Mining Attribute Evolution Rules in Dynamic Attributed Graphs

Philippe Fournier-Viger[1], Ganghuan He[1],
Jerry Chun-Wei Lin[2], and Heitor Murilo Gomes[3]

[1] Harbin Institute of Technology (Shenzhen), Shenzhen, China
[2] Western Norway University of Applied Sciences (HVL), Bergen, Norway
[3] University of Waikato, Waikato, New Zealand
philfv@hit.edu.cn, heganghuan@gmail.com, jerrylin@ieee.org,
heitor.gomes@waikato.ac.nz

Abstract. A dynamic attributed graph is a graph that changes over time and where each vertex is described using multiple continuous attributes. Such graphs are found in numerous domains, e.g., social network analysis. Several studies have been done on discovering patterns in dynamic attributed graphs to reveal how attribute(s) change over time. However, many algorithms restrict all attribute values in a pattern to follow the same trend (e.g. increase) and the set of vertices in a pattern to be fixed, while others consider that a single vertex may influence its neighbors. As a result, these algorithms are unable to find complex patterns that show the influence of multiple vertices on many other vertices in terms of several attributes and different trends. This paper addresses this issue by proposing to discover a novel type of patterns called *attribute evolution rules* (AER). These rules indicate how changes of attribute values of multiple vertices may influence those of others with a high confidence. An efficient algorithm named AER-Miner is proposed to find these rules. Experiments on real data show AER-Miner is efficient and that AERs can provide interesting insights about dynamic attributed graphs.

Keywords: Dynamic Graphs · Attributed Graphs · Pattern Mining · Attribute Evolution Rules.

1 Introduction

In the last decades, more and more data has been collected and stored in databases. In that context, graphs are playing an increasingly important role because they can model complex structures such as chemical molecules, social networks, computer networks, and links between web pages [16, 6, 5, 11, 7]. To discover interesting knowledge in graphs, algorithms have been proposed to mine various types of patterns such as frequent subgraphs, trees, paths, periodic patterns and motifs [10, 16]. However, many studies consider that graphs are static. However, in real life, graphs often evolve, and studying these changes can provide crucial information. Graph data can be encoded as *dynamic graphs* to consider

temporal information, i.e., graphs observed at different timestamps, where edges, vertices, and labels may change. Several traditional pattern mining tasks have been extended to cope with dynamic graphs [3, 8]. However, most algorithms can only handle graphs where each edge or vertex is described using one label. But for many applications such as social network mining, it is desirable to describe graph vertices using multiple attributes (e.g. each person may have attributes such as age, gender, location and musical tastes).

To address this issue, a generalization of dynamic graphs has been studied, called *dynamic attributed graphs*, where vertices are described using multiple continuous attributes [6, 5]. This representation allows to store rich information about vertices. Several algorithms have been designed to mine patterns in dynamic attributed graphs to reveal interesting attribute changes over time [5–7, 11]. Although those algorithms have several useful applications, patterns have a simple structure and the algorithms impose many restrictions. For example, Desmier et al. [6] proposed to discover sets of vertices in a dynamic graph, where attributes change in the same way over consecutive timestamps. Hence, patterns involving different types of changes (trends) cannot be found. Cheng et al. [5] partly solved that problem by proposing to find sequences of vertex sets that can contain different trends and attributes. However, a pattern is not allowed to match with more than a vertex set and no measure of confidence is used. Hence, spurious patterns may be found, containing uncorrelated changes. Algorithms by Kaytoue et al. [11] and Fournier-Viger et al. [7] find patterns involving various trends but focus on the influence of single vertices on their neighbors. In other words, these algorithms cannot find complex patterns that show the influence of multiple vertices on other vertices.

This paper addresses these issues by proposing to discover a novel type of patterns called *Attribute Evolution Rsules* (AER). which indicate how changes of attribute values of multiple vertices may influence those of others with a high confidence. The contributions of this study are as follows. The problem of mining the novel pattern type of AERs is defined and its properties are studied. The algorithm relies on frequency and confidence measures inspired by association rule mining [1] to ensure that changes in patterns have a strong correlation. AERs are easy to interpret. They indicate likely attribute changes for a subgraph following some attribute changes. Such rule can be useful to predict the future status of a subgraph or to compress a subgraph. An efficient algorithm named AER-miner is proposed to extract these pattern from a dynamic attributed graph. An experimental evaluation was done on two real datasets (airport flight and research collaboration graphs), which shows that the algorithm is efficient and that insightful patterns are found that could not be revealed by previous algorithms. Moreover, two synthetic datasets are generated for experiments.

The rest of this paper is organized as follows. Section 2 reviews related work. Section 3 introduces preliminaries and defines the proposed problem of mining attribute evolution rules. Then, Section 4 describes the designed AER-Miner algorithm, Section 5 presents the experimental evaluation, and Section 6 draws a conclusion and discusses future work.

2 Related work

Recently, a large and growing body of work aimed at mining patterns in dynamic attributed graphs, where vertices are annotated with one or more continuous attributes. The first work in this direction was done by Jin et al [10]. They proposed an algorithm, which first transforms a dynamic attributed graph into a trend graph. A trend graph is a representation of a dynamic attributed graph where attribute values are replaced by trends indicating whether an attribute value has increased, decreased or stayed the same for two consecutive timestamps. Then, the algorithm mines a type of patterns called *trend motif* from the trend graph, which is a connected subgraph where all vertices display the same attribute change (e.g. an increase). An important limitation of that algorithm is that it can only process graphs having a single attribute (called a weighted dynamic graph), and all vertices of a pattern must follow the same trend.

Then, several studies proposed to mine other types of patterns in dynamic attributed graphs using the trend graph representation. To consider multiple attributes, Desmier et al [6] proposed to mine *cohesive co-evolution patterns* in dynamic attributed graphs. A cohesive co-evolution pattern is a set of vertices that show a same trend during a time interval for one or more attributes, and appear frequently over time. Limitations of this work are that vertices may not be connected and these patterns do not allow to see how a change may influence another change since patterns describe a single time interval.

To study how some changes may influence the structure of a graph, Kaytoue et al [11] proposed to mine *triggering patterns*. 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 important limitation of this work is that each pattern consider changes for a single node. Thus, these patterns cannot explain how the attributes of one or more nodes may influence each other. Moreover, a strong restriction is that all rules have a fixed consequent.

Then, Cheng et al. [5] addressed some of these limitations with a novel pattern type named *recurrent patterns*. A recurrent pattern indicates how attribute values have evolved for a set of vertices over more than one time interval. However, a major limitation of this study is that the set of vertices is fixed for a pattern. Thus, this approach does not allow finding general patterns occurring for several sets of vertices having the same topological structure. Moreover, there is no measure of confidence that a change will likely be followed by another change. Hence, spurious patterns may be found containing changes that are uncorrelated to the following changes.

Recently, Fournier-Viger et al [7] addressed this latter issue by proposing a pattern named *significant trend sequence* indicating a strong correlation between some attribute changes. But this study only considers the very specific case where a node's attributes influence its neighbors' attributes. Thus, it ignores the case where multiple nodes may influence other node's attributes.

In summary, most of the above studies have one or more of the following important limitations: to consider a single attribute [10], to consider a single time interval [10, 6], to mine a set of vertices that may not be connected [6, 5], to

consider that all vertices must follow the same trend(s) [10, 6], to consider only the influence of a single node on its neighbors [7], and to not assess whether a change is correlated with a following change [5].

This paper address these issues by proposing a new type of patterns named *attribute evolution rules*. It is a type of rules of the form $A \rightarrow C$ where the antecedent and consequent describe how some attributes have changed for a connected subgraph at two consecutive time intervals. This type of rules is designed to reveal the influence of attribute changes from multiple nodes on those of multiple other nodes, a type of relationship that could not be revealed by prior work. To ensure that strong rules are found, a confidence measure and a lift measure are used inspired by studies on association rule mining [1].

The work that is the closest to the current work is that of Berlingerio et al [2], which developed an algorithm to mine rules called *graph evolution rules* (GER) in dynamic graphs. A GER indicates that a subgraph having a given topological structure may evolve into another structure thereafter. Another similar concept is that of *link formation rules* (LFR) [13], proposed to study the conditions that result in edges addition in a dynamic graph. A related study also proposed to find correlation and contrast link formation rules [14]. However, a limitation of these studies is that they only handle simple dynamic graphs for the case of edge addition, and do not consider edge or node deletion and relabeling. To address this problem, Scharwachter et al [15] designed an algorithm named *EvoMiner* to mine rules with both topology and label evolution. But most work on rule mining in dynamic graphs only consider topological evolution rather than label evolution, and are restricted to dynamic graphs containing one attribute. This is unsuitable for real-life applications where graphs have many attributes and studying how they influence each other may reveal useful information.

3 Preliminaries and Problem definition

This section first introduces preliminaries related to dynamic attributed graphs and then defines the proposed problem.

Definition 1 (Graph). *A* graph *is a tuple $G = (V, E)$ where V is a vertex set, and $E \subseteq V \times V$ is an edge set.*

Definition 2 (Dynamic attributed graph). *A* dynamic attributed graph *is a sequence of attributed graphs $\mathcal{G} = \langle G_{t1}, G_{t2}, \ldots, G_{t_{max}} \rangle$ observed at some timestamps $t_1, t_2, \ldots t_{max}$. An attribute graph G_t is a tuple $G_t = (\mathcal{V}_t, \mathcal{A}_t, E_t, \lambda_t)$, where $\mathcal{V}_t$ is a set of vertices, $\mathcal{A}_t$ is a set of attributes, $E_t \subseteq \mathcal{V}_t \times \mathcal{V}_t$ is a set of edges, and $\lambda_t : \mathcal{V}_t \times \mathcal{A}_t \rightarrow \mathbb{R}$ is a function that associates a real value to each vertex-attribute pair, for the timestamp t.*

For instance, Fig. 1 A) shows a dynamic attributed graph observed at timestamps $t_1, t_2 \ldots t_4$, containing two vertices denoted as 1 and 2, connected by a single edge, and where vertices are described using three numerical attributes a,

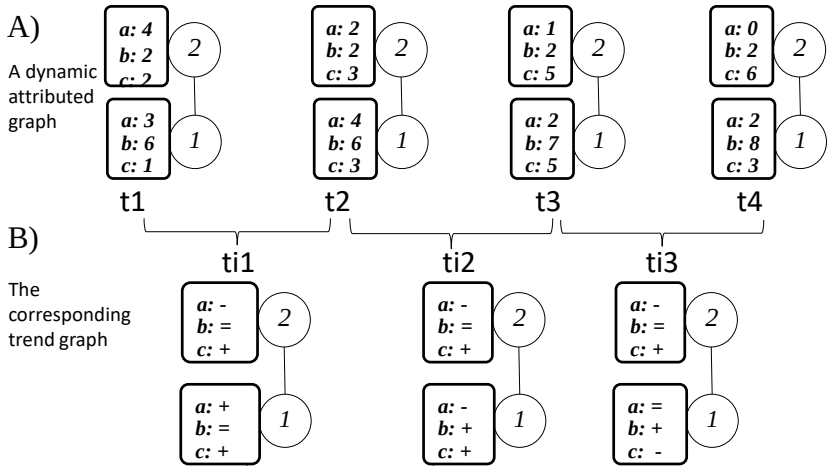

Fig. 1. A) a dynamic attributed graph, B) a trend graph

b and c. It is to be noted that the topological structure is not required to stay the same for different timestamps.

To analyze how attribute values change over time in a dynamic attributed graph, a popular approach is to convert it into a trend graph [5–7, 10]. This transformation consists of calculating trends for each time interval (two consecutive timestamps). A trend indicates whether an attribute value has increased, decreased or stayed the same $\{+, -, =\}$ during a time interval.

Definition 3 (Trend graph). *Let there be a dynamic attributed graph* $\mathcal{G} = \langle G_{t1}, G_{t2}, \ldots, G_{t_{max}} \rangle$ *observed at some timestamps* $t1, t2, \ldots t_{max}$*. Let there be a set of trends* Ω *which are set of discrete value indicating attributes status. The trend graph corresponding to* $\mathcal{G}$ *is a dynamic attributed graph* $\mathcal{G}' = \langle G'_1, G'_2, \ldots, G'_{max-1} \rangle$*, where* $G'_k = (\mathcal{V}'_k, \mathcal{A}'_k, E'_k, \lambda'_k)$ *for* $1 \leq k \leq max-1$ *is an attributed graph where* $\lambda'_k : \mathcal{V}'_k \times \mathcal{A}'_k \rightarrow \Omega$ *is a function that associates a symbol to each vertex-attribute pair and such that (1)* $V'_k = V_k$*, (2)* $E'_k = E_k$*, (3)* $\mathcal{A}'_k = \mathcal{A}_k$*, (4)* $\exists (v, a, +) \in \lambda'_k$ *iff* $\lambda_{k+1}(v, a) - \lambda_k(v, a) > 0$*,* $\exists (v, a, -) \in \lambda'_k$ *iff* $\lambda_{k+1}(v, a) - \lambda_k(v, a) < 0$*, and* $\exists (v, a, =) \in \lambda'_k$ *iff* $\lambda_{k+1}(v, a) = \lambda_k(v, a)$*. In other words, the* k*-th graph* G'_k *of a trend graph indicates how attribute values have changed in the* ***time interval*** *from timestamp* k *to* $k + 1$*.*

For instance, Fig. 1 B) shows the trend graph corresponding to the dynamic attributed graph of Fig. 1 A). The trend graph has three time intervals $ti1$, $ti2$ and $ti3$, representing timestamps $t1$ to $t2$, $t2$ to $t3$, and $t3$ to $t4$, respectively. At $ti1$, the attribute a of vertex 2 has a $-$ value because its value decreased from timestamps $t1$ to $t2$.

In this project, $\Omega = \{+, -, =\}$ indicating that an attribute value has increased, decreased or stayed the same. But without loss of generality, continuous attributes could be mapped to more than three symbols such as $++, +, =, -, --$

to distinguish between small changes and larger ones. Moreover, to avoid detecting very small changes, a constant greater than zero may be used in Definition 3.

Definition 4 (Attribute evolution rule). *An attribute evolution rule is a tuple* $R : (V, E, \lambda_{before}, \lambda_{after})$ *that indicates how the attribute values of a connected subgraph* (V, E) *have evolved for two consecutive time intervals in a trend graph* $\mathcal{G}'$*. The subgraph* (V, E) *is composed of a vertex set* V *and an edge set* $E \subseteq V \times V$*. The relations* λ_{before} *and* λ_{after} *specify the attribute values of the vertices at two consecutive time intervals, and are defined as* $\lambda_{before} : V \times \mathcal{A} \rightarrow \Omega$ *and* $\lambda_{after} : V \times \mathcal{A} \rightarrow \Omega$*, respectively. Furthermore, it is required that for all* $v \in V$*, there exists some attributes* $a, b \in \mathcal{A}$ *and some values* $\omega, \gamma \in \Omega$ *such that* $(v, a, \omega) \in \lambda_{before}$ *and* $(v, b, \gamma) \in \lambda_{after}$*. In other words, each vertex of an AER must be described using at least one attribute. The antecedent and consequent of the rule* R *are defined as* (V, E, λ_{before}) *and* (V, E, λ_{after})*, respectively.*

For instance, consider the four attributes $A = \{a, b, c, d\}$ of the trend graph of Fig. 1 B). Fig. 2 shows an attribute evolution rule consisting of three vertices $V = \{x, y, z\}$ and two edges $E = \{(x, y), (y, z)\}$ indicating how attributes values have changed for two successive time intervals. For instance, it indicates that the attribute a of vertex x has increased $(a+)$ and the attribute b of vertex z has decreased $(b-)$, which then caused vertex y's attribute c to increase and attribute d to decrease at the next timestamp.

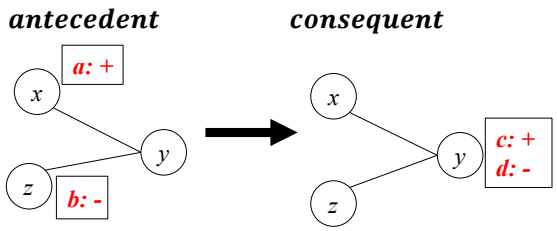

Fig. 2. An attribute evolution rule

An attribute evolution rule R may have multiple occurrences in a trend graph, called *matches*. A match of a rule R is an injective mapping between the vertices of the rule and a set of vertices of two consecutive time intervals of a trend graph. It is formally defined as follows.

Definition 5 (Matches of an attribute evolution rule). *Let there be a trend graph* $\mathcal{G}'$ *and a rule* $R : (V, E, \lambda_{before}, \lambda_{after})$*. The rule* R *is said to have an match* ϕ *from time interval* j *to* $j+1$ *iff there exists a subgraph* (V_S, E_S) *such that* $V_S \subseteq \mathcal{V'}_j \cap \mathcal{V'}_{j+1}$*.* $E_S \subseteq \mathcal{E'}_j \cap \mathcal{E'}_{j+1}$*, and* ϕ *is a bijective mapping* $\phi \subseteq V \times V_S$ *such that* $\exists (v_x, v_y) \in E \Leftrightarrow \exists (\phi(v_x), \phi(v_y)) \in E_S$*. Moreover, it is required that* $\forall (v, a) \in \lambda_{before} \Rightarrow \exists (\phi(v), a) \in \lambda'_j$ *and* $\forall (v, a) \in \lambda_{after} \Rightarrow \exists (\phi(v), a) \in \lambda'_{j+1}$*.*

Based on the concept of match, the support (occurrence frequency) of an AER in a trend graph can be defined. However, one needs to be careful about how to define the support because a subgraph may have many matches and some of them may overlap [4]. If one simply defines the support of a subgraph as its number of matches, then the support measure is neither monotonic nor anti-monotonic, and thus would not allow to reduce the search space, but this is important for developing an efficient pattern mining algorithm. To obtain an anti-monotonic support measure for attribute evolution rules, this paper defines the support as follows, inspired by the concept of minimum image based support used in frequent subgraph mining in a single graph [4].

Definition 6 (C-support of a rule). *Let* $matches_{j,j+1}(R, \mathcal{G}')$ *be all matches of an attribute evolution rule* R *in a trend graph* $\mathcal{G}'$ *for time interval* j *to* $j+1$. *The* c-support *of* R *in* $\mathcal{G}'$ *is defined as* $support(R, \mathcal{G}') = \sum_{j=1...max-2} min_{v \in V} |\{\phi(v) : \phi \in matches_{j,j+1}(R, \mathcal{G}')\}|$. *In other words, the c-support of* R *for two consecutive time intervals is the least number of distinct* consequent *nodes (from* $\mathcal{G}'$*) that a vertex from* R *is mapped to. And the c-support of* R *in the whole trend graph* $\mathcal{G}'$ *is the sum of* R*'s support for all consecutive time intervals.*

For example, consider the small trend graph of Fig. 3 (left), where some irrelevant attribute values have been omitted. Fig. 3 (right) shows the c-support of three AERs, having 3, 2, and 3 matches, respectively, and a c-support of 2. A proof that the c-support measure is anti-monotonic is given below.

Lemma 1 (Anti-monotonicity of the c-support). *If an AER* R_2 *is an attribute extension of another rule* R_1*, then* $support(R_1) \geq support(R_2)$.

Proof. Let γ be the c-support of a rule R_1, and R_2 be an attribute extension of R_1. Since the c-support is the least number of distinct *consequent* nodes, the total number of different consequent of R_1 is γ. To obtain R_2, an attribute is added to R_1, and each mapping of R_1 can either be extended (+1) with the attribute or not (+0) to obtain a mapping for R_2. Hence, $support(R_2) \leq \gamma \leq support(R_1)$.

Though the c-support measure is useful to filter infrequent patterns, it is desirable to also assess how correlated the attributes of a rule are to filter out spurious rules. In pattern mining, some popular measures to assess the correlation between the consequent (C) and antecedent (A) of a rule $A \rightarrow C$ that do not consider time are the confidence and lift [1, 12]. The confidence of a rule is the ratio of its support to that of its antecedent, that is $conf(A \rightarrow C) = sup(A \cap C)/sup(A)$, which is an estimation of $P(C|A) = P(A \cap C)/P(A)$. But a drawback of the confidence is that it does not consider $P(C)$. The lift addresses this problem. For this reason, we use the *lift* as main measure to select interesting rules. The lift is defined as $lift(A \rightarrow C) = P(A \cap C)/[P(A) \times P(C)] = conf(A \rightarrow C)/P(C)$. The confidence can be rewritten as $P(C|A) = [P(A|C) \times P(C)]/P(A)$ by the Bayes's theorem. Based on this observation and because $P(C)$ for AERs is constant while $P(A)$ can contain many vertices and change, we redefine the confidence as $P(A|C)$, which we call the *confidence based on consequent (c-confidence)*. This

measure can also find strongly correlated patterns but is easier to calculate than the original confidence. Then, the lift can be rewritten as $P(A|C)/P(A)$. The proposed AER-Miner algorithm checks both the c-confidence and lift of rules to filter spurious rules.

Definition 7 (C-confidence of a rule). *The c-confidence of an AER* $R : (V, E, \lambda_{before}, \lambda_{after})$ *in a trend graph* $\mathcal{G}'$ *is defined as* $conf(R, \mathcal{G}') = Support(R, \mathcal{G}')\ / Support(Consequent, \mathcal{G}')$.

Definition 8 (Expected confidence of an antecedent attribute). *The Expected confidence of an antecedent attribute a in a trend graph* $\mathcal{G}'$ *is defined as* $expectedConf(a) = P(a)$ *if there is no rule consequent and as* $expectedConf(a) = P(a|c)$ *if there is a consequent attribute c.*

Definition 9 (Lift of a rule). *The lift of an AER R in a trend graph* $\mathcal{G}'$ *is defined as* $lift(R, \mathcal{G}') = conf(R, \mathcal{G}')\ /\ expectedConf(antecedent,\ \mathcal{G}')$. *The expected confidence of the antecedent attribute is the probability that its attributes will appear without other conditional influence. The lift is the ratio of the real probability to the expected probability and it can measure the effect of adding an attribute to the antecedent of a smaller pattern. The lift can thus assess if the consequent and antecedent are correlated. A lift less than, equal to, and greater than 1, indicates a negative correlation, no correlation and a positive correlation, respectively.*

For instance, consider the trend graph of Fig. 3 (left), and that each trend $\{-, =, +\}$ has a uniform occurrence probability (each attribute's expected confidence is 1/3). Fig. 3 (right) shows the c-support and c-confidence of three rules. The lift of rule $< (a+) > \rightarrow < (c+) >$ is $(3/5)/(1/3) = 9/5$. The lift of $< (b-) > \rightarrow < (c+) >$ is $(2/5)/(1/3) = 6/5$. The c-confidence of $< (a+), (b-) > \rightarrow < (c+) >$ can be calculated when adding attribute $b-$ to rule $< (a+) > \rightarrow < (c+) >$, as $conf = P(b- \mid < (a+) > \rightarrow < (c+) >) = 2/3$. Thus, its lift is 5/3.

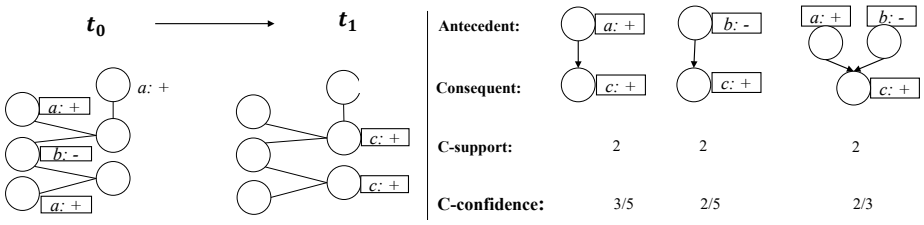

Fig. 3. The c-support and c-confidence of three AERs

Definition 10 (Problem setting). *Given a dynamic attributed graph* $\mathcal{G}$, *a minimum support threshold minsup, a minimum confidence threshold* $minconf$, *and a minimum lift threshold* $minlift$, *the problem of AER mining is to output*

all valid AERs. A rule R is ***frequent*** *if* $support(R, \mathcal{G}') \geq minsup$. *A rule is said to be* ***valid*** *if it is frequent,* $conf(R, \mathcal{G}') \geq minconf$ *and* $lift(R, \mathcal{G}') \geq minlift$.

The traditional problem of mining frequent patterns in a graph is a hard problem because the number of patterns is very large and it requires to do subgraph isomorphism checking, which is an NP-complete problem [4]. The problem of AER mining is more difficult for two reasons. First, the graph is dynamic and thus finding matches of each rule must be done for many time intervals. Second, considering many attributes greatly increases the number of potential patterns. Let there be a trend graph containing v distinct vertices and e distinct edges transformed into a trend graph using a set of trends Ω. The number of possible attribute value combinations for each vertex is $(2^{|a|} - 1) \times |\Omega|$. The number of edges combinations to create a subgraph is $(2^{|e|} - 1)$ if we ignore the requirements that subgraphs must be connected [4]. And since a rule has an antecedent and consequent and there are v vertices, the search space is in the worst case roughly $2 \times (2^{|e|} - 1) \times v \times (2^{|a|} - 1) \times 3$, so a pruning strategy must be used.

4 The AER-Miner Algorithm

This section introduces the proposed AER-Miner algorithm to efficiently find all AERs in a dynamic attributed graph (or trend graph). An attribute evolution rule R is a tuple of the form $R : (V, E, \lambda_{before}, \lambda_{after})$, where V_{before} and V_{after} are relations mapping nodes of the rule's antecedent and consequent to attribute values (for two consecutive timestamps), and where each node may be described using multiple attributes. Hence, the structure of an AER is relatively complex. Thus, rather than trying to enumerate all AERs directly, the proposed algorithm first finds core patterns, which are a simplified form of AERs. AER-Miner performs a breadth-first search using a generate-candidate-and-test strategy to explore the search space of core patterns. Then, core patterns are merged to obtain the AERs. The benefit of using this approach is that part of the AER mining problem can be solved using a modified frequent subgraph mining algorithm. The following paragraphs describe the main steps of AER-Miner and how it reduces the search space. Then, the pseudo-code is presented.

Step 1: Generating 1-size Core Patterns. The algorithm first considers each attribute from the dynamic attributed graph to generate core patterns having that attribute as consequent.

Definition 11 (Core Pattern). *A core pattern is an AER composed of a consequent node and several antecedent nodes, where each consequent-antecedent node pair is connected. Moreover, each node is described using a single attribute. We define a k-size pattern as a core pattern whose total vertice count is k. A core pattern of size k >= 2, can be considered as an attribute evolution rule.*

[4] It was observed using computer simulations that the number of connected labeled graphs with $v = 2, 3, 4, 5, 6, 7$, and 8 nodes is 1, 4, 38, 728, 26,704, 1,866,256, and 251,548,592, respectively (https://oeis.org/A001187).

AER-Miner calculates the expected confidence of each attribute, that is without considering the impact of other attributes.

Step 2: Extending Core Patterns. Then, AER-Miner extends the initial core patterns to generate larger core patterns. A generate-candidate-and-test approach is utilized where a core pattern is extended by adding a new node with an attribute to the antecedent node list of a pattern to obtain a novel core pattern. This is done iteratively following a breadth-first search. During an iteration, k-1 size patterns are combined to generate k-size patterns, and this process ends when no new pattern can be generated. To avoid generating a same pattern more than once, attributes are sorted according to the lexicographical order, and an attribute is used to extend a core pattern only if it is larger than the last attribute in the pattern's antecedent node list. To reduce the search space and filter many uninteresting patterns, attributes that have no changes for an attribute (=) are not used for extending core patterns.

Moreover, the support measure is used for reducing the search space. While some other studies have defined the support of a pattern as the number of its instances (called matches), this measure is not anti-monotonic. In other words, an extension of a core pattern may have more, the same number, or less matches. To be able to reduce the search space, the redefined support measure (Definition 6) is used, which is anti-monotonic. Thus, if a core patterns has a support less than $minsup$, it can be safely ignored as well as all its extensions. This search space pruning property can considerably reduce the search space.

Besides, the lift measure (Definition 9) is also used to reduce the search space. If the lift of a core pattern is less than $minlift$, it is discarded and all its extensions are ignored. The proof that the lift is anti-monotonic w.r.t core pattern extensions is omitted due to the page limitation.

Step 3: Filtering and Merging Core Patterns to Obtain AERs. After obtaining all core patterns, AER-Miner filters core patterns based on their confidence. The reason for filtering patterns at Step 3 rather than Step 2 is that the confidence is not anti-monotonic. In other words, extending a rule antecedent with more attributes may result in a rule having a greater, smaller or equal confidence. For example, the confidence of a rule $(A) \rightarrow (C)$ may be larger than that of a rule $(A, B) \rightarrow (C)$, where A and B are two attributes of different nodes. The reason is that A and B may each have a positive correlation with C.

Thereafter, the remaining core patterns are merged to obtain the AERs that respect the pattern selection conditions. A pair of core patterns is merged to generate an AER if (1) they have the same antecedent attributes and (2) they have the same consequent. A merge operation is considered successful if more than 90% of the support is retained.

The Pseudocode. AER-Miner (Algorithm 1) takes as input a dynamic attributed graph $\mathcal{G}$ or trend graph $\mathcal{G}'$, and the $minsup$, $minlift$ and $minconf$ thresholds. The algorithm first initializes a map structure for storing candidate core patterns. All patterns having a single consequent attribute are added to that map (Line 1). Then, all possible core patterns are generated and stored in a list (Line 2). Then a loop is performed to iteratively generate larger core patterns

(Line 3 to 21). Finally, uninteresting patterns are filtered using $minconf$, and others are merged to obtain the set of AERs.

Algorithm 1: The AER-Miner algorithm

input : a dynamic attributed graph $\mathcal{G}$ or trend graph $\mathcal{G}'$, the $minsup$, $minlift$ and $minconf$ thresholds
output: all the valid attribute evolution rules

1 Initialize a core pattern Map $map_{candidates}$ <core pattern,instances> for growing patterns. Initially, each pattern contains one consequent attribute.
2 Initialize a list $list_{patterns}$ for storing all possible core patterns.
3 **while** $map_{candidates} \neq \emptyset$ **do**
4 $\quad map_{k+1sizecandidate} \Leftarrow \emptyset$
5 $\quad$ **foreach** $candidate \in map_{candidates}$ **do**
6 $\quad\quad pattern \leftarrow candidate.key$
7 $\quad\quad instances \leftarrow candidate.value$
8 $\quad\quad$ **foreach** $attr \in attributelist$ **do**
9 $\quad\quad\quad$ **if** $attr \geq$ *the last attribute of pattern and* $attr \neq' ='$ **then**
10 $\quad\quad\quad\quad newPattern \leftarrow pattern \cup attr$
11 $\quad\quad\quad\quad newInstances \leftarrow extendInstances(pattern, attr, instaces)$
12 $\quad\quad\quad\quad support \Leftarrow sizeofnewInstance$
13 $\quad\quad\quad\quad lift, confidence \leftarrow calLiftAndConfi(pattern, attr, instaces)$
14 $\quad\quad\quad\quad$ **if** $support \geq minsup$ *and* $lift \geq minlift$ **then**
15 $\quad\quad\quad\quad\quad put < newPattern, newInstance > \in map_{k+1sizecandidate}$
16 $\quad\quad\quad\quad$ **end**
17 $\quad\quad\quad$ **end**
18 $\quad\quad$ **end**
19 $\quad$ **end**
20 $\quad list_{patterns} \leftarrow list_{patterns} \cup map_{k+1sizecandidate}$
21 **end**
22 $list_{patterns} \leftarrow filterPatterns(list_{patterns}, minconf)$
23 Return $AERs = list_{patterns} \cup mergePatterns(list_{patterns})$

5 Experimental evaluation

Experiments were performed to evaluate the performance of AER-Miner and its ability to discover interesting patterns in real data. AER-Miner was implemented in Java and experiments were carried out on a 3.6 GHz Intel Xeon PC with 32 GB of main memory, running Windows 10. Two real world datasets were used in the experiments and two synthetic datasets were generated to assess the statistical validity of AERs. The source code of AER-Miner and datasets can be downloaded from the open-source SPMF data mining library http://www.philippe-fournier-viger.com/spmf/. The following datasets were used:

DBLP [6] is a co-authorship dataset containing data from the DBLP online bibliographic service. There are 2,723 vertices, each representing an author having published at least 10 papers in data mining and/or database conferences/journals between 1990 to 2010. This time period is divided into nine timestamps: ([1990-1994][1992-1996]...[2004-2008][2006-2010]). An edge indicates a co-authorship relation between two authors, while an attribute value indicates the number of papers published by an author in a conference/journal.

US Fight [11] contains data about US air traffic during the Katrina hurricane period (from 01/08/2005 to 25/09/2005). Vertices stand for US airports. Two vertices are connected by an edge if there was a flight connecting them during the time period.

Moreover, we randomly generated two datasets, named *synthetic-DBLP* and *synthetic-USFlight*. They have the same vertex, average edge, timestamp and attribute count as the DBLP and US Flight datasets, respectively. But attribute values of each vertex were generated following a Gaussian distribution. Characteristics of the datasets are as follows. The number of vertices, average edges per timestamps, timestamps and attribute count for DBLP and synthetic DBLP is 2,723, 10,737, 9 and 43, while for the US Flight and synthetic-USFlight, it is 280, 1,206, 8 and 8.

Statistical validation experiment. The first experiment was designed to check if AER-Miner can find statistically valid rules and determine an appropriate range of $minlift$ values to obtain valid rules. For this purpose, AER-Miner was run on each real dataset and the corresponding synthetic dataset while varying the $minlift$ parameter. Because an AER describes the correlation between the attributes of a rule's antecedent and consequent, and that synthetic datasets have the same structure as real datasets except for the randomly generated attribute values, no AER should be found in the synthetic datasets for high enough $minlift$ values. The $minlift$ threshold was increased from 1.05 (weak positive correlation) to 1.4 (strong positive correlation) while noting the number of patterns found. The $minsup$ threshold was set to a fixed value (0.004 for DBLP and 0.004 for US Flight) that is high enough to find patterns, but $minsup$ was not varied because it has a small influence on correlation.

Fig. 4 shows results for the (a) DBLP and (b) US Flight datasets. It can be observed that no AER was found in synthetic datasets in most cases, while some were found in real data. This is reasonable as synthetic datasets contain random values that are weakly correlated. Because AERs were found in synthetic data for $minlift < 1.1$, it can be concluded that this parameter should be set to a value of at least 1.1 to find valid rules. Otherwise, random AERs may be found.

Quantitative experiments. Three additional experiments were done to evaluate the influence of dataset characteristics and parameters values on the performance of AER-Miner in terms of runtime and peak memory usage.

First, the influence of a dynamic attributed graph's properties on performance was assessed. Attribute count was first varied, while parameter values where fixed ($minsup = 0.04$, $minlift = 1.3$ and $minconf = 0.3$). Fig. 5 (a) shows the influence of attribute count on runtime and memory for the DBLP

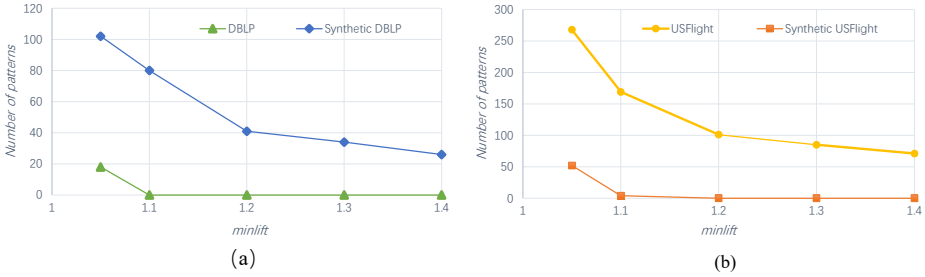

Fig. 4. Statistical validation with real and synthetic datasets.

dataset. As attribute count increased, execution times and memory consumption increased. But there is a difference between the growth rate of execution time and memory. At first, when attribute count is small, execution time increases slowly. Then, when the attribute count becomes quite large, its grows more quickly and then remains stable. For memory, it is the opposite. This is because when the attribute count is small, no attribute correlations are found. Thus, most of the memory is spent for storing the dynamic attributed graph, while as the attribute count increases, much memory is spent to store (candidate) patterns in memory.

Second, the influence of a dynamic attributed graph's graph count (number of timestamps) on performance was evaluated. Fig. 5 (b) shows results for the US Flight dataset when the algorithm's parameters are fixed ($minsup = 0.04$, $minlift = 1.3$ and $minconf = 0.3$). It is found that execution time linearly increases and memory also increases as graph count increases. This shows that AER-Miner has excellent scalability for processing numerous time periods.

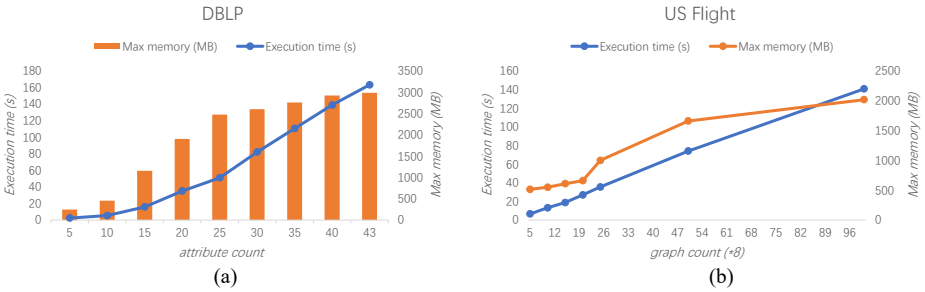

Fig. 5. Influence of attribute count and graph count on performance.

Third, an experiment was done on DBLP to evaluate the influence of the $minsup$ and $minlift$ thresholds on runtime and pattern count. Fig. 6 (a) shows results when $minsup$ is varied while $minlift = 1.3$ and $minconf = 0.3$ are fixed. Fig. 6 (b) presents results when $minlift$ is increased while $minsup = 0.004$ and $minconf = 0.2$ are fixed. It is found in Fig. 6 (a) that as the $minsup$ constraint

is less strict, runtime and pattern count are greater, and that pattern count increases dramatically for $minsup = 0$. It is observed in Fig. 6 (b) that increasing $minlift$ helps reducing the search space and that increasing $minlift$ influences less the performance for values above 1.4. This is because most spurious (random patterns) have a lift smaller than 1.4.

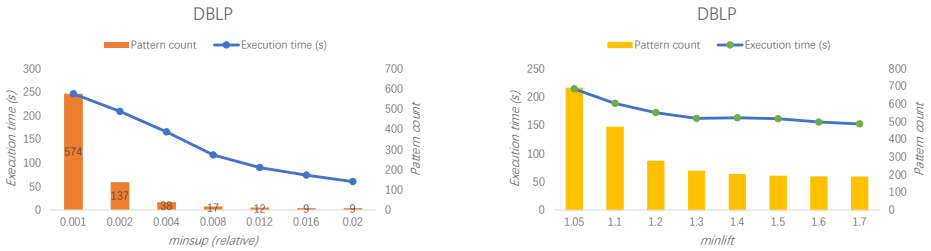

Fig. 6. Influence of $minsup$ and $minlift$ on runtime and pattern count.

Qualitative assessment. An additional evaluation was performed on the DBLP and US Flight datasets to assess the usefulness of patterns found. First, rules were extracted from DBLP using $minsup = 0.004$, $minlift = 1.5$ and $minconf = 0.3$. An example pattern found is $< (PVLDB+), (PVLDB+), (PVLDB+) > \rightarrow < (VLDB-) >$. It indicates that co-authors of an author who published more papers in PVLDB then published less papers in VLDB at the next timestamp. This is reasonable since there is a correlation between VLDB and PVLDB and a person is likely to follow trends of his co-authors. Another pattern is $< (ICDE+) > \rightarrow < (PVLDB+, EDBT+) >$, which indicates that if an author published more ICDE papers, his co-authors are more likely to publish more in PVLDB and EDBT. Second, rules were extracted using $minsup = 0.004$, $minlift = 1.24$ and $minconf = 0.3$ from US Flight to discover rules related to the impact of Hurricane Katrina on flights. The pattern $< (NbDeparture-) > \rightarrow < (NbCancelation+) >$ was found to have many occurrences. It indicates that departure cancellations caused by the hurricane are strongly correlated with a flight cancellation increase at the next timestamp. Another interesting pattern is $< (NbDeparture+), (NbDeparture+), (NbDeparture+) > \rightarrow < (NbDeparture+) >$, which indicates that flights are returning to normal after a hurricane.

6 Conclusion

This paper has proposed a novel type of patterns called attribute evolution rules, indicating how changes of attribute values of multiple vertices may influence those of others with a high confidence. An efficient algorithm named AER-Miner was proposed to find these rules. Moreover, experiments on real data have shown

that the proposed algorithm is efficient and that AERs can provide interesting insights about real-life dynamic attributed graphs. In future work, we plan to extend AER-Miner to let the user specify temporal constraints on AERs and discover concise representations of AERs.

References

1. Agrawal, R., Srikant, R., et al.: Fast algorithms for mining association rules. In: Proc. of 20th int. conf. very large data bases, VLDB. vol. 1215, pp. 487–499 (1994)
2. Berlingerio, M., Bonchi, F., Bringmann, B., Gionis, A.: Mining graph evolution rules. In: Proc. 20th European Conf. on Machine Learning and Knowledge Discovery in Databases. pp. 115–130 (2009)
3. Borgwardt, K., Kriegel, H., Wackersreuther, P.: Pattern mining in frequent dynamic subgraphs. In: Proc. of the 6th IEEE International Conference on Data Mining. pp. 1818–822 (2006)
4. Bringmann, B., Nijssen, S.: What is frequent in a single graph? In: Proc. 11th Pacific-Asia Conf. on Knowledge Discovery and Data Mining (2007)
5. Cheng, Z., Flouvat, F., Selmaoui-Folcher, N.: Mining recurrent patterns in a dynamic attributed graph. In: Proc. of 21st Pacific-Asia Conf. on Knowledge Discovery and Data Mining. pp. 631–643. Springer (2017)
6. Desmier, E., Plantevit, M., Robardet, C., Boulicaut, J.: Cohesive co-evolution patterns in dynamic attributed graphs. In: Proc. of the 15th Intern. Conf. of Discovery Science. pp. 110–124 (2012)
7. Fournier-Viger, P., Cheng, C., Cheng, Z.X., Lin, J.C.W., Selmaoui-Folcher, N.: Mining significant trend sequences in dynamic attributed graphs. Knowledge Based Systems **182**(15), 1–25 (2019)
8. Fournier-Viger, P., He, G., Cheng, C., Li, J., Zhou, M., Lin, J.C.W., Yun, U.: A survey of pattern mining in dynamic graphs. WIREs Data Mining and Knowledge Discovery p. e1372. https://doi.org/10.1002/widm.1372
9. Jin, R., McCallen, S., Almaas, E.: Trend motif: A graph mining approach for analysis of dynamic complex networks. In: Proc. of the 7th IEEE Intern. Conf. on Data Mining. pp. 541–546. IEEE (2007)
10. Kaytoue-Uberall, M., Pitarch, Y., Plantevit, M., Robardet, C.: Triggering patterns of topology changes in dynamic graphs. 2014 IEEE/ACM International Conference on Advances in Social Networks Analysis and Mining pp. 158–165 (2014)
11. Lenca, P., Vaillant, B., Meyer, P., Lallich, S.: Association rule interestingness measures: Experimental and theoretical studies. In: Quality Measures in Data Mining, pp. 51–76. Springer (2007)
12. Leung, C.W., Lim, E., Lo, D., Weng, J.: Mining interesting link formation rules in social networks. In: Proc. 19th ACM Conference on Information and Knowledge Management. pp. 209–218 (2010)
13. Ozaki, T., Etoh, M.: Correlation and contrast link formation patterns in a time evolving graph. In: Proc. Workshops of 11th IEEE International Conference on Data Mining. pp. 1147–1154 (2011)
14. Scharwächter, E., Müller, E., Donges, J.F., Hassani, M., Seidl, T.: Detecting change processes in dynamic networks by frequent graph evolution rule mining. In: Proc.16th IEEE International Conference on Data Mining. pp. 1191–1196 (2016)
15. Yan, X., Han, J.: gspan: graph-based substructure pattern mining. In: 2002 IEEE International Conference on Data Mining, 2002. Proceedings. pp. 721–724 (2002)