Data Min Knowl Disc (2011) 23:63–90
DOI 10.1007/s10618-010-0199-1

Improving constrained pattern mining with first-fail-based heuristics

Christian Desrosiers · Philippe Galinier · Alain Hertz · Pierre Hansen

Received: 9 October 2009 / Accepted: 9 August 2010 / Published online: 21 August 2010
© The Author(s) 2010

Abstract In this paper, we present a general framework to mine patterns with antimonotone constraints. This framework uses a technique that structures the pattern space in a way that facilitates the integration of constraints within the mining process. Furthermore, we also introduce a powerful strategy that uses background information on the data to speed-up the mining process. We illustrate our approach on a popular structured data mining problem, the frequent subgraph mining problem, and show, through experiments on synthetic and real-life data, that this general approach has advantages over state-of-the-art pattern mining algorithms.

Keywords Constraint-based pattern mining · Frequent subgraph mining · First-fail heuristic

Responsible editor: M.J. Zaki.

C. Desrosiers (✉)
Ecole de Technologie Supérieure, 1100, Notre-Dame O., Montreal, QC H3C 1K3, Canada
e-mail: christian.desrosiers@etsmtl.ca

P. Galinier · A. Hertz
Ecole Polytechnique de Montréal,, C.P. 6079 succ. Centre-ville, Montreal, QC H3C 3A7, Canada
e-mail: philippe.galinier@polymtl.ca

A. Hertz
e-mail: alain.hertz@polymtl.ca

P. Hansen
HEC Montréal, 3000, Cote-Sainte-Catherine, Montreal, QC H3T 2A7, Canada
e-mail: pierre.hansen@gerad.ca

1 Introduction

Mining patterns like itemsets, sequences, graphs or first-order logic relations is a complex task that is essential to many important fields such as marketing, bioinformatics, drug design and Web-related technologies. While many algorithms have been proposed over the years to find patterns occurring frequently in the database, only a few of them are able to handle other types of constraints that further specify which patterns are truly interesting. Yet, recent works such as (Zhu et al. 2007) have shown the necessity of allowing various types of constraints and the benefits, in terms of computational efficiency, of integrating these constraints within the mining process.

In this paper, we present a novel framework to mine constrained patterns that generalizes current solutions for this problem. Like other pattern-growth methods proposed for this task, our framework allows one to explore the search space efficiently and without visiting redundant patterns. However, while such methods usually rely on a pattern- and exploration-specific code, our approach uses a more general technique that dynamically maps the search space into a tree. This approach has several advantages. First, it can be described using a few simple and general rules which can be applied to a wide variety of complex patterns and any type of complex (weak) antimonotone constraints. Moreover, because it transforms the search space into a tree, several known exploration strategies, such as depth-, breadth- and best-first search can be used without having to change the underlying framework. Also, since the exploration is pattern independent, efficient strategies to prune uninteresting patterns can be used. Thus, one of the main aspects of our framework is its ability to integrate constraints directly inside the pattern exploration process by mapping the search space such that every pattern satisfying a (weak) antimonotone constraint can be reached without having to visit patterns that do not satisfy this constraint. This strategy, which allows one to discover the interesting patterns efficiently by avoiding uninteresting ones, is taken one step further through the use of heuristics. Based on the first-fail principle (Bessière and Régin 1996), our framework uses such heuristics to push at the lowest levels of the search tree patterns unlikely to satisfy the constraint. As shown in our experiments, this technique can have a significant impact of the efficiency of the pattern discovery.

The rest of this paper is structured as follows. In Sect. 2, a review of the relevant literature is made and the contributions of this work presented. Then, in Sect. 3, a formal definition of the constrained pattern mining problem is given and the basic principles of our approach are presented. Sect. 4 then illustrates this approach on the task of finding subgraph patterns satisfying constraints on minimum edge density and cycle size. The technique using heuristics to speed-up the mining process is then detailed in Sect. 5, and the benefits of this approach over prefix-coding described in Sect. 6. Moreover, Sect. 7 presents experimental results on the impact of integrating constraints and using heuristics in our framework. Finally, a short summary of the contributions and research perspectives of this work is given in Sect. 8.

2 Related work

Most pattern discovery algorithms in the literature focus on finding the frequent patterns of a transactional database. A pattern X is frequent in this database if the number of transactions containing X, known as the *support* of X, is no less than a given threshold. This concept has also been defined in the context of a single transaction, for instance in (Kuramochi and Karypis 2005), where the support of X corresponds to the number of distinct embeddings of X in this transaction. The methods proposed for this problem, essential to several applications like drug design (Borgelt et al. 2005; Sternberg et al. 1995), classification and predication of molecular structures (Zaki et al. 2004), indexing structures (Yan et al. 2004), and mining XML documents and queries (Chen et al. 2004; Termier et al. 2002), can essentially be divided in two categories: *levelwise* and *pattern-growth* methods.

2.1 Levelwise pattern mining

Levelwise pattern mining approaches, such as APRIORI (Agrawal et al. 1993) for itemsets, GSP (Srikant and Agrawal 1996) for sequences, MOLFEA (Kramer et al. 2001) for paths, TREEMINER (Zaki 2002) for trees, AGM (Inokuchi et al. 2000) and FSG (Kuramochi and Karypis 2001) for graphs, as well as WARMR (Dehaspe and Toivonen 1999) and FARMER (Nijssen and Kok 2001) for first-order logic rules, explore the search space level-by-level. This is generally done in a two-step iterative process:

(1) Pairs of frequent patterns of the same level of generality are combined to produce candidates patterns of the next level of generality. In short, a pattern X is more general than another one Y if X is a subpattern of Y.
(2) The support of these candidate patterns is computed in the database and the patterns whose support is inferior to the threshold are pruned.

One of the main advantages of this approach is that it allows one to prune many patterns with a technique that checks the *downward closure* of a *strong-antimonotone* constraint, such as a the one imposing a pattern to be frequent. Briefly, a constraint is strong-antimonotone if all subpatterns of a pattern X satisfying this constraint also satisfy it. Hence, X can be pruned if any of its subpatterns does not satisfy this constraint, a principle known as *Apriori*. Because levelwise methods always explore a pattern after its subpatterns, this principle can be applied to prune many uninteresting patterns without having to evaluate the constraint on these patterns. However, levelwise algorithms tend to suffer from two problems: the candidate generation process can produce an exponential number of redundant patterns, and a large amount of memory may be required to store the frequent subpatterns of the current exploration level (Yan and Han 2002).

2.2 Pattern-growth mining

On the other hand, pattern-growth approaches, like FP-growth (Han et al. 2000) for mining itemsets, PREFIXSPAN (Pei et al. 2001) for sequences, FREQT (Asai et al. 2002) and CHOPPER (Wang et al. 2004) for trees, as well as GSPAN (Yan and Han

2002), FFSM (Huan et al. 2003) and GASTON (Nijssen and Kok 2004) for mining subgraphs, explore the pattern space depth-first. This is done by extending a pattern as much as possible before exploring another pattern on the same level of generality. One of the main advantages of this approach is that infrequent patterns can be implicitly pruned by extending only frequent ones, since all extensions of an infrequent pattern are also infrequent. Also, as only the current branch of the search space needs to be stored, pattern-growth approaches usually require less memory than levelwise ones (Yan and Han 2002).

In some pattern-growth methods like FP-growth (Han et al. 2000), the database can be compressed in a compact representation called *frequent-pattern* (FP) tree. Because this structure contains all the necessary information on the frequent patterns of the database, one can mine it directly instead of the database. A considerable benefit of this is that the reduced information may often be stored entirely in primary memory, resulting in faster access. Also, the possible extensions of a pattern can normally be deduced from the FP-tree, avoiding the generation of candidates which never occur in the database. However, implementing such a structure may be difficult for patterns like subgraphs, where determining isomorphism is a complex task. A solution proposed for this problem, DFS-coding (Yan and Han 2002), is described in Sect. 6.

2.3 Constraint-based pattern mining

While the above algorithms essentially focus on finding frequent patterns, methods allowing one to define other pattern constraints have also been proposed. An algorithm to mine frequent itemsets with monotonicity, antimonotonicity and succinctness constraints, based on the levelwise approach, is presented in (Ng et al. 1998). A constraint on itemsets is *succinct* if the itemsets satisfying it are subsets of the set of individual items satisfying the constraint. A useful consequence of this is that the satisfaction of such constraints can be evaluated implicitly. A related levelwise algorithm, which mines itemsets satisfying a *block* constraint, is CBMINER (Gade et al. 2004). In itemset mining, block constraints generalize constraints on frequency by imposing that the product of the size of an itemset with its support be greater than a minimum threshold. Several algorithms have also be proposed to mine constrained sequences. For example, a levelwise algorithm for mining frequent sequences, in which the time gap between sets of transactions containing consecutive elements of a sequence is restricted, is GSP (Srikant and Agrawal 1996). Two other levelwise algorithms, CSPADE (Zaki 2000) and SPIRIT (Garofalakis et al. 2002), extend this work by allowing one to add other types of constraints, such as width limitations and regular-expressions, on the mined sequences.

Constraint-based methods have also been proposed for more mining more complex patterns like subgraphs. Among the first is the levelwise algorithm MOLFEA (Kramer et al. 2001), which can handle monotone and antimonotone constraints on subgraph patterns. Pattern-growth algorithms for mining subgraph patterns satisfying strong-antimonotone constraints, such as CABGIN (Wang et al. 2005) and GPRUNE (Zhu et al. 2007), are also described in the literature. Other than strong-antimonotone constraints, GPRUNE can also handle *weak-antimonotone* constraints as well as constraints with a property called *pattern-(in)separable antimonot-*

onicity. Weak antimonotonicity, described more formally in Sect. 3.1, requires a least one subpattern of a pattern satisfying the constraint to satisfy it. On the other hand, a constraint is pattern-separable antimonotone when any embedding of a pattern in a transaction does not satisfy the constraint if the transaction itself does not satisfy it. Consequently, any transaction that does not satisfy this constraint can be discarded altogether. However, if the constraint is pattern-*inseparable*, whether a transaction can be dropped depends on the particular pattern embedding. In GPRUNE, such constraints can be handled at the high cost of maintaining the pattern growth tree tracking the pattern embeddings. Finally, a levelwise constraint-based algorithm to find first-order logic sequences is presented in (Lee and De Raedt 2004).

2.4 Our contributions

This work makes two contributions to the field of pattern mining:

- It introduces a general and efficient approach to mine constrained patterns. Since it is based on pattern-growth, like CABGIN and GPRUNE, this approach can explore the search space efficiently, avoiding the generation of redundant candidates and requiring less memory than levelwise methods. Moreover, as GPRUNE, this approach can be used to mine patterns satisfying strong or weak antimonotone constraints, as well as constraints with the property of pattern-(in)separable antimonotonicity. However, unlike this method, our approach does not depend on pattern-specific codes which can be hard to define for complex patterns and constraints. Instead, it is based on general rules that can be used with any type of ordered patterns. Moreover, it provides a simple way to add complex constraints, that requires little effort and allows one to prune uninteresting patterns.
- It presents a novel technique, based on the first-fail principle (Bessière and Régin 1996), which uses heuristics to avoid the exploration of interesting patterns. While this principle is already used by several frequent pattern mining algorithms to explore the least frequent extensions first, our approach is more general and can handle more complex heuristics. Thus, as discussed in Sect. 6, defining a code which take into account complex heuristics may be a difficult task. However, as presented in Sect. 5, such heuristics can be easily defined in our approach.

3 Overview of the framework

This section gives a formal definition of constrained-pattern mining and presents the main principles of our proposed method for this problem.

3.1 Formal problem definition

Let $\mathcal{P}$ be the set containing all the patterns of a given type, e.g. itemsets, sequences or graphs, called the *pattern space*. To mine patterns in $\mathcal{P}$, it is necessary to have a relation which partially orders these patterns according to their generality. Thus, we say that a pattern $X \in \mathcal{P}$ is more general than a pattern $Y \in \mathcal{P}$, and write $X \sqsubset Y$,

if X is a subpattern of Y. For example, the most natural relation of generality used for itemsets is the subset relation: an itemset X is more general than another itemset Y iff $X \subset Y$. In the context of mining subgraph patterns, two common relations of generality are the edge-induced and node-induced subgraph relations, where a graph H is more general than a graph G if it can be obtained by removing from G any of its edges or, respectively, nodes (and their incident edges). Given a relation of generality, we denote $\mathcal{S}(X)$ the set containing the maximal subpatterns of X, i.e.

$$\mathcal{S}(X) = \{Y \in \mathcal{P} \mid Y \sqsubset X \text{ and } \nexists Z \in \mathcal{P} \text{ s.t. } Y \sqsubset Z \sqsubset X\}.$$

Using this concept, it is possible to arrange the patterns of $\mathcal{S}$ in a lattice-like structure, where the root pattern is $\perp$, the *parents* of a pattern X are the patterns in $\mathcal{S}(X)$, and the *children* of X are the patterns Y such that $X \in \mathcal{S}(Y)$.

A general definition of the constrained pattern mining problem, originally described in (Mannila and Toivonen 1997), is the following. Given a predicate $\mathcal{C} : \mathcal{P} \rightarrow \{true, false\}$, called the *pattern constraint*, find the set of patterns $X \in \mathcal{P}$ such that $\mathcal{C}(X) = true$. In this paper, we are interested in constraints that have the antimonotonicity property. Following the terminology used in (Zhu et al. 2007), we say that a constraint $\mathcal{C}$ is *strong-antimonotone* if:

$$\forall X, X' \in \mathcal{P} \text{ such that } X' \sqsubseteq X, \text{ if } \mathcal{C}(X) = true \text{ then } \mathcal{C}(X') = true.$$

Although this property allows one to develop efficient pruning strategies, only a limited number of constraints have this property. A weaker version of this property, called *weak-antimonotone*, is such that:

$$\forall X \in \mathcal{P}, \text{ if } \mathcal{C}(X) = true \text{ then } \exists X' \in \mathcal{S}(X) \text{ such that } \mathcal{C}(X') = true.$$

Although the method proposed in this paper is not specific to a particular data type, we will mainly illustrate its use on the task of finding patterns in *labeled graphs*. Formally, a labeled graph G can be represented as a tuple (V, E, L, l_V, l_E), where V is a set of nodes, E a set of edges, L is a set of labels, and l_V (resp. l_E) is a function assigning to each node (resp. edge) a label of L. While mining labeled graphs has several important applications, such as finding the frequent subgraphs of a database (Huan et al. 2003; Kuramochi and Karypis 2001; Nijssen and Kok 2004; Yan and Han 2002), it is also more general and challenging than mining simpler data types:

- Mining subgraph patterns can be seen as a generalization of mining simpler patterns, such as itemsets, sequences, or trees.
  For example, subgraph mining could be used to find sequences by imposing the subgraph patterns to be paths.
- Mining subgraph patterns is more challenging than mining simpler patterns. While testing the redundancy of a subgraph pattern amounts to solving the non-trivial *graph isomorphism* problem (Fortin 1996), testing whether a subgraph pattern is embedded in a transaction graph, known as the *subgraph isomorphism* problem (Garey and Johnson 1979), is NP-hard in the general case.

Note, however, that our method could also be used for patterns more complex than subgraphs, for instance, hyper-subgraphs or even rules of first-order logic.

3.2 Exploring the pattern space

In this section, we present the pattern exploration approach at the heart of our framework. Recall from the previous section that the patterns of $\mathcal{P}$ can be arranged in a lattice-like structure, where the parents of any pattern X are given by $\mathcal{S}(X)$. Since patterns may have more than one parent, traversing this structure may visit the same pattern more than once. To avoid this, we use an efficient technique, similar to the one used in vSiGraM (Kuramochi and Karypis 2005) to find the subgraphs occurring frequently in a large graph. The main idea of this technique is to transform the lattice search space into a rooted tree $\mathcal{T}_p$ using a function p that assigns to each pattern X a unique parent $p(X) \in \mathcal{S}(X)$ called *canonical parent*. Since, for every pattern X, there is a single path in $\mathcal{T}_p$ connecting X to the root, exploring this tree breadth-first or depth-first will visit every pattern exactly once. However, two isomorphic patterns may still be explored. In short, two patterns X and Y are isomorphic, written $X \simeq Y$ if they are equivalent under a structure preserving transformation. To avoid visiting such redundant patterns, it is necessary to add a condition on p imposing the canonical parents of two isomorphic patterns to also be isomorphic:

$$\forall X, Y \in \mathcal{P}, \text{ if } X \simeq Y \text{ then } p(X) \simeq p(Y). \tag{1}$$

Furthermore, since two extensions of a pattern may also be isomorphic, to avoid redundancy, another condition should also be imposed:

$$\forall X \in \mathcal{P}, \text{ no two isomorphic extensions of } X \text{ are explored.} \tag{2}$$

Thus, if we partition the extensions of X into groups of isomorphic patterns, a single pattern from each of these groups should be explored. Which pattern from each group is explored does not matter.

Unlike vSiGraM, which is limited to subgraph patterns, these general rules can be used to mine many types of patterns, from itemsets to complex patterns like hyper-subgraphs. Moreover, another significant difference between this approach and vSiGraM is the possibility to restrict the exploration to patterns satisfying a given (weak) anti-monotone constraint $\mathcal{C}$. To integrate such constraint directly within the pattern exploration, we use the following observation. For a pattern X to be explored, its canonical parent $p(X)$ should also be explored. However, if $\mathcal{C}$ is weak-antimonotone then $p(X)$ may not satisfy this constraint and, consequently, may not be explored. Therefore, it is necessary to impose another condition on p:

$$\forall X \in \mathcal{P}, \text{ if } X \text{ satisfies } \mathcal{C} \text{ then } p(X) \text{ also satisfies } \mathcal{C}. \tag{3}$$

Conditions (1), (2) and (3) can be combined into an algorithm for mining constrained patterns depth-first or breadth-first. Figure 1 presents the main steps of the

```
Depth-first pattern mining algorithm
  Input: An antimonotone constraint C.
  Output: The set F of patterns satisfying C.
  F := explore(C, ⊥) ;
  return F ;

Procedure explore(C, X)
  Input: An antimonotone constraint C, and a pattern X.
  Output: The set F of patterns satisfying C, in the subspace rooted at X.
  F := {X} ;
  foreach group Y of isomorphic extensions of X do
    Select any pattern Y ∈ Y ;
    if p(Y) ≃ X and C(Y) = true then  F := F ∪ explore(C, Y) ;
  return F ;
```

Fig. 1 Depth-first pattern mining algorithm

depth-first mining algorithm. Starting with the root pattern $\perp$, this algorithm launches the depth-first traversal by calling the recursive procedure *explore*, which takes as input an antimonotone constraint $\mathcal{C}$ and a pattern X, and returns a set $\mathcal{F}$ containing the patterns of the subspace rooted at X that satisfy $\mathcal{C}$. This procedure first adds X to $\mathcal{F}$ and then computes the groups $\mathcal{Y}$ of isomorphic extensions of X. Then, for each of these groups, a pattern Y is selected and tested for two things:

1. The canonical parent of Y is isomorphic to X.
2. Y satisfies $\mathcal{C}$.

The least expensive of these two tests should normally be carried out first. If both conditions are met, then Y is explored. Finally, once all non-isomorphic extensions have been explored, the procedure returns $\mathcal{F}$.

The completeness and soundness of this algorithm are validated in the following proposition.

Proposition 1 *If $\mathcal{C}$ is an antimonotone constraint and conditions (1), (2) and (3) are respected, the depth-first or breadth-first traversal of the search tree $\mathcal{T}_p$ guarantees that:*

(A) Only patterns satisfying $\mathcal{C}$ are explored.

(B) For each group of isomorphic patterns satisfying $\mathcal{C}$, a single *pattern of this group is explored* once.

Proof (A) follows from the fact that patterns not satisfying $\mathcal{C}$ are pruned in the algorithm. We prove (B) by induction. Let $\mathcal{Y}$ be any group of isomorphic patterns satisfying $\mathcal{C}$. From conditions (1) and (3), we know that the canonical parents of every pattern in $\mathcal{Y}$ also satisfy $\mathcal{C}$ and that these canonical parents are all isomorphic. Denote $\mathcal{X}$ the isomorphic group containing these canonical parents and suppose $\mathcal{X}$ satisfies (B). There is a single pattern $X \in \mathcal{X}$ satisfying $\mathcal{C}$ that is explored once. Following condition (2), since all possible non isomorphic extensions of X are explored, a single pattern $Y \in \mathcal{Y}$ will be explored from X. Furthermore, because Y is explored only if it is extended from its canonical parent, i.e. if $p(Y) \simeq X$, this pattern will be explored only once.

We have thus established that a group of isomorphic patterns $\mathcal{Y}$ satisfies (B) if the group $\mathcal{X}$ containing their canonical parents satisfies (B). Since the root pattern is an ancestor of every pattern and satisfies (B), by induction, every group of isomorphic patterns satisfy (B). □

3.3 Computing the canonical parent

3.3.1 The general approach

The main difficulty in defining the canonical parent function p is to ensure that condition (1) is met, i.e. that isomorphic patterns have isomorphic canonical parents. In simple patterns like itemsets, this can be done easily. For instance, if each item has a unique identifier, $p(X)$ can be defined as the set obtained by removing the item with the lowest identifier. For complex patterns, like subgraphs, this can be more complicated. A general solution to this problem is to define a total order on the space of patterns $\mathcal{P}$, and then select the canonical parent of a pattern X as the subpattern $Y \in \mathcal{S}(X)$ satisfying $\mathcal{C}$ which is minimal with respect to this order. This can be done by using the concept of *canonical code*. Let $code$ be a function that uniquely maps every pattern X to a string $code(X)$. The canonical code of X, written $canon(X)$, is the lexicographically minimal string in all the pattern isomorphic to X:

$$canon(X) = \min_{Y \simeq X} code(Y).$$

Of course, two isomorphic patterns will have the same canonical code. The canonical parent of X can then be defined as the parent of X satisfying $\mathcal{C}$, whose canonical code is the smallest. The exploration of X guarantees such parent exists.

To illustrate this idea, consider the case of subgraph patterns. Let $G = (V, E, L, l_V, l_E)$ be a labeled graph with n nodes. Assuming that $l_E(u, v)$ is 0 if u and v are not connected, G can be encoded as

$$\begin{aligned} code(G) = (&l_V(v_1),\ l_V(v_2),\ \ldots,\ l_V(v_n),\ l_E(v_1, v_2),\ l_E(v_1, v_3),\ \ldots, \\ &l_E(v_1, v_n),\ l_E(v_2, v_3),\ \ldots,\ l_E(v_{n-1}, v_n))\,. \end{aligned}$$

Note that if G is unlabeled, we can encode it using its adjacency matrix A

$$code(G) = \left(A_{1,2}, A_{1,3}, \ldots, A_{1,n}, A_{2,3}, \ldots, A_{n-1,n}\right).$$

Moreover, let Γ be the set of all possible numberings of the nodes in V, called *permutations*, and denote $\varphi(v)$ the position assigned to a node $v \in V$ in a permutation $\varphi \in \Gamma$.

The *canonical code* of G can then be taken as the minimal code obtained by any permutation:

$$canon(G) = \min_{\varphi \in \Gamma} code(G^{\varphi}),$$

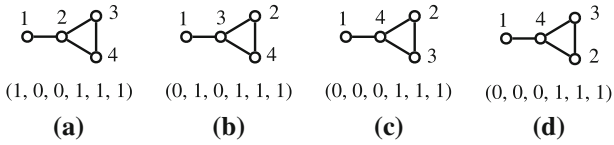

Fig. 2 Four permutations of the vertices of an unlabeled graph and their corresponding code. Permutations (*c*) and (*d*) give the canonical code

where G^{φ} is the graph isomorphic to G permuted by φ. We call *canonical permutations* the permutations leading to the canonical code. For example, the canonical code of the graph showed in Fig. 2, obtained by canonical permutations (*c*) and (*d*) is $(0, 0, 0, 1, 1, 1)$.

While testing all $n!$ permutations of Γ is very inefficient, it is possible to considerably reduce the number of tested permutations using node invariants. The idea is to compute the value of a powerful yet inexpensive invariant for all the nodes of V, and then only consider the node permutations in which the value of the invariant is non-decreasing for consecutive nodes. For instance, if we impose the nodes in the permutation to be of increasing degree, then only permutations *(c)* and *(d)* of Fig. 2 would be considered. This strategy, used in algorithms like NAUTY (McKay 1981) and FSG (Kuramochi and Karypis 2001), allows one to efficiently compute the canonical code of large graphs.

3.3.2 Improving the computation

In order to find the canonical parent of a pattern X, using the above-mentioned approach, one needs to compute the canonical code of every parent of X and then select the parent for which this code is minimum. This can be quite inefficient, especially if computing the canonical code of a pattern is expensive. As we now illustrate with subgraph patterns, the same result can be achieved in a much more efficient way. Before presenting this approach, we need to introduce the concept of *node* and *edge isomorphism*.

Let Γ^* denote the set of canonical permutations. Except for artificial graphs, specifically constructed to have a large amount of symmetry, Γ^* usually contains only a few canonical permutations and can be obtained efficiently. Two nodes u, v, are said *isomorphic* (i.e., topologically equivalent) if there are two canonical permutations $\varphi_1, \varphi_2 \in \Gamma^*$ such that $\varphi_1(u) = \varphi_2(v)$. For example, the two rightmost nodes of the graph in Fig. 2 are isomorphic since inverting the position of these nodes in a canonical permutation (e.g., permutation (*c*)) gives another canonical permutation (e.g., permutation (*d*)). Likewise, two edges (u, v) and (x, y) are *isomorphic* if there are two canonical permutations φ_1, φ_2 such that $\varphi_1(u) = \varphi_2(x)$ *and* $\varphi_1(v) = \varphi_2(y)$. For instance, edges (2, 4) and (3, 4) in Fig. 2d are isomorphic since inverting the order of nodes 2 and 3 in the permutation, while keeping node 4 in its position (i.e., mapping node 4 to itself), leads to another canonical permutation. Intuitively, if two nodes (or edges) are isomorphic, then removing one from the graph produces the same subgraph as would removing the other one.

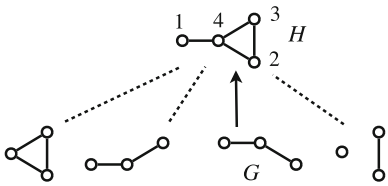

Fig. 3 A graph H explored from a graph G and its node-induced subgraphs. For the canonical permutation showed in this figure, the subgraphs are tested from left to right

3.3.3 Node-induced subgraphs

Consider, first, the case where the parents of a graph G are the subgraphs obtained by removing a single *node*, and let H be a graph extending G with a node w and a set of edges incident to w. Moreover, let φ be any canonical permutation of H. We define the canonical parent of H obtained with φ, i.e. $p_\varphi(H)$, as the graph produced by removing the first node v^*, following the order of φ, such that the node-induced subgraph satisfies $\mathcal{C}$. Since removing isomorphic nodes produces topologically equivalent subgraphs, we can show that the canonical parents obtained with two canonical permutations are isomorphic:

$$p_\varphi(H) \simeq p_{\varphi'}(H), \quad \text{if } \varphi \text{ and } \varphi' \text{ are canonical permutations of } H.$$

Since H is explored only if $p_\varphi(H) \simeq G$, we can explore this graph if and only if v^* is isomorphic to w. To illustrate these concepts, consider the graph H in Fig. 3, explored from a graph G. Suppose the exploration is constrained to connected cycle-free graphs. The node-induced subgraphs of H are considered by removing its nodes, following their order in the canonical permutation showed in the figure (i.e., left to right in this example). Since the second tested subgraph is the first that is connected and cycle-free, it is the canonical parent of H. However, since nodes 2 and 3 of H are isomorphic, we know that $G \simeq p_\varphi(H)$, and therefore H will be explored. Note that this result was obtained without having to compute the canonical code of any subgraph of H, and that we would have reached the same conclusion if we had used another canonical permutation of H (e.g., permutation (c) in Fig. 2).

3.3.4 Edge-induced subgraphs

A similar approach can be used to find the canonical edge-induced parent of a graph H extending G with an edge $e = (u, v)$. Once again, let φ be a canonical permutation of H. We can define a total order on the edges of H such that $e_1 = (u_1, v_1)$ precedes $e_2 = (u_2, v_2)$ if and only if:

- $\varphi(u_1) < \varphi(u_2)$ *OR*;
- $u_1 = u_2$ and $\varphi(v_1) < \varphi(v_2)$.

We define $p_\varphi(H)$ as the graph produced by removing the first edge e^*, following the above-mentioned order, such that the edge-induced subgraph satisfies $\mathcal{C}$. Similar to node-induced subgraphs, we can show that $p_\varphi(H) \simeq G$ if and only if e^* is isomorphic to the extension edge e. As an illustration, consider the graph H in Fig. 4, extended

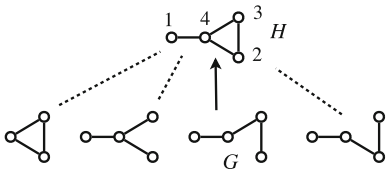

Fig. 4 A graph H explored from a graph G and its edge-induced subgraphs. For the canonical permutation showed in this figure, the subgraphs are tested from left to right

from graph G with edge $(2, 4)$. Removing the edges of H following the order described above, we get the subgraphs shown from left to right in the figure. If, once again, we limit the exploration to connected cycle-free graphs, then the canonical parent of H will be the one obtained by removing edge $(2, 3)$ (i.e, the second subgraph from the left). However, because edges $(2, 3)$ and $(2, 4)$ are *not* isomorphic, we conclude that $p_\varphi(H) \not\simeq G$ and, therefore, that H should *not* be explored.

4 Illustration of constrained pattern mining

In this section, we illustrate our approach on mining subgraph patterns satisfying constraints on minimum cycle size and edge density.

4.1 Minimum cycle constraint

Consider the task of mining the frequent *connected* subgraphs of the database satisfying the following constraint $\mathcal{C}$:

- Either the subgraph contains a cycle of at least n nodes, *OR*
- The subgraph is a path.

It is easy to show that this constraint is weak-antimonotone with respect to the edge-induced subgraph relation. Hence, if G is a path, then removing an edge at either end of the path, gives another path. Otherwise, if G is a proper cycle with at least n nodes, then removing any edge also produces a path. Finally, if G is not such a cycle but contains one, there is a non-disconnecting edge that does not belong to every such cycles of G, and the subgraph obtained by removing this edge will satisfy $\mathcal{C}$.

The simplest way to mine the frequent connected graphs satisfying $\mathcal{C}$, the *generate and test* approach, consists in first finding all frequent connected graphs and then filtering out those that do not satisfy this constraint. As we will see in the experimental section of this paper, such an approach can be highly inefficient, even for small values of n, since the number of graphs satisfying $\mathcal{C}$ can be much less than the number of frequent ones. A somewhat better approach is to find all frequent paths and all frequent subgraphs containing a cycle of $n, n{+}1, n{+}2, \ldots$ nodes, separately. The problem with this approach is that the coherence of support computations, which comes from depth-first exploration, is lost. Thus, even though there are no frequent graphs containing cycles of $k \geq n$ nodes, there might be some that contain more than k nodes and the database has to be mined for every possible cycle size.

On the other hand, mining the graphs satisfying this constraint, using our framework, is a simple task. Consider a graph G visited in the exploration. If G contains a

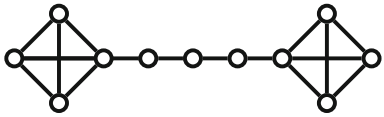

Fig. 5 A graph which does not have a *connected* node-induced subgraph of greater or equal density

cycle of at least n nodes, the canonical parent of G is obtained by removing the minimum non-disconnecting edge, following the order described in Sect. 3.3, that does not belong to this cycle. Otherwise, if G is a proper cycle, then the canonical parent is a path obtained by removing the minimum edge from the cycle. Finally, if G is a path then its canonical parent is obtained by removing the minimum edge at either end of the path. Finally, if none of these cases are true, G can be pruned without fear of missing an interesting subgraph pattern.

4.2 Minimum edge density constraint

Another example of mining patterns with constraints, described in (Zhu et al. 2007), is the task of finding the frequent connected graphs with an edge density greater than a given threshold δ. Let G be a graph with n nodes and m edges, the edge density of G is given by $D(G) = \frac{2m}{n(n-1)}$. Consider the pattern constraint $\mathcal{C}$ imposing that the density of a graph is at least δ, where $0 \leq \delta \leq 1$.

While this constraint is not antimonotone with respect to the *edge*-induced subgraph relation, it satisfies this property for *node*-induced subgraphs. Hence, suppose we remove a node v from G, then the resulting graph H will have a density of $D(H) = \frac{2(m-d_v)}{(n-1)(n-2)}$, where d_v is the number of edges incident to v, called *degree* of v. In order to have $D(H) \geq D(G)$, the degree of node v must therefore be such that $d_v \leq \frac{2m}{n}$. It is easy to verify that any graph G contains such a node. Indeed, suppose this is not true, then, since the minimum degree of a node of G is greater than $\frac{2m}{n}$, the number of edges of G is such that

$$m = \frac{1}{2}\sum_{v \in V} d_v > \frac{1}{2}\left(n \times \frac{2m}{n}\right) = m, \quad \text{a contradiction.}$$

The minimum density constraint, however, loses its weak-antimonotone property if we add to it the restriction that graphs should be connected. To illustrate this, consider the graph shown in Fig. 5. Since this graph has 11 nodes and 16 edges, its density is $\frac{2 \cdot 16}{11 \cdot 10} \approx 0.291$. Furthermore, to have a node-induced subgraph with a density no less than that, we should remove a node v with degree $d_v \leq \frac{2 \cdot 16}{11} \approx 2.91$. However, the only nodes that satisfy this, the three nodes in the middle of the graph, disconnect the graph when removed. In order to have a constraint with the weak-antimonotone property, we must strengthen it by also imposing the graphs satisfying the constraint to have at least one connected node-induced subgraph of greater or equal density. In practice, only a small proportion of graphs do not satisfy this last constraint.

Again, mining the connected subgraphs satisfying this constraint is an easy task with our framework. Let G be a graph of n nodes and m edges, explored in the mining

process. If $D(G) < \delta$ or if G has no non-disconnecting node v such that $d_v \leq \frac{2m}{n}$, then it is pruned. Otherwise, the canonical parent of G can be obtained by removing the minimum non-disconnecting node v, following the canonical node ordering described in Sect. 3.3, such that $d_v \leq \frac{2m}{n}$.

5 Speeding-up the process with heuristics

As mentioned before, testing the downward closure of a pattern X, i.e. testing that every parent of X satisfies a strong-antimonotone constraint $\mathcal{C}$, is a powerful strategy that can considerably speed-up the discovery of the interesting patterns. In this section, we present a similar strategy that avoids the exploration of many patterns that do not satisfy $\mathcal{C}$ by using heuristics.

5.1 Basic principle

An advantage of the approach presented in this paper is its flexibility. As stated by conditions (1) and (3), the only restrictions on function p are that the canonical parents of isomorphic patterns should be isomorphic and the canonical parent of any pattern X satisfying $\mathcal{C}$ should also satisfy $\mathcal{C}$. If $\mathcal{C}$ is strong-antimonotone, however, every parent of X necessarily satisfies $\mathcal{C}$ and only the first condition needs to be verified. In light of this freedom, one may wonder how to define p in order to mine the interesting patterns as efficiently as possible. The following proposition provides an answer to this question.

Proposition 2 *Let $\mathcal{C}$ be a strong-antimonotone constraint. If p is defined such that, for every pattern X with at least one parent not satisfying $\mathcal{C}$, $p(X)$ does not satisfy $\mathcal{C}$, the depth-first or breadth-first exploration of $\mathcal{T}_p$ will only visit patterns that are downward closed under $\mathcal{C}$.*

Proof The proof follows directly from the fact that X is explored only if $p(X)$ satisfies $\mathcal{C}$, which is verified only if X is downward closed under $\mathcal{C}$. □

There is still one problem: we do not know which of the parents of a pattern X satisfy $\mathcal{C}$. In many situations, however, we might have additional knowledge on which parents are more *likely* than others to satisfy this constraint. Suppose that this knowledge takes the form of a heuristic function h that evaluates the likeliness of a pattern to satisfy $\mathcal{C}$, i.e. if $h(Y) > h(Y')$ then Y is more likely to satisfy $\mathcal{C}$ than Y'. Following Proposition 2, the canonical parent of X should be selected as a parent of X minimizing h:

$$p(X) \in \operatorname*{argmin}_{Y \in \mathcal{S}(X)} h(Y).$$

Note that this strategy is similar to the well-known *first-fail principle* (Bessière and Régin 1996) in which a bad decision detected early in the process is less expensive than one detected later on.

Table 1 The most common chemical elements of the NCI datasets and the percentage of atoms in the datasets with these elements

Element	CAN2DA99	AID2DA99
C	72.8	72.5
O	14.0	13.7
N	9.3	9.6
S	1.4	1.8
Cl	1.1	1.0
F	0.7	0.7

While this strategy allows one to reduce the number of explored patterns that are not downward closed, it may also help to reduce the cost of evaluating computationally expensive constraints. Consider, for instance, the problem of finding the frequent patterns of a database, where computing the support of a pattern can be an expensive operation. In this context, h evaluates the likeliness of a pattern to be frequent. Since support is an antimonotone measure, the transactions that may contain a pattern X is a subset of those containing $p(X)$. By selecting $p(X)$ as a subgraph of X likely to have a low support, one can thus limit the number of database transactions to test for embeddings and, consequently, speed-up the discovery of the frequent patterns.

5.2 Defining the heuristic

As with the canonical parent function p, there is one restriction on the choice of the heuristic h: the heuristic should be invariant with respect to isomorphism, i.e. if two patterns X and Y are isomorphic then $h(X) = h(Y)$. In the context of subgraph mining, the heuristic should therefore be a graph invariant. Note, however, that the heuristic does not have to be static: as long as the previous condition is met, one can define a dynamic heuristic that changes according to the pattern being evaluated. For instance, if the mined patterns are subgraphs, a different heuristic could be used for paths or trees than for graphs with cycles.

A simple approach to define the heuristic consists in using some general knowledge on the type of data mined. For instance, consider the problem of mining the frequent subgraphs of a database containing molecular graphs. Such graphs typically share a common property: the distribution of node labels, corresponding to the chemical element of atoms, is skewed due to the fact that elements like *carbon*, *oxygen*, *nitrogen* and *hydrogen* are found in most organic compounds. This fact is outlined in Table 1, which gives the distribution of the chemical elements in the National Cancer Institute (NCI 1999) datasets used in our experiments. Note that the bonds to hydrogen atoms were removed in these datasets.

A more dynamic and data-specific way to proceed is to extract information from the database, in a pre-processing step, and use this information, in combination with general knowledge on the data, to define the heuristic. For example, a simple heuristic that uses the distribution of labels in the database is the following. Denote l_v the label of a node v and $cnt(l, l')$ the number of edges in the database that have incident

nodes with labels l and l'. This information can be computed with little cost while reading the database. The heuristic simply adds this occurrence count for all edges of a graph G:

$$h(G) = \sum_{(u,v)\in E} cnt(l_u, l_v).$$

To minimize this heuristic, the canonical parent of a graph G should thus be a subgraph of G obtained by removing an edge whose incident label count is the *highest*.

Information on the interesting patterns can also be gathered by sampling a subset of the database and mining this data subset. Statistics obtained on the patterns found can then be used to define the heuristic. For instance, in our experiments on molecular graphs, the following two properties were observed in the frequent subgraph patterns:

- Most nodes in the subgraph patterns have a *low degree*. For example, in our experiments on the NCI datasets, the average node degree of the frequent subgraphs found was less than 3, for all tested values of minimum support.
- Most subgraph patterns are *cycle-free*, i.e. are either paths or trees. Thus, we observed that, on average, 95% of the frequent subgraphs found in the NCI datasets are cycle-free.

The second property is exploited by the graph mining algorithm GASTON (Nijssen and Kok 2004) which mines frequent subgraph patterns in three successive steps. In the first step, the frequent paths of the database are found. These paths are then extended to find the frequent trees. Finally, these trees are themselves extended to find the frequent cyclic subgraph patterns. As we will see in the experimental section, we can achieve an efficiency comparable to GASTON with a simple heuristic based on three invariants: the *minimum*, *average* and *maximum* node degree of a graph G. These invariants have interesting properties. For example, if we select the canonical parent as a non-disconnected edge-induced subgraph with the highest average degree then a cyclic graph will always have for ancestor a proper cycle. Similarly, selecting a subgraph with highest maximum degree as canonical parent ensures that a proper tree is grown from a path *only* if all its subgraphs are paths. A possible way to combine these invariants into the heuristic h is the following:

$$h(G) = -\left(d_{\min} + d_{\text{avg}} + d_{\max}\right).$$

To illustrate this heuristic, consider the graph (a) of Fig. 6, which has three parent subgraphs. If we select the subgraph with highest maximum degree, (c) will be selected as the canonical subgraph. However, if the canonical parent is chosen using the above heuristic, subgraph (b) is instead selected.

As described in Sect. 7, this simple heuristic can improve the discovery of frequent subgraph patterns in a database of molecular graphs. For different types of data, however, such heuristic may not yield any improvement. Thus, finding a good heuristic is a *problem-specific* task which may require prior information on the type of data mined. In some cases, heuristics can be deduced directly from the pattern constraint, especially if this constraint imposes restrictions on the mined patterns. For instance,

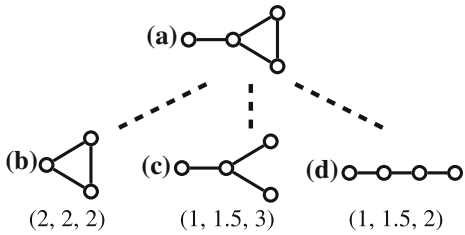

Fig. 6 A graph (*a*) and its three parent subgraphs (*b*–*d*). The minimum, average and maximum node degree of each subgraph is given as a triplet $(d_{\min}, d_{\text{avg}}, d_{\max})$

if this constraint can be expressed as $c(X) \geq \delta$, where $c : \mathcal{P} \rightarrow \mathbb{R}$ is invariant to isomorphism and antimonotone with respect to the order of generality, the heuristic can simply be $h(X) = c(X)$. However, if evaluating c is costly, e.g. c measures the support of a pattern in the database, then this approach may provide no advantage and defining h may need a deeper analysis of the data. As a general rule, the cost of evaluating this heuristic over the parents of every explored pattern should not exceed the benefit of exploring fewer uninteresting patterns. For example, evaluating a simple graph invariant on every parent of a subgraph pattern G is usually negligible compared to the cost of computing the support of G in the database.

6 Relation to prefix-coding

A very popular pattern mining approach, used by many pattern-growth algorithms, is prefix-coding. In this approach, a pattern X is represented by a string, the prefix-code of X, where each element contains the information on a single extension of X. In the exploration, a pattern X is explored only if its prefix-code is lexicographically minimum. A well-known example of this approach, introduced in the frequent subgraph mining algorithm gSpan (Yan and Han 2002), is DFS-coding. In DFS-coding, a labeled graph G is coded into a string whose elements correspond to the edges of G visited in a pre-order traversal. For each visited edge, information like the rank of the incident nodes and their labels is appended to the string. The canonical code of a graph is then the minimum string, following a special lexicographical order detailed in (Yan and Han 2002), for any pre-order traversal of G. One of the main advantages of this approach is that it limits the possible extensions of a canonical graph G. Thus, since G may only be canonical if the order in which the edges of G are explored corresponds to a pre-order traversal, several extensions of G that do not respect this condition can be pruned. However, finding the canonical code is a hard problem which may require to test a large number of traversals, especially if the graph is dense or unlabeled (Desrosiers et al. 2007).

Prefix-coding can be interpreted in the context of our framework as follows. Given a pattern X, the canonical parent of X is the pattern obtained by removing the last element of the canonical code of X. It is easy to show that condition (1) is satisfied: two isomorphic patterns have the same canonical code and, therefore, their canonical parent is the *same* pattern. Note that condition (1) only requires the canonical parents to be isomorphic. While prefix-coding can be seen as a special case of our framework, having a more general pattern mining approach has several advantages:

- Developing a mining algorithm for complex patterns like hyper-subgraphs only requires a canonical parent function which satisfies the condition on isomorphism. On the other hand, developing a specialized prefix-code for mining complex types of patterns may not be such an easy task.
- Since a pattern may have some parents that do not satisfy the pattern constraint, methods based on a fixed coding scheme may require to explore patterns that do not satisfy this constraint. Also, defining a specific code for constraints, such as the ones on density and cycles described in Sect. 4, may be difficult. Because it does not depend on a pattern specific code, this framework allows one to dynamically change the type of constraints without having to the redesign the mining algorithm.
- While the first-fail principle is already used in many prefix-coding algorithms, e.g. frequent itemset miners apply this principle by extending the current itemset with the item found in the least number of transactions, it is rather limited. Hence, since the prefix-code is determined by the individual patterns extensions, such strategy is limited to the choice of the next extension, and can not consider the whole pattern. For example, defining a DFS-code that explores first subgraph patterns with high *algebraic connectivity*[1] would be difficult, since this measure is computed over the entire subgraph.

Note that the approach presented in this paper, like DFS-coding, can also be used to restrict the possible extensions. To illustrate this, suppose the canonical parent of a subgraph G is chosen as a subgraph of G obtained by removing an edge incident to a node of maximum degree. In this case, any extension of a graph by an edge which will not be incident to a node of maximum degree can be discarded. Moreover, using more powerful invariants, one can further restrict the extensions to consider. Of course, the cost of evaluating the invariant should not outweigh the benefit of pre-pruning extensions.

7 Experimentation

In this section, we evaluate experimentally the two main aspects of our framework, i.e. mining patterns under constraints and using heuristics to speed-up the mining process, on the task of finding the frequent subgraph patterns of a database.

The method used in these experiments was implemented in the following way. To compute the canonical permutations of a graph, we used the publicly available implementation of algorithm NAUTY (McKay 1981). Briefly, this algorithm recursively refines a partition of nodes into cells based on their number of incident edges to nodes in other cells of the partition. Once a stable partition is obtained, to find the canonical permutations, the algorithm only tests the permutation of nodes within each cell. Since the partition obtained at the end of the refinement step is often trivial (i.e., each node in its own cell), very few permutations need to be tested in practice. Moreover, once we have the canonical permutations, the isomorphic nodes and edges can be easily found by looking at pairwise mappings of nodes in two permutations (see Sect. 3.3.2 for details). Node and edge isomorphisms are also used to avoid exploring

[1] The algebraic connectivity of a graph is the second-smallest eigenvalue of its Laplacian matrix.

Table 2 The values used for the parameters of the synthetic dataset generator

Param.	Description	Value
T	Avg. size of the database graph	25
F	Avg. nb. of frequent subgraphs	25
I	Avg. size of the frequent subgraphs	15

two isomorphic extensions (condition (2) of the exploration strategy described in Sect. 3.2). Furthermore, to explore the pattern space and integrate the pattern constraints and heuristic in this exploration, we used the depth-first algorithm described in Fig. 1. Also, to compute the support of a graph pattern in the database, we used a simple technique, where the list of database graphs containing the pattern is updated after each extension, and only the graphs of this list are tested for pattern embeddings. However, the approach presented in this paper could also be used with more powerful strategies, such as the memory-expensive technique of storing all embeddings. Finally, all tests were carried out on machines with Intel 2.0 GHz Pentium processors, 512Kb of cache and 1 Gb of RAM.

7.1 Impact of integrating constraints

The first experiment tests our framework on the problem of mining the frequent connected subgraphs of a database satisfying the two constraints described in Sect. 4:

(1) The subgraphs should either contain a cycle with at least n nodes or be paths.
(2) The subgraphs' edge density should be at least δ, where $0 \leq \delta \leq 1$.

For this experiment we have generated 10 different synthetic datasets, each containing 1,000 unlabeled graphs, using the random generator developed by Kuramochi and Karypis (Kuramochi and Karypis 2001). The parameters used for generating these datasets and their values are shown in Table 2. These values were chosen in accordance with similar experiments described in the literature.

To evaluate the impact of integrating the constraints in the mining process, we tested several approaches. In this experiment, we call SYGMA (Cn), where $n \in \{0, 3, 5, 7\}$, the method presented in Sect. 4.1 to mine the connected *edge induced* subgraphs that either contain a cycle of at least n nodes or are paths. Thus, SYGMA (C0) denotes the algorithm that finds *all* frequent connected edge induced subgraphs. Moreover, we call SYGMA (Dδ), where $\delta \in \{0.0, 0.2, 0.3, 0.4, 0.5\}$, the method presented in Sect. 4.2 to mine the connected *node induced* subgraphs with an edge density of at least δ. Again, SYGMA (D.0) is the algorithm which finds all frequent node induced subgraphs.

The results of this experiment are summarized in Fig. 7. This figure presents the average runtimes required to find the frequent subgraphs satisfying the minimum cycle (*top*) and the minimum edge density (*bottom*) constraints, for decreasing values of minimum support. To get a better understanding of these results, since the cost of testing whether these constraints are satisfied is negligible, we can view SYGMA (C0) and SYGMA (D.0) as *generate and test* methods and observe the speed-up obtained by pruning patterns *within* the exploration. As expected, we notice a huge advantage of

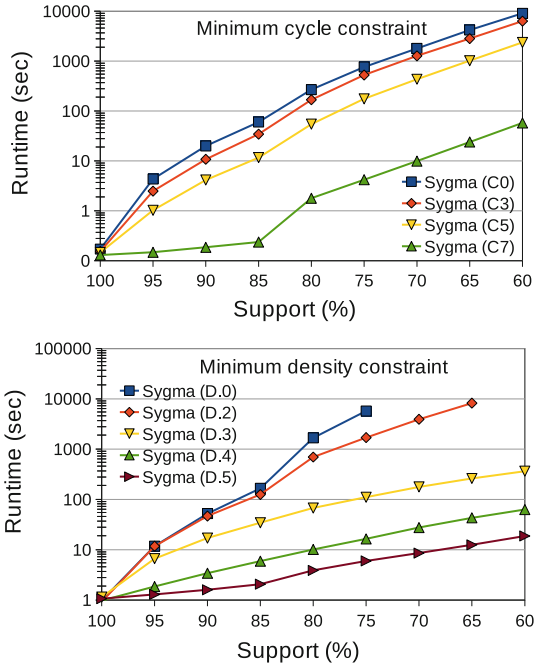

Fig. 7 Average runtimes to mine the frequent subgraphs of the 10 synthetic datasets for constraints on the minimum size of a cycle and the minimum edge density of the patterns, and using and decreasing minimum support thresholds

this approach over the naive generate and test approach. Thus, SYGMA (C3), (C5) and (C7) are, respectively, 1.5, 4.1 and 137.7 times faster on average than SYGMA (C0). Likewise, SYGMA (D.2), (D.3), (D.4) and (D.5) are, respectively, 1.7, 14.5, 93.9 and 251.1 faster, on average, than SYGMA (D.0).

7.2 Impact of using heuristics

This section evaluates the impact of using heuristics to speed-up the task of mining the frequent subgraphs of synthetic and real-life datasets.

7.2.1 Synthetic datasets

In this experiment, we considered the task of finding the frequent connected subgraphs of synthetic datasets whose labels have different distributions. To generate this data, we first produced six label probability distributions D_i, $i = 1, \ldots, 6$. Then, for each D_i, we created six classes C_j, $j = 1, \ldots, 6$, to which we randomly assigned a unique label from the set $\{1, 2, 3, 4, 5, 6\}$. Finally, we obtained the probability p_{ij} of a node in D_i having the label of C_j, using the following formula:

$$p_{ij} = \frac{i^{j-1}}{\sum_{k=1}^{6} i^{k-1}}.$$

Table 3 Probability (%) of having a node with the label of classes C_j, $j = 1, \ldots, 6$, for distributions D_i, $i = 1, \ldots, 6$

Class	Distribution					
	D_1	D_2	D_3	D_4	D_5	D_6
C_1	16.67	4.76	1.10	0.23	0.04	0.01
C_2	16.67	9.52	4.40	1.81	0.70	0.26
C_3	16.67	14.29	9.89	6.12	3.56	1.99
C_4	16.67	19.05	17.58	14.51	11.25	8.39
C_5	16.67	23.81	27.47	28.34	27.47	25.61
C_6	16.67	28.57	39.56	48.98	56.97	63.73

As shown in Table 3, the probability distributions obtained in this way are increasingly skewed. Thus, for the first distribution, all labels are equally probable. However, in the last distribution, the label of class C_6 has a 64% chance of being on a node, while the probability of having a node with the label of class C_1 is only 0.01%.

For each of these distributions, we then generated 10 datasets of 10, 000 unlabeled graphs, using the same random graph generator and parameter values of Table 2, and randomly labeled the nodes of these generated datasets using their respective distribution. We evaluated two approaches on these datasets. The first one, denoted by SYGMA (R), selects the canonical parent of a graph G randomly, by removing any non-disconnecting edge of G. To respect condition (1), however, the random number generator used to select the removed edge should behave the same way for isomorphic graphs. To achieve this, we compute a hash key using the canonical code of G, which is then used to initialize the random number generator. Then, following the edge ordering described in Sect. 3.3, we generate a random number for each of the non-disconnecting edges, and remove the edge which received the highest number.

The second approach, called SYGMA (H1), uses the simple heuristic presented in Sect. 5.2, where the number of database edges having each pair of labels is computed in a pre-processing step, and the canonical parent of a graph G is obtained by removing a non-disconnected edge whose label pair is most frequent in the database. To have a better appreciation of the results, we also compare these two approaches to three popular subgraph mining algorithms: FSG (*version* 1.0.1), GSPAN (2008 *version*) and GASTON (*version* 1.1). We tested two versions of GASTON, one which stores and maintains all embeddings of subgraph patterns in the database (EL), and another which does not use this memory-expensive strategy (RE).

Figures 8 and 9 gives the average runtimes required by the tested methods to find the frequent subgraphs of the datasets, for each label distribution. As expected, the efficiency of SYGMA (H1) increases as the label distribution becomes more skewed. Thus, while GSPAN is twice faster on average than SYGMA (H1) for the uniform distribution, SYGMA (H1) is actually five times faster on average than this algorithm for the last distribution. Moreover, although SYGMA (H1) is only slightly better than GASTON (EL) in the most skewed distribution, there is a more visible improvement

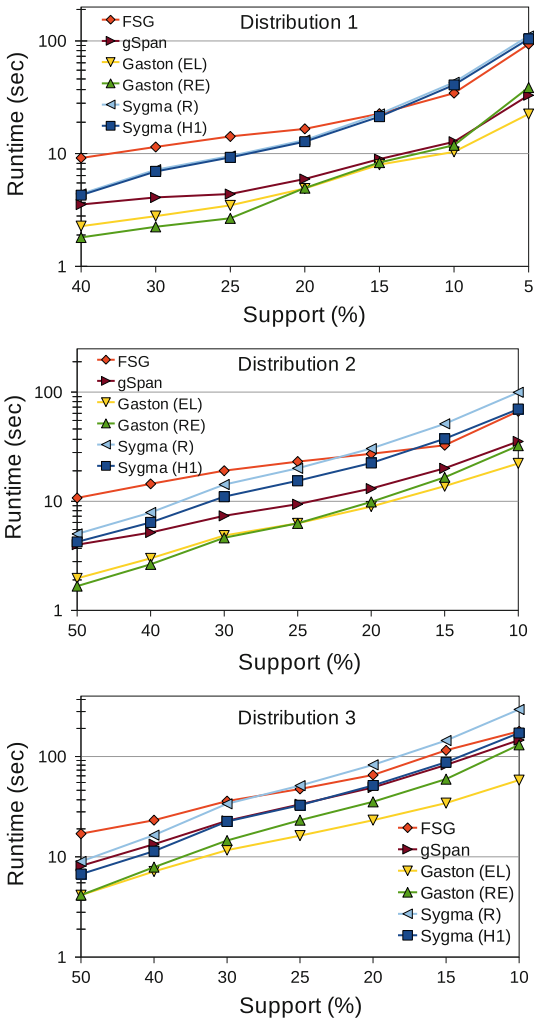

Fig. 8 Average runtimes needed by the tested algorithms to find the frequent subgraphs in the 10 synthetic datasets, using label distributions D_i, $1 \leq i \leq 3$, and decreasing minimum support thresholds

over GASTON (RE), which suggests that adding the strategy of storing embeddings to SYGMA could further enhance its performance.

The comparison of our approach with the other methods may not be reliable, as these methods use different techniques to compute the support of subgraphs in the database. As both SYGMA (H1) and (R) use the same support computation procedure, a more reliable picture can be obtained by comparing these two methods. Thus, we see that SYGMA (H1) becomes noticeably more efficient as the label distribution becomes more skewed: while SYGMA (H1) and SYGMA (R) offer the same performance when all labels are equiprobable, the heuristic version is twice faster on average than the random one for the most skewed distribution.

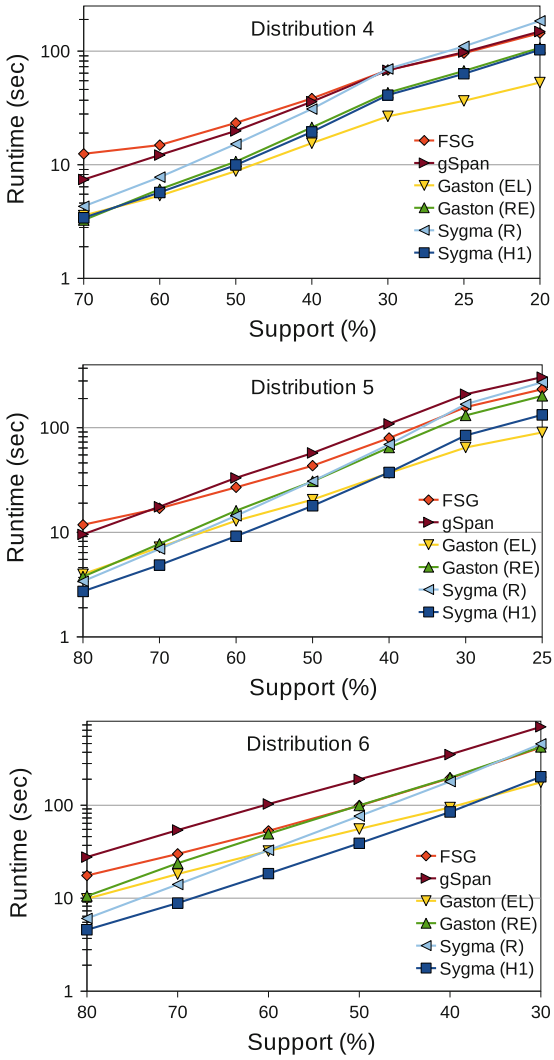

Fig. 9 Average runtimes needed by the tested algorithms to find the frequent subgraphs in the 10 synthetic datasets, using label distributions D_i, $4 \leq i \leq 6$, and decreasing minimum support thresholds

7.2.2 Real-life datasets

This last experiment evaluates the benefits of using heuristics on two real-life datasets from the field of chemoinformatics. These datasets, named CAN2DA99 and AID2DA99, are large molecular databases provided by the National Cancer Institute (NCI 1999), which have been used to benchmark frequent subgraph mining algorithms such as in (Nijssen and Kok 2004). Statistics on these datasets are given in Table 4. As described in Sect. 5.2 and shown in Table 1, the distribution of node labels, corresponding to chemical elements, is highly skewed. In fact, this distribution is even more skewed than all the distributions tested in the previous experiment.

Table 4 Statistics of the NCI datasets

Statistic	CAN2DA99	AID2DA99
Nb. of graphs	32,557	42,689
Nb. of labels	67	65
Avg. nb. of nodes	26.3	25.6
Max. nb. of nodes	229	222
Avg. nb. of edges	28.3	27.5
Max. nb. of edges	236	247
Avg. edge density	0.099	0.103

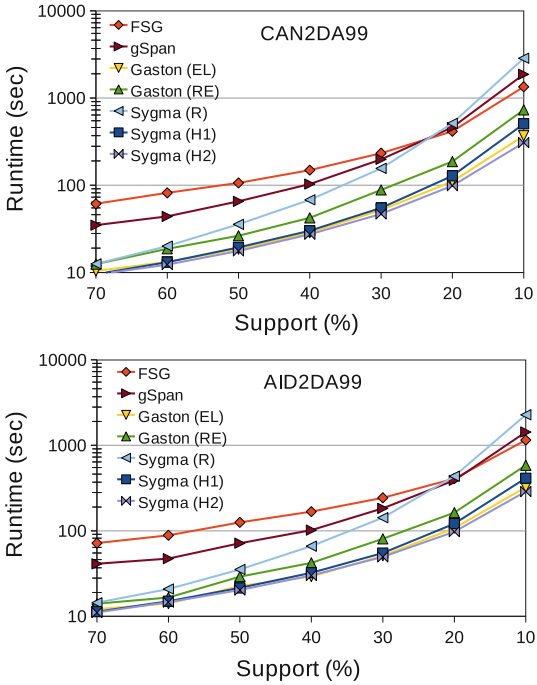

Fig. 10 Runtimes required by the tested algorithms to find the frequent subgraphs of the NCI datasets, for decreasing values of minimum support

For this experiment, we tested three different heuristics. The first two, denoted (R) and (H1), are the ones used in the previous experiment. The third one, called (H2), extends (H1) by also considering the node degrees of the subgraph patterns, as described in Sect. 5.2. In (H2), the canonical parent of a graph G is selected as one of the connected edge-induced subgraph for which the sum of the minimum, average and maximum node degree is the highest. In case of ties, heuristic (H1) is used to narrow down the list of potential canonical subgraphs. Once again, we compared these heuristic with algorithms FSG, GSPAN, GASTON (EL) and GASTON (RE).

The runtimes obtained using the seven tested algorithms on the NCI datasets are given in Fig. 10. Again, we notice that methods using heuristics are more efficient than the one choosing the canonical parent at random. Thus, for the lowest support

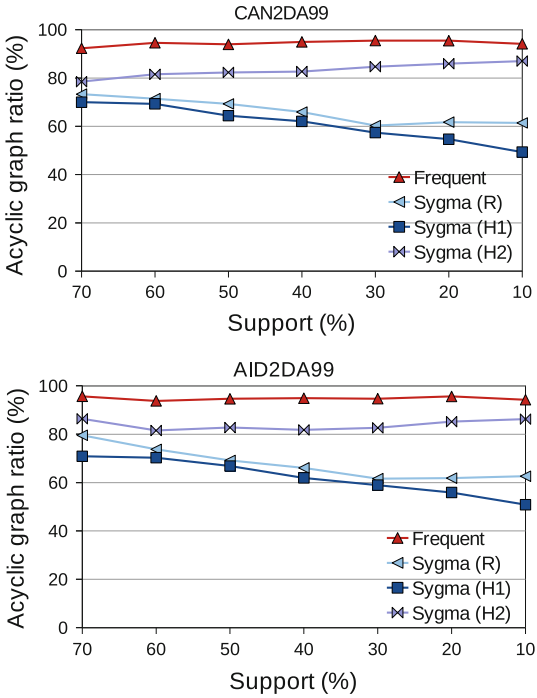

Fig. 11 Percentage of cycle-free graphs among the frequent subgraphs (*Frequent*), as well as the subgraphs explored (i.e., frequent or *not*) using heuristics (R), (H1) and (H2) while mining the NCI datasets with decreasing values of minimum support

threshold of 10%, (H2) is about 9 times faster than (R), on dataset CAN2DA99, and about 8 times faster on dataset AID2DA99. Comparing heuristics (H1) and (H2), as shown in Fig. 11, (H2) explores a greater percentage of cycle-free subgraph patterns than (H1). In the figure, *Frequent* gives the percentage of frequent subgraphs that are cycle-free, which is independent of the exploration heuristic. Since (H2) explores a ration of cycle-free graphs closer to the ratio observed in frequent graphs, this heuristic is up to 50% faster than (H1). Moreover, we observe a significant improvement over algorithms GSPAN and FSG which do not consider the properties of the frequent subgraph patterns. Thus (H2) is 3 to 6 times faster than these methods, over the whole range of minimum support values. Finally, we see that (H2) give results comparable to GASTON (EL), even though our approach does not use specialized methods to mine paths, trees and cyclic graphs, and does not store all pattern embeddings. However, a more significant improvement can be observed for (H2) over GASTON (RE), which uses a more similar approach to compute the support of subgraph patterns.

8 Conclusion

In summary, we have introduced a general framework to mine patterns with antimonotone constraints. This framework extends some of the prior work done on the topic by providing a simple method that allows antimonotone constraints to be used within the mining process, without the risk of pruning interesting patterns or exploring isomorphic patterns. It is also more general than existing approaches, as it provides simple rules to mine any type of patterns with a relation of generality, and allows the use of

various exploration strategies like depth-, breadth- and best-first search. Moreover, the flexibility of this approach makes possible the use of a general technique to boost the mining process with first-fail-based heuristics. Although defining a useful heuristic may require data-specific knowledge, several strategies and examples were given for this task.

To evaluate our framework, we performed experiments in two different settings. In the first setting, we tested the impact of integrating inside the mining process constraints on minimum cycle size and edge density, on the problem of finding the frequent connected subgraphs of a database. The results of this experiment have shown a significant gain in speed over the naive generate and test approach. On the other hand, two experiments were conducted in the second setting, which evaluated the impact of using heuristics on the problem of mining the frequent subgraphs of a database. The first one tested a simple heuristic using the frequency of node labels, on synthetic datasets whose label distributions are increasingly skewed. In the second experiment, this heuristic was compared with another one using information on the node degrees, on two real-life datasets containing molecular graphs. The results of these two experiments have shown a clear advantage of using such heuristics.

In future works, we would like to evaluate the usefulness of our framework for mining other types of patterns. In particular, this framework could be especially useful for more complex types of patterns like hyper-subgraphs or first-order logic rules, for which implementing a prefix-coding method is harder. Moreover, we would also like to test this framework on a wider set of pattern constraints and devise a more formal approach to defining good heuristics for a given constraint. Finally, it would be interesting to evaluate the advantage of using this framework in other pattern mining settings. Specifically, our approach may yield even greater benefits in the task of discovering the patterns occurring frequently in a single transaction or the problem of finding a single best pattern.

References

Agrawal R, Imielinski T, Swami AN (1993) Mining association rules between sets of items in large databases. In: Buneman P, Jajodia S (eds) Proceedings of the 1993 ACM SIGMOD international conference on management of data. Washington, DC, pp 207–216

Asai T, Abe K, Kawasoe S, Arimura H, Sakamoto H, Arikawa S (2002) Efficient substructure discovery from large semi-structured data. In Proceedings of the 2nd annual SIAM symposium on data mining, pp 158–174

Bessière C, Régin J-C (1996) Mac and combined heuristics: two reasons to forsake FC (and CBJ ?) on hard problems

Borgelt C, Berthold MR, Patterson DE (2005) Molecular fragment mining for drug discovery. Number 3571 in lecture notes in AI. Springer Verlag, pp 1002–1013

Chen Y, Yang LH, Wang YG (2004) Incremental mining of frequent xml query patterns. In: Proceedings of 4th IEEE international conference on data mining (ICDM'04), Los Alamitos, CA, USA. IEEE Computer Society, pp 343–346

Dehaspe L, Toivonen H (1999) Discovery of frequent datalog patterns. Data Min Knowl Discov 3(1):7–36

Deshpande M, Kuramochi M, Karypis G (2002) Automated approaches for classifying structures. In: Proceedings of the 2002 workshop on data mining in bioinformatics (BIOKDD'02). Edmonton, Canada, pp 11–18

Desrosiers C, Galinier P, Hansen P, Hertz A (2007) Sygma: reducing symmetry in graph mining. Technical Report G-2007-12, Les Cahiers du GERAD, December 2007

Fortin S (1996) The graph isomorphism problem. Technical Report 96-20, University of Alberta, Edomonton, Alberta, Canada
Gade K, Wang J, Karypis G (2004) Efficient closed pattern mining in the presence of tough block constraints. In: Proceedings of ACM SIGKDD, ACM, pp 138–147
Garey MR, Johnson DS (1979) Computers and intractability: a guide to the theory of NP-completeness. W. H. Freeman & Co Ltd., New York
Garofalakis M, Rastogi R, Shim K (2002) Mining sequential patterns with regular expression constraints. IEEE Trans Knowl Data Eng 14(3):530–552
Han J, Pei J, Yin Y (2000) Mining frequent patterns without candidate generation. In: Chen W, Naughton J, Bernstein PA (eds) 2000 ACM SIGMOD international conference on management of data. ACM Press, pp 1–12
Huan J, Wang W, Prins J (2003) Efficient mining of frequent subgraph in the presence of isomorphism. In: Proceedings of the 3rd IEEE international conference on data mining (ICDM), pp 549–552
Inokuchi A, Washio T, Motoda H (2000) An apriori-based algorithm for mining frequent substructures from graph data. In: Proc of the 4th European conference on principles of data mining and knowledge discovery. Springer-Verlag, pp 13–23
Kramer S, De Raedt L, Helma C (2001) Molecular feature mining in HIV data. In: Proceedings of the seventh ACM SIGKDD international conference on knowledge discovery and data mining (KDD '01). ACM, New York, NY, USA, pp 136–143
Kuramochi M, Karypis G (2001) Frequent subgraph discovery. In: Proceedings of the first IEEE conference on data mining, pp 313–320
Kuramochi M, Karypis G (2005) Finding frequent patterns in a large sparse graph. Data Min Knowl Discov 11(3):243–271
Lee SD, De Raedt L (2004) Constraint based mining of first order sequences in seqlog. In: Database support for data mining application. Springer, pp 155–176
Mannila H, Toivonen H (1997) Levelwise search and borders of theories in knowledge discovery. Data Min Knowl Discov 1(3):241–258
McKay B (1981) Practical graph isomorphism. Congressus Numeratium 30:45–87
National Cancer Institute (NCI) (1999) DTP/2D and 3D structural information. http://cactus.nci.nih.gov/ncidb2/download.html
Ng RT, Lakshmanan LVS, Han J, Pang A (1998) Exploratory mining and pruning optimizations of constrained associations rules. In: SIGMOD'98, pp 13–24
Nijssen S, Kok JN (2001) Faster association rules for multiple relations. In: IJCAI, pp 891–896
Nijssen S, Kok JN (2004) The gaston tool for frequent subgraph mining. In: Proceedings of the international workshop on graph-based tools (Grabats 2004), October 2004. Elsevier, pp 281–285
Pei J, Han J, Mortazavi-Asl B, Pinto H, Chen Q, Dayal U, Hsu M (2001) PrefixSpan: mining sequential patterns efficiently by prefix-projected pattern growth. In: Proceedings of the 17th international conference on data engineering (ICDE '01). IEEE Computer Society, Washington, DC, USA, pp 215–224
Srikant R, Agrawal R (1996) Mining sequential patterns: generalizations and performance improvements. In: EDBT '96: proceedings of the 5th international conference on extending database technology, pp 3–17
Sternberg MJE, King RD, Srinivasan A, Muggleton S (1995) Drug design by machine learning. In: Machine intelligence, vol 15, pp 328–338
Termier A, Rousset M-C, Sebag M (2002) Treefinder: a first step towards xml data mining. In: Proc of int conf on data mining ICDM'02. Maebashi, Japan, pp 450–457
Wang C, Hong M-S, Wang W, Shi B-L (2004) Chopper: efficient algorithm for tree mining. J Comput Sci Technol 19(3):309–319
Wang C, Zhu Y, Wu T, Wang W, Shi B (2005) Constraint-based graph mining in large database. In: Proc of APWeb 2005, pp 133–144
Yan X, Han J (2002) gSpan: graph-based substructure pattern mining. In: Proceedings of the 2002 IEEE international conference on data mining (ICDM'02). IEEE Computer Society, Washington, DC, USA, pp 721–724
Yan X, Yu P, Han J (2004) Graph indexing: a frequent structure-based approach. In: Proceedings of 2004 ACM-SIGMOD international conference management of data (SIGMOD'04). Paris, France, pp 335–346
Zaki MJ (2000) Sequence mining in categorical domains: incorporating constraints. In: Proceedings of the 9th international conference on information and knowledge management, pp 422–429

Zaki MJ (2002) Efficiently mining frequent trees in a forest. In: KDD '02: proceedings of the eighth ACM SIGKDD international conference on knowledge discovery and data mining. 2002 ACM, New York, NY, USA, pp 71–80

Zaki MJ, Nadimpally V, Bardhan D, Bystroff C (2004) Predicting protein folding pathways. Bioinformatics 20(1):386–393

Zhu F, Yan X, Han J, Yu PS (2007) gPrune: a constraint pushing framework for graph pattern mining. In: Advances in knowledge discovery and data mining. PAKDD 2007, pp 388–400