space efficiently. Finally, subsection 4.4 and 4.5 describe the two proposed algorithms, named TSeqMiner$_{dfs-dfs}$ and TSeqMiner$_{dfs-bfs}$, respectively.

4.1. The Search Space

A k-trend sequence is a sequence of trend sets. To find all significant k-trend sequences in a dynamic attributed graph, it is necessary to consider different trend sets and how they can be combined one after the other to form trend sequences. Then, it is necessary to evaluate these trend sequences to find the significant ones. But a key challenge is that the number of trend sets can be very large. Generally, if there are n different trends, $2^n - 1$ trend sets can be created (excluding the empty set). To avoid considering all possible trend sets, the proposed algorithms first use the *minInitSup* threshold to find all frequent trend sets. Then, only these frequent trend sets are combined to form k-trend sequences. By ignoring all infrequent trend sets from further processing, the number of trend sequences that are considered to find the significant k-trend sequences can be greatly reduced.

Initially, the proposed algorithms scan the dynamic attributed graph to build a vertical database for each trend set of size 1 (containing a single trend). For the running example, the result is the size 1 trend sets shown in the top part of Table 3. The vertical database stores the supporting points $SP(ts)$ of each trend set ts. Then, a vertical frequent itemset mining algorithm is applied to identify all the frequent trend sets containing more than one trend. For the running example, the result is the trend sets of size two and three, which are also depicted in Table 3. In the proposed algorithm's implementation, frequent trend sets are mined using the Eclat algorithm as it is simple and efficient [60].

After all frequent trend sets have been found, the search space of k-trend sequences is explored by combining trend sets to form trend sequences. A part of this search space is illustrated in Fig. 6. The search space can be viewed as containing two parts called *inner-levels* and *outer-levels*. The i-th outer-level contains all i-trend sequences, where some of them share a same (i-1)-trend sequences as prefix. The inner-level contains all frequent trend sets (organized in a certain way, which will be explained in Section 4.3).

For example, the first outer level is depicted in the top of Fig. 6, for the running example. It consists of a single inner-level, represented by the topmost dashed box. This box contains all frequent trend sets (1-trend sequences), represented as a tree. The second outer-level consists of all 2-trend sequences. In the illustration, part of this level is represented by the three bottommost boxes. Each box represents an inner-level of the second outer-level. For example, the leftmost box of Fig. 6 shows trend sets that can be appended to the prefix 1-trend sequence $\langle\{a1+\}\rangle$ to form 2-trend sequences. Similarly the bottommost box of Fig. 6 shows trend sets that can be appended to the prefix 1-trend sequence $\langle\{a1+, a2+, a3+\}\rangle$ to form 2-trend sequences. In the illustration, each dashed arrows represent extension(s) of a k-sequence to form $(k + 1)$-trend sequence(s) having a same prefix.

To extract all desired significant trend sequences, the algorithms explore this search space. The output of the algorithms is a tree similar to Fig. 6, storing all significant trend sequences.

It can be observed that if the longest trend sequences appearing in a dynamic attributed graph contain k trend sets, the algorithms may explore up to k outer-levels. The maximal number of outer-levels in the search space depends on the number of time intervals in the dynamic attributed graph and on how trend

Table 3: All frequent trend sets with their supporting points and support values

size	**trend set** ts	$SP(ts)$	**supp**
1	{a1+}	(t_1,1) (t_1,3) (t_1,4) (t_2,4) (t_3,4) (t_3,5) (t_4,1) (t_4,2) (t_4,4) (t_5,3) (t_5,4)	0.44
	{a1-}	(t_1,2) (t_2,1) (t_2,2) (t_3,2) (t_4,3)	0.2
	{a2+}	(t_1,1) (t_2,1) (t_2,4) (t_3,1) (t_4,1) (t_4,5) (t_5,1) (t_5,5)	0.32
	{a2-}	(t_1,4) (t_1,5) (t_2,3) (t_3,3) (t_4,4) (t_5,2)	0.24
	{a3+}	(t_1,1) (t_1,2) (t_1,3) (t_1,4) (t_1,5) (t_2,1) (t_2,3) (t_2,4) (t_2,5) (t_3,2) (t_3,3) (t_3,4) (t_3,5) (t_4,1) (t_4,2) (t_4,4) (t_4,5) (t_5,2) (t_5,3)	0.76
	{a3-}	(t_2,2) (t_3,1) (t_4,3) (t_5,1) (t_5,4)	0.2
2	{a1+,a2+}	(t_1,1) (t_2,4) (t_4,1)	0.12
	{a2+,a3+}	(t_1,1) (t_2,1) (t_2,4) (t_4,1) (t_4,5)	0.2
	{a1+,a3+}	(t_1,1) (t_1,3) (t_1,4) (t_2,4) (t_3,4) (t_3,5) (t_4,1) (t_4,2) (t_4,4) (t_5,3)	0.4
	{a1-,a3+}	(t_1,2) (t_2,1) (t_3,2) (t_4,3)	0.16
	{a2-,a3+}	(t_1,4) (t_1,5) (t_2,3) (t_3,3) (t_4,4) (t_5,2)	0.24
3	{a1+,a2+,a3+}	(t_1,1) (t_2,4) (t_4,1)	0.12

Table 4: Lower bound values on the support obtained using different strategies

level	trend set ts	support	lbs for DFS		lbs for BFS	
			large-grain	medium-grain	level-wise	pair-wise
1	{a1+}	0.44	0.12	0.12	0.12	
	{a2+}	0.32	0.12	0.2	0.12	
	{a3+}	0.76	0.12	0.16	0.16	
	{a1-}	0.2	0.16	0.2	0.16	
	{a2-}	0.24	0.24	0.24	0.24	
	{a3-}	0.2	0.2	0.2	0.2	
2	{a1+,a2+}	0.12	0.12	0.12	0.12	
	{a1+,a3+}	0.4	0.12	0.4	0.12	
	{a2+,a3+}	0.2	0.12	0.2	0.12	
	{a3+,a1-}	0.16	0.16	0.16	0.16	
	{a3+,a2-}	0.24	0.24	0.24	0.24	
3	{a1+,a2+,a3+}	0.12	0.12	0.12	0.12	

sets are associated to nodes. Suppose that there are M frequent trend sets and T_{max} timestamps in the original dynamic attributed graph. Then, at most $D = T_{max} - 1$ outer-levels will be explored. At the k-th outer-level, at most M^k trend sets have to be tested.

4.2. Pruning strategies

The whole search space can be regarded as containing roughly M^D patterns in the worst case. To efficiently mine significant trend sequences, it is thus important to apply pruning strategies to avoid considering all possible trend sequences, while guaranteeing that all the desired trend sequences will be found. The designed algorithms take as input three thresholds, that are *minInitSup*, *minTailSup*, and *minSig*. The proposed algorithms first use the *minInitSup* threshold to find the frequent trend sets, which are then combined to form trend sequences. Because infrequent sequences are then ignored from further processing, the number of trend sequences to be considered is reduced. Thereafter, the number of frequent trend sets M determines the size of the whole search space. The proposed algorithms then explore this search space while using the *minTailSup* and *minSig* thresholds to further reduce the search space when exploring inner and outer-levels, as it will be explained. The proposed pruning strategies are either applied at an outer level or inner level.

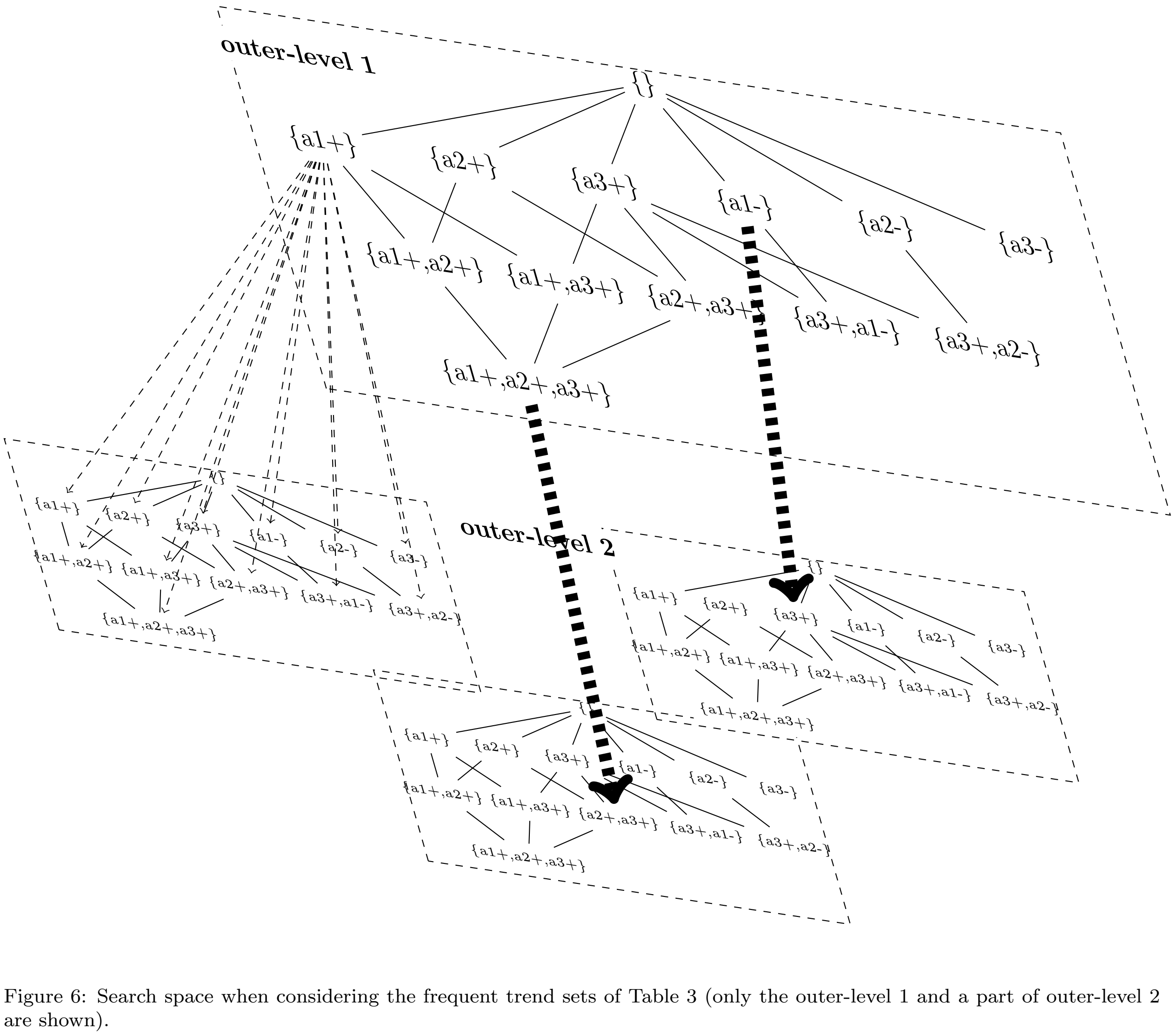

Figure 6: Search space when considering the frequent trend sets of Table 3 (only the outer-level 1 and a part of outer-level 2 are shown).

4.2.1. Outer level pruning

To prune trend sequences at the outer level, Property 1 (anti-monotonicity of $SVGR$) can be used, which states that if a k-trend-sequence is insignificant, all the (k+1)-trend-sequences that extend that trend-sequence are also insignificant. Moreover, Property 2 can also be applied, which states that if a k-trend-sequence is infrequent, all its extensions are also infrequent. As it will be shown in the experiments, applying these two properties can greatly reduce the search space.

4.2.2. Inner level pruning

To reduce the search space of trend sequences at an inner level, relationships between frequent trend sets can be checked. The anti-monotonicity property of the support measure can also be used to reduce the search space when considering *minTailSup*. Because this kind of pruning is simple and widely used in pattern mining, details will not be presented. However, no relationship between trend sets based on the VGR measure (Definition 6) can be directly used to prune the search space. This is shown as follows.

Consider a trend sequence tss and two trend sets R and T such that $T \subseteq R$. Now, consider the two trend sequences obtained by appending the trend sets T and R to tss, respectively. Those trend sequence are denoted as $tss \to T$ and $tss \to R$, respectively. Let $VGR(tss \to T) = VGR(T, Ns(TSP(tss)))$. Then,

$$VGR(tss \to T) = \frac{supp(T, Ns(TSP(tss)))}{supp(T, Ws)}$$

$$VGR(tss \to R) = \frac{supp(R, Ns(TSP(tss)))}{supp(R, Ws)}$$

Because $supp(R, Ns(TSP(tss))) \leq supp(T, Ns(TSP(tss)))$ and $supp(R, Ws) \leq supp(T, Ws)$ always hold, $VGR(tss \to T)$ can be larger than, equal to, or smaller than $VGR(tss \to R)$. Thus, the VGR measure cannot be directly use to reduce the search space.

To be able to reduce the search space indirectly using the VGR measure, the next paragraphs introduce a property that allows to avoid testing $VGR(tss \to R) < minSig$ for the case where $VGR(tss \to T) < minSig$ holds. It is based on a novel upper bound on VGR called UVGR (Property 5). But before presenting the UVGR upper bound and this property, it is necessary to first define a novel lower bound on the support measure named lbs, which is used to define the UVGR upper bound.

Definition 15 (The *lbs* lower bound on the support). Let there be a set I_s of trend sets for which the support has been already measured. For any trend set $T \in I_s$, a lower bound on the support of T, named lbs is defined as:

$$lbs(T) = \begin{cases} \min\limits_{T \subset R \in I_s} lbs(R) & \text{if } \exists R \in I_s \text{ such that } T \subset R \\ supp(T) & \text{else} \end{cases}$$

The lbs lower bound has two important properties.

Property 3. The lbs measure is monotonous. In other words, for two trend sets T_1, $T_2 \in I_s$ such that $T_1 \subseteq T_2$, the relationship $lbs(T_1) \leq lbs(T_2)$ holds.

Proof 3. There are two cases: a) if $T_1 = T_2$, then by definition $lbs(T_1) = lbs(T_2)$. b) if $T_1 \subset T_2$, then $lbs(T_1) = \min\limits_{T_1 \subset R \in I_s} lbs(R) = \min\{lbs(T_2), \min\limits_{T_1 \subset R \in I_s, R \neq T_2} lbs(R)\} \leq lbs(T_2)$. Thus, the property holds.

Property 4. The lbs measure is a lower bound on the support measure. In other words, for any trend set $T \in I_s$, $lbs(T) \leq supp(T)$.

Proof 4. There are two cases: a) if $\nexists R \in I_s$ such that $R \supset T$, then $lbs(T) = supp(T)$ according to Definition 15. b) otherwise, there exists a set $R \in I_s$ such that $R \supset T$ and $\nexists Q \in I_s, Q \supset R$. According to Property 3, $lbs(T) \leq lbs(R)$. Moreover, because R has no proper superset in I_s, $lbs(R) = supp(R)$ according to Definition 15. Thus, $lbs(T) \leq lbs(R) = supp(R) \leq supp(T)$. Hence, the property holds.

In other words, the *lbs* of a trend set T is defined as the smallest value among the support values of the supsersets of T in I_s. Because T's superset cannot have a larger support than that of T, the *lbs* measure is a lower bound on the support of T. Besides, for two trend sets $T_1 \subset T_2$, the set of supsersets of T_2 must be a subset of that of T_1. Hence, $lbs(T_2)$ cannot be larger than $lbs(T_1)$.

Example 13. Consider Is to be the set of all frequent trend sets of Table 4, listed with their support values. Using this information, the *lbs* lower bound can be calculated for each frequent trend set. Because the trend set $\{a1+, a2+, a3+\}$ has no proper superset in Is, $lbs(\{a1+, a2+, a3+\}) = supp(\{a1+, a2+, a3+\}) = 0.12$. Similarly, $lbs(\{a3+, a1-\}) = supp(\{a3+, a1-\}) = 0.16$. For $\{a3+\}$, $lbs(\{a3+\}) = \min\{supp(\{a3+, a1-\}), supp(\{a3+, a2-\}), supp(\{a1+, a2+, a3+\})\} = \min\{0.12, 0.16, 0.24\} = 0.12$. For $\{a2+\}$, $lbs(\{a2+\}) = \min\{supp(\{a1+, a2+, a3+\})\} = 0.12$. This example shows that many trend sets may have a same small value for the *lbs* lower bound.

Based on the *lbs* lower bound, the UVGR upper bound on the VGR measure is defined for reducing the search space at an inner level.

Definition 16 (Upper bound on VGR (UVGR)). Let there be a trend set ts, a neighboring space Ns and a set of trend sets I_s. An upper bound on the VGR of ts, named UVGR, is defined as:

$$UVGR(tss, Ns) = \frac{supp(ts, Ns)}{lbs(ts, Ws)}$$

Property 5. The UVGR upper bound is anti-monotonous and is an upper bound on the VGR. In other words, for two trend sets T and $R \in Is$ such that $T \subseteq R$, $UVGR(T) \geq VGR(T)$, $UVGR(R) \geq VGR(R)$, and $UVGR(T) \geq UVGR(R)$.

Proof 5. The *support* measure is anti-monotonous, and the *lbs* measure is monotonous (Property 3). Hence, the $UVGR$ upper bound is anti-monotonous. Because *lbs* is the denominator of the VGR, and *lbs* is a lower bound on the *support* measure (Property 4), it follows that $UVGR$ is an upper bound on the VGR.

The proposed algorithm apply this property to reduce the search space. When they consider an extension $tss \to T$, the proposed algorithms first compute $VGR(tss \to T)$ to decide if it is significant. If $tss \to T$ is significant, it is output. Then, the algorithms compute $UVGR(tss \to T)$ to determine if trend sequences of the form $tss \to R$ for $T \subset R$ may also be significant. If it is found that $SVGR(tss \to T) < minSig$, then the algorithms do not need to consider these extensions to reduce the search space. Calculating the UVGR upper bound is not costly because the *lbs* lower bound of each frequent trend set is precalculated once.

Example 14. Let $tss = \langle\{a3-\}\rangle$. Consider the extension $tss \to \{a2+\}$. Based on Table 4, it is found that $Ns(TSP(tss)) = \{(t_4, 2), (t_4, 4), (t_4, 5), (t_5, 4), (t_5, 5)\}$. $VGR(\{a2+\}, Ns(TSP(tss))) = \frac{\frac{1}{5}}{0.32} = 0.625 < minSig = 3$. Furthermore, knowing that $lbs(\{a2+\}) = 0.12$ (example 13), a simple calculation is performed, $UVGR(\{a2+\}, Ns(TSP(tss))) = \frac{\frac{1}{5}}{0.12} = 1.67 < minSig = 3$. Therefore, besides $\langle\{a3-\}\rangle \to \{a2+\}$, $\langle\{a3-\}\rangle \to \{a1+, a2+\}$, $\langle\{a3-\}\rangle \to \{a2+, a3+\}$, $\langle\{a3-\}\rangle \to \{a1+, a2+, a3+\}$ all other extensions can be directly removed from the search tree.

4.2.3. Discussion of the pruning effects of the three thresholds

The previous subsections have presented the proposed search space pruning properties, which are applied either at an outer or inner level. This subsection further discusses the pruning effects of the three thresholds ($minSig$, $minTailSup$ and $minInitSup$).

First, consider inner-level pruning. As explained in the previous subsection, inner level pruning can only be used when three conditions are satisfied: $VGR(tss \to T) < minSig$, $VGR(tss \to R) < minSig$ and $UVGR(tss \to T) < minSig$. Therefore, as the $minSig$ threshold is set higher, there is more opportunities for applying the pruning strategy, and the conditions are more likely to be satisfied. In theory, a large enough $minSig$ value can always be chosen to make pruning effective. However, in practice, a user may not want to set the threshold to a very large value to avoid missing interesting patterns. Thus, to prune a large part of the search space, it is important to have other ways of reducing the search space such as a tight upper bound on the VGR measure. Since the proposed $UGVR$ upper bound is obtained by substituting the support by the *lbs* lower bound in the denominator of the VGR measure, if the difference between these two values is small, the UVGR upper bound will be tighter.

It is evident that the significance threshold is important for pruning, especially when outer-level pruning is considered. For a sequence tss, the M extensions of tss in the next outer-level are considered only if tss is significant, which is determined by the $minTailSup$ and $minSig$ thresholds. If the tail support and/or significance values are smaller than their respective thresholds, extensions of tss of the next level are not considered. Thus, large $minTailSup$ and $minSig$ values allow to greatly prune the search space.

To summarize this discussion, $minInitSup$ influences the size of the search space, $minTailSup$ influences outer level pruning, and $minSig$ influences both inner and outer level pruning. In terms of pruning effect, outer level pruning is more effective than inner level pruning, as it will be shown in the experiments. But since outer-level pruning is easier than inner-level pruning, this paper focuses on how to improve the pruning effect of inner level pruning. A pruning strategy was proposed in Section 4.2.2. In the following, the two proposed algorithms will be presented, which adopt different approaches to make pruning more effective. As it will be discussed in the experimental evaluation section, each algorithm performs best in some situations.

4.3. Structures for Search Space Exploration

The previous subsections have described the search space of trend sequences, and proposed pruning properties that can be applied to reduce the search space. This section presents data structures that are used by the proposed algorithms to efficiently explore the search space using a Breadth-First Search (BFS) or Depth-First Search (DFS).

4.3.1. Structure for a Breadth-First Search

After finding all frequent trend sets, a breadth-first search algorithm needs to be able to quickly find all supersets of each trend set to compute the *lbs* lower bound and determine the next trend sets to be explored. Note that although the *lbs* lower bound is recursively defined, it is calculated iteratively (by Procedure 1). Storing a mapping of each trend set to all of its supersets may seem a good idea to quickly compute the *lbs* lower bound. However, if the search space of an inner level contains many large trend sets (containing more than two trends), storing a mapping of each trend set to its supersets can consume a lot of memory. For this reason, the solution adopted in this paper is to only map a trend set to its *direct supsersets* (supersets containing one more trend). This mapping for the running example is illustrated in Fig. 7(a).

4.3.2. Structure for a Depth-First Search

If a DFS is performed instead of a BFS, a different mapping between trend sets must be established to support search space exploration. This mapping is illustrated in Fig. 7(b) for the running example. To obtain this mapping, this paper defines a total order on trend sets and a dominance relationship between trend sets.

Definition 17 (Total order $\prec$ of trend sets). Without loss of generality, assume that elements of any trend set ts are sorted according to the lexicographical order. Let $ts[i]$ denotes the i-th element of ts. For two trend sets $ts1$ and $ts2$, let $minL = \min\{|ts1|, |ts2|\}$. The trend set $ts1$ is said to precede $ts2$, denoted as $ts1 \prec ts2$, if and only if one of the following constraints is satisfied.
1) $\exists i \in [1, minL]$ such that $ts1[j] = ts2[j]$ for $j < i$ and $ts1[i] < ts2[i]$
2) $\forall j \in [1, minL]$, $ts1[j] = ts2[j]$ and $|ts1| < |ts2|$

Definition 18 (Dominated superset). A trend set $ts2$ is said to be a dominated superset of a trend set $ts1$ (and $ts1$ is said to be a dominating subset of $ts2$) if $ts1 \subset ts2$, $|ts1| + 1 = |ts2|$ and $ts1 \prec ts2$.

Example 15. For the running example, assume that the lexicographical order is $a1+ < a2+ < a3+ < a1- < a2- < a3-$. Then, $\{a1+, a2+\} \prec \{a1+, a3+\}$ and $\{a1+, a2+\} \prec \{a1+, a2+, a3+\}$. Moreover, $\{a1+, a2+, a3+\}$ is a dominated superset of $\{a1+, a2+\}$ but $\{a1+, a3+\}$ is not.

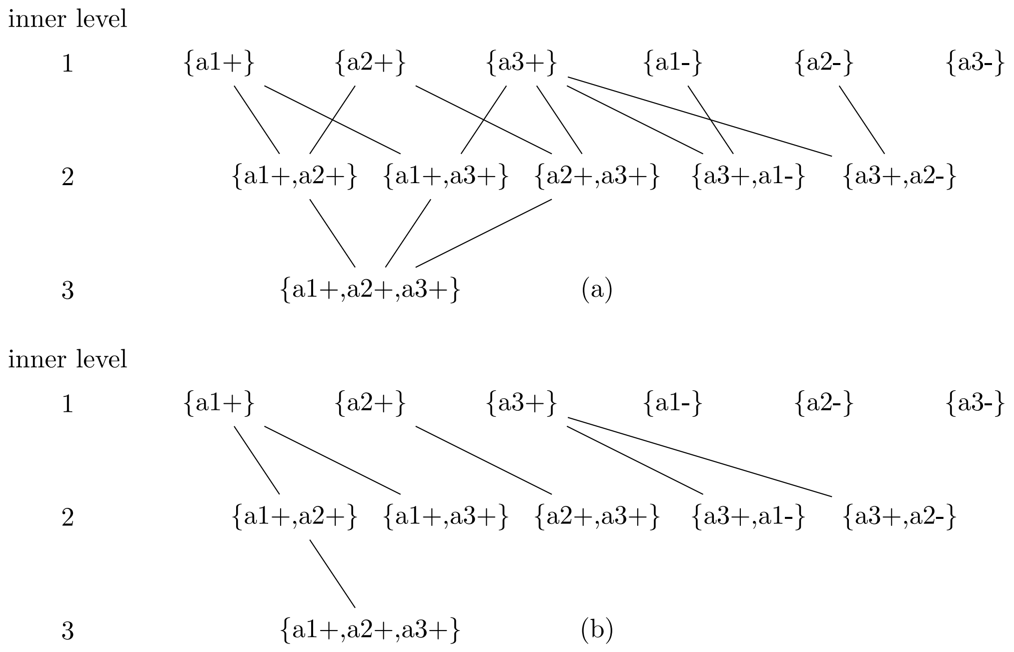

Figure 7: (a) Mapping of trend sets to their direct supersets to calculate the *lbs* lower bound and perform a BFS. (b) Mapping from trend sets to dominated supersets to perform a DFS.

Procedure 1: Calculate the mapping and *lbs* lower bound of trend sets for a BFS

```
input : $levelMapTrendset$: mapping from a level number to its trend sets
        $trendsetMapSuperset$: mapping from a trend set to its direct supersets
        $trendsetMapSup$: mapping from a trend set to its support
output: $trendsetMapLBS$: mapping from a trend set to its lbs
1  $maxL \leftarrow levelMapTrendset.size$;
2  foreach $l \leftarrow maxL$ to 1 do
3      set of trend sets $sts \leftarrow levelMapTrendset.get(l)$;
4      foreach trend set $ts \in sts$ do
5          $lbs \leftarrow trendsetMapSup.get(ts)$;
6          foreach superset $st \in trendsetMapSuperSup.get(ts)$ do
7              $superSup \leftarrow trendsetMapSup.get(st)$;
8              if $superSup < lbs$ then $lbs \leftarrow superSup$;
9          end
10         add $(ts, lbs)$ to $trendsetMapLBS$;
11     end
12 end
13 return $trendsetMapLBS$;
```

As it can be observed in Fig. 7(b), an important property of the proposed mapping is that using that mapping each trend set can only be accessed from a single dominating subset. Moreover, more generally, there exist a single path from the root of the search space (the empty set {}) to each trend set, as shown in Fig. 7(b). Thus, if a depth-first search is performed, instead of mapping a trend set to all direct supersets, a trend set can be mapped to only its dominating supersets, thus reducing memory usage. But the most important is that using this search structure, no redundant pruning test is conducted because each trend set can only be accessed through a single path. If the algorithm considers a trend set during the DFS and it is found that a pruning property can be applied, the subtree rooted at this trend set is pruned. Thus, all trend sets of that subtree can be ignored, thus reducing the search space. Procedure 2 provides the pseudocode for generating this mapping and calculating the *lbs* lower bound on the support.

Procedure 2: Calculate the mapping and lower bound of trend sets for a DFS

```
input : levelMapTrendset: mapping from a level number to its trend sets
        trendsetMapSup:mapping from a trend set to its support
        trendsetMapSuperset: mapping from a trend set to its direct supersets
output: trendsetMapDom: mapping from a trend set to its dominating supersets
        trendsetMapLBS: mapping from a trend set to its lbs
1  maxL ← levelMapTrendset.size;
2  foreach l ← maxL to 1 do
3      set of trend sets sts ← levelMapTrendset.get(l);
4      foreach trend set ts ∈ sts do
5          lbs ← trendsetMapSup.get(ts);
6          doms ← {};
7          foreach superset st ∈ trendsetMapSuperset.get(ts) do
8              if $ts \prec st$ then
9                  add st to doms;
10                 superSup ← trendsetMapSup.get(st);
11                 if superSup < lbs then lbs ← superSup;
12             end
13         end
14         add (ts, doms) to trendsetMapDom;
15         add (ts, lbs) to trendsetMapLBS;
16     end
17 end
18 return trendsetMapDom,trendsetMapLBS;
```

4.4. *The TSeqMiner$_{dfs-dfs}$ algorithm*

As previously pointed out, the search space for patterns can be viewed as consisting of two levels (inner levels and outer levels). Based on the previous discussion, it is clear that using a BFS for outer levels does not bring any benefits and requires more memory than performing a DFS. For this reason, both algorithms proposed in this paper apply a DFS for outer levels. The key difference between the two proposed algorithms is that the first one (named TSeqMiner$_{dfs-dfs}$) applies a DFS to explore inner levels, while the second one (named TSeqMiner$_{dfs-bfs}$) applies a BFS for inner levels. In other words, both algorithm have the same input and output but they search inner levels using different approaches. In this subsection, the TSeqMiner$_{dfs-dfs}$ algorithm is presented. Then, the next subsection will present the TSeqMiner$_{dfs-bfs}$ algorithm. In the following, the notation "A.x" will be used to refer to "Algorithm x".

TSeqMiner$_{dfs-dfs}$ takes as input a set of trend graphs (obtained by preprocessing the original dynamic attributed graph), several thresholds (*minSig*, *minInitSup*, *minTailSup*), and a neighborhood definition $\mathcal{NH}$. The algorithm outputs all significant trend sequences. TSeqMiner$_{dfs-dfs}$ first applies a frequent itemset mining algorithm to find all frequent trend sets with their corresponding supporting points (line 1 A.3). In the TSeqMiner$_{dfs-dfs}$ implementation, a modified version of *Eclat* [60] is used, which takes as input the vertical database and the *minInitSup* threshold and output all trendsets having a support no less than *minInitSup* with supporting points. Then, the frequent trend sets are used to generate the mappings required by the algorithm (line 2 A.3). *levelMapTrendset* maps each level to its trend sets, *trendsetMapSup* maps each trend set to its support, and *trendsetMapSuperset* maps each trendset to its direct supersets. Thereafter, Procedure 2 is applied to obtain a mapping from each trend set to the *lbs* lower bound, which will be used for search space pruning, and a mapping *trendsetMapDom* from each trend set to its dominating superset, which will be used to perform the depth-first search (line 3 A.3).

At the outer level, the TSeqMiner$_{dfs-dfs}$ algorithm explores patterns by appending trend sets to a prefix (a trend sequence) to consider longer trend sequences. The outer level search is implemented by the $outerDFS()$ recursive procedure (Algorithm 4). Its initial parameters are prepared in line 4-8 A.3. The $outerDFS()$ procedure has three parameters. $prefix$ is a sequence of trend sets such that $prefix[1:i] \quad \forall i \leq prefix.size$ is a significant trend sequence. *newTrendsetList* is a list of trend sets that can be appended to $prefix$ to generate larger patterns that may form a significant trend sequence. Each trend set $ts \in newTrendsetList$ has a set of supporting points in *addedSPList*. Based on this information, for each k-trend-sequence tss, if its size is larger than one, it is output (A.4 line 2-3). Then, the method *acquireNeighbors* constructs the neighboring space based on the tail supporting points of tss (A.4 line 4-5). The method receives the neighborhood definition, which can be customized by the user to find various types of patterns. In this paper, we use a simple neighborhood definition, presented in Section 3. For the neighboring subspace, an inner depth-first search $innerDFS()$ is performed to find all new trend sets that are significant in that subspace and their corresponding sets of supporting points (line 6-7 A.4). Finally, all the newly found significant trend sequences are output and the search process continues by calling $outerDFS()$

again (line 8-9 A.4). The last operation is for reusing memory.

Algorithm 3: The TSeqMiner$_{dfs-dfs}$ algorithm

```
input  : trendsetMapSP: a set of trend graphs($tGs$), $minSig$, $minInitSup$, $minTailSup$, $\mathcal{NH}$: a
         neighborhood definition
output: all significant trend sequences

1  $trendsetMapSP \leftarrow Eclat(tGs, minInitSup)$;
2  $levelMapTrendset, trendsetMapSup, trensetMapSuperset \leftarrow$
     $generateMappings(trensetMapSP)$;
3  $trendsetMapDom, trendsetMapLBS \leftarrow$ Procedure 2;
4  $addedTrendsetList$, $addedSPList \leftarrow \{\}$;
5  foreach $(ts, SP) \in trendsetMapSP$ do
6      $addedTrendsetList.add(ts)$;
7      $addedSPList.add(SP)$;
8  end
9  $outerDFS(\{\}, addedTrendsetList, addedSPList)$;
```

Algorithm 4: outerDFS()

```
input:  $prefix$: a sequence of trend sets that form a $k$-trend sequence;
        $newTrendsetList$: a list of trend sets to be appended to $prefix$;
        $tailSPList$: a list of sets of supporting points corresponding to trend sets in
  $newTrendsetList$;
output:  all significant trend sequences that extend $prefix$
1  foreach $ts \in newTrendsetList$ do
2      $tss \leftarrow prefix + ts$;
3      if $tss.size > 1$ then $outputPattern(tss)$ ;
4      $SP \leftarrow$ corresponding set of supporting points in $tailSPList$;
5      $neighborSP \leftarrow acquireNeighbors(SP, \mathcal{NH})$;
6      $addedTsList, addedSPList \leftarrow \{\}$;
7      $innerDFS(neighborSP, addedTsList, addedSPList)$;
8      $prefix \leftarrow tss$;
9      $outerDFS(prefix, addedTsList, addedSPList)$;
10     remove last trend set of $prefix$;
11 end
```

At the inner level, a DFS is applied. For a subspace, the $innerDFS()$ function (Algorithm5) finds all significant trend sets and the corresponding sets of supporting points. $innerDFSHelper()$ is a recursive auxiliary function, initially called by $innerDFS()$ (line 2 A.5). Its main purpose is to process a trend set ts of a given subspace $neighborSP$. First, $innerDFSHelper()$ obtains the support of ts in the whole space, which has been calculated in advance (line 1 A.6). Then, the local support and set of supporting points of ts in the subspace is calculated (line 2 A.6) by calling the $computeLocalSup()$ procedure. It performs the intersection of a subspace and supporting points of ts in the whole space. Thereafter, $VGR(ts, neighborSP)$ is calculated (line 3 A.6). If the significance and number of supporting points are larger than $minSig$ and $minTailSup$, respectively, the trend set and its supporting points are added to the corresponding lists (5-6). Otherwise, the algorithm checks if the pruning strategy can be applied. For this, the upper bound on VGR is calculated by simply substituting lbs by $supp$ in the denominator of the equation of line 3. The resulting equation is shown in line 9. If the upper bound is less than the threshold, all dominated supersets can be ignored. For the DFS, it means that it is not necessary to explore all descendant nodes of the current node (line 10 A.6). Thus, many supersets can be pruned. Otherwise, each dominated superset ts' of ts will be processed (line 12,17 A.6). If medium-grained pruning is activated (line 13 A.6), then lbs of ts is the lbs

of ts' (line 14 A.6). It means that the trend sets of the subtree rooted at ts', together with ts, are used to calculate lbs. In that case, if $UVGR$ of ts is smaller than $minSig$, the subtree rooted at ts' can be pruned (line 15-16 A.6).

Algorithm 5: inner-DFS() function

```
input:  $neighborSP$: a neighboring space consisting of points;
        $addedTsList$: a list of trend sets that are significant in the neighboring space;
        $addedSPList$: a list of set of supporting points corresponding to trend sets in $addedTsList$;
output: $neighborSP$, $addedTsList$, $addedSPList$
1  foreach $trend\ set\ ts \in levelMapTrendset.get(1)$ do
2      $innerDFSHelper(ts, neighborSP, addedTsList, addedSPList)$
3  end
4  innerDFSHelper($ts$, $neighborSP$, $addedTsList$, $addedSPList$):
```

Algorithm 6: innerDFSHelper() function

```
input:  $ts$: current processed trend set;
        $neighborSP$, $addedTsLst$, $addedSPList$: same as in Algorithm 5
output: $neighborSP$, $addedTsList$, $addedSPList$
1  $supp \leftarrow trendsetMapSup.get(ts)$;
2  $localSup, SP \leftarrow computeLocalSup(neighborSP, ts)$;
3  $vGR \leftarrow \frac{localSup}{supp}$;
4  if $vGR \geq minSig\ and\ |SP| \geq minTailSup$ then
5      $addedTsList.add(ts)$;
6      $addedSPList.add(SP)$;
7  else
8      $lbs \leftarrow trendsetMapLBS.get(ts)$;
9      $uVGR \leftarrow \frac{localSup}{lbs}$;
10     if $uVGR < minSig$ then return;
11 end
12 foreach $ts' \in trendsetMapDom.get(ts)$ do
13     if $mGP$ then
14         $lbs \leftarrow trendsetMapLBS.get(ts')$;
15         $uVGR \leftarrow \frac{localSup}{lbs}$;
16         if $uVGR \geq minSig$ then $innerDFSHelper(ts', neighborSP, addedTsList, addedSPList)$ ;
17     else $innerDFSHelper(ts', neighborSP, addedTsList, addedSPList)$;
18 end
```

4.4.1. A detailed example

This section provides a detailed example illustrating how the TSeqMiner$_{dfs-dfs}$ algorithm is applied. Consider the database of Fig. 4, $minInitSup = 3$, $minTailSup = 1$ and $minSig = 3$. The algorithm first identifies frequent trend sets by applying a modified frequent itemset mining algorithm. The frequent trend sets $\{\{a1+\}, \{a2+\}, \{a3+\}, ...\}$, their supporting points $\{SP1, SP2, SP3, ...\}$ and $supp$ values are shown in Table 3. For example, the supporting points of $\{a1+\}$ are $SP1 = \{(t1_1, 1), (t_1, 3)$, $(t_2, 4)$, $(t_3, 4)$, $(t_4, 1)$, $(t_4, 2)$, $(t_4, 4), (t_4, 5), (t_5, 3), (t_5, 4)\}$. Besides, the structure for supporting the DFS is built (illustrated in Fig. 7(b)), and the lbs values of frequent trend sets are calculated for the DFS (shown in Table 4 for medium grain and large grain pruning).

Then, the search for patterns starts from $prefix = \{\}$. At the outer level, TSeqMiner$_{dfs-dfs}$ performs a depth-first search. The trend sequence $ts = \{a1+\}$ is first considered. The neighboring space of {a1+} is caculated using its supporting points $Ns(SP1)$. The result is $\{(t_2,1),(t_2,2),(t_2,3),(t_2,5),(t_3,1),(t_4,1),(t_5,$ $1),(t_5,2),(t_5,3),(t_5,4)\}$. Then, the algorithm explores the subspace of ts by performing an inner depth-first search to find significant trend sets with their sets of supporting points. Based on the search space mapping of Fig. 7(b), {a1+} is first considered. The local supporting points of {a1+} are $LSP1 = Ns(SP1) \cap SP1 = \{(t_4,1),(t_5,3),(t_5,4)\}$. Thus, the significance of {a1+} is $VGR(\{a1+\}, Ns(SP1)) = \frac{\frac{|LSP1|}{|Ns(SP1)|}}{supp(\{a1+\},Ws)} = \frac{\frac{3}{10}}{0.44} = 0.681 < minSig = 3$. Hence, $\{a1+\}$ is not added to the list of significant trend sequences. Then, the $UVGR$ upper bound of $\{a1+\}$ is calculated as $UVGR(\{a1+\}, Ns(SP1)) = \frac{\frac{3}{10}}{lbs(\{a1+\},Ws)} = \frac{0.3}{0.12} = 2.5 < minSig$. The $lbs(\{a1+\}, Ws))$ value used in this calculation is obtained from the column "DFS/large-grain" of Table 4. According to Property 5, the significance of supersets of $\{a1+\}$ is smaller than $minSig$. Therefore, it is unecesary to pursue the DFS from the current node $\{a1+\}$. According to Fig. 7(b), $\{a1+, a2+\}$, $\{a1+, a3+\}$, $\{a1+, a2+, a3+\}$ are pruned. Thereafter, the algorithm considers the next trend set $\{a2+\}$. It is found that $VGR(\{a2+\}, Ns(SP1)) = \frac{\frac{4}{10}}{0.32} = 1.25 < minSig$, and thus $\{a2+\}$ is not a significant trend sequence. But because $UVGR(\{a2+\}, Ns(SP1)) = \frac{\frac{4}{10}}{0.12} = 3.33 > minSig$, the pruning condition is not met and the DFS must be pursued by extending $\{a2+\}$. The algorithm first considers $\{a2+, a3+\}$ at the next level. It is found that $VGR(\{a2+, a3+\}, Ns(SP1)) = \frac{\frac{2}{10}}{0.2} = 1$, and thus that this pattern is not significant. Similarly, $\{a3+\}$, $\{a3+, a1-\}$ and $\{a3+, a2-\}$ are insignificant and the two latter satisfy the search space pruning property. The same happens for $\{a1-\}$, $\{a2-\}$ and $\{a3-\}$. Therefore, $innerDFS()$ returns an empty list of significant trend sequences, indicating that no extensions of $\{a1+\}$ are significant, and it is not necessary to extend the current node $\{a1+\}$ further at the outer level. Next the node $\{a2+\}$ is processed, and this process continues for other patterns in the same way. After the DFS terminates at the outer level, the algorithm stops and returns all significant trend sequences. The result are patterns $\langle\{a1+, a2+, a3+\}, \{a3-\}\rangle$ and $\langle\{a1+, a2+\}, \{a3-\}\rangle$.

4.4.2. An optimization: medium-grained pruning

The above version of the TSeqMiner$_{dfs-dfs}$ algorithm is said to use large-grained pruning. In the example, when pruning is attempted for the node {a1+} in Fig. 7(b), the considered set is $Is = \{\{a1+\}, \{a1+, a2+\}, \{a1+, a2+, a3+\}, \{a1+, a3+\}\}$. In this case, if the significance upper bound $UVGR$ is smaller than $minSig$, neither of the two subtrees rooted at the current node can be pruned. However, there still exists a possibility that one of these sub-trees can be pruned when we consider $Is = \{\{a1+\}, \{a1+, a2+\}, \{a1+, a2+, a3+\}\}$ and $\{\{a1+\}, \{a1+, a3+\}\}$ separately. Based on this consideration, a strategy called medium-grained pruning consists of not storing a unique lbs value for $\{a1+\}$, but to set $lbs(\{a1+\}, Ws) = lbs(\{a1+, a2+\}, Ws)$ when considering pruning the subtree rooted at $\{a1+, a2+\}$. A similar process is performed for the other subtree. Considering each sub-tree separately allows to obtain a tighter upper bound on the significance. Thus, medium-grained pruning can generally improve the algorithm's efficiency. This optimization is activated by setting mGP to true in line 13 of Algorithm 6.

4.4.3. Complexity

The complexity of the TSeqMiner$_{dfs-dfs}$ algorithm is analyzed as follows. The algorithm first discovers frequent trend sets using frequent itemset mining. Frequent itemset mining algorithms generally have a time complexity that is linear to the number of possible patterns (trend sets). If there are w possible trend values in the original database, in the worst case $2^w - 1$ possible trend sets are considered, although in real-life that number is generally much smaller than $2^w - 1$ because not all trends co-occur.

After the set M of frequent trend sets has been identified, the three mappings $levelMapTrendset$, $trendsetMapSuperset$, $trendsetMapSup$ are generated, which require $O(M)$, $O(M^2)$, $O(M)$ time, respectively. Then, $O(M^2)$ operations are performed to generate $trendsetMapDom$ and $trendsetMapLBS$. Suppose that the average depth of the tree is $\overline{D}$. For each node in the tree up to M extensions are attempted. The main cost to evaluate an extension is the intersection of its neighboring space and supporting points of a trend set, whose average cost is assumed to be $\overline{I}$. Therefore, the cost for considering all extensions is $(M + M^2 + ... + M^{\overline{D}}) \cdot M \cdot \overline{I}$. The total time cost is $O(M^{\overline{D}+1} \cdot \overline{I})$. In the worst case, $\overline{D} = T_{max} - 1$.

In terms of space complexity, in the worst case, the maximal depth of the search tree is D_{max}, and each node on a path from the root to the current node has M extensions. Furthermore, the average number of tail supporting points of each extension is $SP_{max} = O(|Ws|)$. Therefore, the total space cost is $O(D_{max} \cdot M \cdot |Ws|)$ if each node in a path maintains information about all extensions. But this is unecessary. Since a depth-first search is adopted on both dimensions, a complete depth-first traversal can be used to address this issue. This can reduce M to 1. The complete depth first search can be implemented without much difficulty, and thus details are omitted.

4.5. The TSeqMiner$_{dfs-bfs}$ algorithm

The second proposed algorithm is TSeqMiner$_{dfs-bfs}$ (Algorithm 7). For outer levels, it performs a depth-first search, while it performs a BFS for inner levels. Searching at the outer level is performed using Algorithm 3 and Algorithm 4 except that two differences are introduced. First, a modified Procedure 1 is applied to generate the mapping from trend sets to the *lbs* lower bound. This replaces line 3 of Algorithm 3. Since the algorithm performs a BFS, trend sets are processed level by level. A trend set of level k is only pruned if all its subsets of level $k-1$ have been pruned. Thus, pruning a trend set is unlikely to happen if it has many subsets. To improve pruning effectiveness, this paper considers direct supersets (supersets containing only one more additional item) to tighten the *lbs* lower bound on the support of each pattern. Thus, line 2 of Procedure 1 is replaced by "foreach ← 1 to $maxL$ do". For this reason, TSeqMiner$_{dfs-bfs}$ is regarded as using a level-wise pruning strategy. The second difference is that line 7 of Algorithm 4 is replaced by a call to $innerBFS()$ to apply a BFS at the inner level.

The innerBFS() procedure processes trend sets iteratively in a level-wise manner (Algorithm 7). At the beginning of an iteration, the set of current trend sets $curLTss$ is initialized as the trend sets found in the previous iteration (line 4,5 A.7). At the first iteration, this set is frequent trend sets containing a single trend. Then, for each unpruned trend set ts of the current level, the algorithm first calculates the support in the whole space and in the neighboring space, to then calculate the significance of ts in the subspace as the ratio of these two values (line 7-9 A.7). For a trend set, if the requirements of significance and supporting points are met, the trend set and its supporting points are added to the corresponding lists (line 10-13 A.7). Otherwise, the algorithm checks if the pruning strategy can be applied (line 14-31 A.7). First, level-wise pruning is applied (line 15-21). By considering the direct supersets of ts and the precalculated lower bound on the support, the $uVGR$ upper bound on the significance is obtained (15-16 A.7). If this upper bound is less than the threshold, all trend sets are removed from $nextTss$ for the current iteration (line 17-20 A.7). In the next iteration, $nextTss$ will become $curLTss$. Otherwise, if pair-wise pruning is activated, an additional pruning attempt is performed (line 22-30 A.7). This optimization will be explained in the next subsection.

4.5.1. An optimization: pair-wise pruning

An important way of improving the performance of the proposed algorithms is to tighten the upper-bound on the significance to eliminate more patterns. That upper bound is based on the *lbs* lower bound on the support, which further depends on the considered trend sets. Let $Is'(ts)$ denote the supersets considered for calculating the lower bound on the support of a trend set ts, that is $Is'(ts) = Is(ts) - \{ts\}$.

For the less optimized TSeqMiner$_{dfs-dfs}$ algorithm, $Is'(ts)$ is a minimal set satisfying the following properties: 1) all dominated supersets of ts are contained in $Is(ts)$;(2) $\forall ts' \in Is(ts)$, all dominated supersets of ts' are contained in $Is'(ts)$. If the upper bound on the significance of ts is smaller than the $minSig$ threshold, all trend sets in $Is'(ts)$ are pruned. The benefits of large-grained pruning comes with the risk that the *lbs* lower bound on the support may be loose. The optimized version of TSeqMiner$_{dfs-dfs}$ provides more stable pruning ability by partitioning $Is(ts)$ into several sets, making it less likely that all trend sets in $Is'(ts)$ share a very loose lower bound on the support.

For the TSeqMiner$_{dfs-bfs}$ algorithm, a similar optimization technique is designed to be used with the BFS. For that algorithm, $Is'(ts)$ consists of all direct supersets of ts. A similar partitioning results in pair-wise pruning (line 22-30 A.7). Knowing that ts is insignificant, the pruning condition for $ts' \in Is'(ts)$ is checked. For each direct superset in ts', if it has not been pruned, it will be considered for calculating the lower bound on the support of ts'. Then, if the pruning condition is satisfied, the considered trend sets are pruned. This pair-wise pruning is useful when level-wise pruning fails. This optimization is mainly based on the observation that calculating UVGR is much less costly than calculating VGR.

4.5.2. A detailed example

A detailed example is given to illustrate the inner BFS process of TSeqMiner$_{dfs-bfs}$, as well as the effect of the pair-wise pruning optimization. Since TSeqMiner$_{dfs-bfs}$ shares a similar outer level search process as TSeqMiner$_{dfs-dfs}$, this example continues the one of section 4.4. Having a neighboring space $Ns(SP1) = \{(t_2,1),(t_2,2),(t_2,3),(t_2,5),(t_3,1),(t_4,1),(t_5,1),(t_5,2),(t_5,3),(t_5,4)\}$, the algorithm will try to find all significant trend sets in this subspace. Initially, $curLTss = \{a1+, a2+, a3+, a1-, a2-, a3-\}$, $nextLTss = \{\{a1+, a2+\}, \{a1+, a3+\}, \{a2+, a3+\}, \{a3+, a1-\}, \{a3+, a2-\}\}$, and $VGR(\{a1+\}, Ns(SP1)) = \frac{\frac{3}{10}}{0.44} = 0.681 < 3$. By level-wise pruning, $Is'(\{a1+\}) = \{\{a1+, a2+\}, \{a1+, a3+\}\}$. Hence $lbs(\{a1+\}, Ws) = \min\{0.12, 0.4\}$, $UVGR(\{a1+\}, Ns(SP1)) = \frac{\frac{3}{10}}{0.12} = 2.5 < 3$. The pruning condition is satistfied. Then, $nextLTss = \{\{a2+, a3+\}, \{a3+, a1-\}, \{a3+, a2-\}\}$. Next, consider $\{a2+\}$. If is found that $VGR(\{a2+\}, Ns(SP1)) = \frac{\frac{4}{10}}{0.32} = 1.25 < 3$, $UVGR(\{a2+\}, Ns(SP1)) = \frac{\frac{4}{10}}{\min\{0.12,0.2\}} = 3.33 > 3$, and thus level-wise pruning fails. Thus, pair-wise pruning is checked. Only $\{a2+, a3+\}$ is contained in *nextLTss* and hence $Is'(\{a2+\}) = \{\{a2+, a3+\}\}$. The corresponding lower bound on the support is 0.2. By considering this scope, $UVGR(\{a2+\}, Ns(SP1)) = \frac{\frac{4}{10}}{0.2} = 2 < 3$. Therefore, pair-wise pruning is successful, and $\{a2+, a3+\}$ is removed from *nextLTss*. Similar calculations show that $\{a3+\}$ is insignificant and $\{a3+, a1-\}, \{a3+, a2-\}$ are pruned by level-wise pruning. *nextLTss* is then empty. Testing the remaining trend sets $\{a1-\}, \{a2-\}, \{a3-\}$ tells that they are not significant. The first loop ends with *nextLTss* being empty, resulting in performing no testing in the second loop. In the third loop, $\{a1+, a2+, a3+\}$ is found to be insignificant. Then, $innerBFS()$ returns two empty lists.

Algorithm 7: TSeqMiner$_{dfs-bfs}$

```
Input, output, operations are the same as Algorithm 3 except that:
$trendsetMapLBS \leftarrow$ Procedure 1 replaces Line 3;
$innerBFS(prefix, addedTsList, addedSPList)$ replaces
 $innerDFS(prefix, addedTsList, addedSPList)$ of Line 7 in Algorithm 4;

innerBFS($prefix$, $addedTsList$, $addedSPList$):
1  $nextLTss \leftarrow \{levelMapTrendset.get(1)\}$;
2  $curLTss \leftarrow \{\}$;
3  foreach $(l, sts) \in levelMapTrendset$ do
4      $curLTss \leftarrow nextLTss$;
5      $nextLTss \leftarrow levelMapTrendset.get(l+1)$;
6      foreach $ts \in sts$ do
7          $supp \leftarrow trendsetMapSup.get(ts)$;
8          $localSup, SP \leftarrow computeLocalSup(neighborSP, ts)$;
9          $vGR \leftarrow \frac{localSup}{supp}$;
10         if $vGR \geq minSig$ and $|SP| \geq minTailSup$ then
11             $addedTsList.add(ts)$;
12             $addedSPList.add(SP)$;
13         end
14         if $vGR < minSig$ then
15             $superMin \leftarrow trendsetMapLBS.get(ts)$;
16             $uVGR \leftarrow \frac{localSup}{superMin}$;
17             if $uVGR < minSig$ then
18                 foreach $st \in trendsetMapSuperset.get(ts)$ do
19                     if $st \in nextLTss$ then remove $st$ in $nextLTss$;
20                 end
21             end
22             else if $pair-wisepruning$ then
23                 foreach $st \in trendsetMapSuperset$ do
24                     if $st \in nextLTss$ then
25                         $superMin \leftarrow trendsetMapSup.get(st)$;
26                         $uVGR \leftarrow \frac{localSup}{superMin}$;
27                         if $uVGR < minSig$ then remove $st$ in $nextLTss$;
28                     end
29                 end
30             end
31         end
32     end
33 end
```

4.5.3. Complexity

The difference between the TSeqMiner$_{dfs-bfs}$ and TSeqMiner$_{dfs-dfs}$ algorithms is mainly about how the search is conducted at inner levels. Different traversal methods result in different visiting orders of trend sets and thus different pruning effects are obtained. In the worst case where no pruning is done, the complexity of TSeqMiner$_{dfs-dfs}$ is approximately the same as TSeqMiner$_{dfs-bfs}$. In practice, which algorithm performs better depends on the data and parameters settings, since both of these factors influence pruning. This will be discussed in more details in the experimental evaluation section.

4.6. How to set the parameters?

The proposed algorithms take as parameters three thresholds and a neighborhood definition. Different neighborhood definitions can be used to find different types of patterns. In this paper, a main definition was presented until now, which is suitable for the real data used in the experimental evaluation of this paper. But this definition could be replaced by other definitions suitable for other applications. For example, a less strict neighborhood definition could be used that also consider nodes that are two edges aways as neighbors.

As for the thresholds, increasing their values can decrease the number of patterns that are output, as the constraints set by the thresholds become more strict. And this contribute to decrease the runtime since more patterns may be pruned by the pruning strategies. But if the thresholds are set too high, it is possible that no patterns may be found, while if they are set too low, many patterns may be found, and it may be time-consuming to look at all the patterns. The optimal threshold values to find enough insightful patterns is dataset dependent. Thus, it is suggested that the user initially set thresholds to high values, and then decrease them until enough interesting patterns are found. This is the approach that has been used for the analysis of patterns found in real data described in the next section. Besides, the thresholds can be set in other ways for specific applications. For example, if the goal is to find rare but influential patterns, $minInitSup$ can be set to a very low value, while $minTailSup$ and $minSig$ can be set relatively high. The influence of thresholds on performance will be studied in the next section.

5. Experimental evaluation

This section reports results of experiments to evaluate the proposed algorithms. Since a new type of patterns has been proposed, as far as we know, there exist no algorithms that can be compared with the proposed algorithms. Therefore, the performance of the designed algorithms is evaluated by considering whether the proposed pruning techniques are efficient, and by comparing different version of the algorithms.

Experiments were conducted using a computer equipped with an Intel(R) Xeon(R) CPU E3-1270 3.60GHz) and 64 GB of RAM running Ubuntu 16.04. Algorithms are implemented in Java. Two real world datasets were used for quantitative experiments and pattern analysis.

- **DBLP dataset.** This dataset [28] consists of bibliographical entries of articles published between 1990 to 2010 by 2,723 authors in 43 conferences/journals. Each author in this dataset has more than 10 publications. The dynamic attributed graph has 9 timestamps ([1990-1994][1992-1996]...[2006-2010]). Each author is represented by a vertice, which is annotated with a value vector of size 43, indicating the publication count of the author for each conference.

- **Domestic US Flight dataset.** This dataset [31] provides data about airport traffic in the US from 01/08/2005 to 25/09/2005. Tree hurricanes have occurred during this period, namely, Irene(04/08-18/08), Katrina(23/08-31/08) and Ophelia(06/09-17/09). Data is aggregated by weeks for a total of 8 timestamps. Each attributed graph has 280 vertices (airport) with 8 attributes (e.g. number of arrivals/departures, average arrival/departure delay). Edges denote flight connections between airports.

In preparation for the experiments, each dataset was transformed into trend graphs. Using trends instead of numeric attribute values is desirable because it allows to find patterns that are robust to small variations. To obtain trends, different discretization strategies were used for each dataset.

For the DBLP dataset, attribute values often vary very slightly, and all attributes are of the same type (publication count). Thus, a simple discretization was applied:

$$trend_1(t,v,a) = \begin{cases} + & \text{if } value(t,v,a) < value(t+1,v,a) \\ = & \text{if } value(t,v,a) = value(t+1,v,a) \neq 0 \\ - & \text{if } value(t,v,a) > value(t+1,v,a) \\ \emptyset & textotherwise \end{cases}$$

where $value(t, v, a)$ denotes the value of attribute a of vertex v at timestamp t.

For the US Flight dataset, different types of attributes are used. Therefore, the standard deviation was used to perform discretization:

$$trend_2(t, v, a) = \begin{cases} ++ & \text{if } value(t+1, v, a) - value(t, v, a) \in scale * [2 * std(a), \infty) \\ + & \text{if } value(t+1, v, a) - value(t, v, a) \in scale * [std(a), 2 * std(a)) \\ = & \text{if } value(t+1, v, a) - value(t, v, a) \in scale * [-std(a), std(a)) \\ - & \text{if } value(t+1, v, a) - value(t, v, a) \in scale * [-2 * std(a), -std(a)) \\ -- & \text{if } value(t+1, v, a) - value(t, v, a) \in scale * [-\infty, -2 * std(a)) \end{cases}$$

where $std(a)$ denotes the standard deviation of attribute a where $scale$ is a coefficient.

To evaluate the impact of optimizations on the performance, the algorithms were compared with versions where some optimizations had been deactivated. In the following, TSeqMiner$_{dfs-dfs}{}'$ and TSeqMiner$_{dfs-bfs}{}'$ denote unoptimized versions of TSeqMiner$_{dfs-dfs}$ and TSeqMiner$_{dfs-bfs}$, respectively. Moreover, TSeqMiner$''$ denotes a version of these algorithms without pruning. The TSeqMiner$''$ algorithm is obtained by removing lines 8-10 and 13-16 in Algorithm 6.

5.1. Quantitative experiment

Experiments were first carried out to empirically evaluate performance of the algorithms in terms of runtime, memory and scalability. Note that performance improvement of outer level pruning is not show explicitly because outer level pruning, which is influenced by $minTailSup$ and $minSig$ cannot be deactivated. Otherwise, algorithms need to consider all possible extensions of trend sequences which makes them unable to terminate on the two datasets. But the following experiments will still offer a rough understanding of how $minTailSup$ and $minSig$ influence the performance of outer level pruning. In the following figures and texts, "NO-PRUNING" denotes the algorithm without inner pruning (TSeqMiner$''$), "BFS" (TSeqMiner$_{dfs-bfs}{}'$) and "BFS-OP" (TSeqMiner$_{dfs-bfs}$) denote unoptimized and optimized version of BFS algorithms respectively, "DFS" (TSeqMiner$_{dfs-dfs}{}'$) and "DFS-OP" (TSeqMiner$_{dfs-dfs}$) denote unoptimized and optimized version of DFS algorithms respectively. Therefore, the improvement provided by inner level pruning can be seen from the differences between "NO-PRUNING" and "BFS", "BFS-OP", "DFS", "DFS-OP".

5.1.1. Influence of minInitSup on runtime and number of patterns

The first experiment assesses the influence of $minInitSup$. Recall that this parameter is used to filter infrequent trend sets. More specifically, this parameter determines the number of frequent trend sets M, which directly influences the size of the inner search space. For this reason, in the following, the number M is directly measured instead of $minInitSup$ to assess the influence of M on algorithms' runtimes and on the number of patterns found.

Fig. 8(a) shows results for the DBLP dataset when $minTailSup = 0.0023(50)$ and $minSig = 8$. The first observation is that as M is increased, both the runtimes and number of patterns of the compared algorithms quickly grow. Another observation is that algorithms using inner level pruning strategies ("BFS" and "DFS") outperform the algorithm without inner level pruning ("NO-PRUNING"), and the optimized algorithms ("BFS-OP" and "DFS-OP") outperform corresponding other algorithms. Fig. 8(b) shows results for the US Flight dataset. It can be observed that the number of patterns and runtimes increase in a similar way as for the DBLP dataset. However, on the US Flight dataset, pruning is more effective for reducing runtimes. Besides, it is found that depth-first search algorithms outperforms other algorithms by a wide margin on that dataset. It is mainly because many trends appear together in the US Flight dataset. Thus, the support of large trend sets is close to that of small trend sets. Therefore, the lower bound on the support is tight, which enhances the effect of large grain pruning. Generally speaking, pruning using the DFS is done at a larger grain, which can be observed from the search structure of Fig. 7.

In this experiment, an approximate exponential growth of runtimes is observed, which is in accordance with the time cost analysis presented in section 4.4.3, where the total cost is $O(M^{\overline{D}+1} \cdot \overline{I})$. Hence, a larger M value indicates a larger search space. In other words, more patterns need to be considered, and more patterns must be extended.

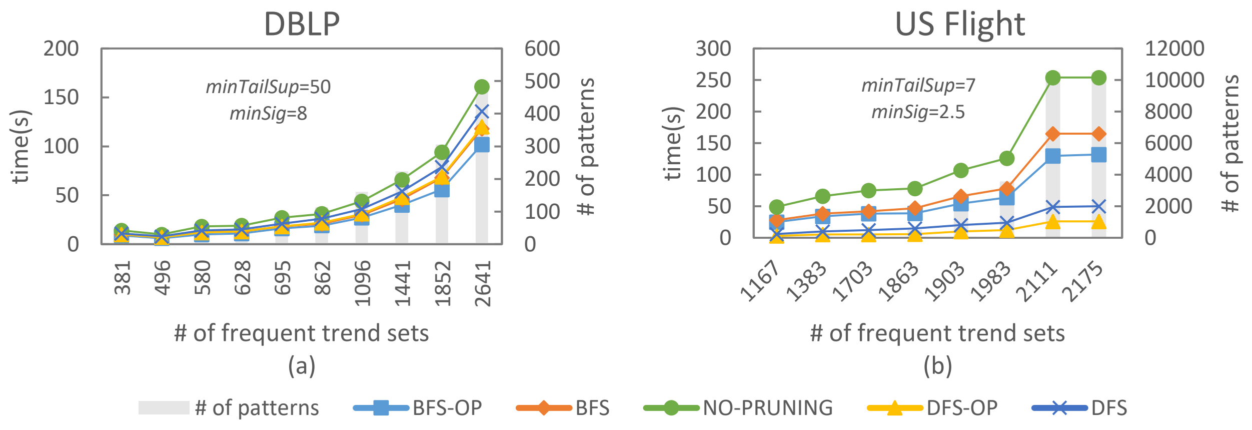

Figure 8: Influence of the number of frequent trend sets on runtime and number of patterns found.

5.1.2. Influence of outer level pruning on runtime and number of patterns

The second experiment assesses the impact of outer level pruning. Recall that while $minInitSup$ determines the size of the search space, the effectiveness of outer level pruning is determined by both $minTailSup$ and $minSig$. Because $minSig$ also influences inner level pruning, this experiment is done with TSeqMiner$''$ to avoid pruning the inner search space.

Fig. 9, (a) (b) show the number of patterns and runtimes for the DBLP dataset, while Fig. 9 (c) (d) show the same for the US Flight dataset. From these four figures, it is found that both $minTailSup$ and $minSig$ greatly influence the performance with pruning. In particular, if at least one of the tresholds is set to a large value, pruning is very effective by reducing the number of patterns and the runtime. This is the case for example when $minSig = 30$ or $minTailSup = 110$ on the DBLP dataset, as shown in Fig. 9 (a). From a practical perspective, setting these parameters to large values may result in not finding some interesting patterns. Thus, the user can select appropriate threshold values to obtain a good trade-off between performance and number of patterns found.

Next, consider the relative influence of $minSig$ and $minTailSup$ on pruning. For a fixed $minSig$ value, setting $minTailSup$ to a large value results in finding very few patterns and small runtimes, suggesting that the search tree was quickly pruned. Generally, setting $minTailSup$ to a large value always results in very effective pruning even for values of $minSig$ that are relatively small. As $minTailSup$ is decreased, it has less influence on pruning, and the $minSig$ parameter becomes the most important for pruning. For relatively small $minTailSup$ values, $minSig$ has a greater effect on pruning than $minSig$. In non-extreme cases, both parameters are important for pruning. This is for example the case for $minSig = 12$ and $minTailSup = 40$, and also for $minSig = 10$ and $minTailSup = 50$. These two parameters thus both contribute to limit the number of pattern extensions to be considered. The third observation is that when both parameters are set to very small values, the number of patterns and runtime can considerably increase. For example, consider Fig. 9 (c). When $minSig \geq 4$, the number of patterns and runtime slowly increase when $minTailSup$ is increased. But when $minSig = 3.5$, the latter increases more quickly. This shows the importance of outer level pruning on performance.

In summary, outer level pruning is extremely important to obtain results in a reasonable time. Using the $minSig$ and $minTailSup$ thresholds greatly contribute to achieving this goal. When also considering inner level pruning, setting $minSig$ to relatively large values also contribute to improving the performance.

5.1.3. Influence of minSig on runtime and number of patterns

In a third experiment, the influence of inner level pruning on the performance of the compared algorithms was assessed. The runtime and the number of patterns was measured when varying the $minSig$ parameter, while other parameters were fixed. For DBLP, $minInitSup$ was fixed to generate 1441 frequent trend sets

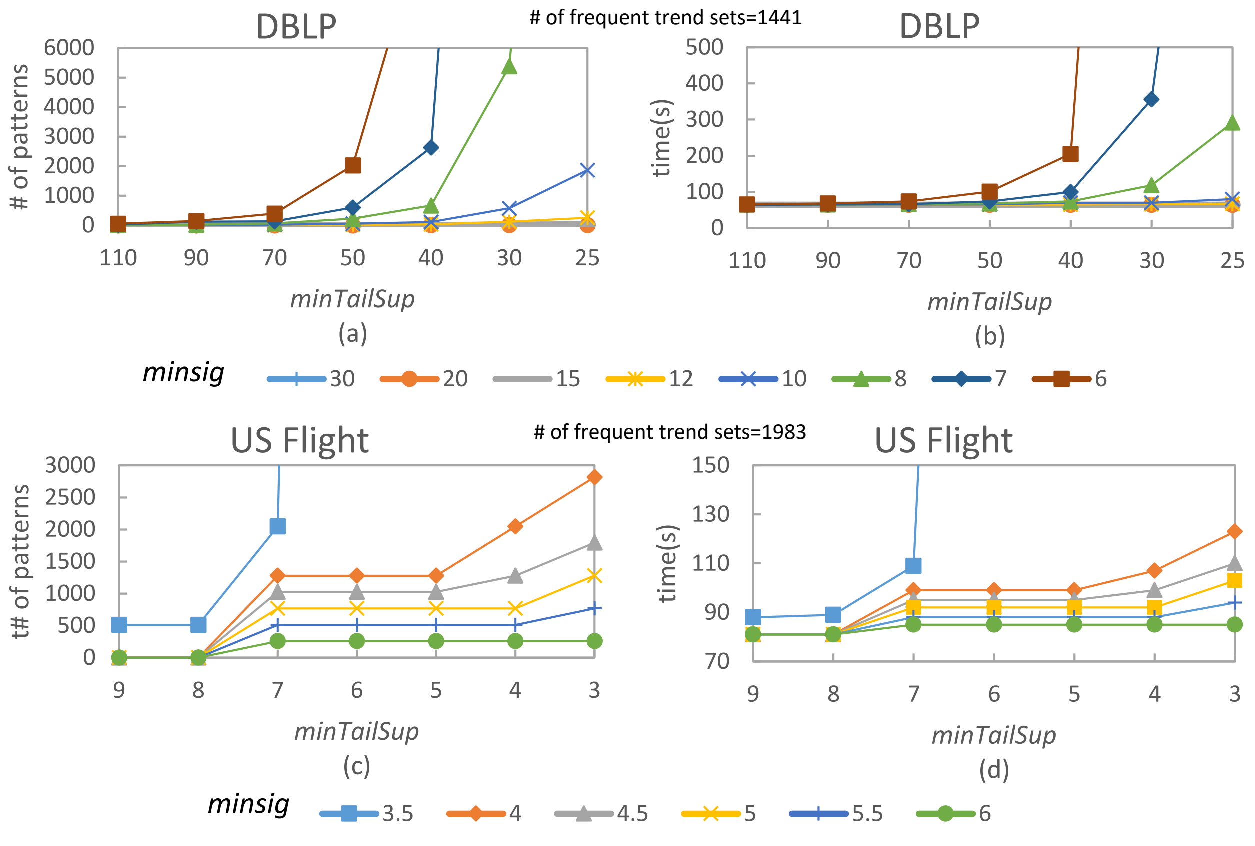

Figure 9: Influence of $minTailSup$ on (a) number of patterns for DBLP, (b) runtime for DBLP, (c) number of patterns for US Flight, and (d) runtime for US Flight.

and $minTailSup$ was set to 70. For the US Flight dataset, $minTailSup$ and $minInitSup$ were set to 5 and to generate 1983 frequent trend sets, respectively. These fixed parameter values were chosen as they can clearly show the influence of $minSig$ and ensure that enough trend sets are considered. Fig. 10 (a) and (b) show results for the DBLP and US Flight datasets, respectively. In these charts, primary and secondary vertical axes indicate runtimes and pattern counts, respectively. It should be noted that although $minSig$ influences both inner and outer level pruning, it is possible to observe its influence on inner level pruning by considering TSeqMiner$''$ as the baseline.

A first observation made from Fig. 10 is that inner level pruning is more effective when $minSig$ is increased. This is because choosing larger threshold values increase the number of patterns that will meet the pruning condition based on $minSig$. In fact, even patterns having loose upper bound values on their significance may be pruned when $minSig$ is large.

A second observation is that optimized versions of the algorithms always outperform the corresponding unoptimized versions. This is because performing the additional pruning tests is not costly while these tests can prune large parts of the search space. For high $minSig$ values, there is no significant difference between the runtime and number of patterns of unoptimized and optimized versions. The reason is that pruning almost always succeed at the first attempt for high $minSig$ values.

A third observation is that there is no guarantee that depth-first search outperform breadth-first search or conversely. On the DBLP dataset, for high $minSig$ values such as 100, TSeqMiner$_{dfs-dfs}$ and TSeqMiner$_{dfs-dfs}{}'$ are faster than TSeqMiner$_{dfs-bfs}$ and TSeqMiner$_{dfs-bfs}{}'$, respectively while for small values, the latter are faster. The reason is that depth-first search generally prunes at a large-grained level, and that this type of

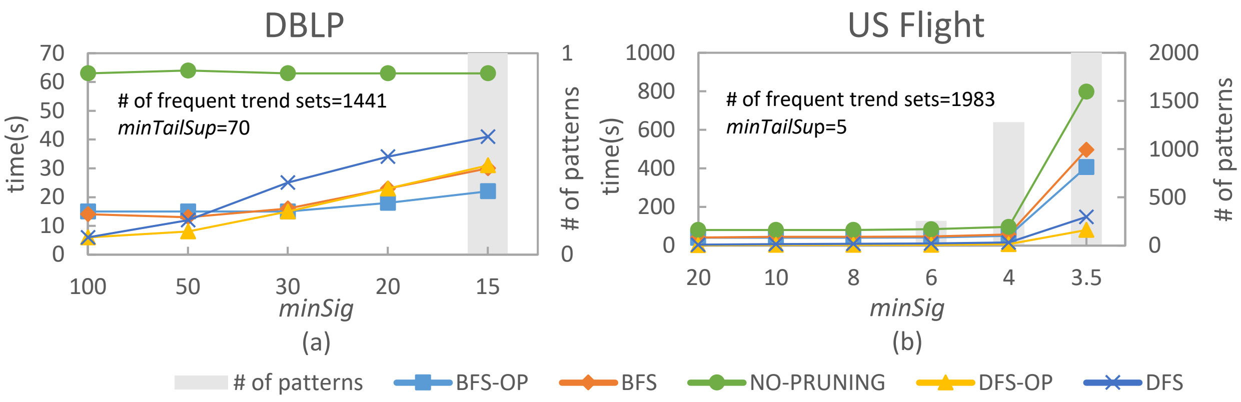

Figure 10: Influence of $minSig$ on runtimes and number of patterns on the (a) DBLP and (b) US Flight datasets.

pruning is more likely to succeed for high threshold values and is more sensitive to that threshold. Optimized versions decrease sensitivity to that threshold because smaller-grained pruning is also performed. Thus, generally TSeqMiner$_{dfs-dfs}$ may have better pruning effect at high threshold values, while the performance of TSeqMiner$_{dfs-bfs}$ is generally more stable when $minSig$ is varied on DBLP. However, this behavior is not observed on the US Flight dataset. In this case, depth-first search algorithms always outperform others by a large margin. The reason is that the 1,441 frequent trend sets of the DBLP dataset are composed of 125 trends, while the 1,983 frequent trend sets of the US Flight dataset are formed of only 20 trends. As a result, the inner search space of US Flight is deeper than that of DBLP, and successfully pruning a high level node can cut off a larger part of the search tree. The combination of this type of search space with a relatively tight lower bound on the support benefits the depth-first search on US Flight. As a result TSeqMiner$_{dfs-dfs}$ is up to one order of magnitude faster than TSeqMiner$_{dfs-bfs}$.

In summary, TSeqMiner$_{dfs-dfs}$ and TSeqMiner$_{dfs-bfs}$ outperform the baseline no pruning algorithm but whether TSeqMiner$_{dfs-dfs}$ is faster than TSeqMiner$_{dfs-bfs}$ depends on the search structure of the dataset and how $minSig$ is set. This shows that both algorithms are useful in some situations.

5.1.4. Influence of the number of timestamps, attributes and database size

In a fourth experiment, the influence of the number of timestamps, number of attributes and database size on performance was evaluated. For this scalability experiment, the real DBLP dataset was used rather than synthetic datasets. The reason is that synthetic databases typically do not contain patterns that are significant and meaningful.

In the experiment, the TSeqMiner$_{dfs-bfs}$ and TSeqMiner$_{dfs-dfs}$ algorithms were ran on different versions of DBLP with $minInitSup = 0.0022$, $minTailSup = 0.0032$ and $minSig = 8$. Fig. 11 (a) compares runtimes and number of patterns for various number of timestamps. It is observed that the runtimes of both algorithms exponentially increase with the number of timestamps. This is reasonable because adding timestamps increases the number of extensions that must be considered for sequences ending at the last timestamp.

Fig. 11 (b) shows runtimes and number of patterns for various number of attributes. It is observed that the runtime increases in an approximately linear way when the number of attributes is increased. Moreover, TSeqMiner$_{dfs-bfs}$ outperforms TSeqMiner$_{dfs-dfs}$, which is consistent with previous results. In fact, the number of attributes influences the shape of the inner search space. Thus, it influences the effects of pruning strategies. But the influence on the overall performance is relatively small.

The influence of database size on runtime was also assessed. The algorithms were ran while increasing the size of the DBLP and US flight datasets by repeating each dataset k times, where k is called the number of repeats. Note that increasing the size of a dataset in this way does not increase the number of timestamps but increases the number of nodes and edges. Fig. 12 (a) and (b) shows respectively the runtime for

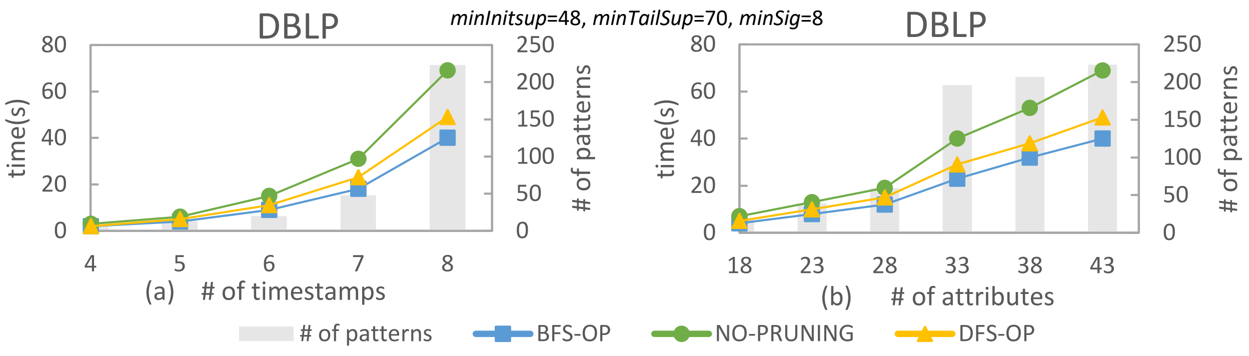

Figure 11: Influence of number of (a) timestamps and (b) attributes on the runtime and number of patterns.

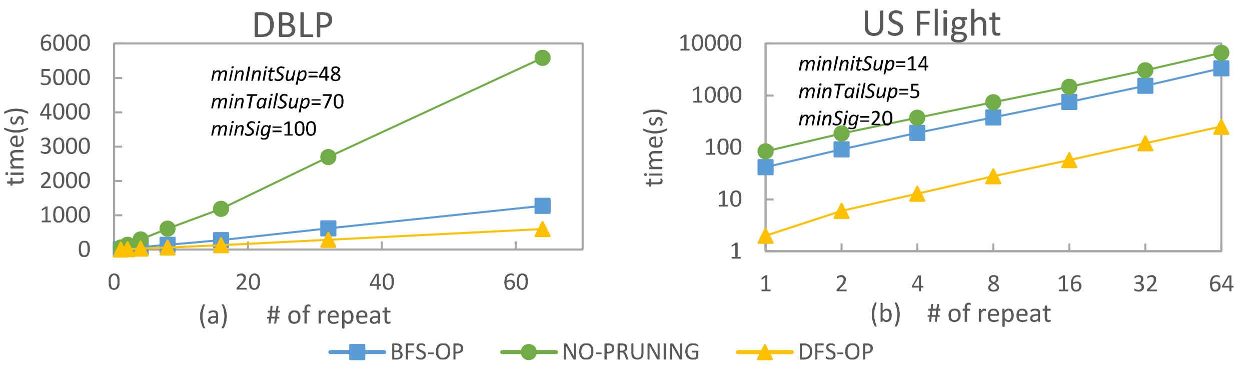

Figure 12: Influence of database size (number of repeats) on runtimes for (a) the DBLP (b) and US Flight datasets.

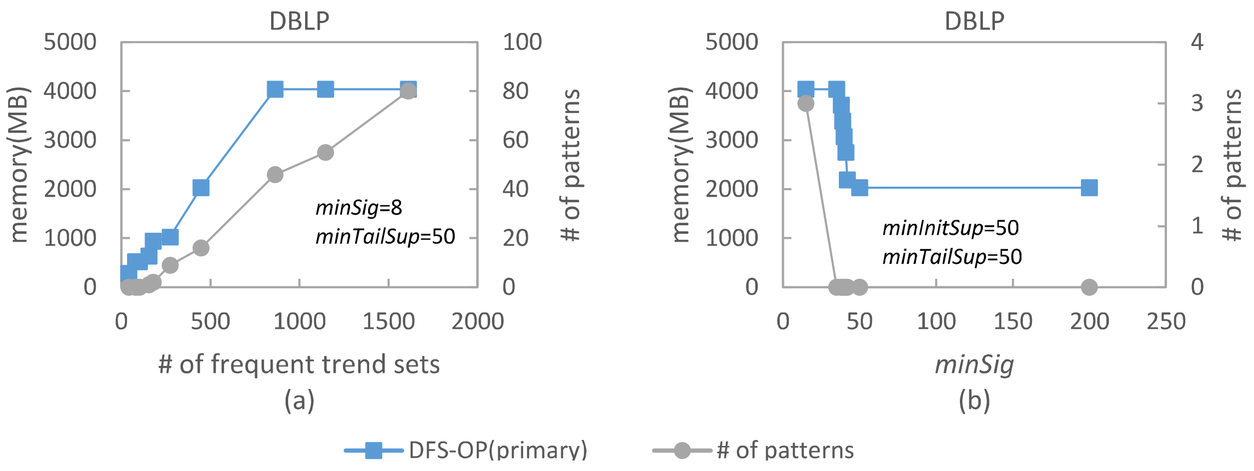

Figure 13: Memory usage with respect to (a) number of frequent trend sets and (b) *minSig*.

the DBLP and US Flight datasets for different number of repeats. Increasing the number of vertices and average number of edges increase the support of each trend set in the whole space and each neighboring space, respectively. As a result, the intersection operation becomes more costly. Therefore, it was decided to investigate how the number of supporting points influence the final time cost. By repeating each dataset k times, the number of supporting point is multiplied by k. The first observation on the DBLP dataset is that runtime linearly increases as the number of supporting points is increased. The second observation is that the ratio of the runtime of TSeqMiner$_{dfs-dfs}$ or TSeqMiner$_{dfs-bfs}$ over the baseline remains stable. This is because the number of intersection operations and cases where pruning succeed remains approximately the same wen datasets are repeated. Result for the US Flight dataset are similar.

5.1.5. Influence of minInitSup and minSig on memory consumption

In a sixth experiment, the influence of *minInitSup* and *minSig* on memory consumption was assessed. Fig. 13 (a) and (b) indicate the memory usage when the number of frequent trend sets (determined by *minInitSup*) and *minSig* are varied, respectively. Because memory usage is not much influenced by the type of search space traversal, results are only shown for TSeqMiner$_{dfs-dfs}$. It is observed In Fig. 13(a) that increasing the number of frequent trend sets M does not always increase memory usage but can have a considerable influence on memory consumption. Moreover, this increase does not exactly follow the increase of the number of patterns found. Besides, it is observed in Fig. 13(b) that memory usage quickly increases in some cases for small variations of *minSig*, and that this increase also does not follows the number of patterns found.

5.2. Pattern analysis

An analysis of patterns found was also performed on the DBLP and US Flight dataset to assess whether interesting patterns are found by the proposed algorithms. As mentioned, the neighborhood definition of the proposed algorithms can be parameterized for the needs of various applications. In the following, the previously discussed neighborhood definition is called $\mathcal{NH}1$. Besides, another definition called $\mathcal{NH}2$ is considered, where each vertex is its neighboor at the following timestamp.

5.2.1. DBLP

Patterns were first extracted from the DBLP dataset. The $\mathcal{NH}1$ function was used, and the parameters were set to $minInitSup = 35$, $minTailSup = 100$ and $minSig = 10$ to obtain a set of interesting patterns. For these values, TSeqMiner$_{dfs-bfs}$ generates 1,610 frequent trend sets and returns 3 patterns in 45 s. These patterns are show in Table 5. The first pattern is of size 2. It indicates that if the number of publications of an author in $ICDE$, $PVLDB$ and $ACMTransDBSys$ is increasing, it is more likely that publications of a co-author will increase in $PVLDB$ while decreasing in $VLDB$ for the next timestamp. In this pattern, the first trend set is supported by 45 authors in the whole space and the second trend set is supported by 107 authors in the neighboring space of the first trend set. In this case, the significance is 11.5, which indicate a strong correlation. The second pattern is similar to the first one. The third pattern indicates that no significant changes between the number of publications in $JMLR$ between two timestamps ($JMLR =$), is correlated with a decrease in the number of publications in $JMLR$ for a co-author at the following timestamp.

Patterns	# of supporting points	significance
$\{(ICDE+, PVLDB+, ACMTransDBSys+), (PVLDB+, VLDB-)\}$	$\{45, 107\}$	$\{11.5\}$
$\{(PVLDB+, VLDB+), (PVLDB+, VLDB-)\}$	$\{57, 120\}$	$\{11.5\}$
$\{(JMLR =), (JMLR-)\}$	$\{283, 147\}$	$\{10.3\}$

Table 5: Three significant trend sequences with their support and significance, mined from the DBLP dataset using the $\mathcal{NH}1$ neighborhood definition.

An analysis of patterns found was also done using the $\mathcal{NH}2$ neighborhood definition. Using this definition, mined patterns indicate how an author's publication influence his/her future publications. The parameter *minTailSup* was set to 200 to obtain interesting patterns. After filtering some trivial patterns such as $\{(a+), (a=)\}$, $\{(a=), (a-)\}$, interesting patterns have been identified, listed in Table 6. The pattern

$\{(VLDB+), (ICDE+, VLDB =)\}$ indicates that an author publishing an increasing number of papers in $VLDB$ is likely to publish more $ICDE$ papers while having a stable number of publications in $VLDB$ at the next timestamp. This pattern is reasonable since ICDE and VLDB are both top conferences in the field of database. If it is found that an author has an increasing number of publications in VLDB, it means that the author has a good knowledge of databases and can write excellent papers. It is thus reasonable that he/she will be more likely to have more publications in another top conference such as ICDE. The number of supporting points is also quite large (> 200) meaning that this pattern that is relevant to a large group of researchers, while the high significance indicates that there is a strong correlation and this pattern do not likely appears by chance.

Patterns	# of supporting points	significance
$\{(VLDB+), (ICDE+, VLDB =)\}$	$\{1364, 247\}$	$\{10.5\}$
$\{(KDD+), (KDD =), (KDD-)\}$	$\{1334, 590, 208\}$	$\{10.9, 12.4\}$
$\{(BioInfo+), (BioInfo+, BMCBio+)\}$	$\{1255, 241\}$	$\{10.1\}$

Table 6: Partial significant trend sequences with their support and significance, mined from the DBLP dataset using the $\mathcal{NH}2$ neighborhood definition.

Furthermore, note that if $\mathcal{NH}2$ is used and topological attributes are considered such as *closeness* and *numCliques* to filter patterns, the proposed algorithms will find all triggering patterns suggesting how variations of attribute values of a vertex can influence its topological properties [31]. More complicated definitions of neighborhood can also be used for other needs.

5.2.2. US Flight

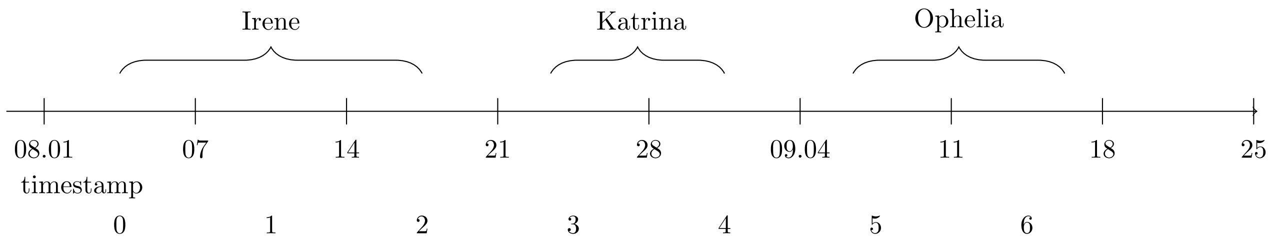

Figure 14: Hurricane dates and corresponding timestamps of the US Flight dataset.

Patterns from the US Flight dataset were also extracted and analyzed. This dataset is about the impact of hurricanes on flight traffic in the United States. To facilitate the discussion, Fig. 14 illustrates the dates of the Irene, Katrina and Ophelia hurricanes, and the corresponding timestamps of the US Flight dataset.

The $\mathcal{NH}2$ neighborhood definition was first considered, which focuses on the evolution of each single vertex. Parameters were set to $minInitSup$ = 8, $minTailSup$ = 10 and $minSig$ = 70 to find an interesting set of patterns. The TSeqMiner$_{dfs-dfs}$ algorithm returned 512 patterns in 2 s. One of them is $\{(NbCancel++, NbDivert++, DelayDepart-), (NbDepart-, NbArriv-, NbCancel--, NbDivert--)\}$, which has a significance of 72.6. The first trend set indicates that a large increase in the number of canceled flights ($NbCancel++$) and diverted flights ($NbDivert$++) was observed, as well as a moderate decrease in the number of delayed departures ($DelayDepart-$) due to an hurricane. The appearance of $DelayDepart$ is explained by the fact that the number of flights is reduced, and thus remaining flights are less likely to be delayed. The second trend set indicates a large decrease in the number of canceled flights ($NbCancel--$) and diverted flights ($NbDivert--$), with a moderate decrease in the number of arrivals ($NbArriv-$) and departures ($NbDepart-$). This second trend set indicates that an airport's activity has returned to normal after recovery from hurricane damage. The presence of $NbArriv-$ and $NbDepart-$ may appear contradictory to this conclusion, but it can be explained by the fact that during the recovery from the previous

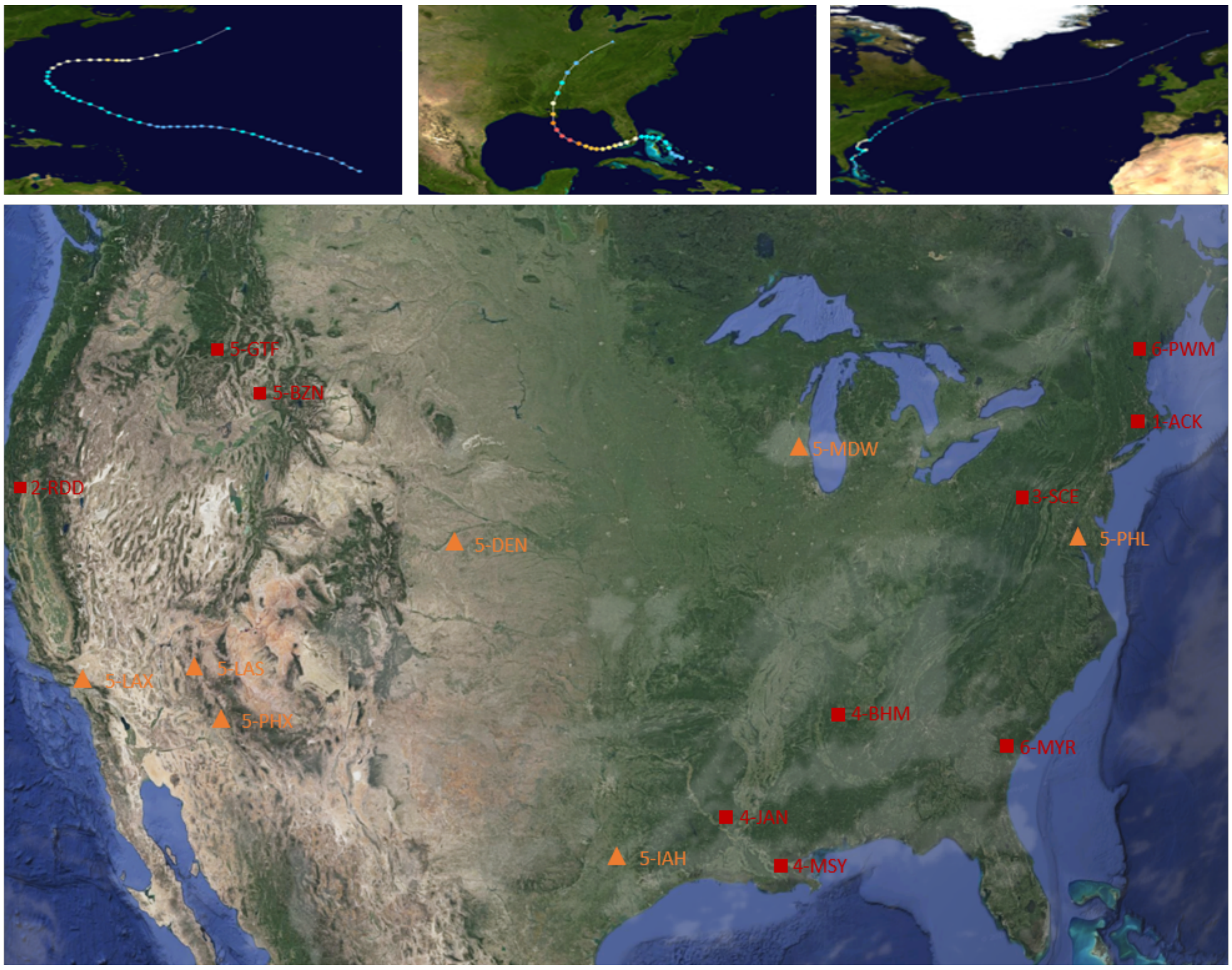

Figure 15: The paths of the Irene (top left), Katrina (top center) and Ophelia (top right) hurricanes. The partial supporting points (bottom) of the pattern $\{(NbCancel--, NbDivert--, DelayDepart-), (NbDepart-, NbCancel-, NbDivert-, DelayDepart-, DelayArriv+)\}$, mined using $\mathcal{NH}1$.

hurricane, part of the flights are influenced by another hurricane. That becomes clearer when we have a look at supporting points of the trend sets.

Timestamp	location
0	"WRG-Wrangell AK" "KTN-Ketchikan AK"
1	"OME-Nome AK"
3	"JAN-Jackson/Vicksburg MS" "PSG-Petersburg AK" "BHM-Birmingham AL" "MSY-New Orleans LA" "JNU-Juneau AK"
4	"GTF-Great Falls MT"
5	"MYR-Myrtle Beach SC"
6	"MYR-Myrtle Beach SC" "DAY-Dayton OH" "STL-St. Louis MO" "EYW-Key West FL" "GJT-Grand Junction CO" "GRR-Grand Rapids MI" "IND-Indianapolis IN" "SNA-Santa Ana CA"

Table 7: Supporting points of the first trend set of significant trend sequence $\{(NbCancel++, NbDivert++, DelayDepart-), (NbDepart-, NbArriv-, NbCancel--, NbDivert--)\}$.

The supporting points of the first trend set are shown in Table 7. In that table, it is observed that abnormal events of timestamps 3 and 4 are caused by the Katrina hurricane, while those of timestamp 5 and 6 are due to hurricane Ophelia. Dates of these two hurricanes are close. Therefore, some airports recovering from Katrina start to be affected by Ophelia. Moreover, affected airports are mostly located on the East coast of the United States, which is consistent with these hurricanes' paths depicted in the top pictures of Fig. 15.

The $\mathcal{NH}1$ neighborhood definition was also considered. The parameters were set to $minInitSup = 8$, $minTailSup = 7$ and $minSig = 6$ to find interesting patterns. The TSeqMiner$_{dfs-dfs}$ algorithm returned 768 patterns in 9 s. An interesting pattern is $\{(NbCancel--, NbDivert--, DelayDepart-), (NbDepart-, NbCancel-, NbDivert-, DelayDepart-, DelayArriv+)\}$, whose supporting points are partially illustrated in Fig. 15. In Fig. 15, red rectangles denote supporting points of the first trend set and orange triangles denote supporting points of the second trend set. The first trend set contains $NbCancel--$ and $NbDivert--$, which indicate a significant decrease of the number of canceled flights and diverted flights, following recovery from an hurricane. Most supporting points are located on the East coast which indicates that it represents aiports that were directly affected by the hurricane. The second trend sets contains $NbCancel-$ and $NbDivert-$, which indicate a moderate recovery from damage. The supporting points of the second trend set are all at timestamp 5 and mostly not located on the East coast. Thus, these points represent airports that were indirectly influenced by the hurricane or have experienced moderate damage. The airports that were directly influenced by the hurricane recovered earlier at time 4, and then were followed by the moderate recovery of related airports at timestamp 5. The $DelayArriv+$ trend indicates an increase of the average delay time of arriving flights, which indicates that flight schedules may still not have completely returned to normal.

On overall, analysis of patterns have shown that interesting patterns can be discovered in two real-life dynamic attributed graphs using the proposed model of significant trend sequences. It was also shown that using different neighborhood definition can provide different insights about the data.

Generally, dynamic graphs are found in many domains. Thus, the proposed algorithms could be applied in many other fields. For example, dynamic graphs can be used to model changes in ontologies [20], processes in complex systems [21], computer networks [22], spatio-temporal changes observed in satellite images [23] and movements of vehicles [24]. Thus, the proposed algorithms could be also applied on such data to find patterns. Studying patterns found for other applications besides the two presented in this paper will be considered in future work.

6. Conclusion

To allows discovering strongly correlated patterns in dynamic attributed graphs, this paper proposed a novel significance measure named *Sequence Virtual Growth Rate*. It allows evaluating if a pattern represents entities that are correlated in terms of their proximity in a graph over time. Based on this measure a novel type of graph patterns called *Significant Trend Sequence* was proposed. To efficiently mine these patterns, two algorithms named TSeqMiner$_{dfs-bfs}$ and TSeqMiner$_{dfs-dfs}$ were proposed. They rely on a novel upper bound and pruning strategy to reduce the search space. An externsive experimental evaluation has show that the algorithms are efficient and can identify interesting patterns in real-world social network and flight data.

There are many possibilities for future work. First, we have introduced a significance measure to find strongly correlated trend sequence. But this measure could also be adapted to find more general types of patterns such as significant subgraph sequences. Second, alternative constraints could be designed to select meaningful patterns and to provide properties for pruning. Third, the proposed algorithms may still generate many similar patterns. Finding a good way to group them can save user much time and increase the applicability of the proposed algorithms. Fourth, the concept of high utility patterns [61, 62, 63, 64, 65, 66] may be considered for dynamic attributed graph mining.

References

[1] C. C. Aggarwal, H. Wang (Eds.), Managing and Mining Graph Data, Vol. 40 of Advances in Database Systems, Springer, 2010. doi:10.1007/978-1-4419-6045-0.
URL https://doi.org/10.1007/978-1-4419-6045-0

[2] Y.-T. Wen, Y. Y. Fan, W.-C. Peng, Mining of location-based social networks for spatio-temporal social influence, in: Pacific-Asia Conference on Knowledge Discovery and Data Mining, Springer, 2017, pp. 799–810.

[3] T. Lv, H. Gao, X. Li, S. Yang, L. Hanzo, Space-time hierarchical-graph based cooperative localization in wireless sensor networks, IEEE Transactions on Signal Processing 64 (2) (2016) 322–334. doi:10.1109/TSP.2015.2480038.

[4] L. B. Holder, D. J. Cook, et al., Learning patterns in the dynamics of biological networks, in: Proceedings of the 15th ACM SIGKDD international conference on Knowledge discovery and data mining, ACM, 2009, pp. 977–986.

[5] F. Fassetti, S. E. Rombo, C. Serrao, Discovering discriminative graph patterns from gene expression data, in: Proceedings of the 31st Annual ACM Symposium on Applied Computing, SAC '16, ACM, New York, NY, USA, 2016, pp. 23–30. doi:10.1145/2851613.2851617.
URL http://doi.acm.org/10.1145/2851613.2851617

[6] Y. Ding, L. Huang, C.-D. Wang, D. Huang, Community detection in graph streams by pruning zombie nodes, in: Pacific-Asia Conference on Knowledge Discovery and Data Mining, Springer, 2017, pp. 574–585.

[7] L. Rashidi, A. Kan, J. Bailey, J. Chan, C. Leckie, W. Liu, S. Rajasegarar, K. Ramamohanarao, Node re-ordering as a means of anomaly detection in time-evolving graphs, in: Joint European Conference on Machine Learning and Knowledge Discovery in Databases, Springer, 2016, pp. 162–178.

[8] S. Ranu, M. Hoang, A. Singh, Mining discriminative subgraphs from global-state networks, in: Proceedings of the 19th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, KDD '13, ACM, New York, NY, USA, 2013, pp. 509–517. doi:10.1145/2487575.2487692.
URL http://doi.acm.org/10.1145/2487575.2487692

[9] Y. Yang, D. Yan, H. Wu, J. Cheng, S. Zhou, J. Lui, Diversified temporal subgraph pattern mining, in: Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, ACM, 2016, pp. 1965–1974.

[10] P. Perner, Mining frequent subgraph pattern over a collection of attributed-graphs and construction of a relation hierarchy for result reporting, in: Industrial Conference on Data Mining, Springer, 2017, pp. 323–344.

[11] D. J. Cook, L. B. Holder, Substructure discovery using minimum description length and background knowledge, Journal of Artificial Intelligence Research 1 (1993) 231–255.

[12] M. Kuramochi, G. Karypis, Finding frequent patterns in a large sparse graph, Data mining and knowledge discovery 11 (3) (2005) 243–271.

[13] X. Yan, J. Han, gspan: graph-based substructure pattern mining, in: 2002 IEEE International Conference on Data Mining, 2002. Proceedings., 2002, pp. 721–724. doi:10.1109/ICDM.2002.1184038.

[14] A. Inokuchi, T. Washio, H. Motoda, Complete mining of frequent patterns from graphs: Mining graph data, Machine Learning 50 (3) (2003) 321–354.

[15] K. M. Borgwardt, H. p. Kriegel, P. Wackersreuther, Pattern mining in frequent dynamic subgraphs, in: Sixth International Conference on Data Mining (ICDM'06), 2006, pp. 818–822. doi:10.1109/ICDM.2006.124.

[16] A. Inokuchi, T. Washio, A fast method to mine frequent subsequences from graph sequence data, in: 2008 Eighth IEEE International Conference on Data Mining, 2008, pp. 303–312. doi:10.1109/ICDM.2008.106.

[17] R. Ahmed, G. Karypis, Algorithms for mining the coevolving relational motifs in dynamic networks, ACM Transactions on Knowledge Discovery from Data (TKDD) 10 (1) (2015) 4.

[18] L. Meng, Y. Hulovatyy, A. Striegel, T. Milenković, On the interplay between individuals' evolving interaction patterns and traits in dynamic multiplex social networks, IEEE Transactions on Network Science and Engineering 3 (1) (2016) 32–43.

[19] S. Halder, M. Samiullah, Y.-K. Lee, Supergraph based periodic pattern mining in dynamic social networks, Expert Systems with Applications 72 (2017) 430–442.

[20] M. Burch, S. Lohmann, Visualizing the evolution of ontologies: A dynamic graph perspective., in: VOILA@ ISWC, 2015, p. 69.

[21] H. Zhao, Dynamic graph embedding for fault detection, Computers &amp; Chemical Engineering 117 (2018) 359–371.

[22] D. Meng, W. Niu, X. Ding, L. Zhao, Network-to-network control over heterogeneous topologies: a dynamic graph approach, IEEE Transactions on Systems, Man, and Cybernetics: Systems (99) (2018) 1–12.

[23] E. Desmier, M. Plantevit, C. Robardet, J.-F. Boulicaut, Trend mining in dynamic attributed graphs, in: Proceedings of the European Conference on Machine Learning and Knowledge Discovery in Databases - Volume 8188, ECML PKDD 2013, Springer-Verlag New York, Inc., New York, NY, USA, 2013, pp. 654–669. doi:10.1007/978-3-642-40988-2_42. URL http://dx.doi.org/10.1007/978-3-642-40988-2_42

[24] Y. Chen, Y. Guo, Y. Wang, Modeling and density estimation of an urban freeway network based on dynamic graph hybrid automata, Sensors 17 (4) (2017) 716.

[25] P.-N. Mougel, C. Rigotti, O. Gandrillon, Finding collections of k-clique percolated components in attributed graphs, in: Pacific-Asia Conference on Knowledge Discovery and Data Mining, Springer, 2012, pp. 181–192.

[26] C. Pasquier, J. Sanhes, F. Flouvat, N. Selmaoui-Folcher, Frequent pattern mining in attributed trees: algorithms and applications, Knowledge and Information Systems 46 (3) (2016) 491–514.

[27] C. Pasquier, F. Flouvat, J. Sanhes, N. Selmaoui-Folcher, Attributed graph mining in the presence of automorphism, Knowledge and Information Systems 50 (2) (2017) 569–584.

[28] E. Desmier, M. Plantevit, C. Robardet, J.-F. Boulicaut, Cohesive co-evolution patterns in dynamic attributed graphs, in: J.-G. Ganascia, P. Lenca, J.-M. Petit (Eds.), Discovery Science, Springer Berlin Heidelberg, Berlin, Heidelberg, 2012, pp. 110–124.

[29] Z. Cheng, F. Flouvat, N. Selmaoui-Folcher, Mining recurrent patterns in a dynamic attributed graph, in: Pacific-Asia Conference on Knowledge Discovery and Data Mining, Springer, 2017, pp. 631–643.

[30] Z. Cheng, Ming recurrent patterns in a dynamic attributed graph, Ph.D. thesis, University of New Caledonia (2018).

[31] M. Kaytoue, Y. Pitarch, M. Plantevit, C. Robardet, Triggering patterns of topology changes in dynamic graphs, in: 2014 IEEE/ACM International Conference on Advances in Social Networks Analysis and Mining (ASONAM 2014), 2014, pp. 158–165. doi:10.1109/ASONAM.2014.6921577.

[32] P. Fournier-Viger, J. C.-W. Lin, T. Dinh, H. B. Le, Mining correlated high-utility itemsets using the bond measure, in: F. Martínez-Álvarez, A. Troncoso, H. Quintián, E. Corchado (Eds.), Hybrid Artificial Intelligent Systems, Springer International Publishing, Cham, 2016, pp. 53–65.

[33] P. Fournier-Viger, J. C.-W. Lin, U. R. Kiran, Y.-S. Koh, A survey of sequential pattern mining, Data Science and Pattern Recognition 1 (1) (2017) 54–77.

[34] Y. Huang, L. Zhang, P. Zhang, A framework for mining sequential patterns from spatio-temporal event data sets, IEEE Transactions on Knowledge & Data Engineering (4) (2007) 433–448.

[35] G. Dong, J. Li, Efficient mining of emerging patterns: Discovering trends and differences, in: Proceedings of the fifth ACM SIGKDD international conference on Knowledge discovery and data mining, ACM, 1999, pp. 43–52.

[36] P. K. Novak, N. Lavrač, G. I. Webb, Supervised descriptive rule discovery: A unifying survey of contrast set, emerging pattern and subgroup mining, Journal of Machine Learning Research 10 (Feb) (2009) 377–403.

[37] R. Ahmed, G. Karypis, Algorithms for mining the evolution of conserved relational states in dynamic networks, Knowledge and Information Systems 33 (3) (2012) 603–630.

[38] C. Zhang, C. Liu, X. Zhang, G. Almpanidis, An up-to-date comparison of state-of-the-art classification algorithms, Expert Syst. Appl. 82 (2017) 128–150. doi:10.1016/j.eswa.2017.04.003.

[39] R. Jin, S. McCallen, E. Almaas, Trend motif: A graph mining approach for analysis of dynamic complex networks, in: Data Mining, 2007. ICDM 2007. Seventh IEEE International Conference on, IEEE, 2007, pp. 541–546.

[40] L. Li, S. Erfani, C. Leckie, A pattern tree based method for mining conditional contrast patterns of multi-source data, in: Data Mining Workshops (ICDMW), 2017 IEEE International Conference on, IEEE, 2017, pp. 916–923.

[41] Z. Zheng, W. Wei, C. Liu, W. Cao, L. Cao, M. Bhatia, An effective contrast sequential pattern mining approach to taxpayer behavior analysis, World Wide Web 19 (4) (2016) 633–651.

[42] T. Gu, L. Wang, Z. Wu, X. Tao, J. Lu, A pattern mining approach to sensor-based human activity recognition, IEEE Transactions on Knowledge and Data Engineering 23 (9) (2011) 1359–1372.

[43] A. Prado, M. Plantevit, C. Robardet, J.-F. Boulicaut, Mining graph topological patterns: Finding covariations among vertex descriptors, IEEE Transactions on Knowledge and Data Engineering 25 (9) (2013) 2090–2104.

[44] M. Celik, Partial spatio-temporal co-occurrence pattern mining, Knowledge and Information Systems 44 (1) (2015) 27–49.

[45] K. G. Pillai, R. A. Angryk, J. M. Banda, M. A. Schuh, T. Wylie, Spatio-temporal co-occurrence pattern mining in data sets with evolving regions, in: Data Mining Workshops (ICDMW), 2012 IEEE 12th International Conference on, IEEE, 2012, pp. 805–812.

[46] K. G. Pillai, R. A. Angryk, J. M. Banda, T. Wylie, M. A. Schuh, Spatiotemporal co-occurrence rules, in: New Trends in Databases and Information Systems, Springer, 2014, pp. 27–35.

[47] J. Sanhes, F. Flouvat, N. Selmaoui-Folcher, C. Pasquier, J.-F. Boulicaut, Weighted path as a condensed pattern in a single attributed dag, in: 23rd International Joint Conference on Artificial Intelligence (IJCAI'13), 2013.

[48] P. Mohan, S. Shekhar, J. A. Shine, J. P. Rogers, Cascading spatio-temporal pattern discovery, IEEE Transactions on Knowledge and Data Engineering 24 (11) (2012) 1977–1992.

[49] F. Beck, M. Burch, S. Diehl, D. Weiskopf, A taxonomy and survey of dynamic graph visualization, in: Computer Graphics Forum, Vol. 36, Wiley Online Library, 2017, pp. 133–159.

[50] A. E. Sizemore, D. S. Bassett, Dynamic graph metrics: Tutorial, toolbox, and tale, NeuroImage 180 (2018) 417–427.

[51] M. Ye, A. J. Ma, L. Zheng, J. Li, P. C. Yuen, Dynamic label graph matching for unsupervised video re-identification, in: Proc. of the 16th IEEE International Conference on Computer Vision, 2017, pp. 5142–5150.

[52] N. Shah, D. Koutra, T. Zou, B. Gallagher, C. Faloutsos, Timecrunch: Interpretable dynamic graph summarization, in: Proc. of the 21th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, ACM, 2015, pp. 1055–1064.

[53] K. Iwabuchi, S. Sallinen, R. Pearce, B. Van Essen, M. Gokhale, S. Matsuoka, Towards a distributed large-scale dynamic graph data store, in: 2016 IEEE International Parallel and Distributed Processing Symposium Workshops (IPDPSW), IEEE, 2016, pp. 892–901.

[54] L. Bulteau, V. Froese, K. Kutzkov, R. Pagh, Triangle counting in dynamic graph streams, Algorithmica 76 (1) (2016) 259–278.

[55] M. T. Son, S. Amer-Yahia, I. Assent, M. Birk, M. S. Dieu, J. Jacobsen, J. Kristensen, Scalable interactive dynamic graph clustering on multicore cpus, IEEE Transactions on Knowledge and Data Engineering.

[56] S. Fernandes, H. Fanaee-T, J. Gama, Dynamic graph summarization: a tensor decomposition approach, Data Mining and Knowledge Discovery 32 (5) (2018) 1397–1420.

[57] B. A. Miller, N. Arcolano, S. Kelley, N. T. Bliss, Analytical models and methods for anomaly detection in dynamic, attributed graphs, Computational Network Analysis with R: Applications in Biology, Medicine and Chemistry 7.

[58] P. Goyal, S. R. Chhetri, N. Mehrabi, E. Ferrara, A. Canedo, Dynamicgem: A library for dynamic graph embedding methods, arXiv preprint arXiv:1811.10734.

[59] P. Fournier-Viger, J. C. Lin, B. Vo, T. C. Truong, J. Zhang, H. B. Le, A survey of itemset mining, Wiley Interdiscip. Rev. Data Min. Knowl. Discov. 7 (4). `doi:10.1002/widm.1207`.
URL `https://doi.org/10.1002/widm.1207`

[60] M. J. Zaki, Scalable algorithms for association mining, IEEE transactions on knowledge and data engineering 12 (3) (2000) 372–390.

[61] P. Fournier-Viger, J. C.-W. Lin, T. Truong-Chi, R. Nkambou, A survey of high utility itemset mining, in: High-Utility Pattern Mining, Springer, 2019, pp. 1–45.

[62] T. Truong-Chi, P. Fournier-Viger, A survey of high utility sequential pattern mining, in: High-Utility Pattern Mining, Springer, 2019, pp. 97–129.

[63] L. T. Nguyen, V. V. Vu, M. T. Lam, T. T. Duong, L. T. Manh, T. T. Nguyen, B. Vo, H. Fujita, An efficient method for mining high utility closed itemsets, Information Sciences.

[64] W. Gan, J. C.-W. Lin, P. Fournier-Viger, H.-C. Chao, H. Fujita, Extracting non-redundant correlated purchase behaviors by utility measure, Knowledge-Based Systems 143 (2018) 30–41.

[65] P. Fournier-Viger, Y. Zhang, J. C.-W. Lin, H. Fujita, Y. S. Koh, Mining local and peak high utility itemsets, Information Sciences 481 (2019) 344–367.

[66] U. Yun, D. Kim, E. Yoon, H. Fujita, Damped window based high average utility pattern mining over data streams, Knowledge-Based Systems 144 (2018) 188–205.