A Survey of Pattern Mining in Dynamic Graphs

Philippe Fournier-Viger [*], Ganghuan He[†], Chao Cheng[‡],
Jiaxuan Li[§], Jerry Chun-Wei Lin[¶], Unil Yun[‖]

Article Type:

Advanced Review

Abstract

Graph data is found in numerous domains, including the analysis of social networks, sensor networks, bioinformatics, industrial systems, and chemistry. Analyzing graphs to find useful and interesting patterns is an important area of research. It can help to understand graphs, and hence support decision making. Since two decades, many graph mining techniques have been proposed to identify patterns such as frequent subgraphs, paths, cliques and trees. But many techniques assume that graphs are static. This simplifying assumption makes it easy to design algorithms but discard information about how graphs evolve. This paper provides a detailed survey of techniques for mining interesting patterns in dynamic graphs, which can serve both as an introduction and as a guide to recent advances and opportunities in this research area. The main tasks related to mining patterns in dynamic graphs are reviewed such as discovering frequent subgraphs, evolution rules, motifs, subgraph sequences, recurrent patterns, triggering patterns and trend sequences. In addition, an overview of strategies and approaches to solve dynamic graph mining problems are presented, and their advantages and limitations are highlighted. Various extensions are also discussed such as to discover patterns in data stream and big data. Lastly, the article mentions several research opportunities.

[*]School of Natural Science and Humanities, Harbin Institute of Technology (Shenzhen), China
[†]School of Computer Science, Harbin Institute of Technology (Shenzhen), China
[‡]School of Computer Science, Harbin Institute of Technology (Shenzhen), China
[§]School of Computer Science, Harbin Institute of Technology (Shenzhen), China
[¶]Department of Computing, Mathematics and Physics, Western Norway Universityof Applied Sciences (HVL), Bergen, Norway
[‖]Department of Computer Engineering, Sejong University, Seoul, Republic of Korea

INTRODUCTION

Pattern mining is a key research area in data mining, which consists of applying algorithms to identify interesting patterns appearing in the data. Generally, a pattern can be considered interesting if it reveals some novel information that is useful to understand the past or predict the future. Over the years, techniques have been designed to extract patterns from several types of data such as transactional data[7,8], time series[12–14], business process logs[9], trajectories[10], spatial data[15], sequences[1,5,6,11] and graphs[2–4]. Depending on the applications, what is an interesting patterns may differ. Thus, algorithms have been proposed to extract patterns based on various criteria such as patterns having high occurrence frequency and confidence[8,16], rare patterns[17,18], and profitable patterns[19]. Pattern mining tasks can be very challenging because the goal is to identify interesting patterns from a potentially huge number of possible patterns. It is thus important to design algorithms that can find the desired patterns without considering all possible patterns. For this purpose, efficient algorithms have been designed based on efficient data structures and search space pruning strategies[1,8].

Among the various types of data studied in pattern mining, graph is one of the most important as graph structures are found in numerous domains such as social networks[72], chemistry[22], vehicular networks[35], computer networks[20], bioinformatics[31], XML data[31], and geographical data[79]. Algorithms have been proposed to find various types of patterns in graphs such as subgraphs[2,22,24,25], trees[31,32] and traversal paths[45–47]. Moreover, algorithms have been proposed to find patterns in various types of graphs such as weighted graphs[45], directed graphs[23,32], attributed graphs[39,66,74,76,78,79], and graph databases[2,3]. A type of graphs that has attracted the interest of many researchers is dynamic graphs. A dynamic graph is a graph that changes over time in terms of its attributes or structures (edges, vertices). Considering the time dimension in graph mining allows to understand how graphs evolve and is key to many applications such as social network analysis. However, mining patterns in dynamic graph is also more challenging.

Graph mining is a very active research field. Although some surveys have been published on discovering patterns in graphs[3], there is none on mining patterns in dynamic graphs. This paper addresses this issue by providing an up-to-date and detailed survey that provides

not only an introduction to the field but also reviews recent advances and opportunities.

It is to be noted that this survey focuses on analyzing graphs to find *patterns*, i.e. interesting sets of value that are appearing several times in data, are correlated or met some other interestingness criteria set by the user. Other useful graph analysis tasks are considered to be outside the scope of this survey such as identifying prestigious nodes[41], detecting communities[41,42], calculating descriptive measures[43], clustering nodes[43], graph summarization[44], automatic node labeling (relational classification)[40], and graph visualization[43].

This survey is organized as follows. It first presents important concepts and challenges related to mining patterns in static graphs. Then, the following sections discusses techniques for discovering patterns in a single dynamic graph, common to several dynamic graphs (a graph database), and in attributed graphs. In these sections, an overview of techniques employed for discovering different kinds of patterns is presented. Then, the paper discusses research opportunities. Finally, a conclusion is drawn.

MINING PATTERNS IN STATIC GRAPH(S)

Before discussing techniques for mining patterns in dynamic graphs, this paper presents a brief overview of techniques for mining patterns in static graphs, as they are related. Then, the next section reviews techniques for mining patterns in dynamic graphs.

PRELIMININARY DEFINITIONS

In its most simple form, a *graph* is a tuple $G = (V, E)$ such that V is a vertex set, and $E \subseteq V \times V$ is an edge set. Some common vocabulary to described graphs is presented below. A graph is said to be *directed* if edges have directions. A graph is *connected* if by following the edges, it is possible to go from any vertex to any other vertices. A graph is *weighted* if weights are assigned to edges and/or vertices. A graph is said to be *simple* if it is not weighted, undirected, has no self-loop (an edge connecting a node to itself) and has no more than one edge between any pair of vertices. A *labeled graph* is a tuple $G = (V, E, L_V, L_E, \phi_V, \phi_E)$ where V is an edge set, $E \subseteq V \times V$ is an edge set, L_V is a set of vertex labels, L_E is a set of edge labels, ϕ_V is a function mapping vertex to labels (ϕ_V:$V \rightarrow L_V$), and ϕ_E is a function

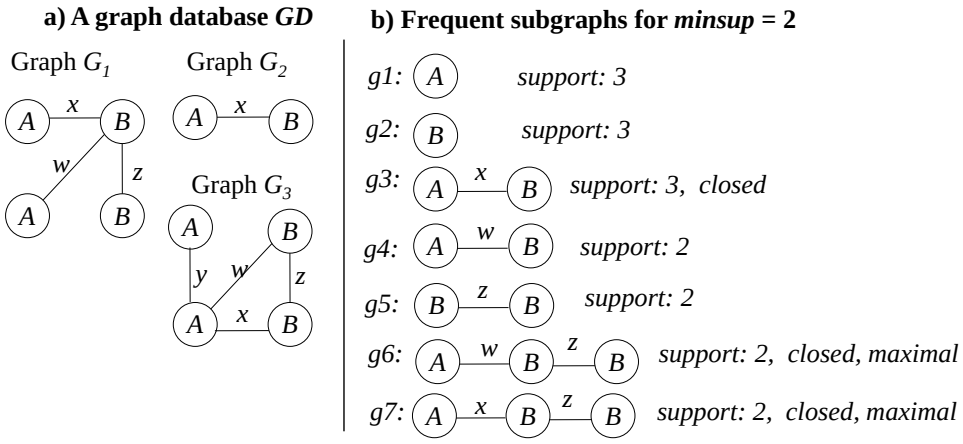

Figure 1: Different types of static graphs

mapping edges to labels (ϕ_E:$E \rightarrow L_E$). An *attributed graph* is a tuple $G = (V, A, E, \lambda)$ where V is a vertex set, A is an attribute set, $E \subseteq V \times V$ is an edge set, and $\lambda : V \times A \rightarrow \mathbb{R}$ is a function, which maps a number to each vertex-attribute pair. Fig. 1 shows examples of these different types of graphs.

MINING PATTERNS IN A STATIC GRAPH DATABASE

Several tasks have been proposed to find patterns in a database of static graphs.

Frequent subgraph mining. It is one of the most popular graph mining task. It aims at finding all subgraphs that appear frequently in a database of simple connected graphs [2–4,20]. Given a parameter called the minimum support threshold (*minsup*), a graph is frequent if it appears in no less than *minsup* input graphs. The assumption of frequent subgraph mining is that a subgraph is interesting if it appears many times in a set of graphs. For example, this task can be useful to find an association between elements that is common to several chemical molecules.

Formally, frequent subgraph mining is defined as follows. Let there be a *graph database* $GD = \{G_1, G_2 \ldots G_n\}$ consisting of n simple labeled graphs. Consider two labeled graphs $G_x = (V_x, E_x, L_{Vx}, L_{Ex}, \phi_{Vx}, \phi_{Ex})$ and $G_y = (V_y, E_y, L_{Vy}, L_{Ey}, \phi_{Vy}, \phi_{Ey})$. The graph G_x is said to be *isomorphic to* G_y if and only if there exists a bijective function $f : V_x \rightarrow V_y$ such that (1) $\forall v \in V_x$, $L_{Vx}(v) = L_{Vy}(f(v))$ and (2) $\forall (u, v) \in E_x$, $(f(u), f(v)) \in E_y$ and $L_{Ex}(u, v) = L_{Ey}(f(u), f(v))$. A graph G_x is said be a *subgraph isomorphism* of (to *appear*

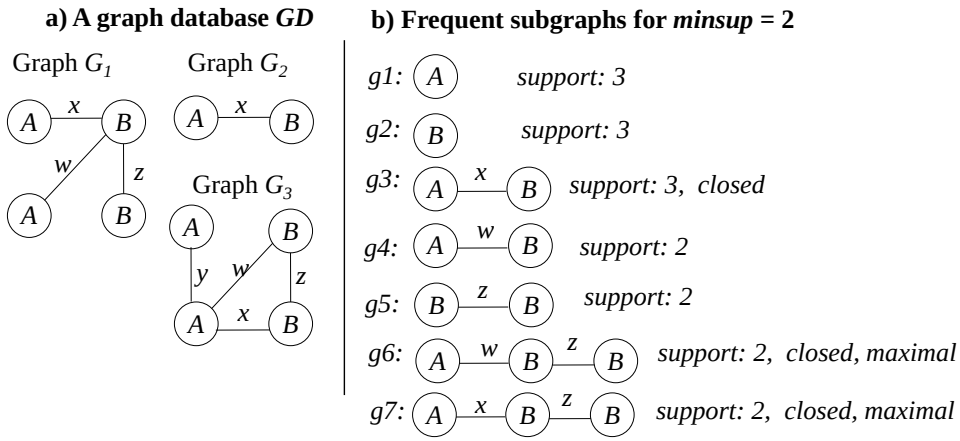

Figure 2: A graph database and frequent subgraphs found for $minsup = 2$

in) a graph G_z, denoted as $G_x \sqsubseteq G_z$, if there exists a subgraph $G_y \subseteq G_z$ such that G_x is isomorphic to G_y. A subgraph isomorphism is also called an *embedding*. The *support* of a graph G_x in a graph database GD is defined as $sup(G_x) = |\{g | g \in GD \wedge G_x \sqsubseteq g\}|$. Given a graph database GD and a $minsup$ threshold ($minsup > 0$), the task of frequent subgraph mining is to enumerate all frequent subgraphs (graphs having a support no less than $minsup$).

For example, consider the graph database of Fig. 2 a) containing three graphs, G_1, G_2 and G_3. If $minsup = 2$, seven frequent subgraphs are found, represented on Fig. 2 b) with their support. For example, the subgraph g_6 has a support of 2 because it appears in G_1 and G_3, while subgraph $g1$ has a support of 3 because it appears in G_1, G_2 and G_3. It is to be noted that the support calculation ignores the fact that a graph may have multiple occurrences in an input graph (e.g. $g1$ appears multiple times in G_1).

The problem of frequent subgraph mining is difficult because a potentially very large number of subgraphs must be considered, and their support must be calculated, to find the frequent subgraphs. Several efficient algorithms have been proposed to find frequent subgraphs efficiently. Generally, they start from graphs each having a single vertex or edge, and recursively append edges to these graphs to obtain larger graphs. Algorithms such as FSG[21] perform a *breadth-first search* to explore the search space of all subgraphs. They first find all subgraphs having one edge. Then, the algorithms grow these subgraphs to find those having two edges. Then, those having three edges are considered and so on, until no

patterns can be generated. Other algorithms adopt a depth-first search. They also start from patterns with single edges but recursively grow a pattern before growing others. Such algorithms are MoFa[22], gSpan[2], FFSM[24], Gaston[25] and FPGraphMiner[20]. These algorithms have the same input and output but use different search strategies and data structures.

Generally, the two key challenges for designing an efficient frequent subgraph mining algorithm is how to explore the search space and how to perform support counting. To avoid exploring the whole search space, a key property is that the support measure is anti-monotonic, i.e. the support of a subgraph is always greater or equal to those of its supergraphs[2,21,24,25]. Thus, if a subgraph is infrequent, all its supergraphs can be eliminated from the search space as they cannot be frequent subgraphs. This property, often called *downward closure property* or *Apriori* property is used by most frequent subgraph mining algorithms, and has been used for several other pattern mining problems such as itemset mining[8,16,96,97] and sequential pattern mining[1]. Another problem related to search space exploration is that an algorithm may generate candidate subgraphs that are isomorphic (equivalent) to subgraphs that it has previously considered. To avoid considering these subgraphs again several algorithms perform *isomorphism checking*. Though both efficient exact and approximate linear time isomorphism checking algorithms have been proposed[26], it remains a computationally expensive task.

Many extensions of the frequent subgraph mining problems have been proposed to find subgraphs using other measures to select patterns such as density, edge connectivity and vertex connectivity[27], and subgraphs having a high correlation[30]. To reduce the number of patterns that are presented to the user, algorithms have been proposed to mine concise representations of frequent subgraphs such as closed and maximal subgraphs. A *frequent closed subgraph* is a frequent subgraph that is not a subgraph of any other frequent subgraph having the same support[23]. A *frequent maximal subgraph* is a frequent subgraph that is not a subgraph of any other frequent subgraph[28,29]. For example, in Fig. 2 (right), closed and maximal frequent subgraphs are indicated. Some frequent subgraph mining algorithms can also be extended with small modifications to mine partially labeled graphs, directed graphs, graphs with self-loops, and multiple edges between vertices[23].

Frequent subtree mining. Another popular task for mining patterns in static graphs is

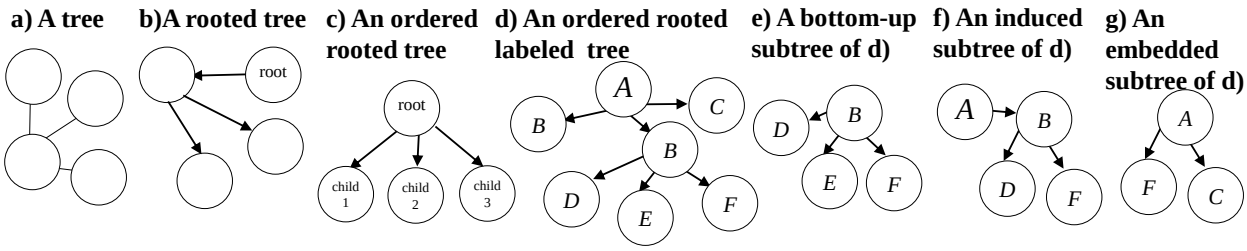

Figure 3: Different types of trees and subtrees

frequent subtree mining[31,32]. The problem consists of identifying a set of subtrees appearing frequently in a database of trees. Several types of trees are considered in the literature[32] (which are graphs). A *free tree* is an undirected connected graph that is acyclic (no sequences of edges can be found that starts from a vertex and leads to the same vertex). A *rooted tree* is a directed acyclic graph where a vertex (node) is called the *root*, the root has no incoming edges, each other node has a single incoming edge, and a path exists from the root to every other node. An *ordered rooted tree* is a rooted tree where childs of a vertex are ordered as a list from first to last. A *labeled tree* is a tree such that each vertex has a label. Fig. 3 (a), (b), (c), (d) shows examples of these different types of trees. Trees can represent many types of data. For example, the tag structure of XML documents and hierarchies of DNS servers can be viewed as ordered rooted trees.

Three main types of subtrees can be extracted from a tree database[31,32]. They are defined as follows. Let there be two ordered labeled trees $T_x = (V_x, E_x, L_{Vx}, L_{Ex}, \phi_{Vx}, \phi_{Ex})$ and $T_y = (V_y, E_y, L_{Vy}, L_{Ey}, \phi_{Vy}, \phi_{Ey})$. The tree T_x is a *bottom-up subtree* of T_y if $E_x \subseteq E_y$, $V_x \subseteq V_y$, labels of nodes/edges and the ordering of child nodes are preserved in T_x, and all descendants of each node $v \in V_y$ are also in T_x. The tree T_x is an *induced subtree* of T_y if $E_x \subseteq E_y$, $V_x \subseteq V_y$, labels of nodes/edges are preserved in T_x, and parent-child relationships between nodes are preserved. The tree T_x is an *embedded subtree* of T_y if $E_x \subseteq E_y$, $V_x \subseteq V_y$, labels of nodes/edges are preserved in T_x, and ancestor/descendant relationships between nodes are preserved. Thus, all bottom-up subtrees are induced subtrees, and all induced subtrees are embedded subtrees ($BottomUpSubtrees \subseteq InducedSubtrees \subseteq EmbeddedSubtrees$). The goal of frequent subtree mining is to find all subtrees that are bottom-up, induced or embedded subtrees of at least *minsup* trees of a tree database. Some classic algorithms are

TreeMiner[31] for mining frequent ordered embedded subtrees in a database of rooted ordered trees, and FREQT[33] to discover all frequent induced subtrees in a database of rooted ordered trees. Tree mining has applications in bioinformatics such as identifying common philogenetic subtrees and RNA structures[31]. Subtree mining can be viewed as a special case of subgraph mining. But the former is a tractable problem while the latter is intractable. Recently, the problem of frequent tree mining has been generalized as *frequent attributed tree mining* to consider trees and subtrees where each node may have multiple labels (*attributed trees*)[39].

Other tasks. Several variations of the above tasks have also been proposed such as to discover frequent sub-DAG from a database of DAG (directed acyclic graphs)[48], and to discover frequent subgraphs in a database of outter-planar graphs (a graph that can be drawn on a plane without any crossing edges, that is a generalization of trees)[49].

MINING PATTERNS IN A SINGLE STATIC GRAPH

The previous subsection has reviewed techniques for mining patterns in a database of static graphs. This section reviews the main tasks for discovering patterns in a single static graph.

Frequent subgraph mining in a single graph. Some frequent subgraph mining algorithms can be adapted to mine frequent subgraphs appearing frequently in a single graph. To do this, it is necessary to define an appropriate support counting function. A simple solution is to define the support of a pattern as its number of occurrences in the graph. For example, consider the graph G of Fig. 4 (left). Fig. 4 (right) shows frequent subgraphs found in G when the minimum threshold is set to 2 occurrences (right). However, this simple definition raises two important problems. First, it allows subgraph occurrences to be overlapping, which may be undesirable for some applications. Second, this support measure is not anti-monotonic (the support of a subgraph may be greater, smaller or equal to the support of its supergraphs), and thus the powerful downward closure property cannot be directly used to reduce the search space[37,38]. This is illustrated with a simple example[38]. In a graph $(X)-(Y)-(X)$, the subgraph (Y) has one occurrence: $(X)-(Y)-(X)$. The subgraph $(X)-(Y)$ has two overlapping occurrences: $(X)-(Y)-(X)$ and $(X)-(Y)-(X)$. And, $(X)-(Y)-(X)$ has a single occurrence: $(X)-(Y)-(X)$. To address this issue, several alternative support counting functions that are anti-monotonic have been defined.

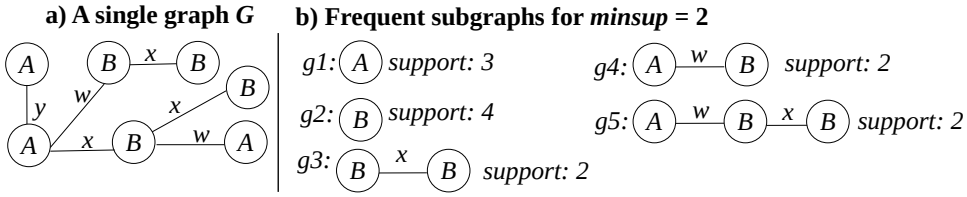

Figure 4: A single graph and frequent subgraphs found for $minsup = 2$

Meng and Tu provide and up-to-date overview of these functions[37]. To improve efficiency of subgraph mining in a single graph, researchers have proposed distributed and approximate algorithms[50].

Frequent subtree mining in a single tree. Some algorithms for mining subtrees in a tree database such as TreeMiner can also mine frequent trees from a single large tree[31].

Traversal pattern mining. This task consists of identifying all frequent sub-paths (*traversal patterns*) in a database of paths over a single graph. This task has applications such as analyzing webpage access patterns on a website (where pages are nodes and links are edges) and finding common car trajectories (where nodes are road intersections and edges are road segments connecting them)[45–47]. The basic problem is to find all sub-paths appearing in at least *minsup* paths, where *minsup* is set by the users[46]. For example, consider the graph of Fig. 5 a) and three trajectories over this graph depicted in Fig. 5 b). By setting $minsup = 2$, frequent subpaths of Fig. 5 c) are obtained. Sub-paths must preserve the visiting order of nodes but may skip some nodes. To obtain these patterns, a sequential pattern mining algorithm[1] can be applied by considering each path as a sequence of symbols. However, such algorithms ignore the graph structure. To exploit the graph structure, Nanopoulos and Manolopoulos[46] proposed three algorithms relying on different search strategies. Then, various extensions of traversal pattern mining have been proposed. The WTPMiner algorithm[45] mines frequent sub-paths from paths over a graph where weights are associated to edges or vertices to indicate their relative importance. A framework was also proposed to find traversal paths in a directed graph for specific time periods (e.g. to find the most frequent paths to reach a destination during lunch time)[47].

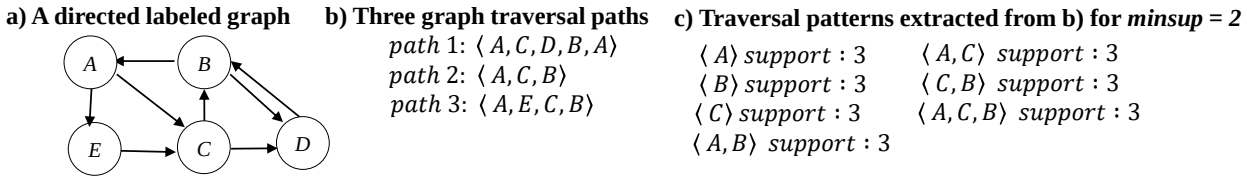

Figure 5: A graph,three traversal paths, and traversal patterns for $minsup = 2$

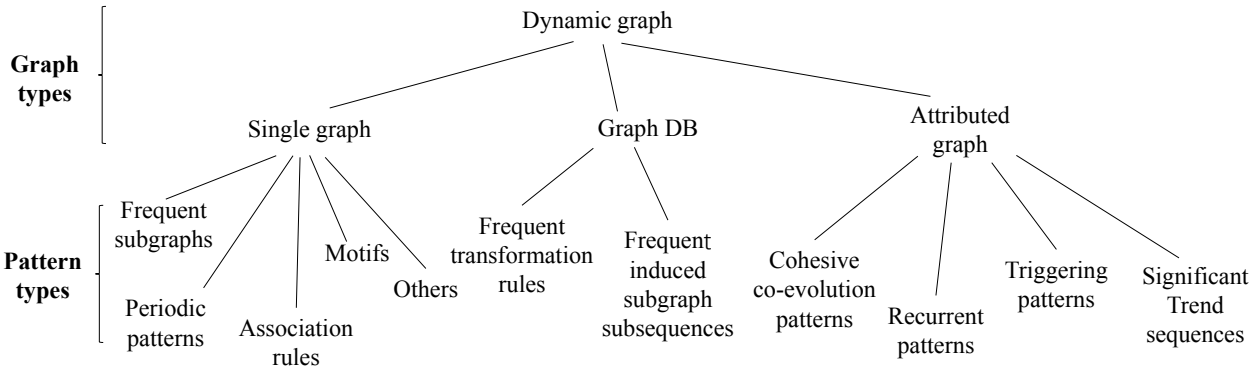

Figure 6: Main types of patterns discovered in dynamic graphs

MINING PATTERNS IN DYNAMIC GRAPHS

Pattern mining algorithms reviewed in the previous section have many applications but do not consider the time dimension. To discover pattern revealing how graph(s) evolve over time, several algorithms have been proposed. The next three subsections review studies on discovering patterns in three main types of data: (1) a dynamic graph, (2) a database of dynamic graphs, and (3) a dynamic attributed graph. Then, the last subsection discusses other extensions. For the convenience of the reader, Fig. 6 shows a tree indicating the main data types and pattern types discussed in this section.

MINING PATTERNS IN A SINGLE DYNAMIC GRAPH

Several studies have been done on finding patterns in a single dynamic graph. Researchers that have contributed to these studies are from various research fields such as data mining, statistics, and network science. Several names have been used to refer to a graph that changes over time such as dynamic graph, dynamic network, evolving graph, temporal graph, graph sequence, time-series of graph, and time-varying graph. There are two main research sub-

areas: (1) techniques to analyze a social network with a focus on community detection[41,42], and (2) pattern mining techniques to identify interesting patterns[51,56,59,66]. As explained in the introduction, this survey focuses on techniques for mining patterns in dynamic graphs. To know more about community detection techniques, the interested reader may refer to recent surveys on this topic[41,42].

This subsection first introduces a formal definition of a dynamic graph and important related terms. Then, a taxonomy of the main types of dynamic graphs is presented based on what is changing in a graph. Lastly, key studies on pattern mining in a dynamic graph are discussed.

Single dynamic graph. Formally, a *single dynamic graph* is a sequence of labeled graphs $G = \langle G_1, G_2, ..., G_T \rangle$ in which a graph G_t represents the state of the dynamic graph at time t. The graph $G_t = (V_t, E_t, \lambda_t)$ consists of a set of vertex V_t at time t, a set of edges $E_t \subseteq V_t \times V_t$ at time t, and a labelling function $\lambda_t : V_t \cup E_t \rightarrow \mathbb{R}$ mapping edges and vertices of G_t to labels (represented by numbers or literals) at time t. A graph at a time t is also called a *snapshot* of the dynamic graph G. For the sake of brevity, this subsection will refer to a single dynamic graph as a dynamic graph.

A taxonomy of dynamic graphs. The above definition is a generic definition of dynamic graph. Different variations of this definition are considered in the literature for the needs of different applications. They can be categorized based on their structure as discussed in the previous section (e.g. directed/undirected graphs, weighted graphs, trees, and attributed graphs), but more importantly they can be described in terms of what is changing in a graph over time. Two main types of evolutions (changes) have been mainly considered. A *topological evolution* refers to changes in a graph topology such as adding and removing vertices and edges. A *label evolution* refers to changes of labels associated to edges and/or vertices.

Based on these concepts, three types of dynamic graphs can be identified: (1) dynamic graphs with only topological evolution[51,53,56,57,59–61], (2) dynamic graphs with only label evolution[66], and (3) dynamic graphs with both topological and label evolution[52,63,64,67,68]. Studies on the third type of dynamic graphs mostly focus on a type of graph called *dynamic attributed graphs*, which will be discussed in a following subsection. Studies on such graphs

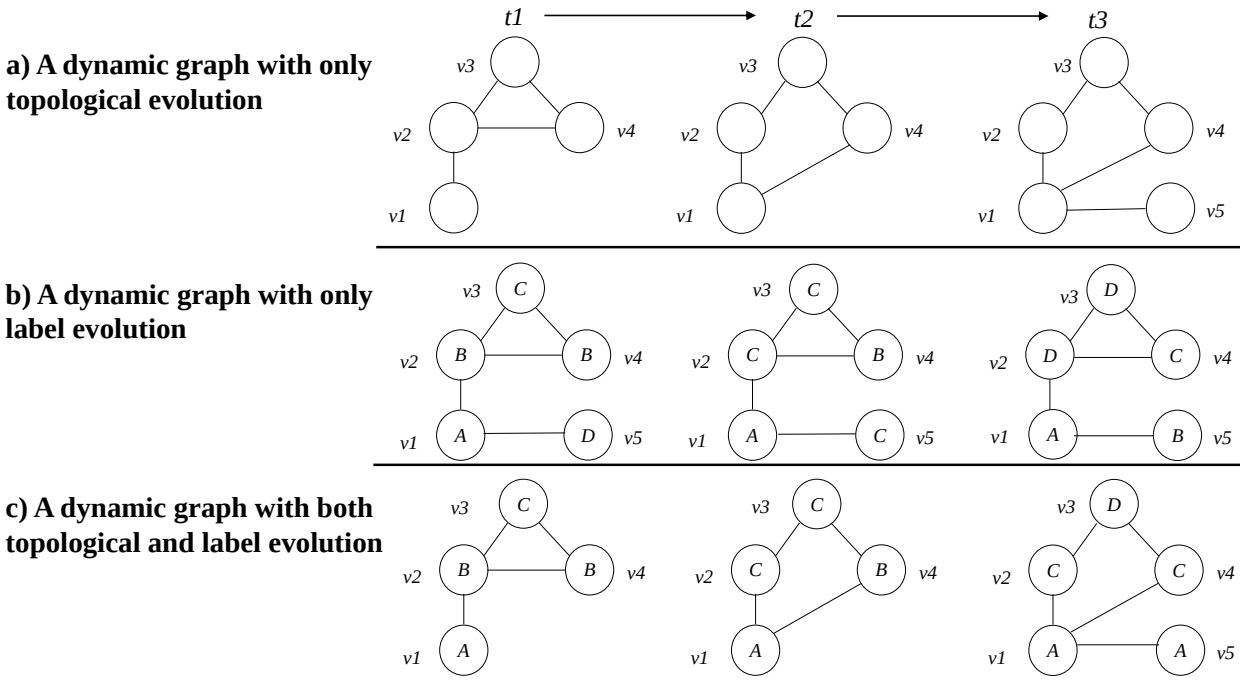

Figure 7: Three types of dynamic graphs

mainly aim to analyze the evolution of multiple attributes and their relationships over time. Fig. 7 provides examples of these different types of dynamic graphs, which evolve over three timestamps t_1, t_2 and t_3. Although only vertices are labelled in this example, edge labels can also be considered. The following paragraphs discuss the main types of patterns that are discovered in a dynamic graph.

Mining frequent subgraphs in a dynamic graph. FSM is the most well-studied problem for mining patterns in a static graph. Several studies have been done on mining frequent subgraphs in a single large graph[37,38] (see previous section). And in general, FSM has inspired most of the work on pattern mining in graphs. Nowadays, with the explosion of data, more and more data has rich temporal information that can be modeled as dynamic graphs. Hence, FSM was naturally extended to analyze dynamic graphs[51].

The first formal definition of FSM in a dynamic graph was proposed by Borgwardt et al.[51]. They designed an algorithm, which internally represents a dynamic graph (a sequence of snapshots) as a single union graph. In this graph, each edge is labeled with a bit string (a sequence of 0s and 1s), which describes the edge's status over time. For instance, an edge labeled as "010" means that this edge did not exist at time 1, appeared at time 2, and did

not exist at time 3. This graph representation is very similar to that used for FSM in a single large static graph[37,38]. The difference is that strings are added to edges to store information about edge evolution. To find frequent subgraphs, Borgwardt et al. extended the GREW algorithm[55], which is designed for FSM in a single graph. The modified algorithm is called Dynamic GREW. It is breadth-first search algorithm, which generates large dynamic subgraphs by joining smaller frequent dynamic subgraphs already found. To find frequent subgraphs, Dynamic GREW generates more candidates than GREW because many frequent substrings may need to be considered for a same subgraph. Moreover, isomorphism checking is more complex in Dynamic GREW because substring checking must be performed when comparing two subgraphs. The output of Dynamic GREW is a series of subgraphs with bit strings representing temporal behaviors over consecutive timestamps that frequently occurred in the input dynamic graph. The algorithm can output either synchronous subgraphs (each occurrence must start at the same time) or asynchronous subgraphs (occurrences of a subgraph are not required to start at the same time). Patterns found using Dynamic GREW can help understanding how graph edges evolve (appear or disappear)[55]. But this approach has three key limitations: it uses a greedy heuristic approach like GREW that trades completeness of results for speed, it considers that edges are unlabelled, and it can have long runtimes[52].

Wackerseuther et al. improved upon that work in several ways[52]. They considered a richer graph model where each graph edge is annotated with a string that contains edge labels for each timestamp, rather than only 0s and 1s. An edge label is either a symbol (e.g. a, b and c) to describe an edge, or a special ϵ label indicating that an edge did not exist at the corresponding timestamp. Before mining patterns, a union graph is created by combining the information of all timestamps. For instance, Fig. 8 a) shows a dynamic graph and Fig. 8 b) shows the corresponding union graph.

Based on that graph representation, Wackerseuther et al. designed a framework to discover dynamic frequent subgraphs where edge strings indicate label evolution. For instance, Fig. 8 c) shows a frequent subgraph found for a minimum support of 2. To discover such patterns, they designed a generic framework that first applies a traditional FSM algorithm to find all frequent subgraphs in the static input graph, while ignoring edge strings. Then, the

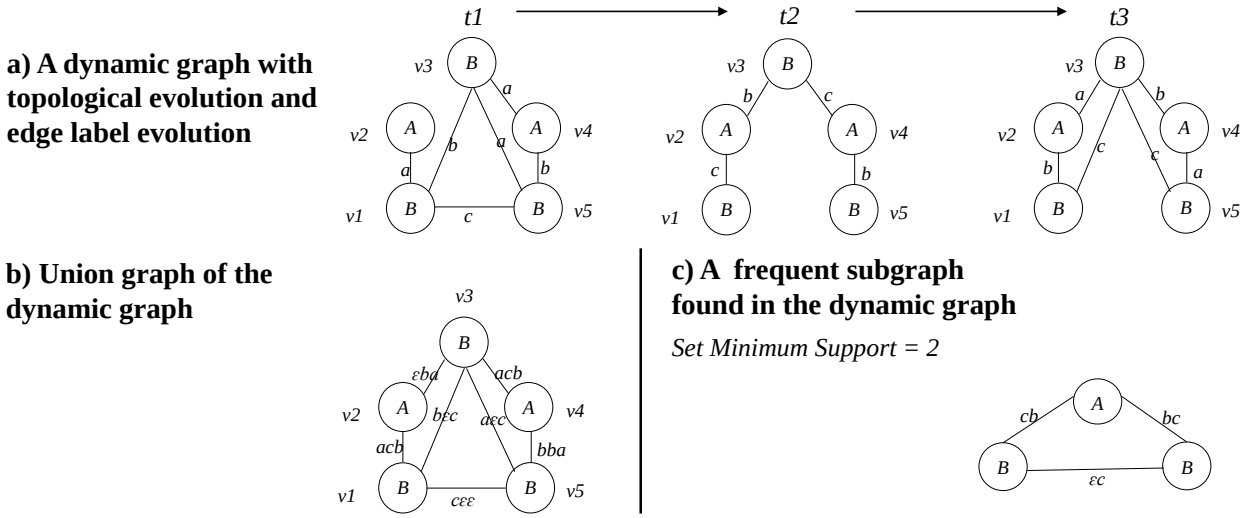

Figure 8: A dynamic graph, the corresponding union graph, and a frequent subgraph

framework searches in each subgraph to find dynamic frequent subgraphs with edge strings. This is done by checking all embeddings (occurrences) of a subgraph to find frequently appearing edge substrings. To efficiently compare graph occurrences, a canonical edge order similar to the one used by gSpan[2] is defined, and a suffix-tree is used to find the frequent longest subtrings in linear time. Advantages of this approach are that it is efficient, it guarantees finding all the desired patterns, its relies on existing FSM algorithms, and consider label evolution.

Contrary to the above two algorithms, which focus on finding frequent subgraphs representing dynamic behaviors, Abedlhamid et al.[53] proposed to identify subgraphs that are frequent in each snapshot of a dynamic graph. To perform this task, a simple solution is to apply a traditional FSM algorithm to each snapshot and then to combine the results. However, this is inefficient because it does not take advantage of the fact that consecutive snapshots are generally similar. To address this problem, Abedlhamid et al. developed an efficient incremental algorithm named IncGM+. Unlike the previous two algorithms, it does not model a dynamic graph as a single graph. It instead processes each new graph one by one using an approach inspired by the Moment algorithm for frequent itemset mining in a stream[54]. This approach consists of mining frequent subgraphs in the first snapshot, and then to only update the "fringe" patterns (those who are at the boundary between

frequent and infrequent patterns) when a new transaction (graph) is processed. This allows to reduce the computational cost of the mining task. IncGM+ also introduces three other performance optimizations. First, instead of storing all embeddings of each fringe subgraph, IncGM+ keeps a minimal number of embeddings for each fringe subgraph which can significantly reduce runtimes and memory usage. Second, a data structure is proposed to efficiently maintain embedding lists to perform fast support calculations. Third, graphs can be processed by batches to further improve efficiency.

Mining periodic patterns in a dynamic graph. The second main type of patterns that has been studied in a single dynamic graph is periodic patterns[56–58]. Lahiri et al.[56] introduced this concept to find repeating interactions in a dynamic graph. For example, one could study a dataset about cellphone calls between people to find interactions that are repeating over time. That study considered finding patterns in a dynamic graph, where snapshots are equally spaced in time (a time-series of graphs), edges representing interactions between vertex are directed or undirected, and vertex labels are unique. This latter assumption greatly simplifies subgraph mining problems as a graph can be represented as a set of integers (each representing the presence of an edge or vertex), which allows to efficiently perform isomorphism checking using the subset operator $\subseteq$. In that study, a subgraph is considered to be *periodic* in a time interval if it appears some minimum number of times in that interval, every consecutive occurrences of the subgraph in that interval is separated by the same amount of time, and the time interval cannot be extended while preserving the previous properties. Furthermore, to eliminate redundancy, it was proposed to only discover closed subgraphs[23], and to find a minimal set of patterns that covers all periodic occurrences of all periodic subgraphs. An example of periodic pattern in the time interval $[t_1, t_5]$ is shown in Fig. 9 c), which appears at timestamps t_1, t_3, and t_5 of the dynamic graph of Fig. 9 a). Fig. 9 b) shows two frequent subgraphs that are found for a minimum support of 3 where the second one is filtered because it is not periodic. To assess the periodic behavior of a , the *Purity* measure is used, which filters out periodic patterns that occur too frequently in a time sub-interval. Moreover, to allow finding periodic patterns that are not eqally spaced in time, it is proposed to apply smoothing to the snapshots as a preprocessing step. An algorithm named PSEMiner was designed to find the desired patterns in polynomial time. It

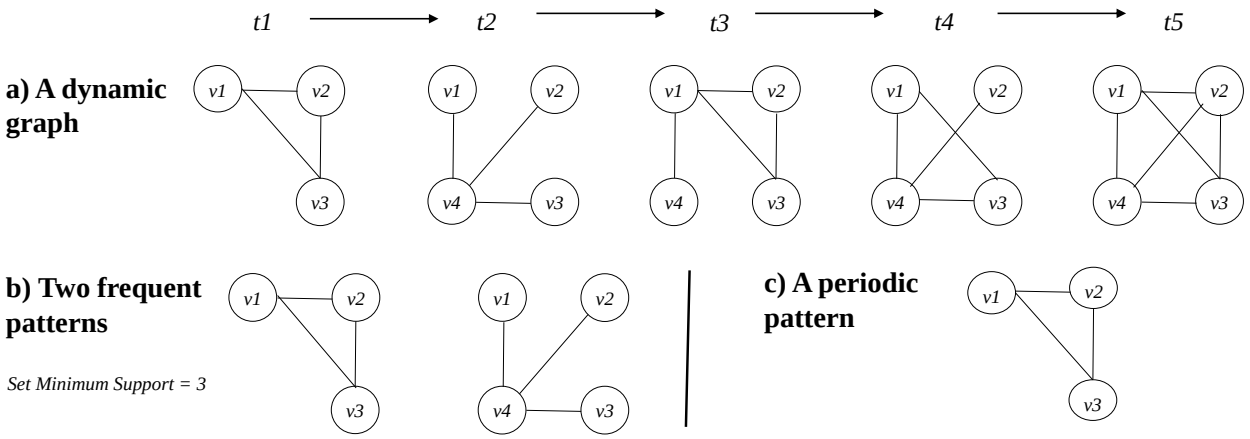

Figure 9: A dynamic graph, some frequent patterns and a periodic pattern

was applied to study e-mail, cellphone, and animal interactions[56]. Several other algorithms have then been proposed to improve the efficiency of mining periodic subgraphs in a dynamic graph[57,58].

Mining association rules in a dynamic graph. The third main type of patterns found in a dynamic graph is rules[59–61,63,64]. Such studies draw inspiration from the traditional data mining task of discovering association rules in transaction databases[8].

Berlingerio et al.[59] introduced an algorithm named GERW to discover a novel type of rules called graph evolution rules (GER) in a dynamic graph. Similar to previous work, GERW first transforms a graph sequence into a single union graph to then mine patterns. In the union graph of GERW, a label on each edge indicates the timestamp of its first appearance. Then, a FSM algorithm for a large static graph can be applied on the union graph to find frequent subgraphs[65]. However, the constraint that labels should be identical for two embeddings of a subgraph should not be enforced because edge labels represent timestamps instead of attribute values. Thus, the GERW algorithm first mines relative-time patterns, that is subgraphs having embeddings that are structurally isomorphic and where edge labels (timestamps) differ only by a constant. For instance, Fig. 10 a) shows a union graph, and Fig. 10 b) shows two embeddings of a subgraph where timestamps differ only by 1 time unit. These embeddings are said to belong to the same equivalence class though their edge labels are not the same. After finding all frequent patterns, the algorithm extracts graph evolution rules from these patterns. A rule has the form $body \rightarrow head$ where *head* is a

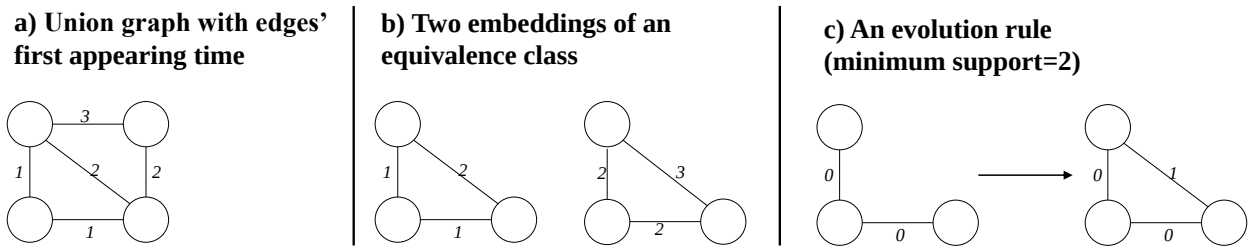

Figure 10: A graph evolution rule found in a dynamic graph

frequent pattern and *body* is obtained by removing the edges having the largest timestamps such that the subgraph remains connected. A rule is output if the confidence between *body* and *head* satisfies a minimum confidence threshold. Thereafter, other algorithms were proposed based on this framework to mine rules in different kinds of dynamic graphs and/or to address limitations of GERW that it does not consider edge deletions and edge relabeling.

Leung et al.[60] proposed to mine Link Formation Rules (LFR) in directed dynamic graphs with multiple edge labels. LFR are discovered by enforcing more strict constraints on rules than GERW does. A LFR indicates how a subgraph is extended by adding a new link from a start node to an end node, with the restriction that every node in the subgraph must have a direct link to the start node or end node. A GER is equivalent to a set of LFR having the same base patterns and where edge timestamps respect the GER's definition. However, these two studies focus on finding association rules in two different types of dynamic graphs. Ozaki et al. [61] then adapted the concept of LFR for simple dynamic graphs. In that work, a LFR represents the addition of an undirected edge to a base pattern (subgraph). Furthermore, two relationships between LFR are defined. First, two LFR are said to be *correlated* if their base pattern is the same and the two extra edges always appear together when the base pattern appears. Second, two LFR are said to be *contrast* when their base pattern is the same and always one or the other extra edge appears when the base pattern appears. To reduce redundancy among patterns and improve pattern mining efficiency, the study only consider δ-closed subgraphs[62] as base patterns.

All the above rule mining algorithms are useful but have the limitation that they cannot handle dynamic graphs with edge or vertex deletion and relabeling. This is because their union graph structure only considers the timestamp of the first appearance of each edge.

To address this limitation, two methods were proposed that keep more information about dynamic graphs by transforming them into a database of union graphs. The first method named DGRMiner was introduced by Vaculik[63]. It can find rules either in a dynamic graph or in a database of dynamic graphs, where any type of changes may appear and where edges may be directed or undirected. To mine patterns in a dynamic graph containing n snapshots, DGRMiner first transforms the dynamic graph into a set of $n-1$ union graphs that will then be treated as a set of static graphs for mining patterns. The k-th union graph contains the union of the first k snapshots and indicates changes between the k-th and the $(k+1)$-th snapshot using relative timestamps. After creating union graphs, a modified frequent subgraph mining algorithm is applied on them to mine high confidence rules. The second method proposed by Scharwachter et al.[64] is named EvoMine and supports dynamic graphs with edge deletions and relabeling. EvoMine has many similarities with DGRMiner, the main difference being that EvoMine relies on bit strings to represent the status of edges and vertices for FSM. Moreover, EvoMine creates union graphs only from pairs of consecutive snapshots.

Mining motifs in a dynamic graph. The fourth main type of patterns found in a dynamic graph are *motifs*, which are small patterns of interconnections that occur more frequently in a large graph than they would in a randomized or reference graph. Motifs thus characterize a graph's structure and can reveal its design principles. The concept of *motifs* was initially proposed by the network science community in the context of static graphs to analyze various types of data such as biological, ecological and web data[87,88]. Then, it was extended for dynamic graphs.

Jin et al.[66] proposed to extract *trend motifs* from a labeled dynamic graph where the topology is fixed, vertices have weights, and only weights are changing over time. A *trend Motif* is defined as a connected subgraph consisting of nodes that show a trend (increasing or decreasing weights) over some time span. The assumption is that a change of a node's weight is not an isolated event and it tends to be correlated with that of its neighbors. Jin et al. designed an algorithm to find all *frequent trend motifs* that are over-represented in a dynamic graph. The algorithm revealed interesting patterns in stock market and micro-array data.

Ahmed and Karypis. then proposed to mine coevolving relational motifs (CRM)[67] in a dynamic labeled graph where labels can change, and edges may be added or removed over time. A CRM is set of motifs (sets of vertices) that changes in a consistent way over time. The biggest difference between *trend motifs* and CRM is that the latter can capture multiple trends over several timestamps (a consecutive sequence of motifs), while each node in a *trend motif* can only have a single trend (increase or decrease) over a time span. An algorithm named CRMminer was proposed to extract all frequent CRM, which uses a depth-first search and canonical labelling. To further improve the efficiency and reduce redundancy, Ahmed and Karypis proposed to mine coevolving induced relational motifs (CIRM)[68] which adds the constraint of finding induced subgraphs for CRM.

The above motif definitions can only capture temporal changes that appear between two consecutive snapshots, which is very restrictive and can fsemiss some important changes. Paranjape et al. proposed to find another type of motifs in dynamic graphs with only topological changes. The proposed method creates an union graph to represent dynamic graphs where edge labels store all appearing timestamp of edges. Considering the edge appearing order, a *δ-temporal motif* is defined as an ordered sequence of edges within a time window δ such that the induced static graph of its edges is connected. The authors developed algorithms to count the number of instances of *δ-temporal motifs*. Though the algorithm is fast and can scale to large graphs, it can only find 2-node motifs and 3-node, 3-edge star or triangle motifs. Moreover, the algorithm only counts the number of instances but do not enumerate them.

Other patterns. Besides the four main types of patterns described above, a few other types of patterns have been studied. The next paragraph discusses four of them.

Robardet et al. proposed a constraint-based pattern mining approach[70] to find dense and isolated subgraphs. Finding such patterns is similar to the task of community detection. Dense and isolated subgraphs are detected in each snapshot and their evolution is studied over time in terms of formation, dissolution, growth, and stability.

Another type of evolution patterns was proposed by Ahmed and Karypis[71]. That study focuses on the evolution of Induced Relational Subgraphs (IRS). An IRS is an induced subgraph where vertices, edges and labels remain unchanged for a long time (a parameter).

They designed an algorithm to find *evolving induced relational states*, which are patterns having the form $S_1 \longrightarrow S_2 \longrightarrow S_3$ where the notation S_i denotes an IRS.

Bogdanov et al.[72] designed an algorithm named MEEDEN to find patterns in a special type of dynamic graphs having binary edge labels $\{-1, 1\}$. The algorithm finds the highest-scoring temporal subgraphs, called *Heaviest Dynamic Subgraph* (HDS). The score of a subgraph is the sum of its edge weights. A HDS in a dynamic graph reprensenting traffic movement can for example indicate the most congested connected streets during some periods of time. While finding high-scoring subgraphs in a static graph is easy, the problem is difficult in a dynamic graph because various time sub-intervals must be considered. To find patterns efficiently, MEEDEN utilizes pruning techniques to ignore unpromising time sub-intervals. The algorithm was applied to analyze transportation, social media and communication graphs.

In another study, Yang et al.[73] proposed to detect areas of an undirected dynamic graph that are frequently changing. This is useful for several applications such as analyzing traffic data as frequently changing areas may face frequent traffic jam. Yang et al. proposed to mine the most frequently changing components of a dynamic graph, where components are dense and connected subgraphs.

MINING PATTERNS IN A GRAPH SEQUENCE DATABASE

Studies reviewed in the previous section have considered mining patterns in a single dynamic graph (a graph sequence). This section reviews studies on mining patterns in a graph sequence database, that is patterns appearing in several graph sequences. The section first introduces important definitions, and then describes the main types of patterns that are found in a graph sequence database. Graph sequence databases are found in several domains such as in social networks and gene networks. For example, in a social network, each person may have a graph sequence representing the evolution of its relationships with others[84–86].

Formally, a *graph sequence database* GDB is a set of tuples $\langle sid, d \rangle$, where sid is a unique sequence ID and d is a graph sequence. Recall that a graph sequence $d = \langle G^{(1)}, G^{(2)}, ..., G^{(m)} \rangle$ is an ordered list of graphs (as defined in previous subsection).

For example, Fig. 11 shows a graph sequence database that contains two graph sequences

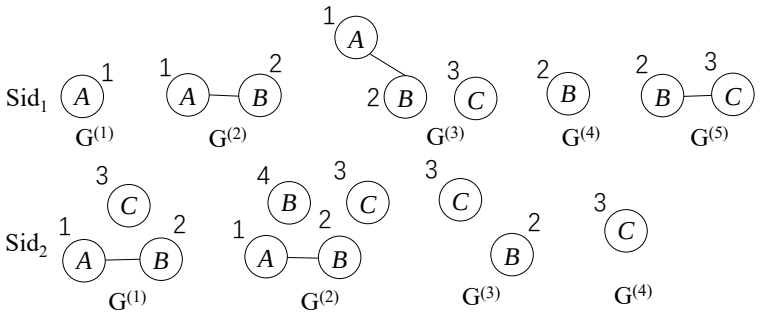

Figure 11: A graph sequence database containing two graph sequences

having the IDs $Sid1$ and $Sid2$. In sequence $Sid1$, there are five elements (graphs), unique IDs (1, 2, 3 and 4) are used to refer to vertices, and each vertex has a label (A, B or C).

Real-life graph sequence databases are often large and sparse. Representing a graph sequence database as a list of graphs can require a considerable amount of space because many parts of a graph may remain unchanged over time. Storing these unmodified elements for consecutive timestamps results in storing redundant information. Using this simple representation can also make it time and memory consuming for mining knowledge in a large and sparse graph sequence database. To compactly represent a graph and facilitate its analysis, several methods have been proposed such as Graph Grammar[82]. However, this latter was designed to be applied to a single graph. To compactly and efficiently represent a graph sequence, Akihiro et al.[80] focused on the differences between each pair of successive graphs and proposed a novel graph grammatical framework which describes a graph sequence as the application of several transformation rules applied to an initial graph.

A transformation rule indicates a change (e.g. adding an edge or changing a label) that appeared between two consecutive snapshots of a graph sequence. In other words, a rule indicates how a snapshot has changed from a timestamp to the next. A graph sequence can then be described by the successive application of transformation rules to the initial snapshot. Such sequence of transformations is called a transformation rule sequence. A more formal definition is given below[80].

Intrastate sequence. Let there be a graph sequence $d = \langle G^{(1)}, G^{(2)}, ..., G^{(m)} \rangle$ containing m snapshots. Consider two graphs $G^{(j)}$ and $G^{(j+1)}$ from consecutive timestamps that have m_j differences. The differences between these two graphs can be described by a sequence

of intermediate graphs $s^{(j)} = \langle G^{(j,1)}, G^{(j,2)}, ..., G^{(j,m_j)} \rangle$ called intrastate sequence such that each intermediate graph has only a single difference with the preceding intermediate graph, and where $G^{(j,1)} = G^{(j)}$ and $G^{(j,m_j)} = G^{(j+1)}$. Based on this idea, the sequence d can be represented as an *interstate sequence* defined as $\langle s^{(1)}, s^{(2)}, ..., s^{(m-1)} \rangle$.

Transformation rule (TR). A transformation rule that transforms a graph $G^{(j,k)}$ into another $G^{(j,k+1)}$ of an intrastate sequence is denoted as $\langle tr^{(j,k)}_{[o_{jk},l_{jk}]} \rangle$ where tr is a literal indicating the transformation type, o_{jk} is the unique ID of the vertex or edge to which the transformation is applied, and l_{jk} is a label that the transformation assigns to a vertex or an edge. Six types of transformation rules are considered, named and denoted as follows: vertex insertion $vi^{(j,k)}_{[u,l]}$, vertex deletion $vd^{(j,k)}_{[u,\cdot]}$, vertex relabeling $vr^{(j,k)}_{[u,l]}$, edge insertion $ei^{(j,k)}_{[(u_1,u_2),l]}$, edge deletion $ed^{(j,k)}_{[(u_1,u_2),\cdot]}$ and edge relabeling $er^{(j,k)}_{[(u_1,u_2),l]}$.

Transformation rule sequence. Based on the concept of transformation rule, an intrastate sequence $s^{(j)} = \langle G^{(j,1)}, G^{(j,2)}, \ldots, G^{(j,m_j)} \rangle$ of a sequence d can be represented by a sequence of transformation rules $seqtr(s^{(j)}) = \langle tr^{(j,1)}_{[o,l]}, tr^{(j,2)}_{[o,l]}, \ldots, tr^{(j,m_j-1)}_{[o,l]} \rangle$. And an interstate sequence d' can be represented as $seqtr(d) = \langle seqtr(s^{(1)}), seqtr(s^{(2)}), \ldots, seqtr(s^{(m-1)}) \rangle$.

Inclusion relation. Let there be an intrastate sequence $s^{(j)}$ of a graph sequence d, and another $s'^{(h)}$ of a graph sequence d'. Their transformation rule sequences are $seqtr(s^{(j)}) = \langle tr^{(j,1)}_{[o,l]}, tr^{(j,2)}_{[o,l]}, \ldots, tr^{(j,m_j-1)}_{[o,l]} \rangle$ and $seqtr(s'^{(h)}) = \langle tr^{(h,1)}_{[o,l]}, tr^{(h,2)}_{[o,l]}, \ldots, tr^{(j,m_h-1)}_{[o,l]} \rangle$, respectively. Iff $\forall tr^{(h,r)}_{[o,l]} \in seqtr(s'^{(h)}), \exists tr^{(j,k)}_{[o,l]} \in seqtr(s^{(j)})$ such that $tr^{(h,r)}_{[o,l]} = tr^{(j,k)}_{[o,l]}$, then $seqtr(s'^{(h)}) \subseteq seqtr(s^{j})$. An inclusion relation is similarly defined for two graph sequences as $seqtr(d) = \langle seqtr(s^{(1)}) and \ldots, seqtr(s^{(m-1)}) \rangle$, $seqtr(d') = \langle seqtr(s'^{(1)}), \ldots, seqtr(s'^{(m'-1)}) \rangle$. Iff there exist integers $1 \leq j_1 \leq \ldots < j_{m-1} \leq m-1$ such that $seqtr(s'^{(h)}) \subseteq seq(s^{(j_h)})$ for $h = 1, 2, \ldots, m'-1$, then $seqtr(d') \subseteq seqtr(d)$.

The **support** (occurrence frequency) of a transformation rule sequence in a graph sequence database GDB is denoted and defined as $sup(seqtr(d)) = \frac{|\{d_i | d_i \in GDB, seqtr(d) \subseteq seqtr(d_i)\}|}{|GDB|}$.

For example, Fig 12 a) and b) show the intrastate sequences corresponding to sequence Sid_1 and Sid_2 of Fig 11, respectively, and Fig 12 c) shows one of their subsequences. Transformation rule sequences of Sid_1 and Sid_2 are $seqtr(Sid_1) = \langle vi^{(1,1)}_{[2,B]} ei^{(1,2)}_{[(1,2),\cdot]} vi^{(1,2)}_{[3,C]} vd^{(3,1)}_{[3,\cdot]} ed^{(3,2)}_{[(1,2),\cdot]}$ $vd^{(3,3)}_{[1,\cdot]}$ $vi^{(4,1)}_{[3,C]}$ $ei^{(4,2)}_{[(2,2),\cdot]} \rangle$ and $seqtr(Sid_2) = \langle vi^{(1,1)}_{[4,B]} vd^{(2,1)}_{[4,\cdot]} ed^{(2,2)}_{[1,2),\cdot]} vd^{(2,3)}_{[1,A]} vd^{(3,1)}_{[3,C]} \rangle$, respectively. The subsequence c) is represented by the transformation rule sequence $\langle vi^{(1,1)}_{[3,C]} ed^{(2,1)}_{[1,2),\cdot]} vd^{(2,2)}_{[1,A]} \rangle$ and

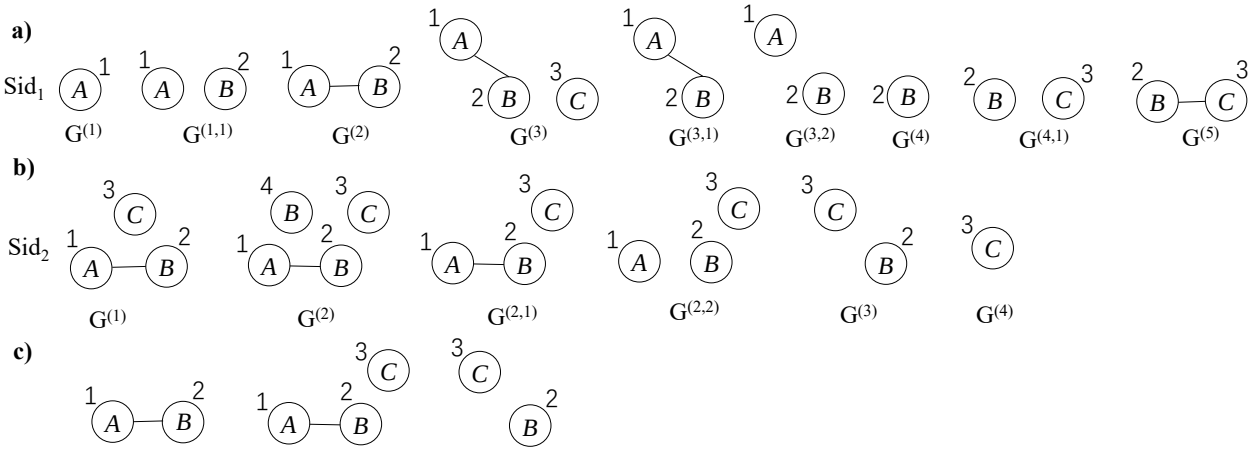

Figure 12: Two intrastate sequences and one of their subsequences

has a support of $2/2 = 1$.

Mining frequent transformation rules. Given a minimum support threshold, Akihiro et al.[80] proposed to mine the transformation rule sequences whose support is no less than a user-defined minimum support threshold, called Frequent Transformation Rule Sequence (FTRS). To efficiently find the FTRSs, Akihiro et al. designed a depth-first pattern growth algorithm based on a sequential pattern mining algorithm named PrefixSpan[83]. The algorithm relies on the anti-monotonicity property of the support measure to reduce the search space, which states that if $seqtr(d_1) \subset seqtr(d_2)$ then $sup(seqtr(d_1)) \geq sup(seqtr(d_2))$. The algorithm considers that each transformation rule is an item in a sequence to transform the problem of mining frequent transformation sequences into that of frequent sequential pattern mining[1]. Akihiro et al.[80] also extended the above problem to mine FTRSs from connected graphs. Given a minimum support threshold, the connected graphs of all graph sequences in a GDB are generated. Then, all frequent connected subgraphs of those connected graphs representing graph sequences are generated by using a traditional frequent graph mining algorithm. Finally, FTRSs are mined from each connected subgraph. FTRSs are interesting because they reveal the evolution sequences that frequently occur in a GDB. Thus, FTRSs may be used to understand a graph and do predictions in some applications.

A limitation of ransformation rule sequence mining is that changes in graphs have to be small or gradual and that there should not be too many vertices, otherwise the performance of the algorithm decreases. To address this problem and discover long sequences in large

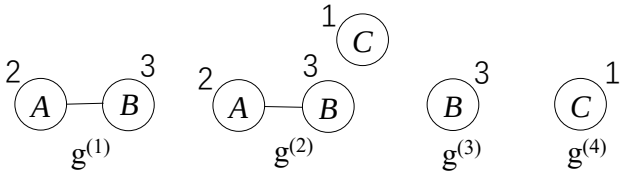

Figure 13: A frequent induced pattern

graphs, Akihiro and Takashi[81] proposed to mine another type of frequent patterns called *induced subgraph subsequence* (ISS).

A graph $G'(V', E', L', l')$ is called a subgraph of $G(V, E, L, l)$ denoted as $g' \subseteq g$ if three conditions are satisfied, which are 1. $(\phi(v_1), \phi(v_2)) \in E$, if $(v_1, v_2) \in E'$, 2. $l'(v) = l(\phi(v))$, and 3. $l'((v_1, v_2)) = l((\phi(v_1)), \phi(v_2))$, where ϕ is a mapping function $\phi : V' \rightarrow V$, and the reverse relation holds. *Induced* means that if two vertices in $V(g')$ are adjacent in g', then they are also adjacent in g.

Formally, an induced subgraph subsequence $b' = (G'^{(1)}, G'^{(2)}, .., G'^{(m)})$ of a graph sequence $b = (G^{(1)}, G^{(2)}, .., G^{(n)})$, where $G'^{(i)} \subseteq G^{(i)}$ and $\phi(L') \rightarrow L$, is denoted as $G' \subseteq_i G$. The support of b' is denoted and defined as $sup_i(b') = |\{sid|\,(\langle sid, d\rangle \in GDB) \wedge (b' \subseteq_i d)\}|$. The anti-monotocity property also holds for ISSs.

For example, a frequent ISS is shown in Fig. 13 for a minimum support of 1. It can be observed that the vertices that are adjacent in the frequent pattern are also adjacent in the GDB. Moreover, there is a correspondence relation between vertex labels in the graph sequence of GDB and the subgraph sequence, where the mapping function is $\phi(1) = 2, \phi(2) = 3, \phi(3) = 1$.

Mining frequent, relevant induced subgraph subsequences. Given a minimum support threshold, Akihiro et al.[81] proposed to mine frequent ISSs from connected graph sequences, which are called frequent, relevant induced subgraph subsequences (FRISS). By defining a mapping function, the main focus is to mine subgraph subsequences that share a same structure with graph sequences in the GDB and where vertex labels match. The proposed approach to mine FRISSs is to first generate connected graphs from all graph sequences of the GDB. Then, all frequent, connected and induced subgraphs in those connected graphs are enumerated by using a conventional graph mining algorithm. Then, frequent, connected and induced subgraphs are given as input to a modified version of Prefixspan[83] to mine

FRISSs. To sum up, because graph sequence databases are often large and contain a large amount of information about vertices and edges of different graph sequences, it is difficult to mine knowledge (patterns) from such database. The above studies mined frequent patterns by compactly representing graph sequences using transformation rules and by considering a mapping relation between vertex labels among different graph sequences. FRISSs can reveal evolution patterns that are common to several sequences of a GDB, which is helpful to understand the distinct and representative features of a GDB.

It is challenging and useful to mine patterns in graph sequence databases. It has many possible applications as graph sequences are found in many domains. For example, it can be used to analyze tranportation data, where a graph sequence and its attributes may represent relationships between roads and indicate their states (e.g. light congestion or blocked) at different times. Furthermore, as there are much fewer algorithms for mining patterns in a graph sequence database than in a single dynamic graph, there are many research opportunities on this topic.

MINING PATTERNS IN A DYNAMIC ATTRIBUTED GRAPH

Until now, this survey has mostly discussed studies for mining patterns in one or more dynamic labelled graphs, where a label may be associated to each vertex or edge. Although labelled graphs are used in many domains, it is desirable in some domains to consider more than one label per edge or vertex. For example, in social network analysis, a social graph may describe relationships (edges) between persons (vertices), where each person may be described using multiple attributes such as age, country and gender. Another example is research collaboration analysis where a research collaboration graph indicates the co-authorship relations (edges) between persons (vertices), and each person may be described using multiple attributes such as a publication count for different journals and conferences. Such data can be viewed as a dynamic graph where attribute values change over time, that is a *dynamic attributed graph*.

Formally, a *dynamic attributed graph* is a sequence of attributed graphs $\mathcal{G} = \langle G_1, G_2, \ldots, G_{t_{max}} \rangle$ where $G_t = (\mathcal{V}_t, \mathcal{A}_t, E_t, \lambda_t)$, $\mathcal{V}_t$ 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 $\lambda_t : \mathcal{V}_t \times \mathcal{A}_t \rightarrow \mathbb{R}$ is a function that associates a real

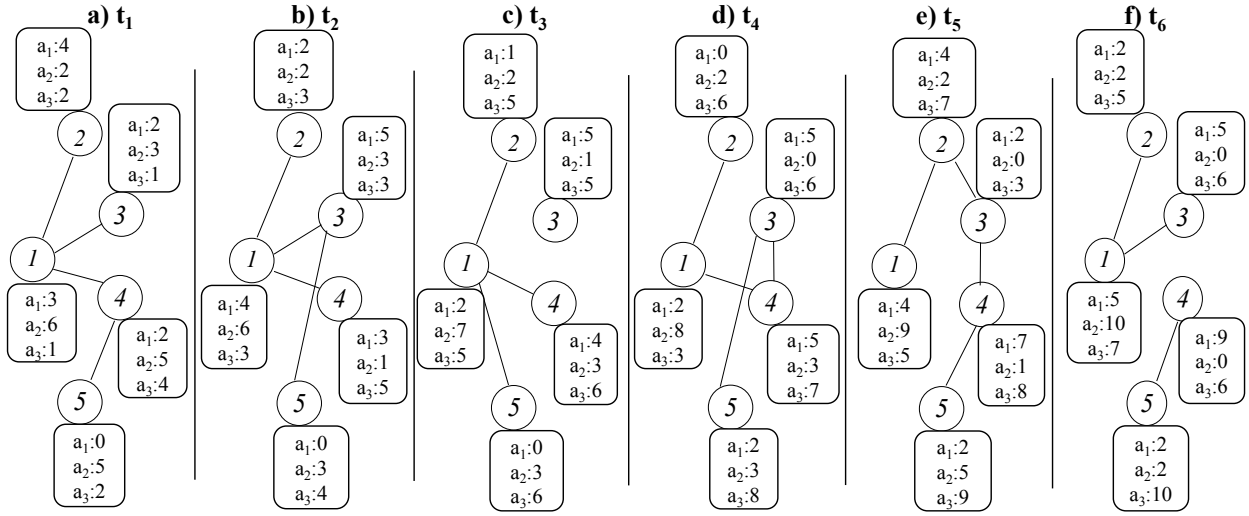

Figure 14: A dynamic attributed graph having six timestamps with raw numerical attribute values.

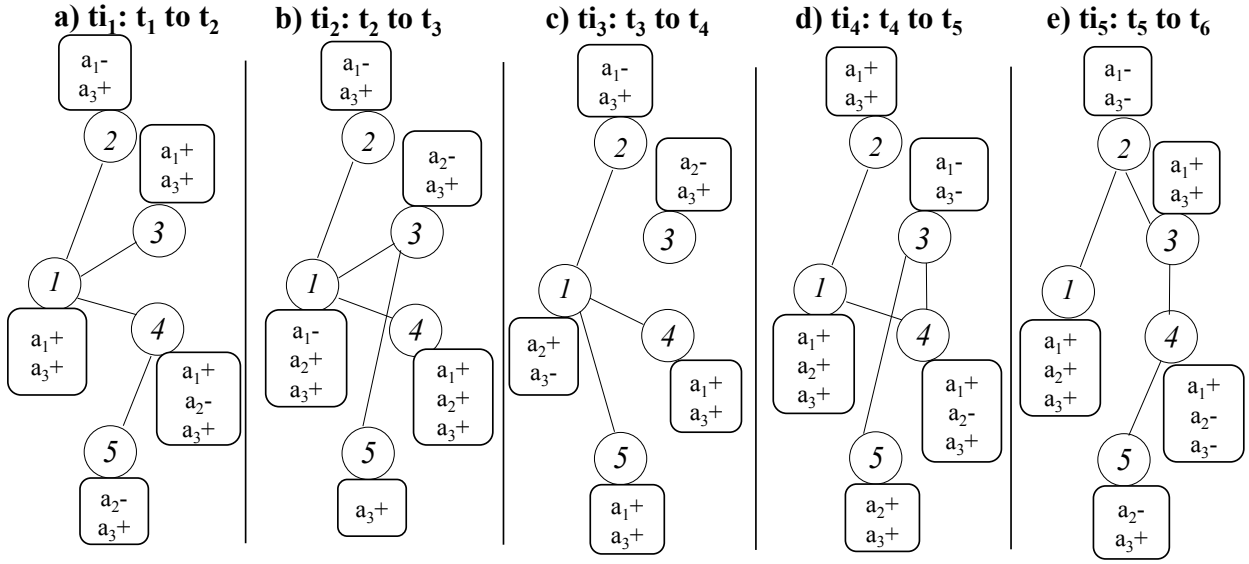

Figure 15: A sequence of trend graphs obtained by preprocessing the dynamic attributed graph of Fig. 14.

value to each vertex-attribute pair, for the timestamp t. To be less influenced by noise when analyzing raw numerical values, a common practice is to convert numerical attribute values of a dynamic attributed graph into trends (attribute variations)[66,79]. For example, Fig. 14 shows a dynamic attributed graph observed at six timestamps. In that figure, vertices are named 1,2,3,4 and 5, attributes are named a_1, a_2 and a_3, and attribute values are numbers. Fig. 15 shows the result of converting that graph into a sequence of trend graphs. In particular, Fig. 15 (a) shows how attribute values have changed (trends) from the timestamp t_1 to t_2 either by increasing (+) or decreasing (−). Similarly, Fig. 15 (b), (c), (d) and (e) shows trends for timestamps t_2 to t_3, t_3 to t_4, t_4 to t_5, and t_5 to t_6. In the following these time intervals are called ti_1, ti_2, ti_3, ti_4 and ti_5, respectively.

A dynamic attributed graph is a generalization of the concept of dynamic graph. Considering more than one attribute makes it possible to find more interesting patterns than when considering a single attribute. The reason is that patterns may involves one or more attributes, which provide information about how a graph evolves over time. Moreover, complex relationships between topological variations and attribute variations may be discovered. Using the concept of dynamic attributed graph also provides more flexibility to the user because rich information can be encoded using several attributes, such as local and global topological properties. The process of creating or selecting attributes for specific needs to then mine patterns is similar to that of feature engineering in machine learning. A user employing an algorithm for mining patterns in a dynamic attribute graph can change attributes and run the algorithm again, without having to change the algorithm. The following paragraph describes the main pattern mining task for dynamic attributed graphs.

Mining trend motifs. Trend motifs a type of pattern found in dynamic attributed graph with a single attribute[66]. A trend motif is a connected subgraph whose vertices show the same trend (e.g. increase or decrease of an attribute value) during a time interval of two consecutive timestamps. Discovering trend motifs allows to find important changes in a dynamic system. A more detailed description of trend motifs was given in the subsection about mining patterns in a single dynamic graph.

Mining cohesive co-evolution patterns. A cohesive co-evolution pattern[76] is a set of vertices that are similar (based on a similarity measure) and display the same trends for

some attribute(s) during a time interval. 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 that may or may not be consecutive. For example, Fig. 16 a) shows a cohesive co-evolution pattern found in the dynamic attributed graph of Fig. 15. This pattern indicates that during time intervals ti_1, ti_2, ti_3 and ti_4, values of attributes a_3 of vertices 1, 2, 4 and 5 have increased.

To filter uninteresting cohesive co-evolution patterns, several constraints have been considered such as a minimum similarity between vertices in a pattern and a volume constraint to ensure that patterns are less influenced by noise. Using these constraints can not only filter patterns but also improve the efficiency of pattern discovery. The proposed algorithm was applied on a landslide dataset consisting of a series of satellite images where decreases of vegetation have been observed[76]. These images were transformed in a dynamic attributed graph and patterns were extracted representing regions having experienced landslides. To make full use of the topological structure among vertices and provide more possibilities for the user, the authors also proposed other interestingness measures. The user can uses these interestingness measures to more precisely express his preferences over patterns, which can improve the efficiency of the algorithm (as using constraints can help to reduce the search space).

Mining recurrent patterns. Cheng et al. [74] generalized the concept of cohesive co-evolution pattern by proposing a new type of patterns called *recurrent patterns*. These patterns capture how attribute values changed for a set of vertices over a sequence of time intervals. While a cohesive pattern is a subgraph, a recurrent pattern is a sequence of subgraphs (vertex sets), where each subgraph can be described using different attributes and trends. For example, Fig. 16 shows a recurrent pattern indicating changes appearing at three time intervals. This pattern has two instances, the first one is appearing at ti_1, ti_2 and ti_3 of Fig. 15, and the second one at ti_3, ti_4 and ti_5. To select interesting recurrent patterns and improve the efficiency of pattern discovery, Cheng et al. also considered several constraints such that a pattern must appear a minimum number of times, non-redundancy, and constraints on the volume of a vertex set and temporal continuity. Recurrent patterns allow to capture the frequent evolutions of trends for nodes in a dynamic attributed graph.

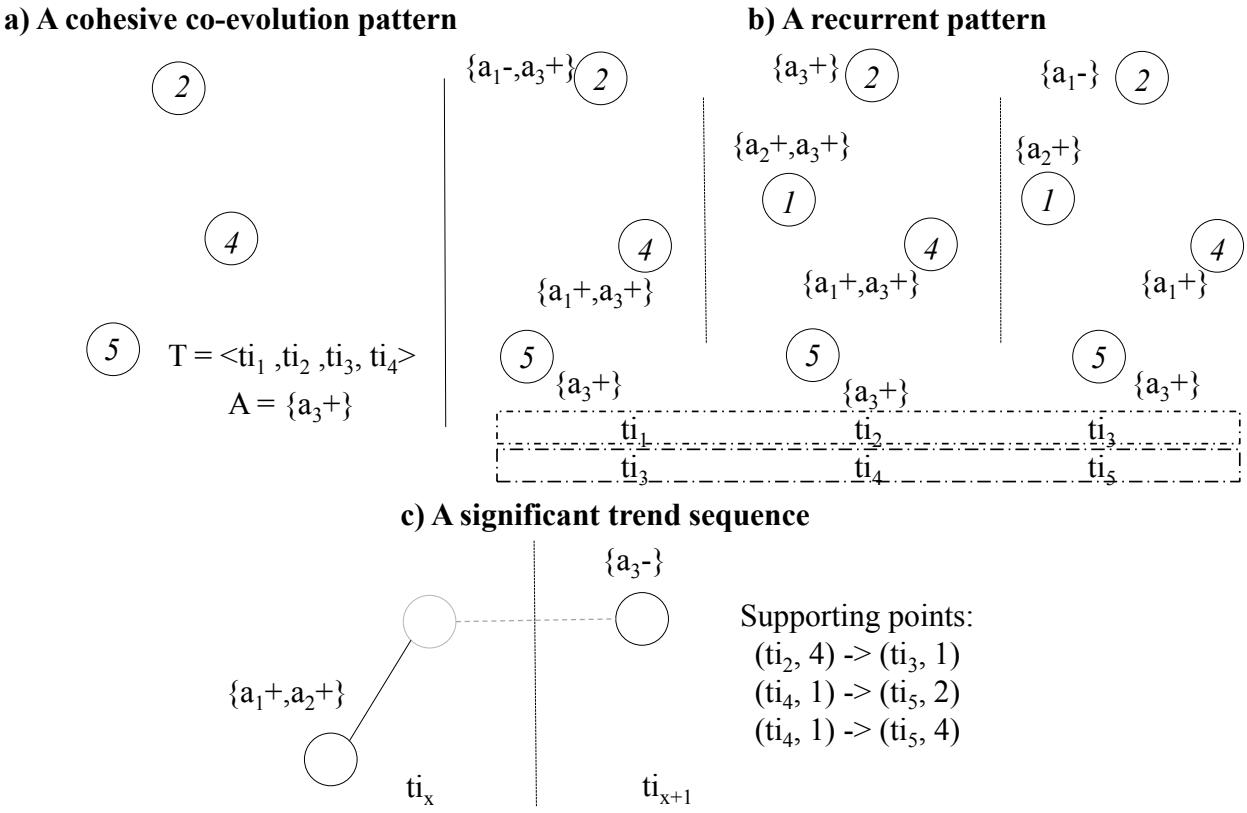

Figure 16: Examples of patterns found in the dynamic attributed graph of Fig. 15

Cheng[75] applied the developed algorithm to extract patterns from a satellite image time series about aquaculture ponds to provide information to experts about how a set of connected ponds evolve together over time.

Mining triggering patterns of topological changes. Another type of patterns are *triggering patterns*[78]. A triggering pattern is a rule of the form $L \rightarrow R$ where L is a sequence of attribute variations followed by a single topological change, R. An example of triggering pattern is $\{a+, b+\}\{c-, d-, e-\} \rightarrow \{closeness-\}$, where a, b, c, d and e are node attributes or topological attributes, the symbol $+$ and $-$ indicate trends, and *closeness* is a topological attribute. This pattern indicates that trends on the left side of the rule triggered the topological change "closeness-" on the right side of rule. Furthermore, the growth rate of a pattern is measured to ensure that attribute variations triggered the topological change. The growth rate is defined in that paper as follows. Consider that vertices of the input graph are divided into two virtual databases. The first one consists of vertices whose attribute variation sequence contain R and the second one consists of the other vertices. The growth rate is the ratio of the frequency of L in first database to its frequency in the second database. In the above example, a, b, c, d, and e can be topological attributes such as closeness, degree, number of cycles since such attributes can be encoded as node attributes.

To discover triggering patterns, it was proposed to convert a dynamic attributed graph into a sequence database and then apply a frequent sequential pattern mining algorithm[1] to extract the desired patterns. In that sequence database representation, each sequence represents the set of attribute variations of a vertex over time. For example, a frequent sequence shared by several vertices may be $\langle\{closeness-, a+\}, \{numcycles+, b-\}, \{eigenvector+, c+\}\rangle$. The proposed approach was applied to analyze a trend graph of social bookmarking activity. Some discovered patterns revealed interesting insights such that if a user increases its number of bookmarks on some topics, it may trigger an increase of his number of followers.

Mining significant trend sequence. Although mining triggering patterns can reveal strong correlations between changes in a dynamic attributed graph, there are two main limitations. The first one is that to identify a triggering pattern, the correlation (growth rate) is only calculated between the last attribute variation and the topological change. Hence, patterns may be found where attribute variations in L are weakly correlated with each other.

Such patterns may be misleading for the user. The second limitation is that relationships between entities are only captured as topological properties, which cause considerable information loss and limit the information that can be expressed by the patterns.

To find more interesting strongly correlated patterns, Fournier-Viger et al.[79] proposed a novel type of patterns called *significant trend sequences*. A significant trend sequence is an ordered list of attribute variations, where consecutive items are strongly correlated. For example, $\langle\{a_1+, a_2+\}, \{a_3-\}\rangle$ in Fig. 16 c) is a significant trend sequence that can be extracted from Fig. 15. To measure the correlation (significance), a novel significance measure is proposed, which is also based on the growth rate but calculated for all consecutive attribute variations. In the above example, $\{a_3-\}$ is strongly correlated with $\{a_1+, a_2+\}$ because $\{a_3-\}$ is not frequent globally but is very often appear following $\{a_1+, a_2+\}$.

Mining trend sequences is not an easy problem as increasing the number of trends can exponentially increase the size of the search space. To efficiently mine the proposed patterns, two projection based algorithms named $TSeqMiner_dfs$ and $TSeqMiner_bfs$ were designed. They rely on several pruning strategies such as upper bound filtering to avoid exploring the whole search space and find all patterns.

The proposed algorithms were first applied to the DBLP dataset, which is a dynamic attributed graph about co-authorship relationships between researchers in different conferences and journals over several years. Some interesting trend sequences were found such as $\langle\{VLDB+\}, \{ICDE+, VLDB\ =\}\rangle$, which 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. The algorithms were also applied on US flight data collected during several years. An interesting pattern found is $\{(NbCancellation--,\ NbDivertedFlights--, NbDelayedDepartures-)$, $(NbDepartures-,\ NbCancellations-,\ NbDivertedFlights-,\ DelayedDepartures-,$ $NbDelayedArrivals+)\}$, which indicates that after an airport recovered from a hurricane's damages the number of cancellations, diverted flights and delayed departures decreased, which then influenced airports that were not damaged by the hurricane (but are connected). In that pattern, the symbols $-$ and $+$ indicate small increase or decrease, while $--$ and $++$ indicate a large increase or decrease.

EXTENSIONS

The previous sections have described the main tasks for discovering patterns in dynamic graphs. There exists various extensions of these tasks that have not been discussed so far. This section gives a brief overview.

Several studies have been done on mining patterns in a data stream of graphs. A stream is an infinite sequence of graphs arriving at high speed. To mine patterns in a stream, adapted algorithms need to be designed because unlike a database, a stream can only be read once and the full stream cannot be kept in memory. Moreover, data distribution may change over time in a stream. For instance, Nishioka et al.[89] proposed two algorithms for mining frequent subgraphs in dense graph streams with limited memory using a disk-based structure. In another study, Ray et al.[90] proposed an algorithm to mine frequent subgraphs in a large attributed streaming graph. The algorithm is an approximate algorithm, which assumes that updates arrive as batches, and where each update is composed of adding nodes and edges. The algorithm timely reports the likely frequent subgraphs, which can help to monitor the network. The study was applied to social network data, movie data, among others.

Some papers have extended graph pattern mining tasks to consider uncertainty of the data. For example, Leung and Cuzzocrea[91] proposed algorithms to mine frequent subgraphs in an uncertain data stream where edges are annotated with existential probabilities. Some papers have also considered finding rare patterns instead of frequent ones[94] and weighted patterns[95].

Other papers have also been proposed to solve pattern mining problems of specific applications. For example, Javel et al.[92] designed a method for detecting sequences of changes in ontologies to reveal how it is edited over time. In that work, an ontology is represented as a dynamic attributed graph.

Another extension is about privacy-preserving data mining, where the goal is to hide sensitive patterns that may reveal important information. For example, Cheng et al.[93] designed a two-phase algorithm for hiding sensitive subgraphs in the context of frequent subgraph mining from a graph database. Although this work is not on dynamic graphs, it is

relevant as some dynamic subgraph mining algorithms rely on traditional subgraph mining algorithms.

RESEARCH OPPORTUNITIES

Mining patterns in dynamic graphs is an active research area. Although several papers have been published in this field, there are numerous research opportunities. Some of the key research opportunities are:

- **Design more efficient algorithms.** Since pattern mining is generally quite computationally expensive, it is important to design more efficient algorithms in terms of runtime and memory. This can be done by developing novel algorithms, search space pruning strategies and data structures. Moreover, GPU, multi-thread, and parallel algorithms can be designed to scale to very large datasets. Moreover, additional constraints may be integrated in algorithms to select more interesting patterns and reduce the search space.

- **Discover patterns in more complex data.** A trend in recent years has been to consider more complex types of data such as attributed graphs[74,76,77,79], streams[89,90], and graphs with uncertainty[91]. The reason is that complex data are found in many applications. Developing models to handle complex data is thus important to address real-life problems.

- **Discover more complex pattern types.** Another important research direction is to develop algorithms to identify more complex patterns that provide more useful information to users. This can be done by extending pattern definitions or considering additional constraints or interestingness measures. A source of inspiration can be other pattern mining tasks such as itemset mining[8] and sequential pattern mining[1], for which many extensions have been developed.

- **Novel applications.** Algorithms for mining patterns in dynamic graphs can be applied to novel applications where data can be represented as dynamic graphs. This is

especially interesting for emerging applications such as the Internet of Things[36], edge computing[34], and Vehicular Ad-hoc NETworks (VANET)[35]. This may not only provide solutions to applied problems but the applications may raise new challenges that may inspire further research.

CONCLUSION

Dynamic graphs are a type of data commonly found in numerous fields. Discovering patterns in such graphs can provide interesting insights on data. This survey has provided an overview of algorithms for discovering patterns in dynamic graphs, including those for mining patterns in a dynamic graph, graph sequence database, and dynamic attributed graphs. Moreover, other extensions have been discussed such as graph mining in streams and uncertain data. Finally, research opportunities have been discussed.

Acknowledgement. This work is supported by the National Science Fundation of China and the Harbin Institute of Technology.

References

1. Fournier-Viger, P, Lin, JCW, Kiran, UR, Koh, YS. A Survey of Sequential Pattern Mining, *Data Science and Pattern Recognition*, 2017, 1(1):54–77.

2. Yan, X, Han, J. gSpan: Graph-Based Substructure Pattern Mining. In: *Proc. 2002 IEEE Intern. Conf. Data Mining*, Maebashi City, Japan, 9-12 December, 2002:721–724

3. Jiang, C, Coenen, F, Zito, M. A survey of frequent subgraph mining algorithms. *The Knowledge Engineering Review*. 2013, 28(1):75–105.

4. Bhuiyan, MA, Al Hasan, M. An iterative MapReduce based frequent subgraph mining algorithm. *IEEE Transactions on Knowledge and Data Engineering*. 2015, 27(3):608–20.

5. Fournier-Viger, P, Gomariz, A, Campos, M, Thomas, R. Fast Vertical Mining of Sequential Patterns Using Co-occurrence Information. In: *Proc. 18th Pacific-Asia Conf. Knowledge Discovery and Data Mining*. Tainan, Taiwan, 13-16 May, 2014:40–52.

6. Mooney, CH, Roddick, JF. Sequential pattern mining–approaches and algorithms. *ACM Computing Surveys*. 2013, 45(2):1–19.

7. Pyun, G, Yun, U, Ryu, KH. Efficient frequent pattern mining based on linear prefix tree. *Knowledge-Based Systems*. 2014, 55:125–39.

8. Fournier-Viger, P, Lin, JCW, Vo, B, Chi, TT, Zhang, J, Le, HB. A Survey of Sequential Pattern Mining, *WIREs Data Mining and Knowledge Discovery*, 2017, doi:10.1002/widm.1207.

9. Bogarin, A, Cerezo, R, Romero, CA survey on educational process mining. *WIREs Data Mining and Knowledge Discovery*, 2018, 8(1):e1230.

10. Feng, Z, Zhu, Y. A survey on trajectory data mining: Techniques and applications. *IEEE Access*. 2016, 4:2056–2067.

11. Mabroukeh, NR, Ezeife, CI. A taxonomy of sequential pattern mining algorithms. *ACM Computing Surveys*. 2010, 43(1):3.

12. Kim, JC, Chung, K. Mining based time-series sleeping pattern analysis for life big-data. *Wireless Personal Communications*, 2019, 105(2):475–489.

13. Yeh, CCM, Zhu, Y, Ulanova, L, Begum, N, Ding, Y, Dau, HA, Zimmerman, Z, Silva, DF, Mueen, A., Keogh, E. Time series joins, motifs, discords and shapelets: a unifying view that exploits the matrix profile. *Data Mining and Knowledge Discovery*, 2018, 32(1):83–123.

14. Wang, H, Wu, J, Zhang, P, Chen, Y. Learning Shapelet Patterns from Network-based Time Series Data. *IEEE Transactions on Industrial Informatics*, 2019, 15(7):3864–3876.

15. Shekhar, S, Evans, MR, Kang, JM, Mohan, P. Identifying patterns in spatial information: A survey of methods. *Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery*, 2011 1(3):193–214.

16. Agrawal, R, Srikant, R. Fast algorithms for mining association rules. In: *Proc. 20th int. conf. very large data bases, VLDB 1994*, Santiago de Chile, Chile, 12-15 September, 1994: 487–499).

17. Szathmary, L, Napoli, A, Valtchev, P. Towards Rare Itemset Mining. In: *Proc. 19th IEEE Intern. Conf. Tools with Artificial Intelligence*, Patras, Greece, 29-31 October, 2007:305–312.

18. Szathmary, L, Valtchev, P, Napoli, A, Godin, R. Efficient Vertical Mining of Minimal Rare Itemsets. In: *Proc. 9th Intern. Conf. Concept Lattices and Their Applications*, Fuengirola, Spain, 11-14 October, 2012:269–280.

19. Fournier-Viger, P, Lin, JCW, Truong-Chi, T, Nkambou, R. A Survey of High Utility Itemset Mining. *High-Utility Pattern Mining*. Cham:Springer, 2019:1–45

20. Vijayalakshmi, R, Nadarajan, R, Roddick, JF, Thilaga, M, Nirmal, P. FP-GraphMiner-A Fast Frequent Pattern Mining Algorithm for Network Graphs. *Journal of Graph Algorithms and Applications*, 2011, 15(6):753–76.

21. Kuramochi, M, Karypis, G.Frequent subgraph discovery. In: *Proc. 1st IEEE Int. Conf. on Data Mining*, San Jose, USA, 29 November - 2 December, 2001:313–320.

22. Borgelt, C, Berthold, MR. Mining molecular fragments: Finding relevant substructures of molecules. In: *Proc. 2nd IEEE Int. Conf. on Data Mining*, Maebashi City, Japan, 9-12 December 2002, 2002:51–58.

23. Yan, X, Han, J. CloseGraph: Mining closed frequent graph patterns. In: *Proc. 9th ACM SIGKDD Intern. Conf. Knowledge Discovery and Data Mining*, Washington DC, USA, 24-27 August, 2003:286– 295.

24. Huan. J, Wang. W, Prins. J. Efficient mining of frequent subgraphs in the presence of isomorphism. In: *Proc. 3rd IEEE Int. Conf. on Data Mining*, Melbourne, Florida, USA, 19-22 December, 2003:549–552.

25. Nijssen, S, Kok, JN. A quickstart in frequent structure mining can make a difference. In: *Proc. 10th ACM SIGKDD Int. Conf. on Knowledge Discovery and Data Mining*, Seattle, WA, USA, 22-25 August, 2004:647–652.

26. Lee, J, Han, WS, Kasperovics, R, Lee, JH. An in-depth comparison of subgraph isomorphism algorithms in graph databases. In: *Proc. 38th Intern. Conf. on Very Large Databases*, Istanbul, Turkey, August 27–31 December, 2012:133–144.

27. Zhu, F, Yan, X, Han, J, Yu, PS. gPrune: a constraint pushing framework for graph pattern mining. In: *Proc. 11th Pacific-Asia Conference on Knowledge Discovery and Data Mining*, Nanjing, China, May 22-25, 2007:388–400.

28. Thomas, LT, Valluri, SR, Karlapalem, K. Margin: Maximal frequent subgraph mining. *ACM Transactions on Knowledge Discovery from Data*, 2010, 4(3):10–42.

29. Kimelfeld, B, Kolaitis, PG. The complexity of mining maximal frequent subgraphs. *ACM Transactions on Database Systems*, 2014, 39(4): 1–32.

30. Ozaki, T, Ohkawa, T. Mining correlated subgraphs in graph databases. In: *Proc. 12th Pacific-Asia Conference on Knowledge Discovery and Data Mining*, Osaka, Japan, May 20-23, 2008:272–283

31. Zaki, MJ. Efficiently mining frequent trees in a forest: Algorithms and applications. *IEEE transactions on knowledge and data engineering*, 2005, 17(8):1021–1035.

32. Chi, Y, Muntz, RR, Nijssen, S, Kok, JN. Frequent subtree miningAn overview. *Fundamenta Informaticae*, 2005, 66(12):161–198.

33. Asai, T, Abe, K, Kawasoe, S, Arimura, H, Satamoto, H, Arikawa, S. Efficient Substructure Discovery from Large Semi-Structured Data. In: *Proc. 2nd SIAM Int. Conf. on Data Mining*, Arlington, VA, USA, April 11-13, 2002:158–174.

34. Shi, W, Cao, J, Zhang, Q, Li, Y, Xu, L. Edge computing: Vision and challenges. *IEEE Internet of Things Journal*, 2016, 3(5):637–646.

35. Hasrouny, H, Samhat, AE, Bassil, C, Laouiti, A. VANet security challenges and solutions: A survey. *Vehicular Communications*, 2017, 7:7–20.

36. Ammar, M, Russello, G, Crispo, B. Internet of Things: A survey on the security of IoT frameworks. *Journal of Information Security and Applications*, 2018, 38:8–27.

37. Meng, J, Tu, YC. Flexible and Feasible Support Measures for Mining Frequent Patterns in Large Labeled Graphs. In: *Proceedings of the 2017 ACM International Conference on Management of Data*. Chicago, IL, USA, 14-19 May, 2017:391–402.

38. Fiedler, M, Borgelt, C. Support Computation for Mining Frequent Subgraphs in a Single Graph. In: *Proc. 5th Intern. Workshop on Mining and Learning with Graphs*. Firenze, Italia, 1-3 August, 2007:1–6.

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

40. Dougnon, YR, Fournier-Viger, P, Lin, JCW., Nkambou, R. Inferring Social Network User Profiles using a Partial Social Graph. *Journal of Intelligent Information Systems*, 2016, 47(2):313–344.

41. Bian, R, Koh, YS, Dobbie, G, Divoli, A. Identifying Top-k Nodes in Social Networks: A Survey. *ACM Computing Surveys*, 2019, 52(1), article 22.

42. Rossetti, G, Cazabet, R. Community discovery in dynamic networks: a survey. *ACM Computing Surveys*, 2018:51(2), article 35.

43. Fouss, F, Saerens, M, Shimbo, M. *Algorithms and models for network data and link analysis*. Cambridge University Press; 2016.

44. Liu, Y, Safavi, T, Dighe, A, Koutra, D. Graph summarization methods and applications: A survey. *ACM Computing Surveys* 2018:51(3), article 62.

45. Geng R, Xu W, Dong X. WTPMiner: Efficient Mining of Weighted Frequent Patterns Based on Graph Traversals. In: *Proc. 2nd Intern. Conf. Knowledge Science, Engineering and Managemen*, Melbourne, Australia, 28-30 November, 2007:412–424.

46. Nanopoulos A, Manolopoulos Y. Mining patterns from graph traversals. *Data and Knowledge Engineering*, 2001, 37(3):243–66.

47. Luo, W, Tan, H, Chen, L, Ni, LM. Finding time period-based most frequent path in big trajectory data. In: *Proc. 2013 ACM Intern. Conf. on management of data*, New York, NY, USA, 22-27 June, 2013:713–724.

48. Termier, A, Tamada, Y, Numata, K, Imoto, S, Washio, T, Higuchi, T. DIGDAG, a First Algorithm to Mine Closed Frequent Embedded Sub-DAGs. In: *Mining and Learning with Graphs*, Firence, Italy, 1-3 August, 2007.

49. Horvath, T, Ramon, J, Wrobel, S. Frequent subgraph mining in outerplanar graphs. Data Mining and Knowledge Discovery, 2010, 21(3):472–508.

50. Abdelhamid, E, Abdelaziz, I, Kalnis, P, Khayyat, Z, Jamour, F. Scalemine: Scalable parallel frequent subgraph mining in a single large graph. In: *Proc. Intern. Conf. on High Performance Computing, Networking, Storage and Analysis*, Salt Lake City, UT, USA, 13-18 November, 2016:717–727.

51. Borgwardt, KM, Kriegel, HP, Wackersreuther, P. Pattern Mining in Frequent Dynamic Subgraphs. In: *Proc. of the 6th IEEE International Conference on Data Mining*, Hong Kong, China, 18–22 December, 2006:818–822.

52. Wackersreuther, B, Wackersreuther, P, Oswald, A, Bohm, C, Borgwardt, KM. Frequent subgraph discovery in dynamic networks. In: *Proc. 8th Workshop on Mining and Learning with Graphs*, Washington, D.C., USA, 24-25 July, 2010:155–162.

53. Abdelhamid, E, Canim, M, Sadoghi, M, Bhattacharjee, B, Chang, Y, Kalnis, P. Incremental Frequent Subgraph Mining on Large Evolving Graphs. In: *Proc. 34th Intern. Conf. on Data Engineering*, Paris, France, 16-19 April, 2018:1767–1768.

54. Chi, Y, Wang, H, Yu, P, Muntz, RR. Moment: maintaining closed frequent itemsets over a stream sliding window. In: *Proc. 4th IEEE International Conference on Data Mining*, Brighton, UK, 1-4 November 2004:59–66.

55. Kuramochi, M, Karypis, G. GREW - a scalable frequent subgraph discovery algorithm. In: *Proc. 4th IEEE International Conference on Data Mining*, Brighton, UK, 1-4 November 2004:439–442.

56. Lahiri, M, Bergerwolf, TY (2009). Mining Periodic Behavior in Dynamic Social Networks. In: *Proc. 8th IEEE International Conference on Data Mining*. Pisa, Italy, 15-19 December, 2008:373–382.

57. Halder, S, Samiullah, M, Lee, YK. Supergraph based Periodic Pattern Mining in Dynamic Social Networks, *Expert Systems With Applications*. 2016, 72:430-442.

58. Apostolico, A, Barbares, M, Pizzi, C. Speedup for a periodic subgraph miner. *Information Processing Letters*, 2011, 111(11):521–523.

59. Berlingerio, M, Bonchi, F, Bringmann, B, Gionis, A. Mining graph evolution rules. In: *Proc. European conference on machine learning and knowledge discovery in databases*. Bled, Slovenia, 7-11 September, 2009:115–130.

60. Leung, CWK, Lim, EP, Lo, D, Weng, J. Mining interesting link formation rules in social networks. In: *Proc. 19th ACM intern. Conf. on Information and knowledge management*. Toronto, Ontario, Canada, 26-30 October, 2010:209–218.

61. Ozaki, T, Etoh, M. Correlation and contrast link formation patterns in a time evolving graph. In: *Proc. Workshops of the 11th International Conference on Data Mining*. Vancouver, BC, Canada, 11 December, 2011:1147–1154.

62. Takigawa, I, Mamitsuka, H. Efficiently mining δ-tolerance closed frequent subgraphs, *Machine Learning*, 2011, 82(2):95–121.

63. Vaculik, K. A Versatile Algorithm for Predictive Graph Rule Mining. In: *Proc. 15th Conference on Information Technologies - Applications and Theory*, Slovensky Raj, Slovakia, 17-21 September, 2015:51–58.

64. Scharwachter, E, Muller, E, Donges, J, Hassani, M, Seidl, T. Detecting change processes in dynamic networks by frequent graph evolution rule mining. In: *Proc. 16th International Conference on Data Mining*. Barcelona, Spain, 12-15 December, 2016:1191–1196.

65. Bringmann, B, Nijssen, S. What is frequent in a single graph?. In: *Proc. 12th Pacific-Asia Conference on Knowledge Discovery and Data Mining*, Osaka, Japan, May 20-23, 2008:858–863.

66. Jin, R, McCallen, S, Almaas, E. Trend motif: A graph mining approach for analysis of dynamic complex networks. In: *Proc. 7th IEEE International Conference on Data Mining*. Omaha, Nebraska, USA, 28-31 October, 2007:541–546.

67. Ahmed, R, Karypis, G. Algorithms for mining the coevolving relational motifs in dynamic networks. *ACM Transactions on Knowledge Discovery from Data*, 2015, 10(1):4–31.

68. Ahmed, R, Karypis, G. Mining coevolving induced relational motifs in dynamic networks. In: *Proceedings of 2nd SDM Workshop on Mining Networks and Graphs*, Vancouver, BC, Canada, 30 April, 2015.

69. Paranjape, A, Benson, AR, Leskovec, J. Motifs in temporal networks. In: *Proc. 10th ACM Intern. Conf. on Web Search and Data Mining*, Cambridge, United Kingdom, 6-10 February, 2017:601–610.

70. Robardet, C. Constraint-based pattern mining in dynamic graphs. In: *Proc. 9th IEEE International Conference on Data Mining*, Miami, Florida, USA, 6-9 December, 2009:950–955.

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

72. Bogdanov, P, Mongiovi, M, Singh, AK. Mining heavy subgraphs in time-evolving networks. In: *Proc.11th International Conference on Data Mining*, Vancouver, BC, Canada, 11-14 December, 2011:81–90.

73. Yang, Y, Xu, JX, Gao, H, Pei, J, Li, J. Mining most frequently changing component in evolving graphs. *World Wide Web*, 2014, 17(3):351–376.

74. Cheng, Z, Flouvat, F, Selmaoui-Folcher, N. Mining recurrent patterns in a dynamic attributed graph. In: *Proc. 21th Pacific-Asia Conference on Knowledge Discovery and Data Mining*, Jeju, South Korea, May 23-26, 2017:631–643.

75. Cheng, Z. Mining recurrent patterns in a dynamic attributed graph, Ph.D. thesis, University of New Caledonia, 2018.

76. Desmier, E, Plantevit, M, Robardet, C, Boulicaut, J. Cohesive Co-evolution Patterns in Dynamic Attributed Graphs. In: *Proc. 15th Intern. Conf. on Discovery Science*. Lyon, France, 29-31 October, 2012:110–124.

77. Desmier, E, Plantevit, M, Robardet, C, Boulicaut, J. Trend mining in dynamic attributed graphs. In: *Proc. 6th Joint European Conference on Machine Learning and Knowledge Discovery in Databases*, Prague, Czech Republic, 22-26 September, 2013:654–669.

78. Kaytoue, M, Pitarch, Y, Plantevit, M, Robardet, C. Triggering patterns of topology changes in dynamic graphs. In: *Proc. 6th IEEE/ACM International Conference on Advances in Social Networks Analysis and Mining*, Beijing, China, 17-20 August, 2014:158–165.

79. Fournier-Viger, P, Cheng, C, Cheng, Z, Lin, JCW, Selmaoui-Folcher, N. Mining Significant Trend Sequences in Dynamic Attributed Graphs. *Knowledge-Based Systems*, 2019, to appear.

80. Inokuchi, A, Washio, T. A fast method to mine frequent subsequences from graph sequence data. In: *Proc. Eighth IEEE International Conference on Data Mining*. Pisa, Italy, 15-19 December, 2008:303–312.

81. Inokuchi, A, Washio, T. Mining frequent graph sequence patterns induced by vertices. In: *Proc. 2010 SIAM Intern. Conf. on Data Mining.*, Columbus, Ohio, USA, 29 April - 1st May, 2010:466–477.

82. Jeltsch, E, Kreowski, HJ. Grammatical inference based on hyperedge replacement. In: *Proc. Intern. Workshop on Graph Grammars and their Application to Computer Science*. Bremen, Germany, 5-9 March, 1990:461–474.

83. Pei, J, Han, J, Mortazavi-Asl, B, Wang, J, Pinto, H, Chen, Q, Dayal, U, Hsu, MC. Mining sequential patterns by pattern-growth: The prefixspan approach. *IEEE Transactions on knowledge and data engineering*. 2004, 16(11):1424–40.

84. Sohail, M, Irshad, A. A Graph Theory Based Method to Extract Social Structure in the Society. In: *Proc. 1st Intern. Conf. on Intelligent Technologies and Applications*, Bahawalpur, Pakistan, 23-25 October, 2018:437-448.

85. Bonchi, F, Gionis, A, Berlingerio, M, Bjorn, B. Network graph evolution rule generation: U.S. *Patent Application* 15/345,242[P]. 2017-2-23.

86. Richter, MJ, Kelly, MW, Haugen, A, Flores, EN. Client-side modification of search results based on social network data: U.S. *Patent Application* 10/296,547[P]. 2019-5-21.

87. Milo, R, Shen-Orr, S., Itzkovitz, S, Kashtan, N, Chklovskii, D, Alon, U. Network motifs: simple building blocks of complex networks, *Science*, 2002, 298(5594):824–827.

88. Yaveroglu, ON, Malod-Dognin, N, Davis, D, Levnajic, Z, Janjic, V, Karapandza, R, Stojmirovic, A, Przulj, N. Revealing the hidden language of complex networks, *Scientific Reports*, 2014, 4:1–9.

89. Nishioka, C, Scherp, A. Analysing the Evolution of Knowledge Graphs for the Purpose of Change Verification. In: *Proc. 2018 IEEE 12th International Conference on Semantic Computing*, Laguna Hills, CA, USA, 31 January - 2 February, 2018:25–32.

90. Ray, A, Holder, LB, Choudhury, S. Frequent subgraph discovery in large attributed streaming graphs. In: *Proceedings 3rd Intern. Conf. on Big Data, Streams and Heterogeneous Source Mining: Algorithms, Systems, Programming Models and Applications*, New York, USA, 24 August, 2014:166–181.

91. Leung, CK, Cuzzocrea, A. Frequent subgraph mining from streams of uncertain data. In: *Proc. 8th Intern. C* Conference on Computer Science and Software Engineering*, Yokohoma, Japan, 13-15 July, 2015:18–27.

92. Javed, M, Abgaz, YM, Pahl, C. Graph-based discovery of ontology change patterns. In: *Proc. Joint Workshop on Knowledge Evolution and Ontology Dynamics*, Bonn, Germany, 24 October, 2011.

93. Cheng, X, Su, S, Xu, S., Xiong, L, Xiao, K., Zhao, M. A Two-Phase Algorithm for Differentially Private Frequent Subgraph Mining. *IEEE Transactions on Knowledge and Data Engineering*, 2015:30(8), 1411–1425.

94. Yun, U, Lee, G, Kim, CH. The smallest valid extension-based efficient, rare graph pattern mining, considering length-decreasing support constraints and symmetry characteristics of graphs. *Symmetry*, 2016, 8(5):32.

95. Lee, G, Yun, U, Kim, D. A weight-based approach: frequent graph pattern mining with length-decreasing support constraints using weighted smallest valid extension. *Advanced Science Letters*, 2016, 22(9):2480–2484.

96. Yun, U, Kim, D, Yoon, E, Fujita, H. Damped window based high average utility pattern mining over data streams. *Knowledge-Based Systems*, 2018, 144:188–205.

97. Lee, J, Yun, U, Lee, G, Yoon, E. Efficient incremental high utility pattern mining based on pre-large concept. *Engineering Applications of Artificial Intelligence*, 2018, 72:111–123.