Knowledge and Information Systems (2020) 62:3931–3971
https://doi.org/10.1007/s10115-020-01478-9

REGULAR PAPER

Mining evolutions of complex spatial objects using a single-attributed Directed Acyclic Graph

Frédéric Flouvat[1] · Nazha Selmaoui-Folcher[1] · Jérémy Sanhes[1] · Chengcheng Mu[1] · Claude Pasquier[2] · Jean-François Boulicaut[3]

Received: 8 June 2018 / Accepted: 20 May 2020 / Published online: 24 June 2020

Abstract

Directed acyclic graphs (DAGs) are used in many domains ranging from computer science to bioinformatics, including industry and geoscience. They enable to model complex evolutions where spatial objects (e.g., soil erosion) may move, (dis)appear, merge or split. We study a new graph-based representation, called attributed DAG (a-DAG). It enables to capture interactions between objects as well as information on objects (e.g., characteristics or events). In this paper, we focus on pattern mining in such data. Our patterns, called weighted paths, offer a good trade-off between expressiveness and complexity. Frequency and compactness constraints are used to filter out uninteresting patterns. These constraints lead to an exact condensed representation (without loss of information) in the single-graph setting. A depth-first search strategy and an optimized data structure are proposed to achieve the efficiency of weighted path discovery. It does a progressive extension of patterns based on database projections. Relevance, scalability and genericity are illustrated by means of qualitative and quantitative results when mining various real and synthetic datasets. In particular, we show how such an approach can be used to monitor soil erosion using remote sensing and geographical information system (GIS) data.

This research was supported by the Project FOSTER ANR-2010-COSI-012-01 funded by the French Ministry of Higher Education and Research.

✉ Frédéric Flouvat
frederic.flouvat@univ-nc.nc

Nazha Selmaoui-Folcher
nazha.selmaoui@univ-nc.nc

Claude Pasquier
claude.pasquier@univ-cotedazur.fr

Jean-François Boulicaut
jean-francois.boulicaut@insa-lyon.fr

1 ISEA, University of New Caledonia, BP R4, 98851 Nouméa, New Caledonia

2 CNRS, I3S, Université Côte d'Azur, Nice, France

3 LIRIS UMR5205, CNRS, INSA de Lyon, 69621 Villeurbanne, France

Keywords Graph mining · Spatiotemporal data · Attributed DAG · Weighted path · Environmental monitoring

1 Introduction

Analysis and management of environmental risks are major issues in today's world. These last years, a huge amount of data has been collected to monitor the environment. One of the main challenges with such data is to understand and predict dynamics of ecosystems affected by global climate change and local human activity. However, underlying phenomena and collected data are complex. A large number of spatial objects evolve over long periods of time. Moreover, each object may be related to several heterogeneous information (e.g., landcover, weather, soil type, topology) that may explain or influence their evolution. Objects may also move and change (as illustrated in Fig. 1). For example, soil erosion may grow, shrink, move, appear, disappear, merge or split. Analysis of these data by scientists and decision makers is difficult and requires advanced data mining approaches. In such a context, supporting pattern discovery seems particularly promising, but analyzing such complex spatiotemporal phenomena remains challenging.

Graph mining has been an important topic for the data mining community over the past years. One reason for this large dissemination is that a graph structure can be used to model many problems since it can easily support entities, their attributes and relationships between entities [19]. Various types of graphs have been studied such as trees or directed acyclic graphs (DAG). The interest of DAGs is to capture dependencies or triggering relations. In this context, DAGs have been used in various application domains such as in computer science, bioinformatic or industry [17,25,49,52,63,73]. For example, References [17,52,73] have used DAGs to model computer networks, web navigation patterns, XML documents or class dependencies in software engineering. Termier et al. [63] use DAGs to represent interactions between genes. Fariha et al. [25] model human interactions in meetings using

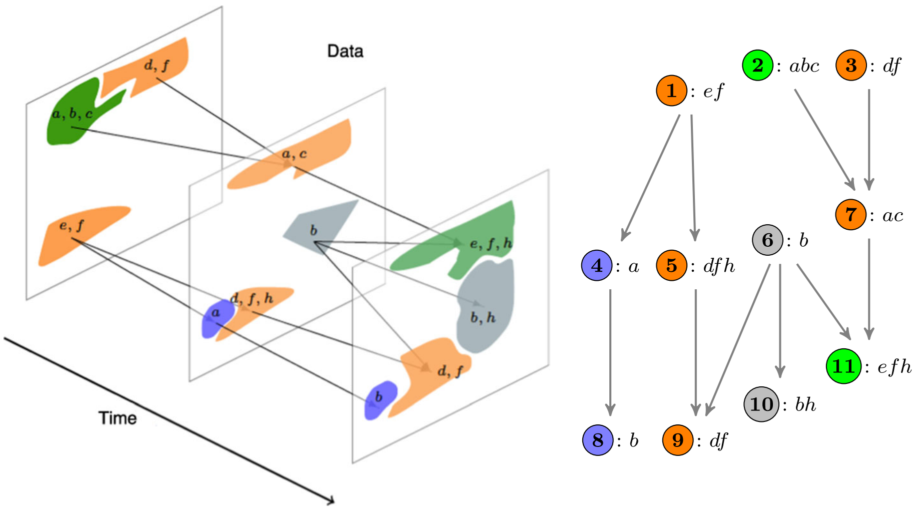


Fig. 1 Example of time series related to environmental monitoring (with moving, appearing, disappearing, merging or splitting objects) and its corresponding DAG representation

DAGs. Spatiotemporal data can also be naturally represented as a single large DAG. Vertices represent spatial objects (e.g., rain, forest, mine, road), and edges represent neighboring objects in consecutive timestamps. For example, Mohan et al. [49] use a single labeled DAG to model crime incidents. All these works focus on labeled graphs and/or transactional databases. In the first case, each vertex can only be associated with a single label, i.e., only one information (a single property or an event). In the second one, only a collection of graphs can be analyzed and not a single large graph. Their application is by construction limited to specific contexts.

To analyze complex spatiotemporal data, an alternative would be to express the studied problem into some available frameworks, such as sequential pattern discovery [2], string pattern extraction [21] or set system mining [4]. However, all these frameworks also focus on transactional databases. A natural approach would be to transform the single input graph into a transactional database (e.g., a collection of graphs or sequences). However, as shown in [60], such an approach is not scalable and it loses structural information. Moreover, it leads to several problems for pattern interpretation. For example, a single input graph can be transformed into a collection of paths (i.e., sequences) by duplicating some nodes or deleting some edges. The first solution will overexpress information and thus add a bias. The second one would result in information loss. Although such problems may be fixed thanks to a postprocessing step, it is not satisfactory.

In this paper, we model evolutions of complex spatial objects using a single-attributed DAG (a-DAG), i.e., a Directed Acyclic Graph in which vertices are labeled by some attributes. Such a data representation enables to capture information on objects and their complex dynamics such that moving, merging/splitting or appearing/disappearing. Therefore, we design a new pattern domain, called the weighted paths, to study frequent evolutions in a single a-DAG. It is based on sequential patterns [77] and path patterns [16]. It offers a good trade-off between expressiveness and complexity. Indeed, complex patterns are more difficult to mine (which impacts algorithm scalability), but also more difficult to interpret by domain experts. A weighted path is a sequence of itemsets in which a value is associated with each transition. This value represents the frequency of this transition/evolution within the path in the a-DAG (see Section 3). In addition to frequency constraint, we also consider a non-redundancy constraint. It leads to a condensed representation of frequent patterns without loss of information (i.e., all the patterns with their frequency can be derived from the computed subset without accessing the data). We propose an algorithm to compute weighted paths in a single a-DAG. It is a generic algorithm that can be used to mine any type of a-DAG and not only the ones related to spatiotemporal phenomena analysis. It is based on a depth-first search strategy, and it performs progressive pattern extensions. It recursively projects the a-DAG, and it progressively expands patterns into all projected a-DAGs. A detailed empirical study has been performed to discuss the scalability and the genericity of our approach. Its added value is mainly illustrated on real-world time series monitoring soil erosion though we also provide experimental results on a patent citation network and some synthetic datasets.

This work is a significant extension of [58]. It modeled a dengue epidemic spread thanks to an attributed DAG, and we were looking for frequent evolutions, i.e., frequent paths. This first paper shows how a-DAGs can be used to model spatiotemporal phenomena, and it introduces the weighted path pattern domain. It also provides a mining strategy based on an n-ary relation transformation. However, the algorithm from [58] is not complete while its scalability for large graph analysis remains an issue. Indeed, the largest graph mined using this first algorithm had 40,000 vertices, 120,000 edges and 7.5 attributes per vertex in average.

Our key contributions in this paper are:

1. A sound and complete depth-first search strategy to mine condensed weighted paths in a single a-DAG (with theoretical proofs for properties and completeness)
2. An optimized data structure and an algorithm to extract patterns more efficiently and in larger graphs (e.g., a graph with 200,000 vertices, 600,000 edges and 7.5 attributes per vertex in average, was mined in less than 8 min with a minimum support threshold of 10%)
3. A detailed performance analysis with various synthetic datasets and a patent citation network
4. A KDD process where this pattern mining approach is integrated to monitor soil erosion using remote sensing and GIS data.

The rest of the paper is organized as follows. Section 2 gives an overview of related works dealing with graph mining and spatiotemporal pattern discovery. Section 3 introduces our theoretical framework, i.e., the data representation, the pattern domain, the constraints and the studied mining task. Section 4 presents our algorithm to mine the weighted path patterns that satisfy user-defined constraints. Section 5 discusses performance issues, and it provides a case study where we use the weighed path pattern to study soil erosion using times series of remotely sensed data. Finally, Sect. 6 concludes the paper and gives some perspectives.

2 Related works

2.1 Spatiotemporal pattern mining

These last years, the explosion of spatial data collected by experts, sensors and satellites opens new challenges for the data mining community. In this context, many works have been done to extract spatiotemporal patterns. Initially, these works have considered the spatial dimension and the temporal dimension independently (see, e.g., colocation mining [36] and time series mining). Lately, these works have been extended to integrate both dimensions. We can distinguish two frameworks: mining trajectories of moving objects [31] and mining evolutions of properties/attributes related to spatial objects [3,5,15,49,64,67]. In our work, we focus on the second framework.

In this framework, Celik et al. [15] generalize the concept of colocation to spatiotemporal data. Spatiotemporal co-occurrence patterns are subsets of properties such that their instances are spatially close for a significant period of time. Such a concept has also been studied in other works. For example, Qian et al. [57] study Spread Patterns of spatiotemporal Co-occurrences Over Zones (SPCOZ) which represent frequent trajectories of spatial colocations. In [78], the authors study spatial patterns frequently occurring at different times (called Spatial Object Association Pattern, or SOAP) and propose to visualize their evolutions using graphs.

More generally, sequence and graph mining frameworks have also been used to analyze spatiotemporal phenomena. For example, Tsoukatos and Gunopulos [64] use sequential pattern mining to extract frequent evolutions of spatial objects. They consider evolutions of geographical regions associated with various occurring events or properties. Thus, the database is a collection of sequences of itemsets (e.g., set of environmental properties or events), each one representing the evolution of a given region. Wang et al. [67] use sequential patterns to study the spatial spread of events in predefined temporal windows. They decompose the temporal dimension in time windows of a given size (e.g., 4 days), represent space as a grid and introduce the concept of flow patterns. A flow pattern is a frequent sequence of events $< \ I_1 \rightarrow I_2 \rightarrow \cdots \rightarrow I_k \ >$ in which I_i is a set of events and each event $e(location) \in I_i$ is composed of an event type e (e.g., rain, wind) and its location. In these patterns, each set I_i is composed of spatially close events occurring at the same time, and two sets I_j and I_k are consecutive if their events are consecutive and in the same time window. In [3], authors propose a new pattern domain, called spatiosequential patterns, to study evolutions of spatial objects/regions and their neighborhood. They introduce the concept of spatial itemset, i.e., a pair of itemsets (I, I') associated with spatially close regions and denoted by $I.I'$. Then, they integrate this notion in the sequence framework. The database is a collection of sequences of itemsets representing evolution of regions such as in [64], but it also integrates a neighborhood relationship. Resulting patterns are frequent sequences of spatial itemsets $< I_1.I_1', \ I_2.I_2', \ \ldots \ I_k.I_k' >$ where I_i are events/properties (an itemset) related to a set of regions and I_i' are events/properties (an itemset) related to their neighborhood.

Graphs have also been used to study spatiotemporal phenomena. References [39,74] use graphs to represent the topology of spatial objects and their relationships. These formalisms enable to capture complex, dynamic, region-based structures, with the component regions inhabited by a multiplicity of dynamic objects. References [23,62] have worked to define qualitative spatial terminology in graph-based embeddings. References [24,46] have also proposed such models to represent complex spatial objects (but again they do not represent changes). Mohan et al. [49] consider a collection of spatiotemporal events modeled as a labeled DAG, and search frequent sub-DAGs representing events located together and occurring serially. They propose a new interest measure called the cascade participation index to represent the frequency of a pattern in such data. This measure is the minimum conditional probability of a pattern given one of its events. Thanks to the monotonicity of this measure, a levelwise algorithm has been developed to mine these patterns. Aydin and Angryk [5] study evolutions of regions w.r.t. event types. They also model such data using a labeled DAG, but they extract frequent spatiotemporal event sequences instead of sub-DAGs. Such as in the previous work, each region is associated with a single event type. They introduce a parametrized spatiotemporal "follow" relationship and a frequency measure based on the minimum participation ratios such as in [36]. The proposed algorithm adopts a pattern-growth strategy to mine frequent sequences.

2.2 Mining patterns over a collection of labeled graphs

Mining graph data has been a topic of interest for several years. Inokuchi et al. [37] propose an Apriori-like algorithm [1] for mining frequent substructures from graph data. The database is a collection of graph transactions. Each graph transaction is a labeled graph represented by its adjacency matrix. A levelwise search on the frequent canonical matrix code is performed. This study focuses on induced subgraph patterns, i.e., subgraphs satisfying the parent-descendant relationship. Kuramochi and Karypis [42] propose an improved version of [37]'s levelwise algorithm based on efficient data structures and various optimizations for candidate generation and counting. In [75], authors also study frequent subgraph mining in a collection of labeled graphs. They investigate a new approach to mine these patterns more efficiently based on a pattern growth strategy. Patterns are mapped to unique sequences (DFS codes) which are extended recursively. The proposed algorithm, called gSpan, performs a depth-first exploration of the search space and avoids costly candidate generation. Several other algorithms have been proposed to efficiently mine frequent subgraphs [22,32,53,68]. Surveys of graph mining approaches can be found in [38,69]. On another hand, many contributions focus on a specific class of graph data (e.g., trees, acyclic graphs, oriented graphs).

Their main goal is to develop dedicated approaches optimized w.r.t a specific representation. For example, several works focus on DAGs and their multiple applications [25,33,48,52,72]. Werth et al. [72] study DAG mining to optimize code compaction in computer programming. They propose an approach to mine frequent patterns in a set of labeled DAGs. Patterns are unconnected, multi- or single-rooted, and induced sub-DAGs. In a very different context, Fariha et al. [25] also exploit DAG mining to find human interaction patterns occurring in meetings. Their database is a set of human interaction flows, each one being represented by a labeled DAG. Vertices have weights to distinguish participant ranks. The aim is to find frequent weighted sub-DAGs in a collection of weighted DAGs. To solve this problem, the authors adapted [72]'s work in order to integrate weights in both the support measure and the mining algorithm.

2.3 Pattern mining in the single-graph setting

Most of the studies referenced previously deal with frequent pattern mining in a graph-transaction database. However, in many application domains, data are represented as a single (large) graph. The graph-transaction setting and the single-graph setting share common properties, but algorithms developed for the former cannot be used for the latter-whereas the opposite is true [43]. One major issue in the single-graph setting is the subgraph isomorphism test. In the graph-transaction setting, this test is stopped when the first occurrence of a subgraph is found in a transaction, whereas when dealing with a single graph, one needs to find all occurrences, which has an important impact on performances. Another problem in the single-graph context is how to define pattern frequency. Indeed, it cannot be defined as the number of transactions in which a pattern occurs. Several authors have studied this issue [12,26,40,43]. Most of them defined a frequency based on pattern occurrences. However, several occurrences may overlap, leading to a non-monotonic frequency measure. Therefore, new frequency measures have been studied.

2.4 Mining attributed graphs

In many applications, using labeled graphs to represent data is not enough to capture all available information. Attributed graphs have been introduced to deal with this limitation. An attributed graph is a graph in which each vertex is labeled by several attributes (i.e., an itemset). However, mining attributed graphs generally leads to a combinatorial explosion (i.e., the exploration of search spaces for both graphs and itemsets), which brings new challenges. Several studies related to attributed graph mining focused on discovering communities sharing similar behavior or interest (e.g., [28,41,50,61]). This problem is central in several application domains such as social network analysis or systems biology. In social networks, an important task is the identification of groups of people with strong social interactions and similar interest. In systems biology, one aim is to identify groups of genes with similar gene expressions. In that context, Moser et al. [50] introduced cohesive patterns, i.e., patterns representing subgraphs with shared itemsets. Such patterns combined principles of dense subgraphs and subspace clusters. The CoPaM algorithm was developed to extract maximal cohesive patterns based on a levelwise pattern enumeration. Fukuzaki et al. [28] worked on similar patterns though they did not consider a density constraint. Their algorithm combined a depth-first search tree and a prefix tree to efficiently extract maximal patterns. Fukuzaki et al. applied their approach to biological network (metabolic pathways) analysis for drug discovery as well as to collaborative research extraction in citation networks. To our knowledge, only [17,47,54] have investigated frequent pattern mining in attributed DAGs. Chen et al. [17] worked on frequent sub-DAG mining in a database composed of several attributed DAGs. However, they concentrated on pyramid patterns, which are DAGs with only one source node, and only considered induced sub-DAGs. Miyoshi et al. [47] mined frequent sub-DAGs in a single-attributed graph. In their work, labels and quantitative itemsets were attributed to vertices. By keeping labels, frequent pattern mining was simplified and decomposed in two steps: mining a labeled graph using gSpan algorithm [75] and mining quantitative itemsets using QFIMiner algorithm [70]. The frequency measure used was based on the minimum number of edge-disjoint occurrences. Pasquier et al. [54] propose a framework to mine frequent subgraphs in rooted DAGs. A graph is rooted if one can find a vertex v such that there exists a path between v and any other vertex of the graph. Thus, this work cannot mine graphs such as the one presented in Fig. 1. Their framework considers the transactional setting and the single graph one. It uses spanning trees of graphs and a new canonical form to explore the search space using an adaptation of a tree mining algorithm. Their work also highlights the complexity and the cost of subgraph isomorphisms in such context.

2.5 Condensed representations in graph mining

In frequent pattern mining, the number of extracted patterns can be huge (exponential in the input size in the worst case). As a consequence, numerous works have sought condensed representations of solutions. In [34], the authors proposed to study maximal frequent itemsets. While all frequent itemsets can be deduced thanks to the frequency constraint's monotonicity property, a major limitation of maximal patterns is that the frequency of sub-patterns cannot be worked out without accessing data. Several other condensed representations have been proposed to reduce the number of extracted solutions without information loss [11,14,55]. Closedness is probably the most studied constraint, and it has been applied to most pattern domains (e.g., itemsets, sequences, trees, graphs). A pattern is closed if there is no super-patterns (w.r.t. a specialization/generalization relation) with the same support (i.e., occurring in the same transactions).

Several algorithms have been designed to mine closed patterns in a collection of graphs [63,76]. Yan and Han [76] proposed to mine closed frequent graphs in graph transactions. They highlighted that optimizations and properties previously developed for itemsets and sequences did not hold in graphs. They introduced new concepts ("equivalent occurrence" and "early termination") to improve pattern pruning. These properties have been integrated in the CloseGraph algorithm, based on gSpan and its rightmost extension approach.

Termier et al. [63] proposed an algorithm called DigDag to mine closed frequent embedded sub-DAGs in a collection of labeled DAGs, where each DAG must have distinct labels. Despite this restriction, results showed that mining such sub-DAGs in a real gene network dataset (with 100 graphs/networks and 50 labels/genes) was not a trivial task. A two-step approach was developed to find the patterns. The first step mined closed frequent ancestor-descendant edges using the itemset mining algorithm LCM [66]. The second step combined these edges to discover closed frequent embedded sub-DAGs.

2.6 Mining path patterns in graphs

In our work, we focused on a particular pattern domain in graphs: paths. Mining paths in graphs has been the subject of previous studies. They highlighted the interest of such patterns in many application fields such as biochemical or gene networks analysis [20,35], communication monitoring in distributed environments (online services) or Web navigation patterns [10,16,30,51].

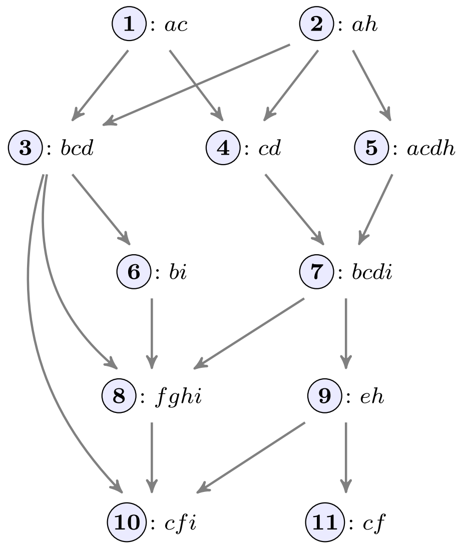
Chen et al. [16] mined frequent paths, called "path traversal patterns," in a labeled directed graph. The graph represented user access patterns in a distributed information environment where documents or objects were linked together (e.g., Web pages, online services). In their work, they did not consider backward references, because they assumed that a backward reference was made for easy navigation but not for browsing. Based on this assumption, they transformed the graph data into a set of maximal forward references, i.e., sequences. Thus, the original problem was transformed into a frequent sequence mining problem, and the graph structure was not considered. Borges and Levene [10] also worked on navigation patterns. They represented a user navigation session as a Markov chain (i.e., a labeled directed graph) and thus sought frequent paths in a collection of Markov chains. They proposed a heuristic to efficiently extract longer paths (often related to low frequency values), since their previous levelwise algorithm did not scale up in such case.

2.7 Position of our work

As highlighted previously, mining a single labeled graph and mining a collection of attributed graphs have been studied in the literature but mainly separately. Several strategies (e.g., depth first or breadth first) have been proposed. A popular strategy due to its efficiency is to progressively extend interesting patterns based on database projections. Such depth-first traversal of the search space has been proposed in [56] for mining sequential patterns. An advantage of such approach is to limit memory consumption (and as a consequence execution time) by generating and testing less patterns at the same time. However, combining structural exploration of DAGs with the combinatorial of itemsets in the single-graph setting is much more difficult. It cannot be resolved by using a simple adaptation of existing frequent pattern mining algorithms. As claimed in [43], "algorithms developed for the graph-transaction setting cannot be used to solve the single-graph setting, whereas the latter algorithms can be easily adapted to solve the former problem".

As discussed in [12,26], the main problem in the single-graph setting is overlapping embeddings. Due to these embeddings, pattern generation and frequency computation are much more difficult. For example, frequency in a transactional database (i.e., a collection of graphs) is "simply" obtained by counting the number of input graphs that contain the pattern. In the single-graph setting, we have to find all occurrences, and not only one per transaction, which is far more complex due to their possible overlapping. Defining an anti-monotone frequency constraint in such context is difficult and its computation is also a problem for scalability. For example, one approach is to find maximum independent node sets (MIS) [43], but this problem is NP-complete.

Defining a condensed representation of interesting patterns, without loss of information, is also more complex in the single-graph setting. Closed patterns in a collection of transactions (sets, sequences or graphs) are the most studied form of condensed representation. It exploits the Galois connexion that holds between transactions and patterns. An important property of the closure operator in this context is the support preservation property: Patterns and their closures have the same support. This property relies on the support definition: A pattern is counted once per transaction. In the single-graph setting, this support definition, this Galois connexion and their properties do not hold anymore. New concepts have to be introduced.



$V_G = \{\textcircled{1}, \textcircled{2}, \textcircled{3}, \ldots, \textcircled{10}, \textcircled{11}\}$
$E_G = \{\textcircled{1} \rightarrowtail \textcircled{3},\ \textcircled{1} \rightarrowtail \textcircled{4}, \ldots\}$
$\mathcal{I} = \{a, b, c, d, e, f, g, h, i\}$

$$\lambda_G : \begin{array}{l} \textcircled{1} \to \{a, c\} \\ \textcircled{2} \to \{a, h\} \\ \textcircled{3} \to \{c, d, e\} \\ \vdots \\ \textcircled{10} \to \{c, f, i\} \\ \textcircled{11} \to \{c, f\} \end{array}$$

Fig. 2 Example of a-DAG

3 Definitions and problem statement

This paragraph introduces more formally the concept of attributed directed acyclic graph (**a-DAG**) and the generic pattern mining approach developed in this paper.

3.1 Attributed directed acyclic graphs (a-DAGs)

An **a-DAG** $G = (V_G, E_G, \lambda_G)$ on a set of items $\mathcal{I}$ consists of a set of vertices V_G, a set of directed edges $E_G \subseteq V_G \times V_G$ and a labelling function $\lambda_G : V_G \to 2^{\mathcal{I}}$ that maps each vertex in V_G to a subset of $\mathcal{I}$. An example of such graph is given in Fig. 2. In the rest of this paper, we simply denote xy the itemset $\{x, y\}$. If there is a path from a u to v in G, then u is called an *ancestor* of v (v is called a *descendant* of u). If $\textcircled{u} \rightarrowtail \textcircled{v} \in E_G$ (i.e., u is an immediate ancestor of v), then u is a *parent* of v (v is a *child* of u).

3.2 Weighted path patterns

Let P be a sequence of itemsets $I_i \in 2^{\mathcal{I}}$ denoted $P = I_1 \rightarrowtail I_2 \rightarrowtail \cdots \rightarrowtail I_{|P|}$. P is called a **path pattern** if and only if there exists a sequence of consecutive vertices $v_1, v_2, \ldots, v_{|P|} \in V_G$ satisfying $\forall I_i \in P$, $I_i \subseteq \lambda_G(v_i)$ and each v_i is a parent of v_{i+1} in G. The sequence of vertices $O = \textcircled{v_1} \rightarrowtail \textcircled{v_2} \rightarrowtail \textcircled{\cdots} \rightarrowtail \textcircled{v_{|P|}}$ is called a **path occurrence** of P. For example, given the a-DAG in Fig. 2, the occurrences of the **size-3** path $ah \rightarrowtail cd \rightarrowtail i$ are $\textcircled{2} \rightarrowtail \textcircled{3} \rightarrowtail \textcircled{6}$, $\textcircled{2} \rightarrowtail \textcircled{3} \rightarrowtail \textcircled{8}$, $\textcircled{2} \rightarrowtail \textcircled{3} \rightarrowtail \textcircled{10}$, $\textcircled{2} \rightarrowtail \textcircled{4} \rightarrowtail \textcircled{7}$, $\textcircled{2} \rightarrowtail \textcircled{5} \rightarrowtail \textcircled{7}$, and $\textcircled{5} \rightarrowtail \textcircled{7} \rightarrowtail \textcircled{8}$. Occurrence $\textcircled{2} \rightarrowtail \textcircled{3} \rightarrowtail \textcircled{6}$ **supports** paths $ah \rightarrowtail bcd \rightarrowtail bi$, $a \rightarrowtail bcd \rightarrowtail bi$, $h \rightarrowtail bcd \rightarrowtail bi$, $h \rightarrowtail b \rightarrowtail bi$ and so on.

In the following, $occur_G(P)$ denotes the set of occurrences of P in G, and G^P the subgraph of G restricted to occurrences of P (i.e., $G^P \subseteq G$).

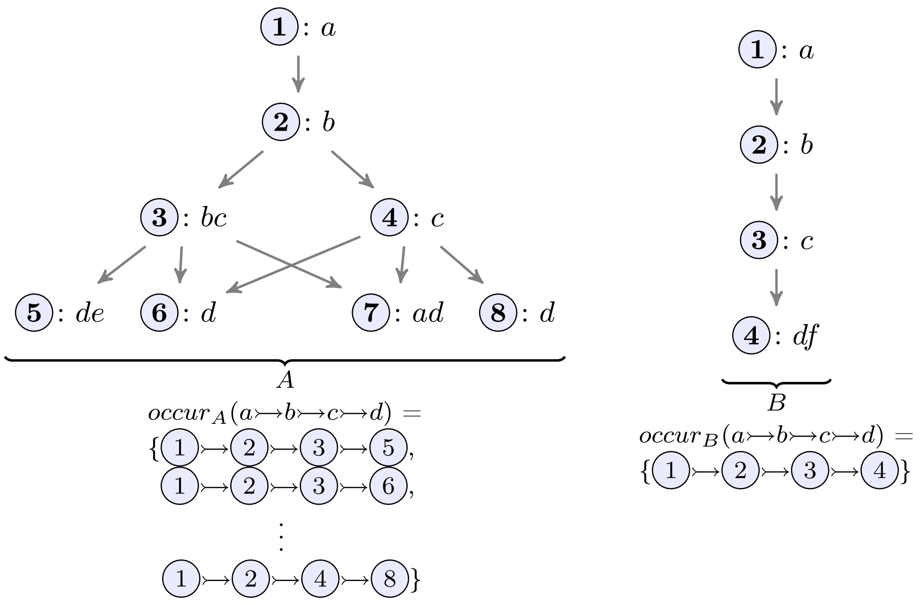


Fig. 3 Two a-DAGs in which path $a \rightarrowtail b \rightarrowtail c \rightarrowtail d$ occurs in very different ways

Path patterns can describe a temporal sequence of events or a temporal evolution of objects. They provide representations of temporal causal relationships that can be easily interpreted by experts. However, as illustrated in Fig. 3, the same path pattern can represent various situations. For example, the path pattern $a{\rightarrowtail}b{\rightarrowtail}c{\rightarrowtail}d$ occurs in the two input databases A and B. In the a-DAG A, this pattern is related to a spread, whereas it represents a single evolution in the a-DAG B. To cope with this limit without increasing pattern language "complexity," we introduce a new pattern domain: **weighted path patterns**.

A weighted path is a path with weight on each edge. The weight represents the number of occurrences of an edge within the path. For example, in Fig. 2, path $P = ah{\rightarrowtail}cd{\rightarrowtail}i$ (whose occurrences have been listed above) provides the weighted path pattern: $ah \overset{4}{\rightarrowtail} cd \overset{6}{\rightarrowtail} i$. Indeed, $occur_{G^P}(ah{\rightarrowtail}cd) = 4$ and $occur_{G^P}(cd{\rightarrowtail}i) = 6$. In other words, itemset cd occurs four times after itemset ah, and itemset i occurs six times after path $ah{\rightarrowtail}cd$. In Fig. 3, we have the weighted path $a \overset{1}{\rightarrowtail} b \overset{2}{\rightarrowtail} c \overset{6}{\rightarrowtail} d$ in a-DAG A and the weighted path $a \overset{1}{\rightarrowtail} b \overset{1}{\rightarrowtail} c \overset{1}{\rightarrowtail} d$ in a-DAG B. These patterns highlight that this sequence has a wider impact in the first graph than in the second one.

A weighted path $P = I_1 \overset{w_1}{\rightarrowtail} I_2 \overset{w_2}{\rightarrowtail} \dots \overset{w_{m-1}}{\rightarrowtail} I_m$ is called a **sub-weighted path** of another weighted path $P' = I'_1 \overset{w'_1}{\rightarrowtail} I'_2 \overset{w'_2}{\rightarrowtail} \dots \overset{w'_{n-1}}{\rightarrowtail} I'_n$ with $m \leq n$, (P' is a **super-weighted path** of P), denoted $P \sqsubseteq P'$, if $\exists k \in \{1, \dots, n\}$ such that $\forall i \in \{1, \dots, m\}$, $I_i \subseteq I'_{k+i-1}$ and $w_i = w'_{k+i-1}$. For example, in Fig. 2, pattern $c \overset{3}{\rightarrowtail} i \overset{2}{\rightarrowtail} fg$ is a sub-weighted path of $ah \overset{3}{\rightarrowtail} cd \overset{3}{\rightarrowtail} i \overset{2}{\rightarrowtail} fg \overset{1}{\rightarrowtail} cfi$ (but $c \overset{5}{\rightarrowtail} i \overset{4}{\rightarrowtail} f$ is not).

Given a weighted path $P' = I'_1 \overset{w'_1}{\rightarrowtail} I'_2 \overset{w'_2}{\rightarrowtail} \dots \overset{w'_{l-1}}{\rightarrowtail} I'_l$, a pattern $P = I_1 \overset{w_1}{\rightarrowtail} I_2 \overset{w_2}{\rightarrowtail} \dots \overset{w_{k-1}}{\rightarrowtail} I_k$ (with $k \leq l$) is called **prefix** of P' if and only if (i) $I_i = I'_i$ and (ii) $w_i = w'_i$

($\forall i \leq k$). For example, patterns $ah \overset{1}{\rightarrowtail} cd$ and $ah \overset{1}{\rightarrowtail} cd \overset{2}{\rightarrowtail} bcdi$ are prefixes of $ah \overset{1}{\rightarrowtail} cd \overset{2}{\rightarrowtail} bcdi \overset{1}{\rightarrowtail} eh \overset{2}{\rightarrowtail} cf$.

3.3 Frequency and non-redundancy constraints

In data mining, frequent pattern discovery focuses on patterns whose support/frequency is greater than a given threshold. Based on [12], we define a monotonic **support** value of a pattern $P = I_1 \overset{w_1}{\rightarrowtail} I_2 \overset{w_2}{\rightarrowtail} \ldots \overset{w_{k-1}}{\rightarrowtail} I_k$ in an a-DAG G, denoted $\sigma_G(P)$, by:

$$\sigma_G(P) = \min_{1 \leq i < |P|} w_i = \min_{1 \leq i < |P|} \left| occur_{G^P}(I_i \rightarrowtail I_{i+1}) \right|$$

This support definition is well adapted to frequency in a single a-DAG, and it can be relatively easily computed. Indeed, it distinguishes paths occurring at several locations in an a-DAG from those occurring at few locations but representing the evolution of the same node (i.e., patterns beginning from, or finishing on, one or few vertices and that spread). For example in Fig. 3, pattern $a \overset{1}{\rightarrowtail} b \overset{2}{\rightarrowtail} c \overset{6}{\rightarrowtail} d$ begins by one vertex but spreads after. Its support is 1. In other words, paths occurring at several distinct locations have a higher support.

Based on this definition of support, we can define the following **minimal support constraint** to only keep patterns covered by a minimum number of instances $minsup$:

$$C_{freq} \equiv \sigma_G(P) \geq minsup$$

However, frequent paths in G can be numerous and redundant. It thus makes sense to define non-redundancy constraints [13]. A popular non-redundancy constraint is related to the concept of closedness. Efficient algorithms have been designed based on theoretical properties of closure operators such as "closure of a pattern is unique" (see, e.g., [29,65]). However, in the single-graph setting, there is no transaction. Moreover, closure of a pattern is not necessarily unique [58]. Thus, the definition of closed pattern as proposed in the transactional setting does not hold anymore in the single-graph setting. In this work, we adapt this definition and propose a new non-redundancy constraint in the single-graph setting (the first one). Intuitively, patterns satisfying this constraint can be seen as "locally closed" because it only keeps maximal patterns representing different path occurrences (such as closed patterns in the transactional setting).

Based on the notations introduced in [9], we denote $Th(G, C_{freq})$ the set of all frequent weighted paths in an a-DAG G (i.e., the theory of G w.r.t. the minimal support constraint). We introduce the following **non-redundancy constraint** to filter redundant frequent patterns:

$$C_{nonRedund} \equiv P \in Th(G, C_{freq}) \mid \nexists P' \in Th(G, C_{freq}) \text{ such that } P \sqsubseteq P'$$

This constraint discards patterns that are a sub-weighted paths (w.r.t. definition in Sect. 3.2) of a longer frequent weighted path. For example, if pattern $ah \overset{3}{\rightarrowtail} cd \overset{3}{\rightarrowtail} bi \overset{2}{\rightarrowtail} fg \overset{1}{\rightarrowtail} cfi$ is frequent in the a-DAG of Fig. 2, then it is not necessary to keep pattern $c \overset{3}{\rightarrowtail} i \overset{2}{\rightarrowtail} fg$ because it can be deduced from the first one. On the contrary, pattern $c \overset{5}{\rightarrowtail} i \overset{4}{\rightarrowtail} f$ has to be kept because weights (i.e., occurrences) are different. In other words, this non-redundancy constraint only keeps patterns associated with different path occurrences in the a-DAG.

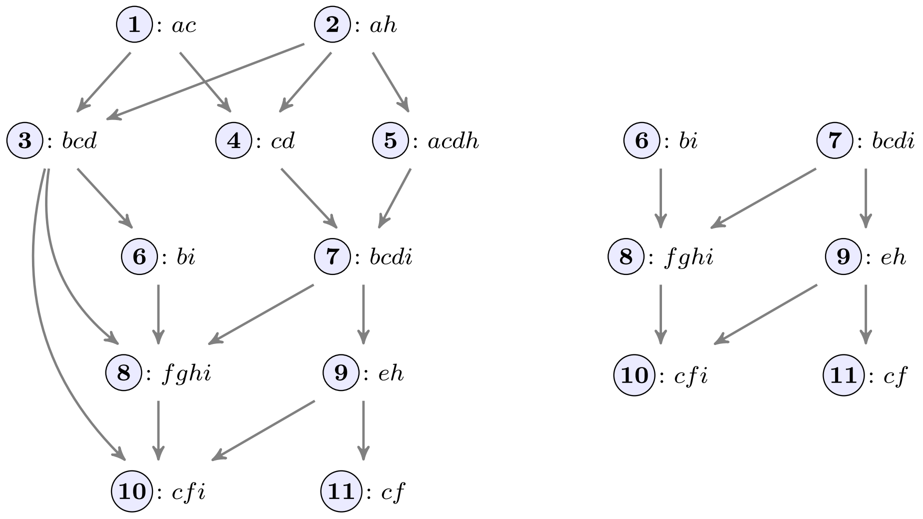


Fig. 4 Example of a-DAG G (left) and its P-projected a-DAG (right), with $P = cd \overset{3}{\rightarrowtail} bi$

Problem statement Given a single a-DAG G, the problem consists in enumerating the set of all frequent non-redundant weighted paths in G, denoted $Th(G, C_{freq} \wedge C_{nonRedund})$.

4 Mining weighted path in a single a-DAG

In this section, we introduce an approach to mine frequent non-redundant weighted paths in a single a-DAG. This complete and efficient approach is based on progressive pattern extensions and a depth-first traversal of the input a-DAG.

4.1 Pattern extensions and graph projections

A weighted path extension is performed by adding a weighted edge and an itemset at the end of a pattern. It is done based on the occurrences of the pattern in the input a-DAG. Last vertices of each occurrence, and their descendant vertices and edges, are more particularly determinant. In other words, given a pattern P, extensions of P can be found in the projection of the input a-DAG w.r.t. P. For example, let us consider the pattern $P = ah \overset{3}{\rightarrowtail} cd \overset{3}{\rightarrowtail} bi$ in the a-DAG G of Fig. 2. This pattern is associated with the occurrences $\{⓶ \rightarrowtail ⓷ \rightarrowtail ⓺, ⓶ \rightarrowtail ⓸ \rightarrowtail ⓻, ⓶ \rightarrowtail ⓹ \rightarrowtail ⓻\}$. As illustrated in Fig. 4, the projection of G w.r.t. P (right figure) represents all possible extensions of this pattern.

Definition 1 (*Projected a-DAG*) Let $P = I_1 \overset{w_1}{\rightarrowtail} I_2 \overset{w_2}{\rightarrowtail} \ldots \overset{w_{k-1}}{\rightarrowtail} I_k$ be a weighted path in an a-DAG $G = (V_G, E_G, \lambda_G)$. The P-projected a-DAG of G, denoted $G|_P = (V|_P, E|_P, \lambda_G)$, is the subgraph of G composed of the last vertices in each occurrence of P in G with all their descendants, and their corresponding edges in E_G.

To avoid construction of redundant patterns, extension must be done in such a way that there is no other possible extension with the same weight and a larger itemset. In other

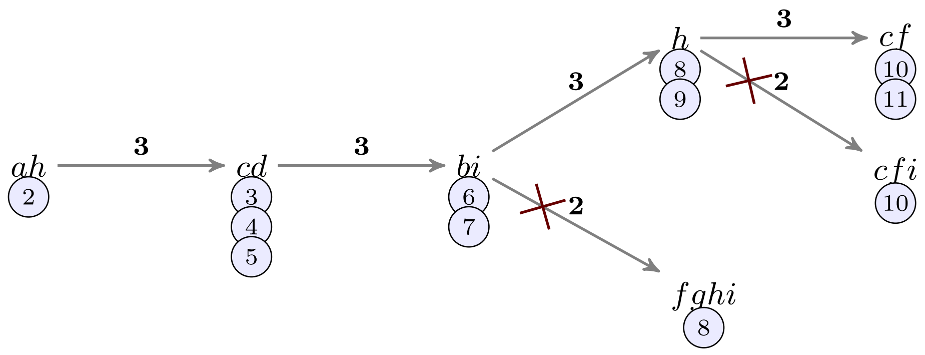


Fig. 5 Complete extensions of $ah \overset{3}{\rightarrowtail} cd \overset{3}{\rightarrowtail} bi$ (minsup=3)

words, resulting patterns must be maximal w.r.t. itemsets and associated with different sets of occurrences in the a-DAG. Figure 5 illustrates all possible maximal—in term of itemsets—extensions of size-3 weighted path $ah \overset{3}{\rightarrowtail} cd \overset{3}{\rightarrowtail} bi$. Based on the projected a-DAG of Fig. 4, we see that this pattern can be extended with $\overset{3}{\rightarrowtail} h$ related to the edges $\{\textcircled{6} \rightarrowtail \textcircled{8}, \textcircled{7} \rightarrowtail \textcircled{8}, \textcircled{7} \rightarrowtail \textcircled{9}\}$ and $\overset{2}{\rightarrowtail} fghi$ related to the edges $\{\textcircled{6} \rightarrowtail \textcircled{8}, \textcircled{7} \rightarrowtail \textcircled{8}\}$. Note that only the first extension is frequent with $minsup = 3$. The resulting pattern $ah \overset{3}{\rightarrowtail} cd \overset{3}{\rightarrowtail} bi \overset{3}{\rightarrowtail} h$ is associated with the occurrences $\{\textcircled{2} \rightarrowtail \textcircled{3} \rightarrowtail \textcircled{6} \rightarrowtail \textcircled{8}, \textcircled{2} \rightarrowtail \textcircled{4} \rightarrowtail \textcircled{7} \rightarrowtail \textcircled{8}, \textcircled{2} \rightarrowtail \textcircled{4} \rightarrowtail \textcircled{7} \rightarrowtail \textcircled{9}, \textcircled{2} \rightarrowtail \textcircled{5} \rightarrowtail \textcircled{7} \rightarrowtail \textcircled{8}, \textcircled{2} \rightarrowtail \textcircled{5} \rightarrowtail \textcircled{7} \rightarrowtail \textcircled{9}\}$. Based on the last vertices ($\textcircled{8}$ and $\textcircled{9}$), this pattern can be extended with $\overset{3}{\rightarrowtail} cf$ and $\overset{2}{\rightarrowtail} cfi$, resulting in the frequent pattern $ah \overset{3}{\rightarrowtail} cd \overset{3}{\rightarrowtail} bi \overset{3}{\rightarrowtail} h \overset{3}{\rightarrowtail} cf$. This last pattern is related to 7 occurrences ($\textcircled{2} \rightarrowtail \textcircled{3} \rightarrowtail \textcircled{6} \rightarrowtail \textcircled{8} \rightarrowtail \textcircled{10}$, $\textcircled{2} \rightarrowtail \textcircled{4} \rightarrowtail \textcircled{7} \rightarrowtail \textcircled{8} \rightarrowtail \textcircled{10}$, $\textcircled{2} \rightarrowtail \textcircled{4} \rightarrowtail \textcircled{7} \rightarrowtail \textcircled{9} \rightarrowtail \textcircled{10}$, $\textcircled{2} \rightarrowtail \textcircled{4} \rightarrowtail \textcircled{7} \rightarrowtail \textcircled{9} \rightarrowtail \textcircled{11}$ $\textcircled{2} \rightarrowtail \textcircled{5} \rightarrowtail \textcircled{7} \rightarrowtail \textcircled{8} \rightarrowtail \textcircled{10}$, $\textcircled{2} \rightarrowtail \textcircled{5} \rightarrowtail \textcircled{7} \rightarrowtail \textcircled{9} \rightarrowtail \textcircled{10}$, $\textcircled{2} \rightarrowtail \textcircled{5} \rightarrowtail \textcircled{7} \rightarrowtail \textcircled{9} \rightarrowtail \textcircled{11}$).

Note that this last pattern would be expressed as $< (cd)(bi)(h)(cf) >$ using classical sequential pattern domains (sequences, paths or strings). Its frequency would be 7 since classical frequency definitions are based on the number of occurrences. Our pattern language by representing this pattern by the weighted path $cd \overset{3}{\rightarrowtail} bi \overset{3}{\rightarrowtail} h \overset{3}{\rightarrowtail} cf$ is closer to reality. Moreover, our frequency definition is monotonically decreasing, while the number of occurrences may not (as illustrated in this example). Thus, our minimum frequency constraint can be used to efficiently prune the search space.

We call "**complete extensions**" the extensions done in the previous example because all occurrences of the pattern can be extended in the input a-DAG with the new edge and itemset.

Definition 2 (*Complete extension*) Let $P = I_1 \overset{w_1}{\rightarrowtail} I_2 \overset{w_2}{\rightarrowtail} \dots \overset{w_{k-1}}{\rightarrowtail} I_k$ a pattern to be extended with $\overset{w_k}{\rightarrowtail} X$, $X \in 2^{\mathcal{I}}$, resulting in a pattern denoted by $P \overset{w_k}{\rightarrowtail} X$. Let $V_{|P_k} = \{u \in V_G \mid \textcircled{t} \rightarrowtail \textcircled{u} \in occur_{G^P}(I_{k-1} \rightarrowtail I_k)\}$ be the last vertices of each occurrence of P in the a-DAG G.

$\overset{w_k}{\rightarrowtail} X$ is a complete extension of P if and only if $\forall u \in V_{|P_k}$, $\exists v \in V_G$ such that $\textcircled{u} \rightarrowtail \textcircled{v} \in E_G$, $X \subseteq \lambda(v)$ and $w_k = \mid occur_{G^P \overset{w_k}{\rightarrowtail} X}(I_k \rightarrowtail X) \mid$.

An interest of a complete extension is that it does not modify the prefix of the pattern being extended. It does not change its weights nor its itemsets. For example, extending $ah \overset{3}{\rightarrowtail} cd \overset{3}{\rightarrowtail} bi$ with h does not change the prefix of this pattern because all its occurrences in the a-DAG ($\textcircled{2} \rightarrowtail \textcircled{3} \rightarrowtail \textcircled{6}$, $\textcircled{2} \rightarrowtail \textcircled{4} \rightarrowtail \textcircled{7}$, $\textcircled{2} \rightarrowtail \textcircled{5} \rightarrowtail \textcircled{7}$) have h in a child vertex. As a consequence, if the initial pattern is frequent and non-redundant, then frequency and non-redundancy of the extended pattern only depend on the extension (which simplifies tests).

Theorem 1 *Given a frequent pattern $P = I_1 \overset{w_1}{\rightarrowtail} I_2 \overset{w_2}{\rightarrowtail} \ldots \overset{w_{k-1}}{\rightarrowtail} I_k$ such that P is non-redundant in a set of patterns $\mathcal{T}$. The complete extension of P with $\overset{w_k}{\rightarrowtail} X$, i.e., $P \overset{w_k}{\rightarrowtail} X$, is also frequent and non-redundant in $\mathcal{T}$ iff*

1. $w_k \geq minsup$
2. $\nexists Y \in 2^{\mathcal{I}}$ *such that* $X \subset Y$, $occur_{G^P \overset{w_k}{\rightarrowtail} X}(I_k \rightarrowtail X) = occur_{G^P \overset{w_k}{\rightarrowtail} Y}(I_k \rightarrowtail Y)$

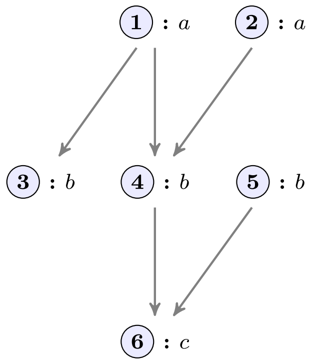
Proof According to Definition 2, a complete extension extends all the path occurrences of the studied pattern. In other words, if $P \overset{w_k}{\rightarrowtail} X$ is a complete extension then P is not modified by this extension, i.e., itemsets and weights in $P \overset{w_k}{\rightarrowtail} X$ remain the same. The first condition of the theorem ensures that $P \overset{w_k}{\rightarrowtail} X$ is frequent. Since P is frequent, all the weights w_i of P are greater or equal than the minimum support threshold $minsup$. Thanks to complete extension, we still have $w_i \geq minsup, i = 1, 2, \ldots k-1$, in $P \overset{w_k}{\rightarrowtail} X$. Thus, if $w_k \geq minsup$, then $P \overset{w_k}{\rightarrowtail} X$ is also frequent. The second condition ensures that $P \overset{w_k}{\rightarrowtail} X$ is non-redundant in $\mathcal{T} \backslash \{P\}$. Indeed, suppose that $P \overset{w_k}{\rightarrowtail} X$ is redundant and the second condition is satisfied, then there exists a path P' such that $P \sqsubseteq P'$, which leads to a contradiction since P is a non-redundant pattern. □

The complete extension principle was first introduced in [58]. However, this approach alone may miss patterns. For example, there are three solutions in Fig. 6 : $a \overset{3}{\rightarrowtail} b$, $b \overset{2}{\rightarrowtail} c$ and $a \overset{2}{\rightarrowtail} b \overset{1}{\rightarrowtail} c$. The two firsts can be generated using a complete extension of patterns a and b, but the last one cannot. Indeed, it cannot be generated using $a \overset{3}{\rightarrowtail} b$ because its occurrences are $\{\textcircled{1} \rightarrowtail \textcircled{3}$, $\textcircled{1} \rightarrowtail \textcircled{4}$, $\textcircled{2} \rightarrowtail \textcircled{4}\}$ and c is not in a child vertex of vertex 3.

To find these patterns, we have to consider **partial extensions**, i.e., extensions that expand only some of the occurrences of the pattern.

Definition 3 (*Partial extension*) Let $P = I_1 \overset{w_1}{\rightarrowtail} I_2 \overset{w_2}{\rightarrowtail} \ldots \overset{w_{k-1}}{\rightarrowtail} I_k$ be a pattern to extend with $\overset{w_k}{\rightarrowtail} X$, $X \in 2^{\mathcal{I}}$. Let $V_{|P_k} = \{u \in V_G \mid \textcircled{t} \rightarrowtail \textcircled{u} \in occur_{G^P}(I_{k-1} \rightarrowtail I_k)\}$ be the set of the last vertices of P in the a-DAG G.
$\overset{w_k}{\rightarrowtail} X$ is a partial extension of P if and only if $\exists u \in V_{|P_k}$ such that $\forall v \in V_G$, $\textcircled{u} \rightarrowtail \textcircled{v} \in E_G$ and $X \nsubseteq \lambda(v)$.

However, such extensions are more difficult to handle because they may impact weights or itemsets of the prefix. In the previous example, $a \overset{3}{\rightarrowtail} b$ can be partially extended with



Frequent non-redundant weighted paths (minsup=1) :
$a \overset{3}{\rightarrowtail} b$
$b \overset{2}{\rightarrowtail} c$
$a \overset{2}{\rightarrowtail} b \overset{1}{\rightarrowtail} c$
Frequent non-redundant weighted paths found by complete extension:
$a \overset{3}{\rightarrowtail} b$
$b \overset{2}{\rightarrowtail} c$
Frequent non-redundant weighted paths found by partial extension:
$a \overset{2}{\rightarrowtail} b \overset{1}{\rightarrowtail} c$

Fig. 6 Example of a-DAG illustrating the importance of partial extensions

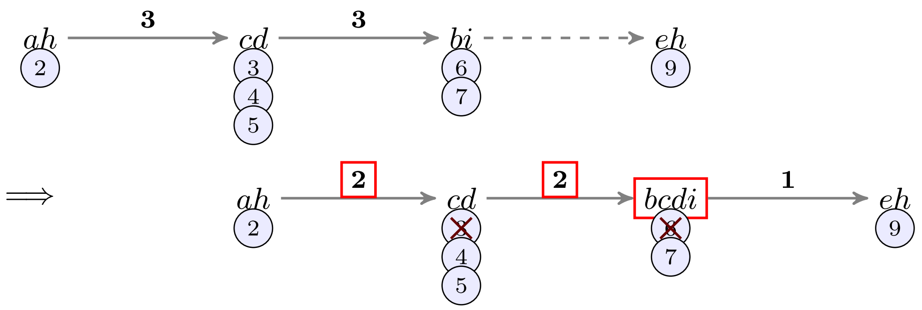


Fig. 7 Partial extensions of $ah \overset{3}{\rightarrowtail} cd \overset{3}{\rightarrowtail} bi$

$\overset{2}{\rightarrowtail}$ c, but this discards one occurrence of $a \overset{3}{\rightarrowtail} b$ (① $\rightarrowtail$ ③). Thus, the weight of $a \rightarrowtail b$ has to be changed and we obtain the pattern $a \overset{2}{\rightarrowtail} b \overset{2}{\rightarrowtail} c$. Such a case also occurs with the pattern $ah \overset{3}{\rightarrowtail} cd \overset{3}{\rightarrowtail} bi$ from the a-DAG of Fig. 2. As illustrated in Fig. 7, this pattern can be "partially" extended with $\overset{1}{\rightarrowtail} eh$. Aside from being infrequent, this extension has also an impact on $ah \overset{3}{\rightarrowtail} cd \overset{3}{\rightarrowtail} bi$, because eh is only associated with one child vertex of ⑦. This extension discards the occurrence ② $\rightarrowtail$ ③ $\rightarrowtail$ ⑥ of $ah \overset{3}{\rightarrowtail} cd \overset{3}{\rightarrowtail} bi$. Thus, we have the pattern $ah \overset{2}{\rightarrowtail} cd \overset{2}{\rightarrowtail} bi \overset{1}{\rightarrowtail} eh$, with occurrences {② $\rightarrowtail$ ④ $\rightarrowtail$ ⑦ $\rightarrowtail$ ⑨, ② $\rightarrowtail$ ⑤ $\rightarrowtail$ ⑦ $\rightarrowtail$ ⑨}. However, this pattern is not maximal w.r.t. these path occurrences, i.e., it is redundant. The pattern $ah \overset{2}{\rightarrowtail} cd \overset{2}{\rightarrowtail} bcdi \overset{1}{\rightarrowtail} eh$ is associated with the same occurrences, and it includes the previous one.

A partial extension may lead to solutions not generated by complete extensions. It may also lead to a redundant or infrequent pattern because it may change the set of occurrences associated with the prefix (occurrences may be removed). After a complete extension, we only have to test the extension to know if the generated pattern is a solution. After a partial extension, it is more complex. As formalized in the next theorem, we have to test the extension of the pattern but also its prefix.

Theorem 2 *Given a frequent pattern* $P = I_1 \overset{w_1}{\rightarrowtail} I_2 \overset{w_2}{\rightarrowtail} \dots \overset{w_{k-1}}{\rightarrowtail} I_k$ *such that* P *is non-redundant in a set of patterns* $\mathcal{T}$. *Given* $P' = I_1 \overset{w'_1}{\rightarrowtail} I_2 \overset{w'_2}{\rightarrowtail} \dots \overset{w'_{k-1}}{\rightarrowtail} I_k \overset{w_k}{\rightarrowtail} X$ *a pattern generated by partial extension of* P *with* $\overset{w_k}{\rightarrowtail} X$ *(i.e.,* $w_i \geq w'_i$*).* P' *is frequent and non-redundant in* $\mathcal{T}$ *iff*

1. $I_1 \overset{w'_1}{\rightarrowtail} I_2 \overset{w'_2}{\rightarrowtail} \dots \overset{w'_{k-1}}{\rightarrowtail} I_k$ *is a frequent and non-redundant pattern*
2. $\overset{w_k}{\rightarrowtail} X$ *is a frequent and non-redundant extension*

Proof $P' = I_1 \overset{w'_1}{\rightarrowtail} I_2 \overset{w'_2}{\rightarrowtail} \dots \overset{w'_{k-1}}{\rightarrowtail} I_k \overset{w_k}{\rightarrowtail} X$ is generated after a partial extension of P with $\overset{w_k}{\rightarrowtail} X$. Thus, for all path occurrences $\textcircled{$v_1$} \rightarrowtail \textcircled{v_2} \rightarrowtail \textcircled{$\cdots$} \rightarrowtail \textcircled{v_k}$ of P', we have $X \subseteq \lambda_G(v_k)$. Let us consider the prefixes $\textcircled{$v_1$} \rightarrowtail \textcircled{v_2} \rightarrowtail \textcircled{$\cdots$} \rightarrowtail \textcircled{v_{k-1}}$ of this set of occurrences. X is a associated in G to a child vertex of each $\textcircled{$v_1$} \rightarrowtail \textcircled{v_2} \rightarrowtail \textcircled{$\cdots$} \rightarrowtail \textcircled{v_{k-1}}$.

In other words, $\overset{w_k}{\rightarrowtail} X$ is a complete extension of $I_1 \overset{w'_1}{\rightarrowtail} I_2 \overset{w'_2}{\rightarrowtail} \dots \overset{w'_{k-1}}{\rightarrowtail} I_k$. According to Theorem 1, P' is frequent and non-redundant iff $I_1 \overset{w'_1}{\rightarrowtail} I_2 \overset{w'_2}{\rightarrowtail} \dots \overset{w'_{k-1}}{\rightarrowtail} I_k$ and its complete extension $\overset{w_k}{\rightarrowtail} X$ are frequent and non-redundant. □

The frequency of the new prefix (the one after partial extension) depends on the number of occurrences that cannot be extended. Thus, it is not necessary to recompute all. We can use the weights of the original pattern, and the number of occurrences discarded, to compute the frequency of the generated pattern (see Property 1).

Property 1 *Let* $P = I_1 \overset{w_1}{\rightarrowtail} I_2 \overset{w_2}{\rightarrowtail} \dots \overset{w_{k-1}}{\rightarrowtail} I_k$ *be a frequent pattern and* $P' = I_1 \overset{w'_1}{\rightarrowtail} I_2 \overset{w'_2}{\rightarrowtail} \dots \overset{w'_{k-1}}{\rightarrowtail} I_k \overset{w_k}{\rightarrowtail} X$ *be the pattern generated by partial extension of* P *with* $\overset{w_k}{\rightarrowtail} X$. *Let* $Dis_i = occur_{G^P}(I_i \overset{w_i}{\rightarrowtail} I_{i+1}) \backslash occur_{G^{P'}}(I_i \overset{w_i}{\rightarrowtail} I_{i+1})$ *be the set of edges of* $occur_G(P)$ *discarded for* $I_i \overset{w_i}{\rightarrowtail} I_{i+1}$ *after the partial extension of* P *with* X. *The pattern* P' *is frequent iff* $w_k \geq minsup$ *and* $w'_i = w_i - |Dis_i| \geq minsup$, $\forall i = 1, 2, \dots, k-1$.

Moreover, given a set of path occurrences O, it is also possible to construct the maximal pattern associated with O by intersecting itemsets related to vertices of O. Let us consider the previous example with the partial extension of $ah \overset{3}{\rightarrowtail} cd \overset{3}{\rightarrowtail} bi$ with $\overset{1}{\rightarrowtail} eh$. The set of occurrences associated with $ah \overset{3}{\rightarrowtail} cd \overset{3}{\rightarrowtail} bi$ is restricted to $\{\textcircled{2} \rightarrowtail \textcircled{4} \rightarrowtail \textcircled{7} \rightarrowtail \textcircled{9}, \textcircled{2} \rightarrowtail \textcircled{5} \rightarrowtail \textcircled{7} \rightarrowtail \textcircled{9}\}$. If we intersect itemsets of these two paths, we obtain the non-redundant pattern $ah \overset{2}{\rightarrowtail} cd \overset{2}{\rightarrowtail} bcdi \overset{1}{\rightarrowtail} eh$. This remark can be expressed more formally by the following property.

Property 2 *Let* $P = I_1 \overset{w_1}{\rightarrowtail} I_2 \overset{w_2}{\rightarrowtail} \dots \overset{w_{k-1}}{\rightarrowtail} I_k$ *be a pattern and* $V_i \subseteq V_G$ *be the set of vertices supporting* I_i *in the occurrences of* P*, i.e.,* $V_i = occur_{G^P}(I_i)$. *The pattern* P *is non-redundant iff* $I_i = \bigcap_{u \in V_i} \lambda_G(u)$, $\forall i = 1, 2, \dots, k$.

Proof If $I_i = \bigcap_{u \in V_i} \lambda_G(u)$, then I_i is the maximal itemset (w.r.t. inclusion) common to this set of vertices V_i. Thus, Q is the maximal pattern (w.r.t. itemset inclusion) related to this subgraph of G, i.e., it is non-redundant. □

Suppose that the prefix of P' remains frequent after partial extension of P. According to the previous property, the set of occurrences of P' can be used to generate a (non-redundant) solution, even if P' is redundant. In other words, all frequent partial extension can be used to generate a solution.

These theorems and properties are the building blocks of the exploration strategy presented in the next section.

4.2 A depth-first search exploration strategy

The exploration strategy used to extract these patterns relies on a depth-first traversal of the search space. It progressively extends patterns based on a-DAG projections and complete/partial extensions. At each step, the algorithm focuses on a projection of the input a-DAG. Then, it uses complete and partial extensions to generate frequent non-redundant patterns related to this projection.

This approach enables to enumerate all solutions. As illustrated previously, given a pattern P, we can construct all the solutions with prefix P using the P-projected a-DAG $G|_P$ and complete extensions. Moreover, Property 2 shows that occurrences of P can also be used to recursively construct all the solutions related to a subset of these occurrences (thanks to partial extensions). Lemma 1 formalizes this by decomposing the original pattern mining problem into a set of subproblems.

Lemma 1 (Prefix-based recursive generation)

- *Let $\{I_1, I_2, \ldots, I_n\}$ be the complete set of size-1 frequent non-redundant patterns in an a-DAG G, and $\{G|_{I_1}, G|_{I_2}, \ldots, G|_{I_n}\}$ be the set of projected a-DAGs associated with these patterns. The complete set of size-2 frequent non-redundant patterns in G can be generated from $\{G|_{I_1}, G|_{I_2}, \ldots, G|_{I_n}\}$.*
- *Let $\{P_1, P_2, \ldots, P_m\}$ be the complete set of size-k frequent non-redundant patterns in an a-DAG G, and $\{G|_{P_1}, G|_{P_2}, \ldots, G|_{P_m}\}$ be the set of projected a-DAGs associated with these patterns. The complete set of size-(k+1) frequent non-redundant patterns in G can be generated from $\{G|_{P_1}, G|_{P_2}, \ldots, G|_{P_m}\}$.*

Proof Let us consider size-1 patterns first. Suppose that there exists a frequent non-redundant pattern $P = Y_1 \overset{w_1}{\rightarrowtail} Y_2$ s.t. P cannot be generated from any $G|_{I_i}$ ($1 \leq i \leq n$). This means that Y_1 is not a frequent non-redundant pattern but also that $occur_G(Y_1) \nsubseteq occur_G(I_i)$. Indeed, if $occur_G(Y_1) \subseteq occur_G(I_i)$, then $G|_{Y_1} \subseteq G|_{I_i}$ and P can be generated from $G|_{I_i}$ (using Property 2 and partial extensions). This is not possible. Firstly, Y_1 is necessarily frequent (because P is frequent). Secondly, if it is redundant, then there exists a frequent non-redundant pattern I_i s.t. $Y_1 \subseteq I_i$ and both patterns have the same support, i.e., $occur_G(Y_1) = occur_G(I_i)$.

This reasoning can be transposed to size-k patterns. Suppose that there exists a size-(k+1) frequent non-redundant pattern $P' = Y_1 \overset{w_1}{\rightarrowtail} Y_2 \overset{w_2}{\rightarrowtail} \ldots \overset{w_{k-1}}{\rightarrowtail} Y_k \overset{w_k}{\rightarrowtail} X$ s.t. P' cannot be generated from any $G|_{P_i}$ ($1 \leq i \leq n$). Let us denote by Q the size-k prefix of P', i.e., $Q = Y_1 \overset{w_1}{\rightarrowtail} Y_2 \overset{w_2}{\rightarrowtail} \ldots \overset{w_{k-1}}{\rightarrowtail} Y_k$. This means that Q is not a frequent non-redundant pattern but also that $occur_G(Q) \nsubseteq occur_G(P_i)$. Indeed, if $occur_G(Q) \subseteq occur_G(P_i)$, then $G|_Q \subseteq G|_{P_i}$ and P' can be generated from $G|_{P_i}$ (using Property 2 and partial extensions with X). This is not possible. Firstly, Q is necessarily frequent (because P' is frequent). Secondly, if it is redundant, then there exists a frequent non-redundant pattern P_i s.t. $Q \sqsubseteq P_i$, i.e., $occur_G(Q) = occur_G(P_i)$. □

A basic approach to find all solutions would be to recursively construct a complete projection of the a-DAG for each prefix. However, the cost of this approach w.r.t. main memory would be very high. To deal with this problem, we do not keep all the edges of the projected a-DAG, but only the ones necessary for first path extensions, also called candidate edges.

Definition 4 (*Candidate edges*) Let P be a size-k weighted path and $G|_P = (V|_P, E|_P, \lambda_G)$ be the P-projected a-DAG of G. The set of candidate edges for P, denoted $cand(E|_P)$, is the set of edges in $E|_P$ that can be used to extend an occurrence of P, i.e., $cand(E|_P) = \{(v_k, u) \in E|_P \mid \textcircled{v_1} \rightarrowtail \textcircled{...} \rightarrowtail \textcircled{v_k} \in occur_G(P)\}$.

For example, candidate edges for $P = ah \overset{3}{\rightarrowtail} cd \overset{3}{\rightarrowtail} bi$ (Fig. 4) are $\{(6, 8), (7, 8), (7, 9)\}$. All possible size-1 complete and partial extensions ($P \overset{3}{\rightarrowtail} h$, $P \overset{2}{\rightarrowtail} fghi$ and $P \overset{1}{\rightarrowtail} eh$) can be generated from this set of edges. This example also shows that the extension problem can be transformed in a closed itemset mining problem based on this set of candidate edges. In this example, the transactional database would be $\{fghi, fghi, eh\}$ because there are two edges leading to $fghi$ and one for eh. Closed itemsets would be h (support: 3), eh (support: 1) and $fghi$ (support: 2). These three closed itemsets enable to generate the three possible non-redundant extensions of P.

Definition 5 (*Transactional database of candidate extensions*) Let P be a weighted path in an a-DAG G, and $G|_P = (V|_P, E|_P, \lambda_G)$ be the P-projected a-DAG of G. The transactional database $\mathcal{D}_{G|_P}$ associated with the P-projected a-DAG of G is the collection of transactions $t \subseteq \mathcal{I}$ such that $t = \lambda(v)$, with $v \in V|_P$ and $(u, v) \in cand(E|_P)$.

In Fig. 2, occurrences of pattern $ac \overset{3}{\rightarrowtail} cd$ are $\textcircled{1} \rightarrowtail \textcircled{3}$, $\textcircled{1} \rightarrowtail \textcircled{4}$ and $\textcircled{5} \rightarrowtail \textcircled{7}$. Thus, candidate edges for this pattern are $\{(3, 6), (3, 8), (3, 10), (4, 7), (7, 8), (7, 9)\}$. The transactional database is $\mathcal{D}_{G|ac \overset{3}{\rightarrowtail} cd} = \{bi, fghi, cfi, bcdi, fghi, eh\}$. There are two transactions with itemset $fghi$, because $fghi$ is associated with the edges (3, 8) and (7, 8). With $minsup = 3$, frequent closed itemsets, i.e., frequent non-redundant extensions, are i (support: 5), h (support: 3) and fi (support: 3). The first one leads to the complete extension $\overset{5}{\rightarrowtail} i$ and frequent non-redundant pattern $ac \overset{3}{\rightarrowtail} cd \overset{5}{\rightarrowtail} i$. The second one leads to the partial extension $\overset{3}{\rightarrowtail} h$ and infrequent (non-redundant) pattern $ac \overset{2}{\rightarrowtail} bcd \overset{3}{\rightarrowtail} h$ (using Property 2). The last one leads to the partial extension $\overset{3}{\rightarrowtail} fi$ and infrequent (non-redundant) pattern $ac \overset{2}{\rightarrowtail} bcd \overset{3}{\rightarrowtail} fi$ (using Property 2). The following theorem formalizes the generation of frequent non-redundant weighted paths using these "local" closed itemsets.

Theorem 3 *Let $P = I_1 \overset{w_1}{\rightarrowtail} I_2 \overset{w_2}{\rightarrowtail} ... \overset{w_{k-1}}{\rightarrowtail} I_k$ be the prefix of a frequent non-redundant weighted path in an a-DAG G, and $\mathcal{D}_{G|_P}$ be the transactional database related to the projection of G w.r.t. P. Let $X \in 2^{\mathcal{I}}$ be a frequent closed itemset in $\mathcal{D}_{G|_P}$, and $w = support(X)$ be its frequency in $\mathcal{D}_{G|_P}$.*

If $\overset{w}{\rightarrowtail} X$ is a complete extension of P, then $I_1 \overset{w_1}{\rightarrowtail} I_2 \overset{w_2}{\rightarrowtail} ... \overset{w_{k-1}}{\rightarrowtail} I_k \overset{w}{\rightarrowtail} X$ is the prefix of a frequent non-redundant weighted path.

If $\overset{w}{\rightarrowtail} X$ is a partial extension of P, then $P' = I'_1 \overset{w'_1}{\rightarrowtail} I'_2 \overset{w'_2}{\rightarrowtail} ... \overset{w'_{k-1}}{\rightarrowtail} I'_k \overset{w}{\rightarrowtail} X$ is the prefix of a non-redundant weighted path, with $I'_i = \bigcap_{u \in V'_i} \lambda_G(u)$ and $V'_i = occur_{G^{P'}}(I_i)$ ($\forall i = 1, 2, \ldots, k$).

Proof Let $\overset{w}{\rightarrowtail} X$ be a complete extension of P. $I_1 \overset{w_1}{\rightarrowtail} I_2 \overset{w_2}{\rightarrowtail} \ldots \overset{w_{k-1}}{\rightarrowtail} I_k \overset{w}{\rightarrowtail} X$ is frequent and non-redundant, by Theorem 1, iff $w \geq minsup$ and $\nexists Y \in 2^{\mathcal{I}}$ such that $X \subset Y$, $occur_{G^P \overset{w}{\rightarrowtail} X}(I_k \rightarrowtail X) = occur_{G^P \overset{w}{\rightarrowtail} Y}(I_k \rightarrowtail Y)$. X is frequent in $\mathcal{D}_{G|_P}$. Thus, $support(X) \geq minsup$, i.e., $w \geq minsup$ (since $support(X) = w$). X is also closed in $\mathcal{D}_{G|_P}$, i.e., $\nexists Y \in 2^{\mathcal{I}}$ such that $X \subset Y$ and $support(X) = support(Y)$ in $\mathcal{D}_{G|_P}$. Thus, the set of transactions supporting X in $\mathcal{D}_{G|_P}$ is different from the one supporting Y. In other words, $\{(u, v) \in cand(E|_P) \mid X \subseteq \lambda_G(v)\} \neq \{(u, v) \in cand(E|_P) \mid Y \subseteq \lambda_G(v)\}$, i.e., $occur_{G^P \overset{w}{\rightarrowtail} X}(I_k \rightarrowtail X) \neq occur_{G^P \overset{w}{\rightarrowtail} Y}(I_k \rightarrowtail Y)$. Thus, X is a non-redundant extension.

The second part of the theorem is a corollary of Theorem 2 and Property 2. As shown previously, if X is a closed itemset in $\mathcal{D}_{G|_P}$, then $\overset{w}{\rightarrowtail} X$ is a non-redundant extension, since $\nexists Y \in 2^{\mathcal{I}}$ such that $occur_{G^P \overset{w}{\rightarrowtail} X}(I_k \rightarrowtail X) = occur_{G^P \overset{w}{\rightarrowtail} Y}(I_k \rightarrowtail Y)$. Moreover, $I'_1 \overset{w'_1}{\rightarrowtail} I'_2 \overset{w'_2}{\rightarrowtail} \ldots \overset{w'_{k-1}}{\rightarrowtail} I'_k$ is non-redundant by definition (Property 2), since $I'_i = \bigcap_{u \in V'_i} \lambda_G(u)$, $\forall i = 1, 2, \ldots, k$.

Note that $occur_G(P') \subseteq occur_G(P)$ because P' is generated after a partial extension of P. Thus, P' is generated using a subset of vertices supporting each I_i in occurrences of P, i.e., $occur_{G^{P'}}(I_i) \subseteq occur_{G^P}(I_i)$. As a consequence, $V'_i = occur_{G^{P'}}(I'_i) = occur_{G^{P'}}(I_i)$ and $I_i \subseteq I'_i$, i.e., I'_i is the maximal itemset supported by this set of occurrences. □

Based on this theorem, we recursively generate each pattern based on the closed frequent itemsets extracted from candidate edges. If an itemset leads to a partial extension, we first update the weights of the prefix using Property 1. Then, if the resulting pattern stills frequent, we recompute the itemsets of the prefix using Property 2. After, we continue its recursive extension and save the final pattern. All frequent non-redundant patterns are generated using this depth-first algorithm. However, some patterns (or part of patterns) may be generated twice. For example, in Fig. 8, $a \overset{3}{\rightarrowtail} bcd \overset{3}{\rightarrowtail} fi$ is generated based on a prefix growth of a, and $bcd \overset{3}{\rightarrowtail} fi$ is generated based on a prefix growth of bcd. To avoid this, we have to do inclusion tests (and potentially update some solutions) before recursively extending the current pattern. Indeed, if $a \overset{3}{\rightarrowtail} bcd \overset{3}{\rightarrowtail} fi$ is generated first, we have to stop the extension of $bcd \overset{3}{\rightarrowtail} fi$ and do not save this last pattern. If $bcd \overset{4}{\rightarrowtail} i$ is generated first, we have to update this pattern with prefix $a \overset{3}{\rightarrowtail}$ in the set of solutions and stop its extensions (because they have already been processed when extending $bcd \overset{4}{\rightarrowtail} i$).

This approach is described in Algorithm 1. Figure 8 illustrates the different steps of this algorithm with the a-DAG introduced in Fig. 2 and $minsup = 3$. The complete set of frequent non-redundant weighted path can be extracted in the following steps:

1. **Mine size-1 patterns:** Scan G once to find all the size-1 frequent non-redundant weighted paths. To do this, the algorithm generates a transactional database composed of the itemsets associated with the origin of each edge of G, and mine the frequent closed itemsets of this database. The first size-1 patterns of this set are $a(6)$, $ac(3)$, $ah(4)$, $b(6)$, $bcd(5)$, $bi(3)$ (with frequency in parenthesis).
2. **Divide the search space:** The search space can be divided based on the size-1 frequent patterns previously extracted. Each size-1 pattern leads to one prefix and to one projected a-DAG. To limit memory occupation, only the candidate edges are computed. For example, the first solutions being explored are the ones with prefix a, and they are computed based on the candidate edges of a, i.e., $cand(E|_a) =$

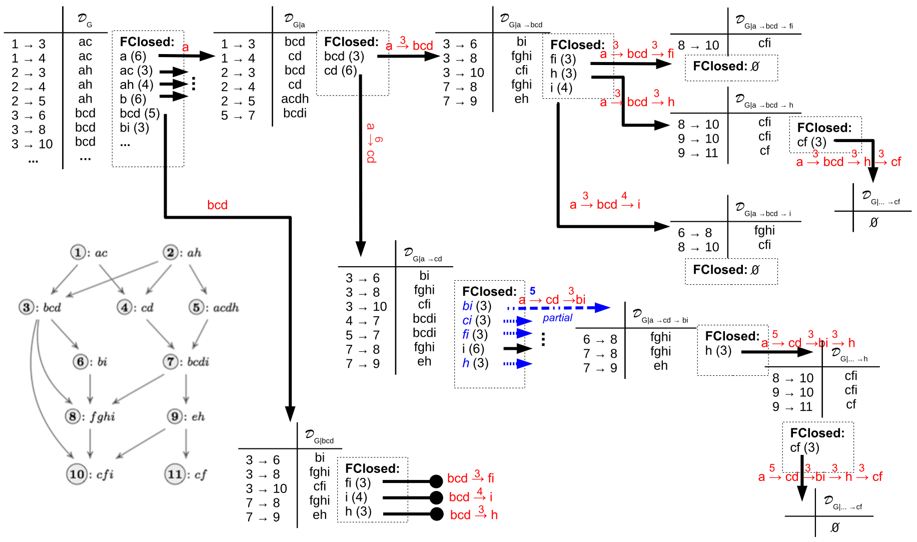


Fig. 8 Example of PrefixPathGrowth execution with $minsup = 3$

ALGORITHM 1: PrefixPathGrowth(P, G, $minsup$, Th, $cand(E|_P)$)

```
Input : A pattern $P$, an a-DAG $G = (V_G, E_G, \lambda_G)$, the minimum support threshold $minsup$, the set of
        solutions $Th$, the set of candidate edges $cand(E|_P)$
1  if $|cand(E|_P)| < minsup$ then
2  |   $Th = Th \cup \{P\}$;
3  else
4  |   $FClosed = MiningFreqClosedItemset( cand(E|_P), minsup )$
5  |   if $FClosed == \emptyset$ then
6  |   |   $Th = Th \cup \{P\}$;
7  |   else
8  |   |   foreach $X \in FClosed$ do
9  |   |   |   $P' = P \overset{w}{\rightarrowtail} X$, with $w = support(X)$
10 |   |   |   if $|P| > 1$ and $IsPartialExtension( X, P, cand(E|_P) )$ then
11 |   |   |   |   $BackwardUpdate( P', G, cand(E|_P) )$
12 |   |   |   |   if $\sigma_G(P') < minsup$ then
13 |   |   |   |   |   Continue     // skip the current loop iteration and go to next
   |   |   |   |   |   extension
14 |   |   |   |   end
15 |   |   |   end
16 |   |   |   if $\neg Include( Th, P' )$ and $\neg UpdatePrefixes( Th, P' )$ then
17 |   |   |   |   PrefixPathGrowth ( $P', G, minsup, Th, cand(E|_{P'})$ )
18 |   |   |   end
19 |   |   end
20 |   end
21 end
```

$\{(1,3), (1,4), (2,3), (2,4), (2,5), (5,7)\}$. Once all the solutions with prefix a are mined, the algorithm explores the solutions with prefix ac (the next size-1 pattern generated), etc. Note that these projected a-DAG are not explored if the number of candidate edges is lower than the minimum support threshold value (line 1 of the algorithm), because in such case the resulting patterns will not be frequent.

3. **Recursive extension of prefixes:** Each subset of solutions is generated recursively by extending progressively patterns based on candidate edges. More precisely, for each prefix, its candidate edges are transformed into a transactional database and closed frequent itemsets are mined (line 4 of the algorithm). Each closed itemset is used to extend the current prefix (line 9 of the algorithm). Each new prefix (and its candidate edges) is then used in input of another recursive call (line 17 of the algorithm). Lines 10–15 of the algorithm correspond to a partial extension of the prefix. As mentioned before, in such case, the prefix has to be updated (function $BackwardUpdate$) and frequency has to be checked. Line 16 checks if prefix P' has not already been studied (totally or partially). If there is a pattern in Th which includes P', this prefix extension is stopped (function $Include$). If there is a pattern in Th such that its prefix is a suffix of P', the pattern in Th is updated with P' and the extension is stopped (function $UpdatePrefixes$).
This process on the example of Fig. 8 is detailed below:
   (a) **Mine patterns with prefix** a **:** The transactional database related to this projection of G is $\mathcal{D}_{G|_a} = \{bcd, cd, bcd, cd, acdh, bcdi\}$. Two frequent closed itemsets are mined: bcd (with a support of 3) and cd (with a support of 6). Thus, we have two possible prefix extensions: $a \overset{3}{\rightarrowtail} bcd$ and $a \overset{6}{\rightarrowtail} cd$ (complete extensions).
      i. **Mine patterns with prefix** $a \rightarrowtail bcd$: The transactional database related to this projection of G is $\mathcal{D}_{G|_a \overset{3}{\rightarrowtail} bcd} = \{bi, fghi, cfi, fghi, eh\}$. The three frequent closed itemsets in this database are fi (with a support of 3), h (with a support of 3) and i (with a support of 4). Thus, we have three possible prefix extensions: $a \overset{3}{\rightarrowtail} bcd \overset{3}{\rightarrowtail} fi$, $a \overset{3}{\rightarrowtail} bcd \overset{3}{\rightarrowtail} h$ and $a \overset{3}{\rightarrowtail} bcd \overset{4}{\rightarrowtail} i$ (complete extensions).
         – **Mine patterns with prefix** $a \rightarrowtail bcd \rightarrowtail fi$: The transactional database related to this projection of G is $\mathcal{D}_{G|_a \overset{3}{\rightarrowtail} bcd \overset{3}{\rightarrowtail} fi} = \{cfi\}$. There is no frequent closed itemset in this database. Thus, this pattern is not extended.
         – **Mine patterns with prefix** $a \rightarrowtail bcd \rightarrowtail h$: The transactional database related to this projection of G is $\mathcal{D}_{G|_a \overset{3}{\rightarrowtail} bcd \overset{3}{\rightarrowtail} h} = \{cfi, cfi, cf\}$. There is only one frequent closed itemset in this database: cf (with a support of 3). Thus, we have one possible prefix extension: $a \overset{3}{\rightarrowtail} bcd \overset{3}{\rightarrowtail} h \overset{3}{\rightarrowtail} cf$ (complete extension).
            • **Mine patterns with prefix** $a \rightarrowtail bcd \rightarrowtail h \rightarrowtail cf$: The transactional database related to this projection of G is $\mathcal{D}_{G|_a \overset{3}{\rightarrowtail} bcd \overset{3}{\rightarrowtail} h \overset{3}{\rightarrowtail} cf} = \emptyset$. There is no frequent closed itemset in this database. Thus, this pattern is not extended.
         – **Mine patterns with prefix** $a \rightarrowtail bcd \rightarrowtail i$ The transactional database related to this projection of G is $\mathcal{D}_{G|_a \overset{3}{\rightarrowtail} bcd \overset{4}{\rightarrowtail} i} = \{fghi, cfi\}$. There is no frequent closed itemset in this database. Thus, this pattern is not extended.
      ii **Mine patterns with prefix** $a \rightarrowtail cd$: The transactional database related to this projection of G is $\mathcal{D}_{G|_a \overset{6}{\rightarrowtail} cd} = \{bi, fghi, cfi, bcdi, bcdi, fghi, eh\}$. The frequent closed itemsets in this database are $bi(3)$, $ci(3)$, $fi(3)$, $i(6)$, $h(3)$. Thus,

we have 5 possible prefix extensions (and only i is a complete extension). After backward update of itemsets and weights for partial extensions, we have prefixes $a \overset{5}{\rightarrowtail} cd \overset{3}{\rightarrowtail} bi$, $a \overset{5}{\rightarrowtail} cd \overset{3}{\rightarrowtail} ci$ and $a \overset{6}{\rightarrowtail} cd \overset{6}{\rightarrowtail} i$, which are recursively explored. Note that $a \overset{2}{\rightarrowtail} cd \overset{3}{\rightarrowtail} fi$ and $a \overset{2}{\rightarrowtail} cd \overset{3}{\rightarrowtail} h$ are not frequents.

– **Mine patterns with prefix** $a{\rightarrowtail}cd{\rightarrowtail}bi$, $a{\rightarrowtail}cd{\rightarrowtail}ci$ and $a{\rightarrowtail}cd{\rightarrowtail}i$:...

(b) **Mine patterns with prefix** ac, ah, b, bcd, bi **...**

Note that, in this example, the algorithm doesn't need to explore $bcd \overset{3}{\rightarrowtail} fi$, $bcd \overset{4}{\rightarrowtail} i$ and $bcd \overset{3}{\rightarrowtail} h$ (test done in line 16 of the algorithm), because super-patterns have already been explored when mining solutions prefixed with a.

4.3 Data structure and optimizations

The previous algorithm presents the global strategy of our approach. However, some steps of this strategy may be particularly time consuming without an adapted data structure. In particular, it is the case of the *BackwardUpdate*, *Include* and *UpdatePrefixes* functions because they do costly comparisons between the current pattern and its occurrences in the input graph.

To cope with this difficulty, we propose a data structure to efficiently mine solutions. This data structure is called the *pattern graph*. Each path in this graph represents a solution, i.e., a frequent non-redundant pattern with its occurrences. Given a solution $P = I_1{\rightarrowtail}I_2{\rightarrowtail}\cdots{\rightarrowtail}I_{|P|}$. There is a path in the pattern graph representing this pattern. Each vertex of this path contains an itemset of P and its occurrences in the input graph. Directed edges of this path represent the successive itemsets of P. More formally, this graph, denoted by G_{Th}, consists of a set of vertices V_{Th}, a set of edges $E_{Th} \subseteq V_{Th} \times V_{Th}$, a labelling function $\lambda_w : E_{Th} \rightarrow \mathbb{Z}$ that maps each edge in E_{Th} to a weight, and a labelling function $\lambda_{Th} : V_{Th} \rightarrow (2^{\mathcal{I}}, 2^{V_G})$ that maps each vertex in V_{Th} to an itemset and a subset of vertices of G. For example, Fig. 9 shows the pattern graph of the 5 solutions extracted in Fig. 8. Each path represents one solution. For example, path $\boxed{1} \rightarrowtail \boxed{2} \rightarrowtail \boxed{3}$ represents pattern $a \overset{3}{\rightarrowtail} bcd \overset{3}{\rightarrowtail} fi$ and its occurrences $\{\textcircled{1} \rightarrowtail \textcircled{3} \rightarrowtail \textcircled{8}, \textcircled{1} \rightarrowtail \textcircled{3} \rightarrowtail \textcircled{10}, \textcircled{2} \rightarrowtail \textcircled{3} \rightarrowtail \textcircled{8}, \textcircled{2} \rightarrowtail \textcircled{3} \rightarrowtail \textcircled{10}, \textcircled{5} \rightarrowtail \textcircled{7} \rightarrowtail \textcircled{8}\}$ in G. As illustrated by this figure, a vertex of the pattern graph can be shared across several solutions. For example, vertices $\boxed{1}$, $\boxed{4}$ and $\boxed{5}$ are used to represent patterns $a \overset{3}{\rightarrowtail} bcd \overset{3}{\rightarrowtail} h \overset{3}{\rightarrowtail} cf$ and $a \overset{5}{\rightarrowtail} cd \overset{3}{\rightarrowtail} bi \overset{3}{\rightarrowtail} h \overset{3}{\rightarrowtail} cf$. To the opposite, we need different vertices ($\boxed{7}$ and $\boxed{8}$) to represent cd in patterns $a \overset{6}{\rightarrowtail} cd$ and $a \overset{5}{\rightarrowtail} cd \overset{3}{\rightarrowtail} bi \overset{3}{\rightarrowtail} h \overset{3}{\rightarrowtail} cf$, because they are not associated with exactly the same occurrences in G.

This data structure enables to efficiently store patterns but also to efficiently generate them. Each pattern extension corresponds to the generation of a new vertex and a new edge in the pattern graph. If the vertex is already in the pattern graph, we simply add a new edge and stop extensions for this pattern because they have already been previously processed. We have such example in Fig. 9 when extending pattern $a \overset{5}{\rightarrowtail} cd \overset{3}{\rightarrowtail} bi$ (path $\boxed{1} \rightarrowtail \boxed{8} \rightarrowtail \boxed{9}$) with $\overset{3}{\rightarrowtail} h \overset{3}{\rightarrowtail} cf$ (path $\boxed{4} \rightarrowtail \boxed{5}$). Thanks to this data structure, inclusion tests (function $Include$) and pattern updates (function *UpdatePrefixes*) in Algorithm 1 are much more easy. They consist only in searching a vertex in a set of vertices. This search can be

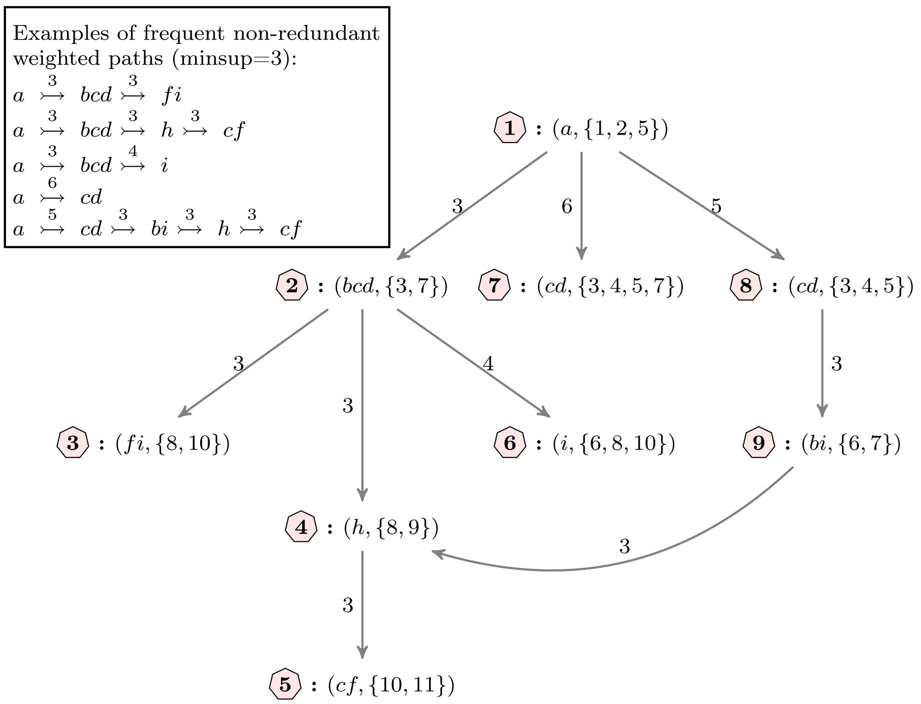

Examples of frequent non-redundant weighted paths (minsup=3):
$a \overset{3}{\rightarrowtail} bcd \overset{3}{\rightarrowtail} fi$
$a \overset{3}{\rightarrowtail} bcd \overset{3}{\rightarrowtail} h \overset{3}{\rightarrowtail} cf$
$a \overset{3}{\rightarrowtail} bcd \overset{4}{\rightarrowtail} i$
$a \overset{6}{\rightarrowtail} cd$
$a \overset{5}{\rightarrowtail} cd \overset{3}{\rightarrowtail} bi \overset{3}{\rightarrowtail} h \overset{3}{\rightarrowtail} cf$
1 : $(a, \{1, 2, 5\})$
2 : $(bcd, \{3, 7\})$
7 : $(cd, \{3, 4, 5, 7\})$
8 : $(cd, \{3, 4, 5\})$
3 : $(fi, \{8, 10\})$
6 : $(i, \{6, 8, 10\})$
9 : $(bi, \{6, 7\})$
4 : $(h, \{8, 9\})$
5 : $(cf, \{10, 11\})$
3
6
5
3
3
4
3
3
3

Fig. 9 Pattern graph of the solutions extracted in Fig. 8

simplified because each vertex $u \in V_{Th}$ is uniquely identified by its subset of vertices V, with $\lambda_{Th}(u) = (X, V)$. It is a direct consequence of Property 2.

Algorithm 2 describes our algorithm using this pattern graph data structure. The global strategy has not changed, but several operations are simplified using the data structure. The initial call of this recursive algorithm is done with $u_k = u_0$, $\lambda_{Th}(u_0) = (\emptyset, V_G)$.

Given a pattern $P = I_1 \overset{w_1}{\rightarrowtail} I_2 \overset{w_2}{\rightarrowtail} \ldots \overset{w_{k-1}}{\rightarrowtail} I_k$. The algorithm extends this pattern with all possible frequent non-redundant extensions X. It generates patterns $P' = I'_1 \overset{w'_1}{\rightarrowtail} I'_2 \overset{w'_2}{\rightarrowtail} \ldots \overset{w'_{k-1}}{\rightarrowtail} I'_k \overset{supp_X}{\rightarrowtail} X$, with $I'_i = I_i$ and $w'_i = w_i$ if it is a complete extension, or $I_i \subseteq I'_i$ and $w'_i \leq w_i$ if it is a partial extension ($i = 1, \ldots, k$). Lines 5–9 correspond to the generation of the first itemset I_1 of P, i.e., the generation of a vertex (I_1, V_1) in the pattern graph. Line 6 checks if this vertex is already in the pattern graph, i.e., if a super-pattern has already been generated. This simple test substitues the costly $Include$ function. Lines 11–25 represent the general case where I_k is the last itemset of P and V_k its occurrences. Line 11 processes V'_k, i.e., the subset of vertices of V_k that can be extended with X. If $V_k \neq V'_k$ (line 12), then we have a partial extension of P with X. This test substitues the previous $IsPartialExtension$ function. The *BackwardUpdate* function in line 13 processes the new prefix $I'_1 \overset{w'_1}{\rightarrowtail} I'_2 \overset{w'_2}{\rightarrowtail} \ldots \overset{w'_{k-1}}{\rightarrowtail} I'_k$ w.r.t. V'_k and returns true if it is frequent. New vertices and edges may be inserted in the pattern graph to represent this new prefix. Lines 17–24 represent the extension of the pattern graph with X. u'_k is the current vertex in the pattern graph. If the extension is complete, then $u'_k = u_k$. If the extension is partial, u'_k is the vertex

ALGORITHM 2: PrefixPathGrowth-PatternGraph($G, minsup, G_{Th}, u_k$)

```
Input : An a-DAG $G = (V_G, E_G, \lambda_G)$, a minimum support threshold $minsup$, a pattern graph
        $G_{Th} = (V_{Th}, E_{Th}, \lambda_{Th}, \lambda_w)$, a vertex $u_k \in V_{Th}$ with $\lambda_{Th}(u_k) = (I_k, V_k)$
1  $cand(E|_P) = \{(u, v) \in E_G \mid u \in V_k\}$
2  $FClosed = MiningFreqClosedItemset\ (\ cand(E|_P), minsup\ )$
3  foreach $X \in FClosed$ do
4      $V_X = \{v \in V_G \mid (u, v) \in cand(E|_P)$ and $X \subseteq \lambda_G(v)\}$
5      if $I_k == \emptyset$ then
6          if $\nexists u_X \in V_{Th}$ s.t. $\lambda_{Th}(u_X) = (X, V_X)$ then
7              Insert $u_X$ in $V_{Th}$ with $\lambda_{Th}\ (u_X) = (X, V_X)$
8              PrefixPathGrowth-PatternGraph ( $G, minsup, G_{Th}, u_X$ )
9          end
10     else
11         $V'_k = \{u \in V_k \mid (u, v) \in cand(E|_P)$ and $X \subseteq \lambda_G(v)\}$
12         if $V_k \neq V'_k$ then                                   // partial extension
13             if $BackwardUpdate(\ G, minsup, G_{Th}, u_k, V'_k\ ) == False$ then
14                 Continue        // skip the current loop iteration and go to next
                   extension
15             end
16         end
17         Let $u'_k \in V_{Th}$ s.t. $\lambda_{Th}(u'_k) = (I'_k, V'_k)$
18         if $\nexists u_X \in V_{Th}$ s.t. $\lambda_{Th}(u_X) = (X, V_X)$ then
19             Insert $u_X$ in $V_{Th}$ with $\lambda_{Th}\ (u_X) = (X, V_X)$
20             Insert $(u'_k, u_X)$ in $E_{Th}$ with $\lambda_w\left((u'_k, u_X)\right) = support(X)$
21             PrefixPathGrowth-PatternGraph ( $G, minsup, G_{Th}, u_X$ )
22         else
23             Insert $(u'_k, u_X)$ in $E_{Th}$ with $\lambda_w\left((u'_k, u_X)\right) = support(X)$
24         end
25     end
26 end
```

of the pattern graph associated with V'_k (an existing vertex or a newly created one by the *BackwardUpdate* function). Line 18 tests if (X, V_X) is already in the pattern graph, i.e., if this part of the pattern (and its extensions) has already been generated. If it is not in the pattern graph, the algorithm inserts vertex u_X, generates an edge (u'_k, u_X) and continues the extension of P'. If it is already in the pattern graph, the algorithm only generates an edge (u'_k, u_X) and stops extensions. These operations substitute the $UpdatePrefixes$ function. It avoids the generation of the same extension twice, such as in the previous example with $a \overset{5}{\rightarrowtail} cd \overset{3}{\rightarrowtail} bi$ and $\overset{3}{\rightarrowtail} h \overset{3}{\rightarrowtail} cf$ (direct extension of $\boxed{1} \rightarrowtail \boxed{8} \rightarrowtail \boxed{9}$ with $\boxed{4} \rightarrowtail \boxed{5}$).

Algorithm 3 describes in detail the *BackwardUpdate* function. The input parameter u_k is the last vertex of $P = I_1 \overset{w_1}{\rightarrowtail} I_2 \overset{w_2}{\rightarrowtail} \ldots \overset{w_{k-1}}{\rightarrowtail} I_k$ in the pattern graph. V'_k is the set of occurrences that can be partially extended with X in Algorithm 2. The principle of Algorithm 3 is to recursively explore all ancestors of u_k and to update their occurrences according to V'_k (cascading updates). This exploration continues until the prefix remains frequent. At the end, if $I'_1 \overset{w'_1}{\rightarrowtail} I'_2 \overset{w'_2}{\rightarrowtail} \ldots \overset{w'_{k-1}}{\rightarrowtail} I'_k$ is frequent, the algorithm inserts all necessary vertices and edges in the pattern graph. Lines 1–4 correspond to I'_1 processing, i.e., insertion of (I'_1, V'_1) in the pattern graph. In such case, all the pattern is frequent and all (I'_i, V'_i) vertices (and the corresponding edges) can be recursively inserted in the pattern graph (lines 16–19). Property 2 is used to process all I'_i (lines 2 and 17). Line 13 is the

ALGORITHM 3: BackwardUpdate($G, minsup, G_{Th}, u_k, V_k'$)

Input : An a-DAG $G = (V_G, E_G, \lambda_G)$, a minimum support threshold $minsup$, a pattern graph $G_{Th} = (V_{Th}, E_{Th}, \lambda_{Th}, \lambda_w)$, a vertex $u_k \in V_{Th}$ with $\lambda_{Th}(u_k) = (I_k, V_k)$, a subset of vertices V_k' representing a partial extension of I_k ($V_k' \subset V_k$)

Output: True if the the prefix is frequent, False elsewhere

```
if $\nexists u_{k-1} \in V_{Th}$ s.t. $(u_{k-1}, u_k) \in E_{Th}$ then
    Insert $u_k'$ in $V_{Th}$ with $\lambda_{Th}(u_k') = (I_k', V_k')$ and $I_k' = I_k \cup \bigcap_{u \in V_k'} \lambda_G(u)$
    return True
else
    frequent = False
    foreach $u_{k-1} \in V_{Th}$ s.t. $(u_{k-1}, u_k) \in E_{Th}$ , with $\lambda_{Th}(u_{k-1}) = (I_{k-1}, V_{k-1})$ do
        $V_{k-1}' = \{u \in V_{k-1} \mid (u, v) \in E_G$ and $v \in V_k'\}$
        if $\exists (u_{k-1}', u_k') \in E_{Th}$ s.t. $\lambda_{Th}(u_{k-1}') = (I_{k-1}', V_{k-1}')$ and $\lambda_{Th}(u_k') = (I_k', V_k')$ then
            if BackwardUpdate( G, minsup, $G_{Th}$, $u_{k-1}$, $V_{k-1}'$ ) == True then
                frequent = True
            end
        else
            $w_k' = \lambda_w\left((u_{k-1}, u_k)\right) - \left|\{(u, v) \in E_G \mid u \in V_{k-1}, u \notin V_{k-1}'$ and $v \in V_k'\}\right|$
            if $w_k' \geq minsup$ then
                if BackwardUpdate( G, minsup, $G_{Th}$, $u_{k-1}$, $V_{k-1}'$ ) == True then
                    frequent = True
                    Insert $u_k'$ in $V_{Th}$ with $\lambda_{Th}(u_k') = (I_k', V_k')$ and $I_k' = I_k \cup \bigcap_{u \in V_k'} \lambda_G(u)$
                    Insert $(u_{k-1}', u_k')$ in $E_{Th}$ s.t. $\lambda_{Th}(u_{k-1}') = (I_{k-1}', V_{k-1}')$, with $\lambda_w\left((u_{k-1}', u_k')\right) = w_k'$
                end
            end
        end
    end
    return frequent
end
```

frequency computation of $I_{k-1}' \overset{w_{k-1}'}{\rightarrowtail} I_k'$ in P based on Property 1. Note that if $I_{k-1}' \overset{w_{k-1}'}{\rightarrowtail} I_k'$ is already in the pattern graph, then we continue the recursive exploration of the prefix without processing anything for I_k' (lines 8–11).

4.4 Size of the search space and computational complexity

Size of the search space Basically, a weighted path pattern is a sequence of itemsets representing a set of sub-paths in the input graph. The number of possible itemsets for each element of such pattern is $2^n - 1$, with n the number of different items. Given a path of size k in the input graph, with all vertices labelled by the n items. The number of possible sub-patterns is $(2^n - 1)^k$. In the worst case, the number of paths of size k in a directed acyclic graph is the number of k-combinations of vertices, i.e., $\binom{|V_G|}{k}$. Thus, we have $\binom{|V_G|}{k} \times (2^n - 1)^k$ possible patterns of size k. Let k_{max} be the longest path in the input graph. The size of search space is $\sum_{k=2}^{k_{max}} \left(\binom{|V_G|}{k} \times (2^n - 1)^k\right)$.

Computational complexity Algorithm 2 recursively extends patterns to find the solutions. To study the computational complexity of this algorithm, we focus on the complexity of generating one solution. Each recursive call generates all the valid extensions of the current pattern w.r.t. frequency and non-redundancy constraints. These extensions increase the size of the current pattern by one itemset.

Line 1 lists all outgoing edges from the occurrences of the current pattern. In the worst case, its time complexity is linear in the number of edges in the input graph, i.e., $O(|E_G|)$. Line 2 extracts all frequent closed itemsets from the database generated from these outgoing edges. To our knowledge, one of the most efficient algorithm for this task is LCM [65]. Its time complexity is linear in the number of frequent closed itemsets, i.e., $O(2^n)$ in the worst case. Then, lines 3–26 correspond to the enumeration of all frequent non-redundant extensions based on closed itemsets. Let us study one iteration of this loop (because we study the computational complexity of generating one solution).

Line 4 searches all the occurrences of a given itemset extension in the projected database $cand(E|_P)$. Its time complexity is $O(|E_G| \times n)$ in the worst case. Lines 5–9 represent the generation of the first element of the pattern. First, the algorithm searches if this element has not been previously generated in the pattern graph (lines 4–6). Vertices of the pattern graph are stored in a tree-based data structure, and finding a element in such data structure is logarithmic in the input size, i.e., $O(log(|V_{Th}|)$. Then, the algorithm generates a new vertex and inserts it in the pattern graph (if necessary), with a time complexity of $O(1)$.

In lines 11–12, the algorithm uses the current vertex of the pattern graph to check if the current extension is a partial extension. The time complexity of this operation is $O(|V_G| \times n)$. If it is a partial extension, the algorithm executes a backward update (line 13), i.e., it inserts new vertices in the pattern graph (Algorithm 3). The time complexity of this algorithm for one itemset extension is $O(log(|E_{Th}|) + |E_G| \times n)$ in the worst case (such as lines 4–6). If it is not a partial extension, lines 17–25 check if this extension has already been generated in the pattern graph. The time complexity of these operations is $O(log(|V_{Th}|))$ for the search and $O(1)$ for the new vertex/edge generation.

At the end, the time complexity of our algorithm to extract one pattern of size k is $O(2^n + |E_G| \times n + |V_G| \times n + log(|V_{Th}|))$ in the worst case. Note that it is much less in practice because we don't have all vertices and all edges to check at each iteration (due to graph projections).

The space complexity corresponds mainly to the size of the pattern graph, i.e., $O(|E_{Th}| + |V_{Th}| \times (n + |V_G|))$ (the space complexity of LCM is linear in the input size). In the worst case, this size depends on the number of solutions which can be very big. However, as highlighted in our experiments (see next section), memory usage remains acceptable because vertices and edges are often shared across several solutions.

5 Experimental results

Our `PrefixPathGrowth-PatternGraph` algorithm has been implemented in C++. Experiments were performed on a computer running Ubuntu 14.04 LTS with an Intel Core i5 @ 3.10 GHz and 16 GB of main memory. First, we use our approach to study spatiotemporal data dealing with soil erosion monitoring. Second, we demonstrate its genericity and scalability by using data from a patent citation network as well as nine synthetic datasets. Performance analysis is done w.r.t. execution times, memory usage and number of solutions.

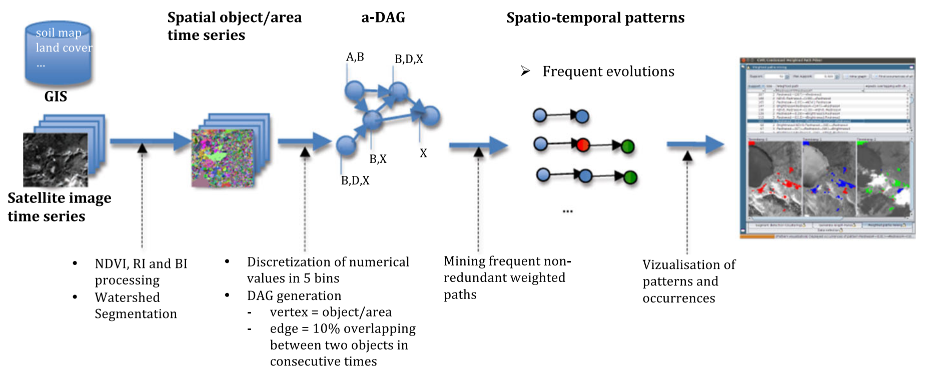

soil map
land cover
...
GIS
Satellite image time series
Spatial object/area time series
a-DAG
A,B
B,D,X
B,X
X
B,D,X
Spatio-temporal patterns
➢ Frequent evolutions
...
• NDVI, RI and BI processing
• Watershed Segmentation
• Discretization of numerical values in 5 bins
• DAG generation
- vertex = object/area
- edge = 10% overlapping between two objects in consecutive times
Mining frequent non-redundant weighted paths
Vizualisation of patterns and occurrences

Fig. 10 The KDD process on the soil erosion dataset

5.1 Application to soil erosion monitoring: qualitative and performance analysis

Soil erosion is a worldwide major issue which affects both environment and economy. This phenomenon is natural, but it is greatly accelerated by anthropic activities (e.g., bush fires, deforestation, mining projects) and climate change (resulting in intense precipitation events). It has also a strong impact on connected terrestrial and coastal ecosystems such as mangrove and coral reefs. Identifying key components of these erosion processes is essential to good environmental management and sustainable development.

In this work, we study a satellite image time series of a region impacted by soil erosion. This time series is composed of five images (SPOT4 and SPOT5) dated in 1999, 2002, 2005, 2008 and 2009. Spatial resolution of these images is 10 meters. Size of the studied area is 794×660 pixels, i.e., $52.5\,\text{km}^2$ ($20.3\,\text{mi}^2$). In addition to these raster data, we also have the following vector data: a soil map, land cover data and a digital elevation model from which we derived the slope.

To analyze such heterogenous data, several pre- and postprocessing are necessary. First, remote sensing indicators related to soil erosion monitoring are processed from the images. Three indicators related to soil erosion have been used: NDVI (Natural Difference Vegetation Index),3 RI (Redness Index) and BI (Brightness Index). The NDVI is a common measure to observe vegetation. The RI is used to quantify bare soils (in the studied areas, soils are rich in irons and thus are rusty red). The BI is also a measure to study degraded lands [6]. We also integrate the vector data available on the studied area (i.e., soil, land cover and slope). Then, each satellite image is segmented, i.e., pixels sharing similar values are grouped in order to obtain homogenous regions/objects. We use the watershed method [8] to group pixels sharing similar values into objects. We choose this classical method because it generates lots of small very homogenous regions (over-segmentation). Finally, numerical attributes are discretized and the a-DAG is generated. In our experiments, numerical attributes are discretized into 5 classes representing 5 intensity levels from 0 to 4. Vertices of the a-DAG represent the different spatial objects/regions in the segmented images. Each vertex is labeled with a set of attributes which are discretized remote sensing indicators and attributes of the vector data (land cover, soil type, etc). Edges of the a-DAG represent the possible influence/evolution of an object at time t on/into an object at time $t + 1$. Two objects/vertices are linked through an edge if they are overlapping in two consecutive times. In these experiments, we chose to link two objects o_1 at time t and o_2 at time $t + 1$ if at

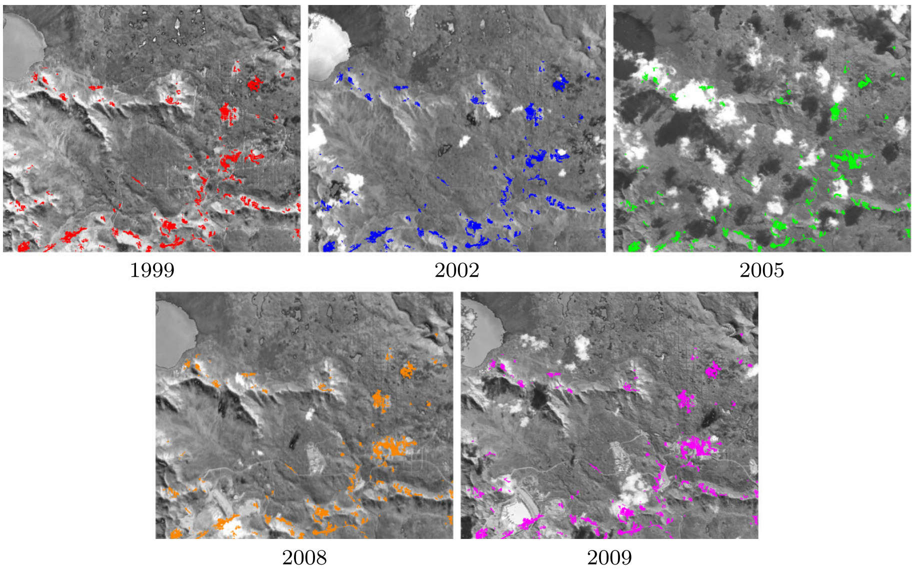


Fig. 11 Pattern Redness4, Slope$[3.6;\ 30]$ $\overset{246}{\rightarrowtail}$ Redness4, Slope$[3.6;\ 30]$ $\overset{244}{\rightarrowtail}$ Redness4, Slope$[3.6;\ 30]$ $\overset{252}{\rightarrowtail}$ Redness4, Slope$[3.6;\ 30]$ $\overset{259}{\rightarrowtail}$ Redness4, Slope$[3.6;\ 30]$

least 10% of o_1's pixels are in o_2. This threshold has been chosen after several experiments and feedbacks from the experts. It enables to capture fine interactions and important evolutions, while avoiding many meaningless relations to be considered. At the end, extracted patterns were analyzed with a geologist specialist of soil erosion. To facilitate interpretation, we developed a tool to display occurrences of a given pattern on the original images (in order to position the pattern in its geographical context). Thus, the expert can visualize where, when and how frequent non-redundant weighted paths (i.e., frequent evolutions) occurred. Since pattern occurrences might start at different times, we use different colors to identify temporal positions.

This workflow illustrated in Fig.10 has been integrated in the KNIME Analytics Platform [7] through the development of several "nodes". A detailed description of this KDD process and its implementation in a KNIME plugin called PaTSI (PAttern mining of Time Series of satellite Images) can be found in [18].

5.1.1 Qualitative results

This subsection details three interesting patterns obtained by our pattern mining algorithm for this area. First, it was interesting to notice that most of the zones impacted by soil erosion (with a *Redness4* value) had a low slope as illustrated in Fig. 11.

Figure 11 shows the zones supporting the pattern $Redness4, Slope[3.6; 30] \overset{246}{\rightarrowtail} Redness4, Slope[3.6; 30] \overset{244}{\rightarrowtail} Redness4, Slope[3.6; 30] \overset{252}{\rightarrowtail} Redness4, Slope[3.6; 30] \overset{259}{\rightarrowtail} Redness4, Slope[3.6; 30]$. When we compared this pattern to $Redness4 \overset{358}{\rightarrowtail} Redness4 \overset{356}{\rightarrowtail} Redness4 \overset{362}{\rightarrowtail} Redness4 \overset{373}{\rightarrowtail} Redness4$ (another extracted pattern), we noticed that

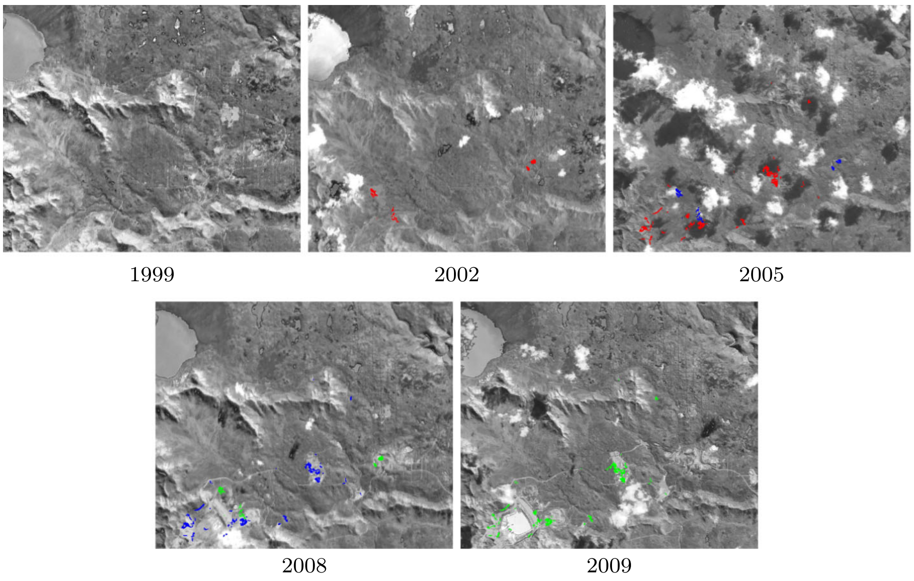


Fig. 12 Pattern Redness1 $\overset{58}{\rightarrowtail}$ Redness4, NDVI0 $\overset{61}{\rightarrowtail}$ Redness4, NDVI0, Brightness4

its frequency (244) represented almost 69% of the second pattern frequency (356). Moreover, spatial distribution of this pattern showed that impacted zones were situated at the base of more sloppy zones. Presence of a strong Redness Index in such zones showed that these zones represented deposition zones for sediments coming from higher places.

Figure 12 shows zones impacted by mining activities. New mines and buildings appeared in this area between 2005 and 2008. We can see them in the center and in the bottom-left of the 2008 and 2009 images. As a consequence, soil erosion increased in this area during this period (Redness1 to Redness4). This soil degradation is confirmed by a low vegetation index (NDVI0) and a high brightness index (Brightness4). We can notice that this degradation extends progressively. It began in 2005 with few areas (in blue color in the 2005 image) and continued in 2008 with other nearby (in blue color in the 2008 image). We also observe that once an area became degraded, it stayed degraded for a long period of time (Redness4 followed by Redness4 in the pattern).

A small part of this area was characterized by an increase in vegetation. Figure 13 shows the pattern representing this evolution. Vegetation index (NDVI) is gradually growing from medium (*NDVI2*) to very high (*NDVI4*). As shown in images, only few areas followed such an evolution (in the southwest from 1999 to 2005 and in the northeast from 2005 to 2009). Moreover, some of these zones (denoted by β in the 2009 image) were small lakes. In this case, an increase in NDVI may only represent an algae proliferation. Other zones (denoted by γ in the 2009 image) were old trails. Since those trails were less frequented, vegetation had grown. The α zone became a forest. Its evolution may be related to a densification of the vegetation due to new plants. It may also be related to an increase in tree size. However, in such a case, it is strange that other forests had not experienced a similar evolution. Field investigations are required to actually explain those changes in detail.

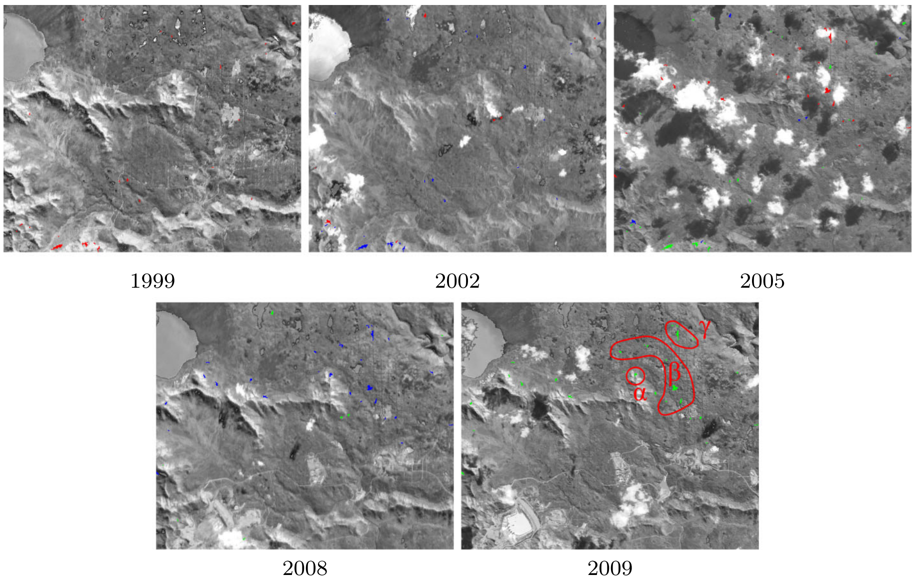


Fig. 13 Pattern NDVI2 $\overset{57}{\rightarrowtail}$ NDVI3 $\overset{58}{\rightarrowtail}$ NDVI4

Table 1 Characteristics of the soil erosion dataset

Dataset	# of edges	# of vertices	# of items per vertex	Total # of items	graph density [71]
Soil erosion	41 166	25 618	6	262	0.000063

5.1.2 Quantitative results

Table 1 presents the main characteristics of the a-DAG generated for this dataset.

Figure 14 shows execution times, memory usage and number of solutions for several minimum support thresholds. It demonstrates that our algorithm is efficient on this dataset until down to very low support thresholds (lower than 1%). For very low support thresholds, the algorithm can generate an important number of solutions. For example, it extracted almost one million solutions with a minimum support threshold of 0.1%. That highlights the importance of considering other constraints (e.g., constraints defined by experts) during pattern mining or in postprocessing. In the present work, we only take into account expert constraints in postprocessing (using filters and regular expressions).

5.2 Application to a patent citation network and synthetic datasets : genericity and detailed performance analysis

To demonstrate that our approach is generic and scalable, we also study real-world data from a patent citation network and nine synthetic datasets. A summary of these datasets and their parameters can be found in Table 2.

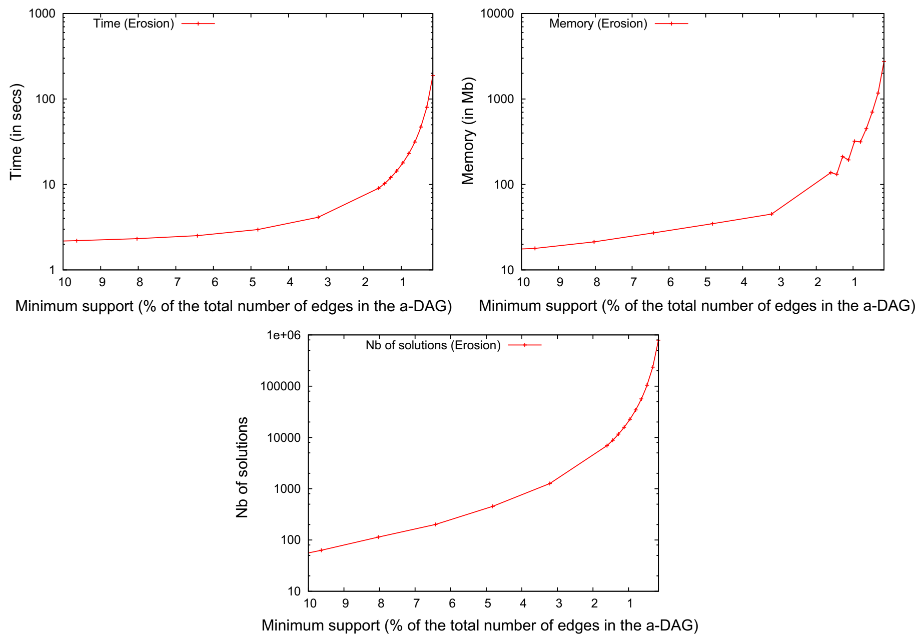


Fig. 14 Execution times, memory usage and number of solutions for the soil erosion dataset

Table 2 Characteristics of the patent citation network and the synthetic datasets

Dataset	# of edges (E)	# of vertices (v)	# of items (λ) per vertex	Total # of items	Graph density [71]
Patent citation network	414,487	184,284	5–7	506	0.000012
V20K E60K λ1–5	60,000	20,000	1–5	15	0.00015
V40K E120K λ1–5	120,000	40,000	1–5	15	0.000075
V200K E600K λ1–5	600,000	200,000	1–5	15	0.000015
V20K E60K λ5–10	60,000	20,000	5–10	15	0.00015
V40K E120K λ5–10	120,000	40,000	5–10	15	0.000075
V200K E600K λ5–10	600,000	200,000	5–10	15	0.000015
V20K E60K λ5–10 in 10 layers	60,000	20,000	5–10	15	0.00015
V40K E120K λ5–10 in 10 layers	120,000	40,000	5–10	15	0.000075
V200K E600K λ5–10 in 10 layers	600,000	200,000	5–10	15	0.000015

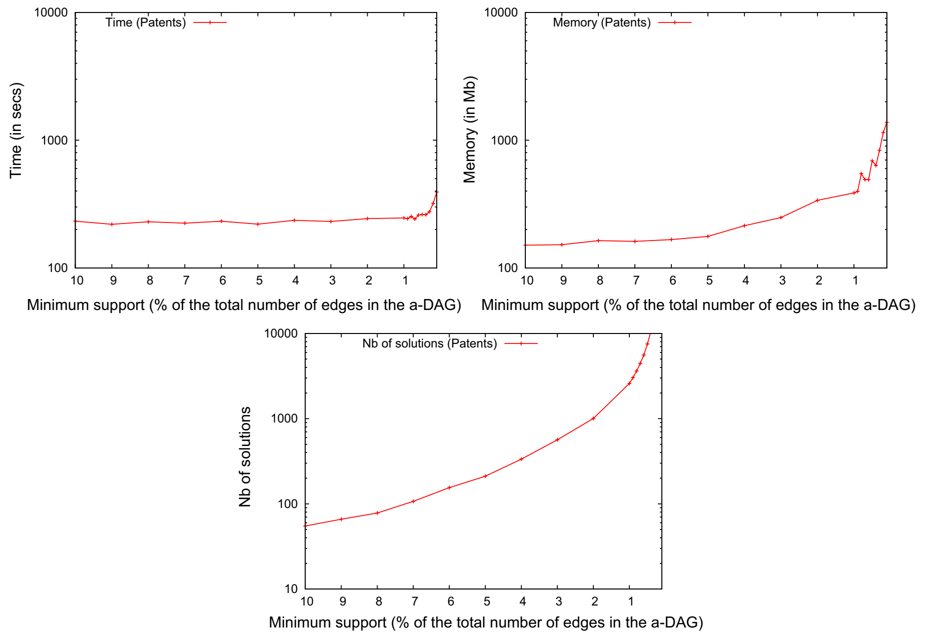


Fig. 15 Execution times, memory usage and number of solutions for the patent citation network

5.2.1 The patent citation network: genericity and scalability

The patent citation network used is a subgraph of the *cit-Patents* graph from the Stanford Large Network Dataset Collection [44,45]. In this dataset, vertices represent patents granted in the USA between 1975 and 1999, and edges represent citations. Each vertex is labeled with 5 to 7 items corresponding to country, state, year, type, category and sub-category of a patent. As shown in Table 2, this dataset is quite large but highly sparse (w.r.t. structure and items).

Figure 15 shows execution times, memory usage and number of solutions generated for this dataset. Execution times are higher than with the previous dataset, but do not exceed 3 min down to a minimum support threshold of 0.1%. Memory usage is smaller (1 Gb in the worst case against 3 Gb previously) since the graph is less dense. Number of solutions is relatively stable (around 500 patterns) down to a minimum support threshold of 1%. Those results demonstrate that our algorithm can efficiently deal with relatively large sparse datasets. They also show that our approach can be used to mine any attributed DAG, and not only spatiotemporal data.

5.2.2 The synthetic datasets: influence of data characteristics on performances

To study more in detail the influence of different parameters on performances, we generate nine synthetic datasets using the *DigraphGenerator* proposed in [59]. This approach constructs a labelled DAG containing a given number of vertices and edges. Edges are randomly generated (following an uniform distribution). To obtain attributed DAGs, vertices are simply labeled with itemsets. For each vertex, the number of items is randomly chosen (following a

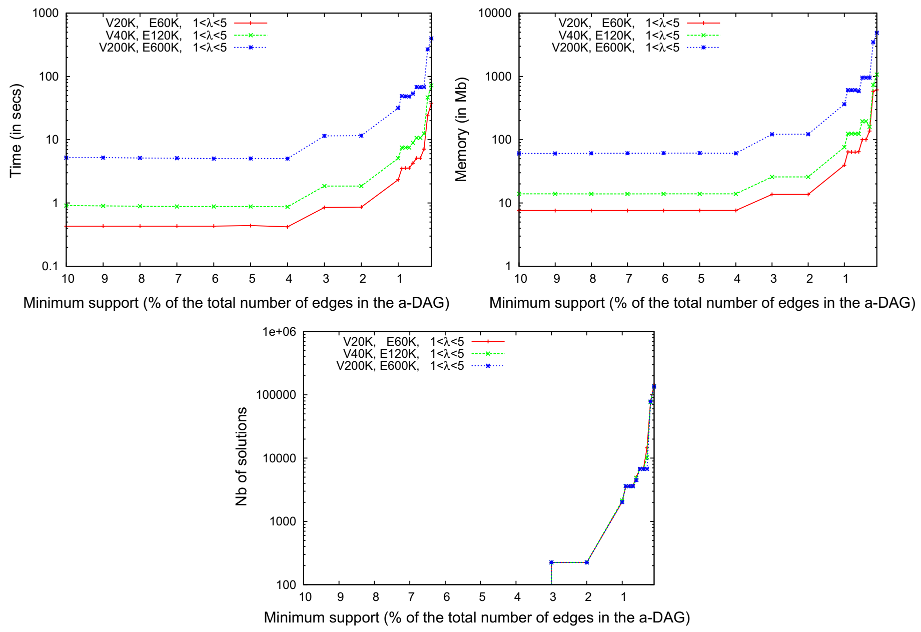


Fig. 16 Execution times, memory usage and number of solutions for synthetic datasets when increasing graph size

gaussian distribution). Then, items are selected among a set of 15 ones (following an uniform distribution).

Figure 16 shows execution times, memory usage and number of solutions for the first three synthetic datasets. The left-hand graph highlights the impact of graph size on execution times. As expected, execution times increase when graphs grow. Note that our algorithm is still efficient for large graph (600 000 edges) down to very low supports (1%). The right-hand graph shows memory usage, and it increases in the same way than execution times. The bottom graph shows that number of solutions are very similar (but not identical if we look at detailed results) for all those datasets. It highlights the problem of synthetic dataset generation, which is a well-known problem in pattern mining.

Figure 17 illustrates the impact of the number of items per vertex on performances. The total number of items is not changed in these experiments. We have only more frequent items, more frequent itemsets and longer ones when the number of items per vertex is between 5 and 10. Once again, as expected, execution times and memory usage increase when itemset sizes increase. However, if we compare with previous results, we notice that the impact is more important. As shown in Fig. 18, the number of solutions dramatically increases when we add items. It also points out the difference between (labeled) graph mining and attributed graph mining. Combinatorial complexity of attributed graph mining is much higher than labeled graph mining, since complexity of itemset mining, which is exponential in the number of items, is "added to each vertex". It has to be noticed that in Fig. 18, we only present results for one of the studied datasets because number of solutions were relatively similar (as discussed previously).

Finally, we studied performances on synthetic a-DAGs produced by an adapted version of *DigraphGenerator*. Those a-DAGs have the same number of vertices and edges, and the

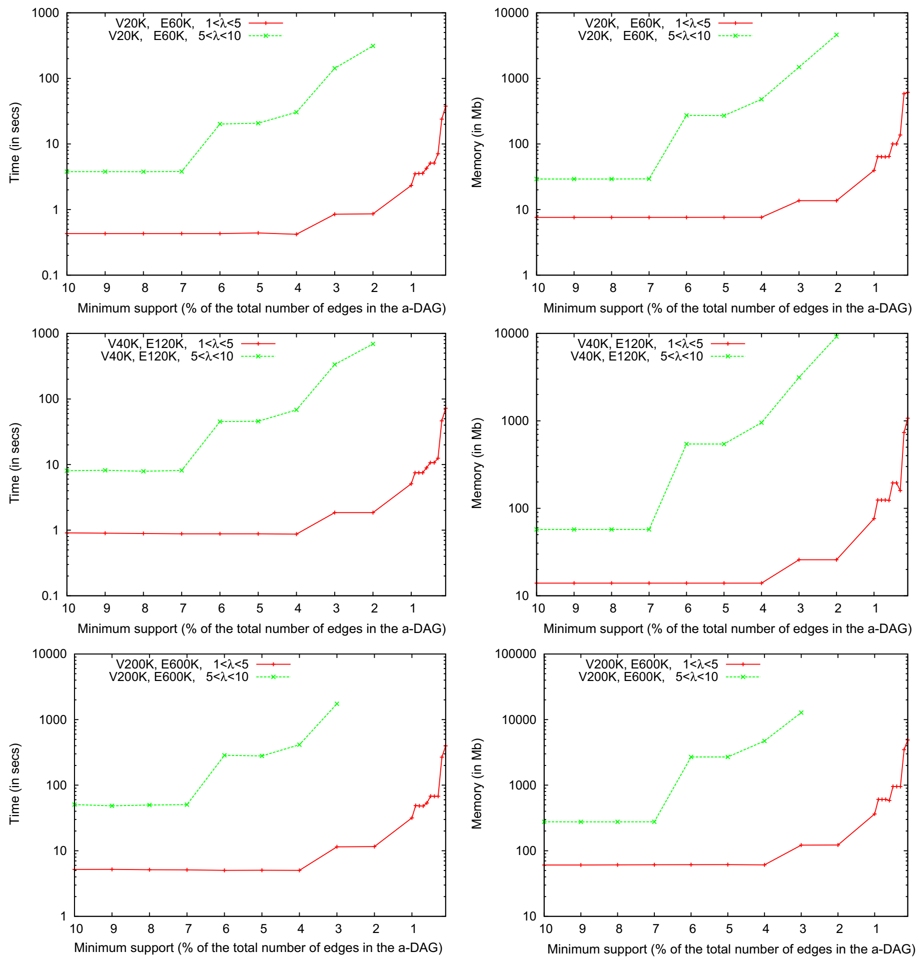


Fig. 17 Execution times and memory usage for synthetic datasets when increasing number of items per vertex

same distribution w.r.t. itemsets, than the previous synthetic datasets, but the graph structure is different. Instead of randomly generating edges, we partitioned vertices in 10 sets (called “layers”), we defined a total order between those sets, and we only generated edges between vertices of two consecutive layers. As a consequence, we have longer paths and a greater number of patterns mined. That graph structure is a typical characteristic of a-DAGs obtained in spatiotemporal applications. Those graphs have a particular structure due to their temporal dimension. As shown in Fig. 19, execution times and memory usage were higher with layer-based graph structures. However, the algorithm is still efficient down to low minimum support threshold (9–7%).

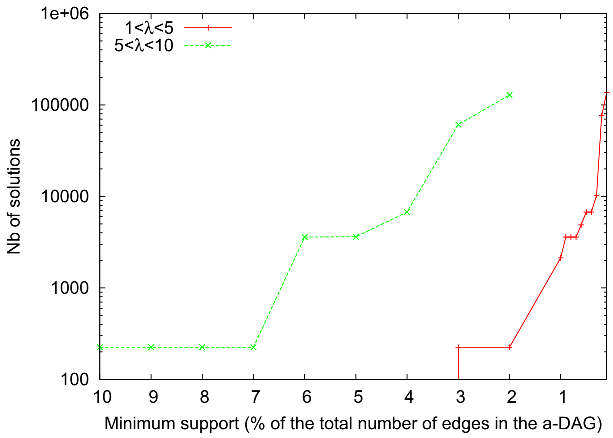


Fig. 18 Number of solutions for synthetic datasets when increasing number of items per vertex

6 Conclusion and perspectives

In this paper, we propose a new algorithm to mine frequent patterns in a single a-DAG, and its application to spatiotemporal environmental data. We show that this new graph-based representation, called attributed DAGs, can capture complex interactions between objects with all their characteristics/events. We use this data representation to represent complex spatiotemporal phenomena, in particular ones with moving and changing objects. In our application example related to soil erosion, objects may move, appear, disappear, merge or split. We use a new constrained pattern domain, frequent non-redundant weighted paths, to study interesting substructures in such data. We propose a new algorithm based on pattern-growth strategy and an optimized data structure to efficiently mine such graph. Our experiments on real and synthetic datasets not only demonstrated the relevance of extracted patterns for soil erosion issues, but also the scalability and genericity of our approach. Indeed, results obtained for the spatiotemporal dataset successfully highlighted the evolution of soil erosion w.r.t. vegetation, slope and mining activities. Results also show that our approach can be used to mine larger graph (e.g., graphs with 200,000 vertices, 600,000 edges, and 7.5 attributes per vertex in average) and that it can be used in other contexts such as citation networks.

The present work offers several perspectives. In the specific setting of spatiotemporal data, a future work could be to study the addition of spatial neighbors in the analysis. Spatial relationships at a given time could be integrated in the data representation and in the pattern language. Such integration is challenging since it would impact the scalability of algorithms. Moreover, extracted patterns would be more complexes to interpret by domain experts, requiring dedicated visualization approaches. Another perspective would be to integrate other statistical constraints and domain knowledge (through domain constraints) during pattern mining. A first work has been done in the itemset mining setting in [27]. It takes advantage of mathematical domain models to improve efficiency and relevancy of itemset mining. Finally, another possibility would be to extend this work to propose a condensed representation for frequent subgraph mining in the single-graph setting. Currently, only maximal patterns (w.r.t. structure inclusion only) have been studied in this

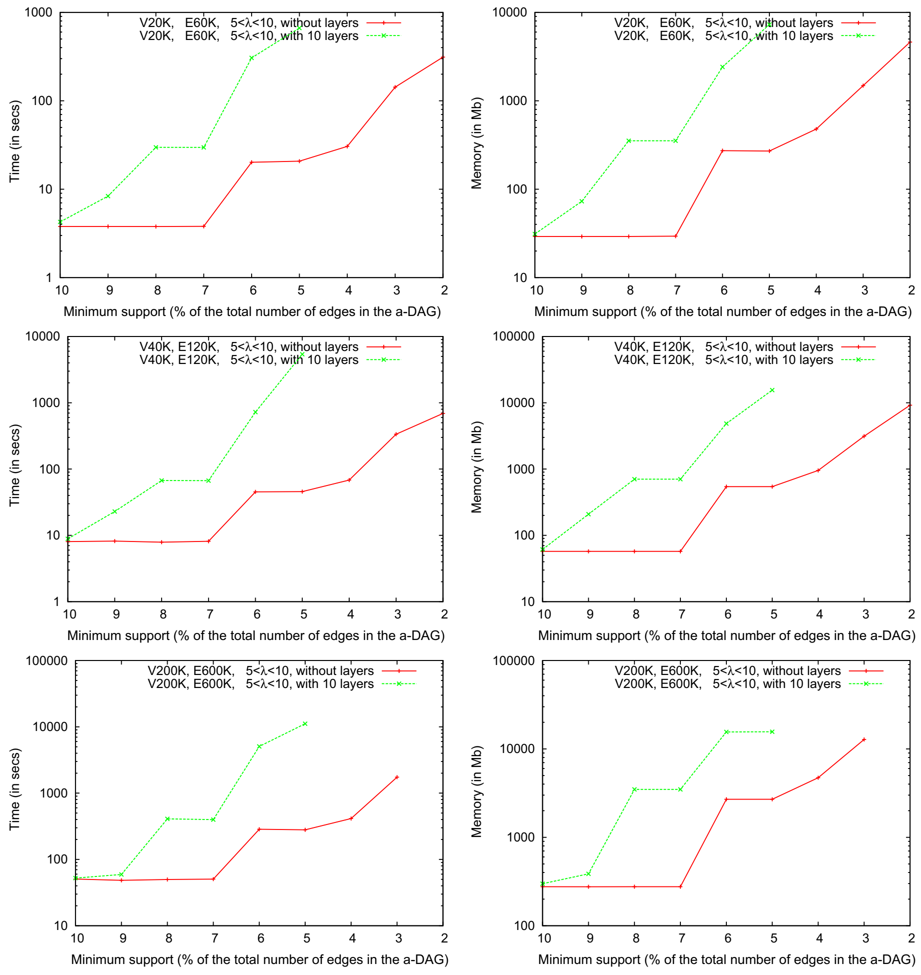


Fig. 19 Execution times and memory usage for synthetic datasets with a layer-based structure

setting. However, with such representation, we loose the information about frequency of sub-patterns.

References

1. Agrawal R, Srikant R (1994) Fast algorithms for mining association rules in large databases. In: Proceedings of the 20th international conference on very large data bases (VLDB). Morgan Kaufmann, pp 487–499
2. Agrawal R, Srikant R (1995) Mining sequential patterns. In: Proceedings of the eleventh international conference on data engineering (ICDE). IEEE Computer Society, pp 3–14
3. Alatrista-Salas H, Bringay S, Flouvat F, Selmaoui-Folcher N, Teisseire M (2012) The pattern next door: towards spatio-sequential pattern discovery. In: Advances in knowledge discovery and data mining. Springer, pp 157–168

4. Arimura H, Uno T (2009) Polynomial-delay and polynomial-space algorithms for mining closed sequences, graphs, and pictures in accessible set systems. In: Proceedings of the SIAM international conference on data mining (SDM). SIAM, pp 1088–1099
5. Aydin B, Angryk RA (2016) A graph-based approach to spatiotemporal event sequence mining. In: Proceedings of the IEEE international conference on data mining workshops (ICDMW). IEEE Computer Society, pp 1090–1097
6. Bannari A, Morin D, Bonn F, Huete A (1995) A review of vegetation indices. Remote Sens Rev 13(1–2):95–120
7. Berthold MR, Cebron N, Dill F, Gabriel TR, Kötter T, Meinl T, Ohl P, Sieb C, Thiel K, Wiswedel B (2007) KNIME: the Konstanz information miner. In: Studies in classification, data analysis, and knowledge organization (GfKL 2007). Springer
8. Beucher S, Meyer F (1993) The morphological approach to segmentation: the watershed transformation. Mathematical morphology in image processing. Opt Eng 34:433–481
9. Bonchi F, Lucchese C (2004) On closed constrained frequent pattern mining. In: Proceedings of the IEEE international conference on data mining (ICDM). IEEE Computer Society, pp 35–42
10. Borges J, Levene M (2000) A fine grained heuristic to capture web navigation patterns. ACM SIGKDD Explor 2(1):40–50
11. Boulicaut JF, Bykowski A, Rigotti C (2003) Free-sets: a condensed representation of boolean data for the approximation of frequency queries. Data Min Knowl Discov 7(1):5–22
12. Bringmann B, Nijssen S (2008) What is frequent in a single graph? In: Proceedings of the Pacific-Asia conference on advances in knowledge discovery and data mining (PAKDD). Springer, pp 858–863
13. Calders T, Rigotti C, Boulicaut JF (2004) A survey on condensed representations for frequent sets. In: Constraint-based mining and inductive databases. Springer, pp 64–80
14. Casali A, Cicchetti R, Lakhal L (2005) Essential patterns: a perfect cover of frequent patterns. In: Proceedings of the international conference on data warehousing and knowledge discovery (DaWaK). Springer, pp 428–437
15. Celik M, Shekhar S, Rogers JP, Shine JA (2008) Mixed-drove spatiotemporal co-occurrence pattern mining. IEEE Trans Knowl Data Eng 20(10):1322–1335
16. Chen MS, Park JS, Yu PS (1998) Efficient data mining for path traversal patterns. IEEE Trans Knowl Data Eng 10(2):209–221
17. Chen Yl, Kao Hp, Ko Mt (2004) Mining DAG patterns from DAG databases. In: Advances in web-age information management, pp 579–588
18. Collin M, Flouvat F, Selmaoui-Folcher N (2016) Patsi: pattern mining of time series of satellite images in knime. In: Proceedings of the IEEE international conference on data mining workshops (ICDMW). IEEE Computer Society, pp 1292–1295
19. Cook D, Holder L (2006) Mining graph data. Wiley, New York
20. De Raedt L, Kramer S (2001) The levelwise version space algorithm and its application to molecular fragment finding. In: Proceedings of the international joint conference on artificial intelligence (IJCAI), vol 2. Morgan Kaufmann, pp 853–859
21. De Raedt L, Jaeger M, Lee SD, Mannila H (2002) A theory of inductive query answering. In: Proceedings of the IEEE international conference on data mining (ICDM). IEEE Computer Society, pp 123–130
22. Douar B, Liquiere M, Latiri C, Slimani Y (2015) Lc-mine: a framework for frequent subgraph mining with local consistency techniques. Knowl Inf Syst 44(1):1–25
23. Dube MP, Egenhofer MJ (2014) Surrounds in partitions. In: Proceedings of the ACM international conference on advances in geographic information systems (SIGSPATIAL). ACM, pp 233–242
24. Dube MP, Barrett JV, Egenhofer MJ (2015) From metric to topology: determining relations in discrete space. In: International workshop on spatial information theory. Springer, pp 151–171
25. Fariha A, Ahmed CF, Leung CKS, Abdullah S, Cao L (2013) Mining frequent patterns from human interactions in meetings using directed acyclic graphs. In: Proceedings of the Pacific-Asia conference on advances in knowledge discovery and data mining (PAKDD). Springer, pp 38–49
26. Fiedler M, Borgelt C (2007) Support computation for mining frequent subgraphs in a single graph. In: Mining and learning with graphs
27. Flouvat F, Sanhes J, Pasquier C, Selmaoui-Folcher N, Boulicaut JF (2014) Improving pattern discovery relevancy by deriving constraints from expert models. In: Proceedings of the European conference on artificial intelligence (ECAI). IOS Press, pp 327–332
28. Fukuzaki M, Seki M, Kashima H, Sese J (2010) Finding itemset-sharing patterns in a large itemset-associated graph. In: Proceedings of the Pacific-Asia conference on advances in knowledge discovery and data mining (PAKDD). Springer, pp 147–159
29. Garriga GC, Khardon R, De Raedt L (2012) Mining closed patterns in relational, graph and network data. In: Annals of mathematics and artificial intelligence, pp 1–28

30. Geng R, Xu W, Dong X (2007) WTPMiner: efficient mining of weighted frequent patterns based on graph traversals. In: Proceedings of the international conference on knowledge science, engineering and management (KSEM). Springer, pp 412–424
31. Giannotti F, Pedreschi D (eds) (2008) Mobility, data mining and privacy—geographic knowledge discovery. Springer, Berlin
32. Gudes E, Shimony SE, Vanetik N (2006) Discovering frequent graph patterns using disjoint paths. IEEE Trans Knowl Data Eng 18(11):1441–1456
33. Günnemann S, Seidl T (2010) Subgraph mining on directed and weighted graphs. In: Proceedings of the Pacific-Asia conference on advances in knowledge discovery and data mining (PAKDD). Springer, pp 133–146
34. Gunopulos D, Mannila H, Saluja S (1997) Discovering all most specific sentences by randomized algorithms extended abstract. Springer, Berlin
35. Haas BJ, Delcher AL, Wortman JR, Salzberg SL (2004) Dagchainer: a tool for mining segmental genome duplications and synteny. Bioinformatics 20(18):3643–3646
36. Huang Y, Shekhar S, Xiong H (2004) Discovering colocation patterns from spatial data sets: a general approach. IEEE Trans Knowl Data Eng 16(12):1472–1485
37. Inokuchi A, Washio T, Motoda H (2000) An apriori-based algorithm for mining frequent substructures from graph data. In: Proceedings of the European conference on principles of data mining and knowledge discovery (PKDD). Springer, vol 1910, pp 13–23
38. Jiang C, Coenen F, Zito M (2013) A survey of frequent subgraph mining algorithms. Knowl Eng Rev 28(01):75–105
39. Jiang J, Worboys M (2009) Event-based topology for dynamic planar areal objects. Int J Geogr Inf Sci 23(1):33–60
40. Jiang X, Xiong H, Wang C, Tan AH (2009) Mining globally distributed frequent subgraphs in a single labeled graph. Data Knowl Eng 68(10):1034–1058
41. Khan A, Yan X, Wu KL (2010) Towards proximity pattern mining in large graphs. In: Proceedings of the ACM international conference on management of data (SIGMOD). ACM Press, pp 867–878
42. Kuramochi M, Karypis G (2001) Frequent subgraph discovery. In: Proceedings of the IEEE international conference on data mining (ICDM). IEEE Computer Society, pp 313–320
43. Kuramochi M, Karypis G (2005) Finding frequent patterns in a large sparse graph*. Data Min Knowl Discov 11(3):243–271
44. Leskovec J, Krevl A (2014) SNAP Datasets: Stanford large network dataset collection. http://snap.stanford.edu/data
45. Leskovec J, Kleinberg J, Faloutsos C (2005) Graphs over time: densification laws, shrinking diameters and possible explanations. In: Proceedings of the ACM international conference on knowledge discovery in data mining (SIGKDD). ACM, pp 177–187
46. Lewis JA, Dube MP, Egenhofer MJ (2013) The topology of spatial scenes in r2. In: International conference on spatial information theory. Springer, pp 495–515
47. Miyoshi Y, Ozaki T, Ohkawa T (2009) Frequent pattern discovery from a single graph with quantitative itemsets. In: Proceedings of the IEEE international conference on data mining workshops (ICDMW), pp 527–532
48. Mohan P, Shekhar S, Shine JA, Rogers JP (2010) Cascading spatio-temporal pattern discovery: a summary of results. In: Proceedings of the SIAM international conference on data mining (SDM), pp 327–338
49. Mohan P, Shekhar S, Shine JA, Rogers JP (2012) Cascading spatio-temporal pattern discovery. IEEE Trans Knowl Data Eng 24(11):1977–1992
50. Moser F, Colak R, Rafiey A, Ester M (2009) Mining cohesive patterns from graphs with feature vectors. In: Proceedings of the SIAM international conference on data mining (SDM), pp 593–604
51. Nanopoulos A, Manolopoulos Y (2001) Mining patterns from graph traversals. Data Knowl Eng 37(3):243–266
52. Nguyen TT, Nguyen HA, Pham NH, Al-Kofahi JM, Nguyen TN (2009) Graph-based mining of multiple object usage patterns. In: Proceedings of the the joint meeting of the European software engineering conference and the ACM SIGSOFT symposium on the foundations of software engineering. ACM Press, pp 383–392
53. Nijssen S, Kok JN (2004) A quickstart in frequent structure mining can make a difference. In: Proceedings of the ACM international conference on knowledge discovery and data mining (SIGKDD). ACM, pp 647–652
54. Pasquier C, Flouvat F, Sanhes J, Selmaoui-Folcher N (2017) Attributed graph mining in the presence of automorphism. Knowl Inf Syst 50(2):569–584
55. Pasquier N, Bastide Y, Taouil R, Lakhal L (1999) Discovering frequent closed itemsets for association rules. In: Proceedings of the international conference on database theory (ICDT). Springer, pp 398–416

56. Pei J, Han J, Mortazavi-Asl B, Wang J, Pinto H, Chen Q, Dayal U, Hsu M (2004) Mining sequential patterns by pattern-growth: the prefixspan approach. IEEE Trans Knowl Data Eng 16(11):1424–1440
57. Qian F, He Q, He J (2009) Mining spread patterns of spatio-temporal co-occurrences over zones. In: Proceedings of the international conference on computational science and its applications (ICCSA). Springer, vol 5593, pp 677–692
58. Sanhes J, Flouvat F, Pasquier C, Selmaoui-Folcher N, Boulicaut J (2013) Weighted path as a condensed pattern in a single attributed DAG. In: Proceedings of the international joint conference on artificial intelligence (IJCAI)
59. Sedgewick R, Wayne K (2011) Algorithms, 4th edn. Addison-Wesley, Reading
60. Selmaoui-Folcher N, Flouvat F (2011) How to use classical tree mining algorithms to find complex spatio-temporal patterns? In: Proceedings of the international conference on database and expert systems applications (DEXA). Springer, pp 107–117
61. Silva A, Meira W Jr, Zaki MJ (2012) Mining attribute-structure correlated patterns in large attributed graphs. Proceedings of the VLDB Endowment 5(5):466–477
62. Sindoni G, Stell JG (2017) The logic of discrete qualitative relations. In: Proceedings of the international conference on spatial information theory (COSIT). Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik, vol 86, pp 1–15
63. Termier A, Tamada Y, Numata K, Imoto S, Washio T, Higushi T, Higuchi T (2007) DigDag, a first algorithm to mine closed frequent embedded sub-DAGs. In: Proceedings of mining and learning with graphs (MLG), pp 1–5
64. Tsoukatos I, Gunopulos D (2001) Efficient mining of spatiotemporal patterns. In: Proceedings of the international symposium on spatial and temporal databases (SSTD). Springer, vol 2121, pp 425–442
65. Uno T, Asai T, Uchida Y, Arimura H (2003) LCM: an efficient algorithm for enumerating frequent closed item sets. In: Proceedings of the IEEE international conference on data mining workshop on frequent itemset mining implementations (FIMI). CEUR-WS.org, vol 90
66. Uno T, Asai T, Uchida Y, Arimura H (2004) An efficient algorithm for enumerating closed patterns in transaction databases. In: Proceedings of the international conference on discovery science (DS). Springer, pp 16–31
67. Wang J, Hsu W, Lee ML, Wang JTL (2004) FlowMiner: finding flow patterns in spatio-temporal databases. In: Proceedings of the IEEE international conference on tools with artificial intelligence (ICTAI). IEEE Computer Society, pp 14–21
68. Wang J, Hsu W, Lee ML, Sheng C (2006) A partition-based approach to graph mining. In: Proceedings of the IEEE international conference on data engineering (ICDE). IEEE Computer Society, pp 74—-74
69. Washio T, Motoda H (2003) State of the art of graph-based data mining. SIGKDD Explora Newsl 5(1):59–68
70. Washio T, Mitsunaga Y, Motoda H (2005) Mining quantitative frequent itemsets using adaptive density-based subspace clustering. In: Proceedings of the IEEE international conference on data mining (ICDM). IEEE Computer Society, pp 793–796
71. Wasserman S, Faust K (1994) Social network analysis: methods and applications, vol 8. Cambridge University Press, Cambridge
72. Werth T, Dreweke A, Wörlein M, Fischer I, Philippsen M (2008) Dagma: mining directed acyclic graphs. In: Proceedings of the IADIS European conference on data mining. IADIS Press, pp 11–18
73. Werth T, Wörlein M, Dreweke A, Fischer I, Philippsen M (2009) Dag mining for code compaction. In: Data mining for business applications. Springer, pp 209–223
74. Worboys M (2012) The maptree: a fine-grained formal representation of space. In: International conference on geographic information science. Springer, pp 298–310
75. Yan X, Han J (2002) gSpan: Graph-bases substructure pattern mining. In: Proceedings of the IEEE international conference on data mining (ICDM). IEEE Computer Society, vol 3, pp 721–724
76. Yan X, Han J (2003) CloseGraph. In: Proceedings of the ACM international conference on knowledge discovery and data mining (SIGKDD). ACM Press, vol 6, p 286
77. Yan X, Han J, Afshar R (2003) Clospan: mining: closed sequential patterns in large datasets. In: Proceedings of the SIAM international conference on data mining (SDM), pp 166–177
78. Yang H, Parthasarathy S, Mehta S (2005) A generalized framework for mining spatio-temporal patterns in scientific data. In: Proceedings of the ACM international conference on knowledge discovery and data mining (SIGKDD). ACM Press, pp 716–721

Frederic Flouvat is a specialist in data mining at the University of New Caledonia. He obtained his PhD in computer sciences in 2006 at the University of Clermont-Ferrand (France), where he studied the development of adaptive and generic solutions for interesting pattern mining in data. After his PhD, he joined the University of Lyon (France) and its database team. Since 2008, he has been an associate professor at the University of New Caledonia, and a member of the Institute of Exact and Applied Sciences. This laboratory brings together biologists, geologists, physicists, chemists, mathematicians and computer scientists to address both fundamental and applied questions related to the concepts of risk and sustainable development. In this multidisciplinary context, his work is focused on spatiotemporal data mining (e.g. environmental databases, sensor data, time series of satellite images), with a particular interest in expert knowledge integration. He participated in several regional, national and local research projects dealing with various questions s.t. soil erosion, dengue fever spread, animal trajectories or dynamic of informal settlements.

Nazha Selmaoui-Folcher is Associate Professor (HDR) at the University of New Caledonia since 1998. She was the leader of a multidisciplinary laboratory on matter and environment 2012–2016 (PPME). She was the coordinator for the ANR FOSTER projects dedicated to Knowledge Discovery on spatio-temporal Databases and Application to Soil Erosion (2011–2014). She participates to ANR project SpiRAL (Soils, rainfall and leptospirosis Understanding leptospirosis environmental contamination). She is member of LABEX-CORAIL (laboratory of excellence, Funded by French Agency) and associate member of IPAL (International Research Lab in Singapore). She received her PhD degree in 1992 from the "Institute National des Sciences Appliques" at Lyon (France) and her Habilitation degree in 2012 from the University of Lyon I. Her current research interests are image analysis, machine learning, artificial intelligence, graph mining, data mining on temporal series of satellite images, spatio-temporal data and its application to health and environmental sciences. She is involved in the program committees of many data mining conferences and Journals.

Jérémy Sanhes obtained his PhD in computer sciences in 20014 at the University of New Caledonia, where he studied spatio-temporal pattern mining. His work focused more particularly on graph representations and integration of domain knowledge. He is currently a data scientist and engineer at Reelevant, a company developing technologies related to smart emails.

Chengcheng Mu is currently a lead engineer at Versent, a consulting company in cloud technologies. In the past, he also worked as a junior engineer in the FOSTER project dealing with pattern mining and soil erosion in New Caledonia.

Claude Pasquier is a researcher at French National Center for Scientific Research (CNRS). He received a PhD degree in Computer Science from the University of Nice–Sophia Antipolis (now Université Côte d'Azur), France, in 1994. During his thesis, he explored the use of software engineering paradigms in the field of structured document manipulation. Subsequently, he was a postdoctoral researcher at the Biophysics and Bioinformatics Laboratory of the University of Athens, Greece, where he conducted research on protein structure prediction. He held positions at the National Institute for Research in Digital Science and Technology (INRIA) and with Schlumberger, Smart Cards & Terminals division (now Gemalto) where he worked on generative programming. Since 2002 when he joined CNRS, he successively worked at Villefranche Oceanographic Laboratory (LOV), the Institute of Biology Valrose (iBV) and New Caledonia Institute of Exact and Applied Sciences (ISEA) where he addressed topics as diverse as semantic data integration, omics data mining and attributed graph mining. Currently at I3S laboratory, he is conducting research focused on complex network mining that combines computer science and systems biology.

Jean-François Boulicaut is currently a professor of computer science at INSA Lyon, and was the director of its Computer Science and Information Technology department between 2015 and 2018. He has been the founding leader of a Data Mining research group at INSA Lyon from 1998 until 2012. He is now a senior member of the DM2L group (Data Mining and Machine Learning) within the large Research Unit LIRIS UMR CNRS 5205. His own expertise concerns the design of constraint-based mining algorithms (e.g., supporting pattern discovery like rules or sequential patterns, subgroup discovery, clustering and co-clustering). He has been a member of the Data Mining and Knowledge Discovery journal editorial board from 2006 until 2016 and has served in the program committees of every major data mining conference.