Mining Tree Patterns with Partially Injective Homomorphisms

Till Hendrik Schulz[1], Tamás Horváth[1,2,3], Pascal Welke[1], and Stefan Wrobel[1,2,3]

[1]Dept. of Computer Science, University of Bonn, Bonn, Germany
[2]Fraunhofer IAIS, Schloss Birlinghoven, Sankt Augustin, Germany
[3]Fraunhofer Center for Machine Learning, Sankt Augustin, Germany

Abstract. One of the main differences between inductive logic programming (ILP) and graph mining lies in the pattern matching operator applied: While it is mainly defined by relational homomorphism (i.e., subsumption) in ILP, subgraph isomorphism is the most common pattern matching operator in graph mining. Using the fact that subgraph isomorphisms are injective homomorphisms, we bridge the gap between ILP and graph mining by considering a natural transition from homomorphisms to subgraph isomorphisms that is defined by *partially injective homomorphisms*, i.e., which require injectivity only for subsets of the vertex pairs in the pattern. Utilizing positive complexity results on deciding homomorphisms from bounded tree-width graphs, we present an algorithm mining frequent trees from *arbitrary* graphs w.r.t. partially injective homomorphisms. Our experimental results show that the predictive performance of the patterns obtained is comparable to that of ordinary frequent subgraphs. Thus, by preserving much from the advantageous properties of homomorphisms and subgraph isomorphisms, our approach provides a trade-off between efficiency and predictive power.

1 Introduction

Despite the facts that graphs can be considered as relational structures and graph patterns as function-free first-order goal clauses, *inductive logic programming* (ILP) [10] and *graph mining* are typically regarded as independent research fields. One of the reasons for this separation lies in the relative simplicity of the vocabularies for graphs as relational structures. Another important difference is that while the pattern matching operator in ILP is defined by *subsumption* (cf. [10]), a weakening of *first-order implication*, it is mainly the *subgraph isomorphism* in graph mining. For first-order function-free clauses, subsumptions are in fact homomorphisms between relational structures (see, e.g., [8]). Thus, ILP (mainly) applies *relational homomorphisms*, while graph mining deploys *subgraph isomorphisms*.

Our goal is to propose a binary feature space for *predictive graph mining* that is spanned by *frequent patterns*. Due to the lack of injectivity, frequent patterns w.r.t. *homomorphism* result, on the one hand, in a loss in predictive performance compared to *subgraph isomorphism* when used for classification tasks. On the other hand, however, homomorphism is decidable in polynomial time for a broad class of patterns for which subgraph isomorphism remains persistently NP-complete (e.g., while the existence of a homomorphism from a path into a graph can be decided in polynomial time, this problem is NP-complete for subgraph isomorphism). As a trade-off between expressiveness and complexity, our goal is to preserve from the rigidity of subgraph isomorphism as much as possible, while utilizing the efficiency of homomorphisms for tractable graph classes. The difference between these two pattern matching operators is that any subgraph isomorphism is in fact an *injective* homomorphism. We therefore consider a natural transition from homomorphisms to subgraph isomorphisms that is defined by *partially injective homomorphisms*, i.e., which require injectivity only for a *subset* of the vertex pairs.

For loop-free graphs, any partially injective homomorphism can polynomially be reduced to an *ordinary* homomorphism by extending the graphs with additional edges corresponding to the injectivity constraints. To distinguish between original and constraint edges, we use edge colors. It holds that distinct sets of injectivity constraints define different partially injective homomorphisms problems between the same pattern and target graphs. In particular, the empty (resp. maximum) set of injectivity constraints corresponds to ordinary homomorphism (resp. subgraph isomorphism). By means of partially injective homomorphisms we can thus relax the rigid conception of having the binary choice between homomorphism and subgraph isomorphism, and are flexible to *dynamically* choose the degree of injectivity in the pattern matching operator. To the best of our knowledge, the application of *dynamic* pattern matching operators is an entirely novel characteristic in pattern mining, distinguishing it from all other traditional pattern matching operators used in ILP and graph mining.

Our approach can *efficiently* be applied to all pattern classes *from* which homomorphisms can be decided efficiently. For this work we consider the class of *bounded tree-width* graphs [12] (cf. [4] for the positive result on the complexity of homomorphisms from bounded tree-width graphs). More precisely, we restrict the patterns to *trees* and require the tree together with the additional constraint edges to form a graph of *tree-width* at most k, where $k > 0$ is some (small) constant. While this kind of partially injective homomorphisms from trees into arbitrary graphs is decidable in polynomial time, ordinary subgraph isomorphism from a tree remains NP-complete.

Using this idea, we propose an algorithm mining *frequent trees* w.r.t. partially injective homomorphisms. The rationale behind the choice of tree patterns is that the predictive performance achieved with frequent trees compares well to that of frequent connected subgraphs [14]. As the set of injectivity constraints depends on the particular pattern at hand, the output of the mining algorithm contains not only the tree patterns, but also the injectivity constraints. A complete enumeration of *all* frequent patterns is, however, practically infeasible for the potentially huge number of injectivity constraint sets. We overcome this problem by considering only patterns which are *k-trees* [13], i.e., edge maximal graphs of tree-width at most k. Utilizing the algorithmic definition of k-trees (cf.

[13]), we arrive at a natural refinement operator for the corresponding pattern mining problem, allowing for an *efficient* frequent pattern enumeration.

We have empirically evaluated the predictive performance and the runtime of our approach on real-world and artificial datasets. The predictive performance obtained by frequent subtrees w.r.t. partially injective homomorphism was very close to that of ordinary frequent subtrees and hence, to that of ordinary frequent subgraphs [14], and was achieved already for tree-width at most 3. Regarding the runtime, our algorithm is slower on molecular graphs than GASTON [11] and FSG [5] which seem to be specifically designed for this kind of graphs. However, already on slightly more complex structures beyond chemical graphs, our method always terminates and is faster by at least 1 (up to 3) orders of magnitude than GASTON and FSG (when they terminate at all). Thus, our approach offers a trade-off between runtime and predictive power via the choice of tree-width.

The rest of the paper is organized as follows. We collect the necessary notions in Sect. 2, introduce the concept of partially injective homomorphisms in Sect. 3, and present our mining algorithm in Sect. 4. We report our empirical results in Sect. 5 and conclude in Sect. 6. For page limitations, proofs are omitted in this short version.

2 Notions and Notation

In this section we collect the necessary notions from graph theory (see, e.g., [6]) and fix the notation. For a set S, let $[S]^2 = \{X \subseteq S : |X| = 2\}$. The set $\{u, v\} \in [S]^2$ is denoted by uv. An *undirected* (resp. *directed*) *labeled graph* over an alphabet Σ is a triple $G = (V, E, \ell)$ consisting of a set V of vertices, a set $E \subseteq [V]^2$ (resp. $E \subseteq V \times V$) of edges, and a labeling function $\ell : V \cup E \to \Sigma$ assigning a label from Σ to each vertex and edge. We often denote the set of vertices of G by $V(G)$ and the set of edges by $E(G)$. For simplicity we present our result for undirected graphs, by noting that any undirected graph G can be regarded as a directed graph such that all edges $uv \in E(G)$ are replaced by the directed edges (u, v) and (v, u). Accordingly, unless otherwise stated, by graphs we always mean undirected graphs.

A *homomorphism* from a graph $G = (V, E, \ell)$ into a graph $G' = (V', E', \ell')$ is a function $\varphi : V \to V'$ preserving all edges and labels, i.e., (i) $\varphi(u)\varphi(v) \in E'$ for all $uv \in E$, (ii) $\ell(v) = \ell'(\varphi(v))$ for all $v \in V$, and (iii) $\ell(uv) = \ell'(\varphi(u)\varphi(v))$ for all $uv \in E$. If, in addition, φ is *injective* then it is a *subgraph isomorphism* from G to G'. The graphs G and G' above are *isomorphic* if there exists a bijection φ between $V(G)$ and $V(G')$ such that φ and its inverse φ^{-1} are both homomorphisms. To enforce the morphisms above to satisfy certain injectivity constraints, the edges of the graphs, in addition to their labels, will have some color as well. In such cases homomorphisms (and hence, subgraph isomorphisms and isomorphisms) are required to preserve the edge colors as well. The definitions of homomorphism, subgraph isomorphism, and isomorphism between directed graphs are analogous with the additional constraint that φ must preserve not only the edges, but also their directions. As the generalization of our method from unlabeled graphs to labeled graphs is straightforward, for simplicity we present our algorithms for the unlabeled case.

We will pay a special attention to graphs of bounded *tree-width* [12]. For the definition and basic properties of bounded tree-width graphs, the reader is referred e.g. to [6]. Given a pattern graph H and a target graph G, it is NP-complete to decide whether there exists a homomorphism from H to G. If, however, the tree-width of H is bounded by some constant, then this problem can be decided in polynomial time for any graph G (see, e.g, [4]). For an integer $k > 0$, a *k-tree* [13] is defined recursively as follows: (i) A clique (i.e., fully connected graph) of $k+1$ vertices is a k-tree. (ii) Given a k-tree G with n vertices, a k-tree with $n+1$ vertices is obtained from G by adding a new vertex v to G and connecting v to all vertices of a k-clique (i.e., a clique of size k) of G. It is well-known that a k-tree has always tree-width k and that it is edge maximal w.r.t. this property, i.e., adding any further edge to a k-tree results in a graph of tree-width $k+1$ (cf. [13]). Furthermore, all k-trees have $k|V(G)| - \binom{k+1}{2}$ edges. Finally, a *partial k-tree* is a subgraph of a k-tree and hence has tree-width at most k.

Let $\preceq$ be a preorder (i.e., reflexive and transitive relation) on a set S. For $a, b \in S$ we say that $a \prec b$ iff $a \preceq b$ and $a \not\succeq b$, and define the *equivalence* relation $\equiv$ on S by $a \equiv b$ iff $a \preceq b$ and $b \preceq a$. A function $\rho : S \to 2^S$ is called a *refinement operator* for $(S, \preceq)$ if for every $a \in S$ we have $\rho(a) \subseteq \{b \in S | a \preceq b\}$. Furthermore, ρ is (i) *locally finite* if $\rho(a)$ is finite for every $a \in S$, (ii) *complete* if for every $a, b \in S$ with $a \prec b$ there exist $a \equiv c_0, c_1, \ldots, c_n \equiv b$ with $c_i \in \rho(c_{i-1})$ for all $i = 1, \ldots, n$, (iii) *proper* if for all $a \in S$, $\rho(a) \subseteq \{b \in S | a \prec b\}$, and (iv) *ideal* if ρ is locally finite, complete and proper. For basic properties of refinement operators, the reader is referred e.g. to [10].

3 Partially Injective Homomorphisms

In this section we define *partially injective homomorphisms*, the central notion for this work, and discuss some of its properties. On the one hand, while the homomorphism and subgraph isomorphism problems are both NP-complete in general, their complexity behaves differently on special subproblems. In particular, homomorphism can be decided in polynomial time for a broad range of pattern classes for which subgraph isomorphism remains NP-complete. As an example, whereas the homomorphism problem from graphs of bounded tree-width can be decided in polynomial time [4], subgraph isomorphism remains NP-complete not only from trees (which have tree-width 1), but even from paths. On the other hand, however, as we empirically demonstrate in Section 5, frequent patterns generated w.r.t. subgraph isomorphism yield a much higher predictive performance when used for classification purposes than those w.r.t. homomorphism. As subgraph isomorphisms are injective homomorphisms, these empirical results clearly highlight the importance of the *injectivity* of the pattern matching operator in practice. Motivated by our experiments, we propose a trade-off between complexity and predictive performance by closing the gap between homomorphisms and subgraph isomorphisms. It follows from the remarks above that a natural transition from homomorphisms to subgraph isomorphisms can be obtained by *partial* injectivity, i.e., by requiring the injectivity constraint *not* for all vertex pairs in the pattern, but only for a subset of them. Formally, a pattern graph H can be embedded into a target graph G by a *partially injective homomorphism* satisfying a set $\mathcal{C} \subseteq [V(H)]^2$ of injectivity constraints, denoted by $H \underset{\mathcal{C}}{\rightarrow} G$, if there exists a homomorphism φ from H to G such that $\varphi(u) \neq \varphi(v)$ for all $uv \in \mathcal{C}$. In case of ordinary homomorphisms, i.e., when $\mathcal{C} = \emptyset$, the above notation reduces to $H \rightarrow G$. We refer to the corresponding decision problem, denoted PIHom$(H, G, \mathcal{C})$, as PIHom *problem* and call the pair $(H, \mathcal{C})$ a PIHom *pattern*.

Our key idea is to consider such PIHom problems that can *polynomially* be reduced to *efficiently* decidable ordinary homomorphism problems. For the efficiency, we consider graphs of bounded tree-width, utilizing positive complexity results on deciding homomorphisms from this graph class [4]. Regarding the reduction, for H, G, and $\mathcal{C}$ above we transform H and G into two edge colored graphs by the following steps:

1. Color all (original) edges of H and G in *blue*,
2. for all $uv \in E(H) \cup \mathcal{C}$, connect u and v by a red edge, and
3. for *all* $u, v \in V(G)$ with $u \neq v$, connect u and v by a red edge.

Let the graphs obtained be denoted by $H\langle\mathcal{C}\rangle$ and $G\langle\top\rangle$. As homomorphisms between colored graphs preserve also the edge colors, the proof of the claim below is immediate from the definitions, by noting that all graphs considered in this work are loop free:

Proposition 1. *For $H, G, \mathcal{C}$ and $H\langle\mathcal{C}\rangle, G\langle\top\rangle$ above we have $H \underset{\mathcal{C}}{\rightarrow} G$ if and only if $H\langle\mathcal{C}\rangle \rightarrow G\langle\top\rangle$.*

Notice that for loop-freeness it suffices to consider the injectivity constraints only for unconnected vertex pairs in the original pattern graph H (i.e., we can assume w.l.o.g. that $\mathcal{C} \cap E(H) = \emptyset$). Note also that for H and G above, each subset $\mathcal{C} \subseteq [V]^2 \setminus E(H)$ defines a distinct problem PIHom$(H, G, \mathcal{C})$ and that the set of all PIHom patterns over H form a (complete) lattice $(\mathcal{L}_H, \preceq)$ with

$$\mathcal{L}_H = \{(H, \mathcal{C}) : \mathcal{C} \subseteq [V(H)]^2 \setminus E(H)\}$$

and with partial order $\preceq$ defined as follows: For all $\mathcal{C}_1, \mathcal{C}_2 \subseteq [V(H)]^2 \setminus E(H)$, $(H, \mathcal{C}_1) \preceq (H, \mathcal{C}_2)$ iff $\mathcal{C}_1 \subseteq \mathcal{C}_2$. The least element $(H, \emptyset)$ (resp. greatest element $(H, [V]^2 \setminus E(H))$) of $\mathcal{L}_H$ corresponds to ordinary homomorphism (resp. subgraph isomorphism) from H. For any target graph G, $(\mathcal{L}_H, \preceq)$ is closed downwards in the sense that $H \underset{\mathcal{C}_1}{\longrightarrow} G$ and $\mathcal{C}_2 \subseteq \mathcal{C}_1$ implies $H \underset{\mathcal{C}_2}{\longrightarrow} G$.

Fig. 1 visualizes three partially injective homomorphisms from H into G. The graphs H and G are identical in all three examples, while the sets of injectivity constraints differ. We explicitly visualize the constraints in the depiction of patterns. For instance, the partially injective homomorphism depicted in Fig. 1(b)

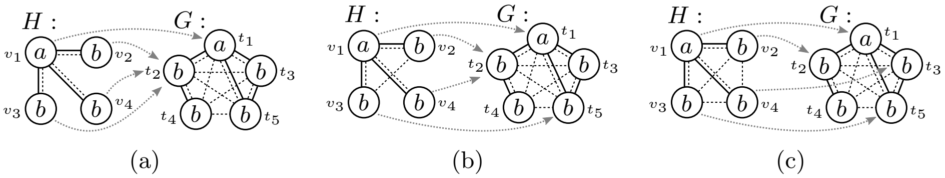

Fig. 1: Examples for different partially injective homomorphisms from H into G. Solid lines represent blue edges whereas dashed ones represent red edges.

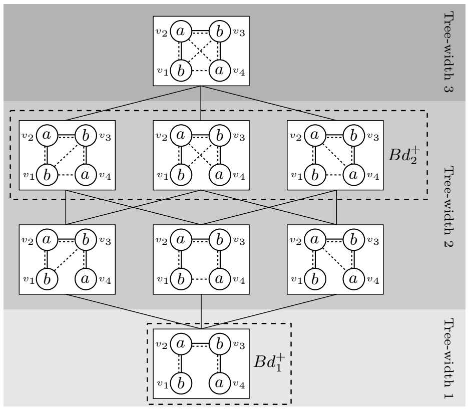

Fig. 2: Visualization of maximal borders in the lattice of PIHom patterns for a path.

requires the homomorphism φ from H to G to satisfy the injectivity constraint $\varphi(v_2) \neq \varphi(v_3)$.

We will consider lattices of PIHom *tree* patterns, i.e., when the first component in the PIHom pattern $(H, \mathcal{C})$ is a tree. In particular, when H is a tree with n vertices, the cardinality of the corresponding PIHom pattern lattice $\mathcal{L}_H$ is $2^{O(n^2)}$, i.e., *exponential* in the size of H. Fig. 2 illustrates such a lattice $(\mathcal{L}_H, \preceq)$ for a path H of length 3. Each depicted graph corresponds to a PIHom tree pattern with a specific set of injectivity constraints.

4 Pattern Mining

In this section we define the problem of mining frequent maximally constrained PIHom tree patterns. We first claim that there exists no locally finite and complete refinement operator for a natural representative set of this kind of patterns if we want to avoid redundancy. We therefore relax the problem definition and propose an efficient pattern mining algorithm tolerating redundancies in the output. The restriction of patterns to *trees* is motivated by complexity issues and by the remarkable predictive performance which is achieved by using frequent subtrees w.r.t. subgraph isomorphism [14]. In general, given a *tree* H and target graph G, PIHom$(H, G, \mathcal{C})$ can be decided in polynomial time if $\mathcal{C} = \emptyset$ (i.e., for ordinary homomorphism), but is NP-complete whenever $\mathcal{C} = [V(H)]^2 \setminus E(H)$ (i.e., for subgraph isomorphism). We bridge this complexity gap by considering a distinguished subset of $\mathcal{L}_H$ for which the corresponding PIHom problems are decidable in polynomial time for any target graph G. More precisely, for a tree H and some constant $k > 0$ we consider the pattern set $\mathcal{L}_H^k = \{(H, \mathcal{C}) \in \mathcal{L}_H : H\langle\mathcal{C}\rangle$ has tree-width at most $k\}$. Proposition 1 together with the positive complexity result on deciding homomorphisms from graphs of bounded tree-width [4] implies that for all $(H, \mathcal{C}) \in \mathcal{L}_H^k$ and for all target graphs G, PIHom$(H, G, \mathcal{C})$ can be decided in polynomial time.

Our experiments (cf. Sect. 5) clearly demonstrate that, besides the structural gap between homomorphisms and subgraph isomorphisms discussed earlier, there is a large gap between their predictive performances as well. Furthermore, this gap vanishes as the number of injectivity constraints increases. Motivated by this empirical observation and the negative result formulated in Section 4.1 below, we will pay a special attention to the *maximal* elements of $\mathcal{L}_H^k$, i.e., to such PIHom patterns $(H, \mathcal{C}) \in \mathcal{L}_H^k$ for which $H\langle\mathcal{C}\rangle$ is a complete graph whenever $|V(H)| \leq k + 1$; o/w it is a k-tree. In other words, we will consider the *positive border* of $(\mathcal{L}_H^k, \preceq)$ w.r.t. the following *interestingness predicate*: $(H, \mathcal{C}) \in \mathcal{L}_H$ is *interesting* if $H\langle\mathcal{C}\rangle$ has tree-width at most k.

As an example, all patterns $(H, \mathcal{C})$ with $|\mathcal{C}| = 2$ in Fig. 2 are maximally constrained w.r.t. tree-width 2, as they have tree-width 2 and adding any further constraint would increase their tree-width. (A 4-clique has tree-width 3.) The positive borders for $k = 1$ and $k = 2$ are denoted by Bd_1^+ and Bd_2^+, respectively. Clearly, $Bd_1^+ = \{(H, \emptyset)\}$ since adding any further injectivity constraint would result in a pattern of tree-width 2. In the example in Fig. 2 there are e.g. seven constraint sets for the path H connecting v_1 and v_4. For all of these constraint sets $\mathcal{C}$, $H\langle\mathcal{C}\rangle$ has tree-width at most 2. Yet, we are most interested in the three patterns with $|\mathcal{C}| = 2$ (i.e., Bd_2^+), as they have the highest degree of partial injectivity.

4.1 PIHom Core Patterns: A Negative Result

Using the concepts introduced above, in this section we consider the pattern language $\mathcal{L}^k$ defined by the union of the $\mathcal{L}_H^k$s over all tree patterns H. Given a database $\mathcal{D}$ of graphs, our goal is to generate a subset S of $\mathcal{L}^k$ on the basis of $\mathcal{D}$ that will span the binary feature space in which the elements of $\mathcal{D}$, as well as further unseen graphs will be embedded. For this application purpose, it is desirable to avoid "redundancies" among the patterns in S. There are various ways of defining redundancy. Perhaps the most natural one is the following definition: S contains *no* two *model equivalent* patterns, where the patterns $(H_1, \mathcal{C}_1), (H_2, \mathcal{C}_2) \in \mathcal{L}^k$ are model equivalent, denoted $(H_1, \mathcal{C}_1) \equiv_{\mathrm{m}} (H_2, \mathcal{C}_2)$, if the equivalence $H_1 \underset{\mathcal{C}_1}{\longrightarrow} G \iff H_2 \underset{\mathcal{C}_2}{\longrightarrow} G$ holds for all graphs G.

For complexity reasons, we regard a relaxed notion of redundancy and show that even this weaker form raises severe algorithmic issues. More precisely, consider the order $\preceq_{\mathrm{h}}$ on $\mathcal{L}^k$ defined as follows: For all $(H_1, C_1), (H_2, C_2) \in \mathcal{L}^k$,

$(H_1, C_1) \preceq_{\mathrm{h}} (H_2, C_2)$ iff $H_1\langle \mathcal{C}_1 \rangle \rightarrow H_2\langle C_2 \rangle$. One can easily see that $\preceq_{\mathrm{h}}$ is a preorder on $\mathcal{L}^k$. Using this definition, a set $S \subseteq \mathcal{L}^k$ is regarded as *non-redundant* if it contains no two *homomorphism equivalent* patterns, where $(H_1, \mathcal{C}_1), (H_2, \mathcal{C}_2) \in \mathcal{L}^k$ are homomorphism equivalent, denoted $(H_1, \mathcal{C}_1) \equiv_{\mathrm{h}} (H_2, \mathcal{C}_2)$, iff $(H_1, C_1) \preceq_{\mathrm{h}} (H_2, C_2)$ and $(H_2, C_2) \preceq_{\mathrm{h}} (H_1, C_1)$. Clearly, $\equiv_{\mathrm{m}}$ and $\equiv_{\mathrm{h}}$ are both equivalence relations and $\equiv_{\mathrm{h}}$ is finer than $\equiv_{\mathrm{m}}$, i.e., the partition of $\mathcal{L}^k$ induced by $\equiv_{\mathrm{h}}$ is a refinement of that induced by $\equiv_{\mathrm{m}}$. Thus, model equivalent patterns are not necessarily homomorphism equivalent, implying that $\equiv_{\mathrm{h}}$ may allow a certain amount of redundancies w.r.t. $\equiv_{\mathrm{m}}$.

Let $\mathcal{L}^k/{\equiv_{\mathrm{h}}}$ denote the set of equivalence classes induced by $\equiv_{\mathrm{h}}$. For all $c \in \mathcal{L}^k/{\equiv_{\mathrm{h}}}$, one can consider the *core* of c, a canonical representative element of c, defined as follows: Select an arbitrary pattern $(H, \mathcal{C}) \in c$ and take the smallest subgraph $H'\langle \mathcal{C}' \rangle$ of $H\langle \mathcal{C} \rangle$ such that $(H, \mathcal{C}) \equiv_{\mathrm{h}} (H', \mathcal{C}')$. It holds that $H'\langle \mathcal{C}' \rangle$ (and hence, $(H', \mathcal{C}')$) can be computed by a greedy algorithm removing the redundant edges one by one. The properties of tree-width together with the results of [2, 4] imply that $(H', \mathcal{C}')$ is a core of c, it always exists and is unique modulo isomorphism independently of the choice of $(H, \mathcal{C})$, and can be calculated in time polynomial in the size of $(H, \mathcal{C})$. The reason of using cores is that in such cases when the pattern matching operator is defined by (relational) homomorphism (e.g. in ILP [10]), the equivalence classes in $\mathcal{L}^k/{\equiv_{\mathrm{h}}}$ may contain infinitely many patterns, due to the fact that $\preceq_{\mathrm{h}}$ is not anti-symmetric.

While subgraph isomorphism as the pattern matching operator allows for a very natural refinement operator on the pattern language, this is typically not the case for homomorphism, caused also by the difference in the anti-symmetry. Indeed, in case of subgraph isomorphism, the pattern language along with the partial order defined by subgraph isomorphism can directly be translated into an *ideal* refinement operator (assuming that all patterns are connected); just extend the pattern at hand in every possible way either by a single edge or by a single vertex connected to one of the old vertices. In contrast, the preorder on $\mathcal{L}^k$ defined by homomorphism does not impose such an algebraic structure that could be turned into a (natural) algorithmic definition of a refinement operator on $\mathcal{L}^k_{\mathrm{c}}$. In fact, $\mathcal{L}^k_{\mathrm{c}}$ may contain cores having infinitely many "direct" refinements, implying the following negative result:

Theorem 1. *For all $k \geq 1$, there exists no finite and complete refinement operator for the preordered set $(\mathcal{L}^k_c, \preceq_h)$.*

The negative result formulated in Theorem 1 does not imply that PIHom core patterns cannot be enumerated *efficiently*; this question is an open problem.[1] However, it indicates that traditional pattern generation paradigms based on refinement operators are not applicable to $(\mathcal{L}^k_{\mathrm{c}}, \preceq_{\mathrm{h}})$. In the next section we therefore relax our problem setting and tolerate further redundancies in the output pattern set.

[1] In the long version of the paper we show that if the pattern language can further be restricted structurally then frequent cores w.r.t. a database cannot be generated in output polynomial time (unless P = NP).

4.2 The Problem Definition

Theorem 1 implies that we have to consider a different pattern language in place of $\mathcal{L}_{\mathrm{c}}^k$ if we want to generate the output patterns by using some algorithmically appropriate refinement operator. To achieve this goal, we consider PIHom tree patterns that are *maximally* constrained w.r.t. tree-width k, i.e., the set $\mathcal{L}_{\max}^k \subseteq \mathcal{L}^k$ defined as follows: For all $(H, \mathcal{C}) \in \mathcal{L}^k$, $(H, \mathcal{C}) \in \mathcal{L}_{\max}^k$ iff $|V(H)| \leq k+1$ and $H\langle\mathcal{C}\rangle$ is a complete graph or $|V(H)| > k+1$ and $H\langle\mathcal{C}\rangle$ is a k-tree. The partially injective homomorphisms obtained in this way are as close as possible to subgraph isomorphism subject to bounded tree-width, resulting in a pattern set of higher predictive performance as shown empirically in Section 5. Using this definition, we consider the following pattern generation problem:

Frequent Maximally Constrained Tree Mining (FMCTM) Problem: *Given* a finite set $\mathcal{D}$ of graphs and integers $t, h, k > 0$, *list* all $(H, \mathcal{C}) \in \mathcal{L}_{\max}^k$ such that $|V(H)| \leq h$ and $\mathrm{freq}((H, \mathcal{C}), \mathcal{D}) \geq t$, where $\mathrm{freq}((H, \mathcal{C}), \mathcal{D})$ denotes the (absolute) *frequency* of the PIHom tree pattern $(H, \mathcal{C})$ in $\mathcal{D}$, i.e., $\mathrm{freq}((H, \mathcal{C}), \mathcal{D}) = |\{G \in \mathcal{D} : H \underset{\mathcal{C}}{\rightarrow} G\}|$.

PIHom tree patterns satisfying the frequency constraint in the definition above will be referred to as *frequent* PIHom tree patterns, or simply frequent patterns. As mentioned earlier, one of the most important distinguishing features of the problem setting above is that the pattern matching operator is *not* static (i.e., fixed in advance), but *dynamic*, in contrast to all traditional frequent graph mining algorithms. Clearly, whether a tree pattern is frequent or not directly depends on the underlying set of constraints applied in the embedding operator. It is therefore necessary to output a frequent tree along with the injectivity constraints defining the (dynamic) pattern matching operator, i.e., the output is always a pair $(H, \mathcal{C})$, instead of H only. Notice that in this work we are not explicitly interested in the semantical properties of the patterns but rather in the binary feature space spanned by them.

The parameter h in the problem definition provides an upper bound on the size of the output patterns. It ensures that the algorithm solving the FMCTM problem will always terminate. It is not difficult to see that without h, the output of the FMCTM problem may contain infinitely many frequent patterns. We also note that the output $\mathcal{O}$ of the FMCTM problem may contain frequent patterns that are homomorphism equivalent. Furthermore, the elements of $\mathcal{O}$ are not necessarily cores. In case the output is required to be a non-redundant subset of $\mathcal{L}_{\mathrm{c}}^k$, after the computation of $\mathcal{O}$, one can first remove all patterns from it that are redundant w.r.t. homomorphism equivalence and then calculate the core for each pattern remaining in $\mathcal{O}$. Since all patterns in $\mathcal{O}$ have bounded tree-width, both steps can be performed in time polynomial in the size of $\mathcal{O}$.

4.3 The Mining Algorithm

In this section, we present our algorithm solving the FMCTM problem and prove that it is correct and enumerates the output patterns in incremental polynomial

Algorithm 1 Listing Frequent Maximally Constrained Trees

```
input:  graph dataset $\mathcal{D}$, integers $t, k, h > 0$
output: all $t$-frequent patterns of $\mathcal{L}^k_{\max}$ with size at least 1 and at most $h$

Enumerate($(H, \mathcal{C})$):
 1: $R :=$ Refinements($(H, \mathcal{C}), k$)
 2: for all $(H', \mathcal{C}') \in R$ do
 3:     if $|V(H')| \leq h \wedge (H', \mathcal{C}') \notin \mathcal{O} \wedge \mathrm{freq}((H', \mathcal{C}'), \mathcal{D}) \geq t$ then
 4:         print $(H', \mathcal{C}')$ and add it to $\mathcal{O}$
 5:         Enumerate($(H', \mathcal{C}')$)

Main:
 1: $\mathcal{O} := \emptyset$
 2: Enumerate($(\perp, \emptyset)$)                       // $\perp$ denotes the empty graph
```

time. For the reader's convenience we ignore a number of (standard) optimization, implementation, and further simplification issues from this short paper and provide only a simplified version of the algorithm implemented.

We guarantee efficiency (i.e., incremental polynomial time) by considering the *partial* order $\preceq_{\mathrm{si}}$ on $\mathcal{L}^k_{\max}$, instead of the *preorder* $\preceq_{\mathrm{h}}$, where $\preceq_{\mathrm{si}}$ is defined as follows: For all $(H, \mathcal{C}), (H', \mathcal{C}') \in \mathcal{L}^k_{\max}$, $(H, \mathcal{C}) \preceq_{\mathrm{si}} (H', \mathcal{C}')$ iff there exists a subgraph isomorphism from $H\langle\mathcal{C}\rangle$ into $H'\langle\mathcal{C}'\rangle$. Clearly, $\mathrm{freq}((H, \mathcal{C}), \mathcal{D}) \geq \mathrm{freq}((H', \mathcal{C}'), \mathcal{D})$ whenever $H\langle\mathcal{C}\rangle \preceq_{\mathrm{si}} H'\langle\mathcal{C}'\rangle$, i.e., frequency is *anti-monotonic* on the poset $(\mathcal{L}^k_{\max}, \preceq_{\mathrm{si}})$. Thus, maximal PIHom tree patterns are closed downwards w.r.t. frequency. While the poset $(\mathcal{L}^k_{\max}, \preceq_{\mathrm{si}})$ allows for efficient pattern enumeration, the output may contain patterns that are homomorphism equivalent. That is, the price we have to pay for the positive complexity result is that the output may contain some redundant patterns.

Algorithm 1 is based on the recursive function Enumerate generating the output patterns in a DFS manner. Its input consists of the same parameters $\mathcal{D}$, t, h, and constant k as the FMCTM problem. The output patterns already generated are stored in the global variable $\mathcal{O}$. The algorithm calls Enumerate with the empty pattern $(\perp, \emptyset)$, where $\perp$ denotes the empty graph (line 2 of Main). As a first step (line 1 of Enumerate), function Refinements generates the set of refinements for the input pattern $(H, \mathcal{C})$; the process governing how new candidate patterns are generated is determined by the refinement operator sketched below. If a newly generated candidate pattern $(H', \mathcal{C}')$ (i) fulfills the size constraint (i.e, $|V(H')| \leq h$), (ii) has not been generated before (i.e., $(H', \mathcal{C}') \notin \mathcal{O}$), and (iii) is t-frequent (i.e., $\mathrm{freq}((H', \mathcal{C}'), \mathcal{D}) \geq t$), we print it, store it in $\mathcal{O}$, and call Enumerate recursively for this new frequent pattern (lines 3–5).

Refinement Operator Function Refinements in Algorithm 1 returns the set R of *refinements* for a pattern $(H, \mathcal{C}) \in \mathcal{L}^k_{\max}$ and $k > 0$. All patterns $(H', \mathcal{C}') \in R$ are required to satisfy the following conditions: (i) H' is a supertree of H obtained by extending H with a new vertex and edge, (ii) $\mathcal{C} \subseteq \mathcal{C}'$, and (iii) $(H', \mathcal{C}') \in \mathcal{L}^k_{\max}$, i.e, it is maximal w.r.t. tree-width k. That is, trees of size n are extended into trees of size $n+1$ by condition (i). Furthermore, condition (iii) implies that $H'\langle\mathcal{C}'\rangle$ is a k-tree if $|V(H')| > k+1$; o/w it is a complete graph.

The algorithmic characterization of k-trees (cf. Section 2) gives rise to the following natural refinement operator on $\mathcal{L}^k_{\max}$: A pattern $(H', \mathcal{C}')$ of size $n+1$ is among the refinements of a pattern $(H, \mathcal{C}) \in \mathcal{L}^k_{\max}$ of size n iff $(H', \mathcal{C}')$ can be obtained from $(H, \mathcal{C})$ in the following way: If $n = 0$ (i.e., $(H, \mathcal{C}) = (\perp, \emptyset)$), we define the refinements of $(H, \mathcal{C})$ by the set of graphs consisting of a single vertex (and no edges, as we consider loop-free graphs).[2] Otherwise, i.e., for $n > 0$, we proceed as follows: (i) Introduce a new vertex u. (ii) If $n \leq k$, then connect u to a vertex $v \in V(H)$ and add an injectivity constraint uv' to $\mathcal{C}$ for every $v' \in V(H) \setminus \{v\}$; o/w select a k-clique C in $H\langle\mathcal{C}\rangle$, connect u to a vertex v of C in H, and add an injectivity constraint uv' to $\mathcal{C}$ for every $v' \in V(C) \setminus \{v\}$.

We are ready to state our main result:

Theorem 2. *For any $\mathcal{D}$, t, k, and h, Algorithm 1 is correct and generates the output patterns in incremental polynomial time.*

5 Experimental Evaluation

This section is concerned with the empirical evaluation of our approach. In particular, we evaluate the predictive performance of frequent maximally constrained PIHom tree patterns and the runtime of the algorithm presented in Sect. 4. For our experiments we used a modified variant of Algorithm 1 making it practically feasible. We omit these technical details from this short version.

In the experiments below we first analyze how the degree of injectivity in PIHom tree patterns impacts the resulting predictive performance. To answer this question, we fix a set of tree patterns and embed the data into feature spaces spanned by the tree patterns w.r.t. partially injective homomorphisms with increasing degree of injectivity, ranging from ordinary homomorphism to subgraph isomorphism. Our results clearly indicate a strong correlation between degree of injectivity and predictive performance. We then compare the predictive performance of the patterns generated by our practical algorithm with that of ordinary frequent subtrees on different benchmark datasets. We found that the predictive performance achieved by PIHom tree patterns for already moderate amounts of injectivity constraints compares favorably to that of ordinary frequent subtrees, which, in turn, are very close to frequent subgraphs in predictive power. Finally, we provide runtime measures comparing our algorithm to state-of-the-art graph miners like GASTON [11] and FSG [5]. We show that while these algorithms are practical only for very restricted graph types, our approach performs well on *arbitrary* graph datasets. In particular, while our implementation was generally slower on the real-world benchmark chemical graph datasets considered, it

[2] In case of labeled graphs, the number of such singleton graphs is equal to that of the different vertex labels in $\mathcal{D}$.

clearly outperforms GASTON [11] and FSG [5] on artificial datasets containing only slightly more complex graph structures beyond molecular graphs.

Datasets To evaluate the predictive performance, we use four benchmark molecular graph datasets. This choice is mainly of practical nature. In particular, the molecules considered are of a fairly simple structure [9], allowing for the application of state-of-the-art frequent subgraph mining systems that are generally very efficient on molecular graph data. This enables us to compare the predictive performance of our method to "gold-standard" results achieved by frequent subgraphs (w.r.t subgraph isomorphism).

The experiments are conducted on the well-established real-world datasets NCI1, NCI109, MUTAG, and PTC annotated for different binary target properties. NCI1 and NCI109 contain 4,110 (resp. 4,127) compounds. The graphs in these two datasets have 30 vertices and 32 edges on average. The MUTAG dataset contains merely 188 compounds; the graphs in this dataset have 18 vertices and 20 edges on average. Finally, the PTC dataset contains a total of 344 graphs, with an average number of 26 vertices and edges.

As most frequent subgraph mining systems like GASTON [11] are specifically designed to cope with graphs of simple structure (such as molecular graphs), runtime comparisons on only chemical graph datasets are not expressive enough. We therefore consider also artificial graph datasets generated according to the Erdős-Rényi random graph model. Each such dataset consists of 50 graphs with an average size of 25, corresponding to the chemical datasets above. The structural complexity of these graphs G corresponds to the edge/vertex ratio $q = |E(G)|/|V(G)|$. In our experiments, only connected graphs are considered.

Experimental Setup Using 10-fold cross-validation, the predictive performance was measured in terms of AUC obtained by support vector machines (SVM) with the RBF kernel. In all experiments we used LIBSVM [3]. We applied SVM to the images of the graphs in the binary feature space spanned by the frequent patterns generated. The parameters of SVM (i.e. C and γ) were fixed throughout all classification tasks for a given database to avoid overfitting by chance. For all frequent pattern generation methods, we chose a frequency threshold value of 5%. Prior experiments have shown that patterns of fairly low sizes achieve the overall best predictive performances. We therefore limit the size of patterns to at most 9 vertices.

5.1 Degree of Injectivity vs. Predictive Performance

We first report our results investigating the influence of the degree of injectivity in PIHom tree patterns on the predictive performance. To exclude possible side-effects caused by *different* pattern sets, for all datasets in our experiments we first fix a set S of tree patterns by selecting some random subset of the frequent trees generated w.r.t. subgraph isomorphism. Then, for all $H \in S$ we consider a PIHom tree pattern $H_k = (H, \mathcal{C}_k)$ that is maximally constrained w.r.t. tree-width k if $k = 1, \ldots, 4$; o/w it is fully constrained, i.e., $\mathcal{C}_\top = [V(H)]^2 \setminus E(H)$. Since H is a tree, $\mathcal{C}_1 = \emptyset$ and hence H_1 is the least element in the lattice

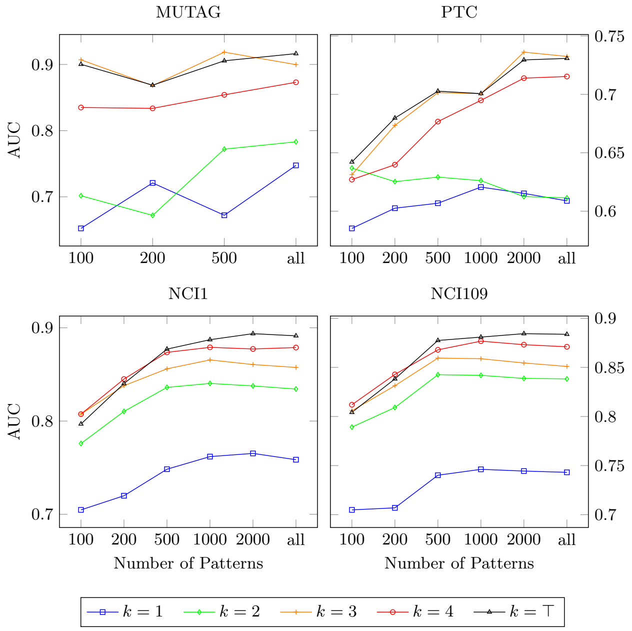

Fig. 3: Predictive performances in AUC for different degrees of injectivity governed by tree-width k. $k = \top$ corresponds to ordinary subgraph isomorphism.

$(\mathcal{L}_H, \preceq)$, corresponding to ordinary homomorphism from H. Analogously, $H_\top$ contains all possible injectivity constraints and is therefore the greatest element of $(\mathcal{L}_H, \preceq)$, corresponding to ordinary subgraph isomorphism from H. In this way we can simulate monotonically increasing degrees of injectivity in the pattern matching operator for a tree pattern H, leading from ordinary homomorphisms to subgraph isomorphisms. The degree of injectivity in the pattern matching operator is directly governed by k. For all $k \in \{1, \ldots, 4, \top\}$ we take the feature set $S_k = \{H_k : H \in S\}$ and embed all target graphs into the binary feature space spanned by S_k in the usual manner.

Figure 3 shows the predictive performances achieved for different degrees of injectivity defined by k. All four datasets indicate a significant difference between employing homomorphism ($k = 1$) and subgraph isomorphism ($k = \top$) as the pattern matching operators. While the results for MUTAG and PTC are fairly volatile due to their small size, with $k = 3$ and subgraph isomorphism yielding the best results, the datasets NCI1 and NCI109 clearly show a direct correlation between the degree of injectivity and predictive performance: Increasing values of k in almost every case yield an improvement in predictive performance. Notice that the gap between $k = 1$ (i.e., ordinary homomorphism) and $k = 2$ is already substantial for the datasets NCI1 and NCI109. For MUTAG and PTC it seems that $k = 3$ is necessary to significantly improve the predictive performance over

Dataset	Frequent Patterns	$\lvert V\rvert = 5$	$\lvert V\rvert = 6$	$\lvert V\rvert = 7$	$\lvert V\rvert = 8$	$\lvert V\rvert = 9$
MUTAG	s.g.i. graphs	73.18 ± 16.46	87.94 ± 10.70	88.29 ± 7.78	90.90 ± 6.83	91.99 ± 6.65
	s.g.i. trees	73.18 ± 16.46	84.07 ± 11.71	86.87 ± 6.23	89.84 ± 6.34	91.63 ± 5.89
	p.i.h. trees ($k = 4$)	73.18 ± 16.46	83.39 ± 11.20	87.10 ± 5.55	89.32 ± 7.66	90.49 ± 6.98
	p.i.h. trees ($k = 3$)	75.43 ± 13.69	83.51 ± 11.11	88.17 ± 6.38	89.15 ± 7.88	90.21 ± 8.36
	p.i.h. trees ($k = 2$)	61.44 ± 15.28	71.72 ± 10.08	76.03 ± 9.03	76.35 ± 12.28	76.92 ± 14.90
PTC	s.g.i. graphs	65.11 ± 8.32	66.82 ± 7.65	71.81 ± 8.27	73.69 ± 9.13	73.07 ± 9.34
	s.g.i. trees	65.11 ± 8.32	66.82 ± 7.65	71.91 ± 8.18	73.76 ± 9.09	73.08 ± 9.39
	p.i.h. trees ($k = 4$)	65.11 ± 8.32	69.26 ± 8.25	72.94 ± 8.55	72.54 ± 8.53	72.38 ± 8.02
	p.i.h. trees ($k = 3$)	64.30 ± 7.41	68.12 ± 8.03	71.87 ± 8.64	73.37 ± 9.01	73.24 ± 8.55
	p.i.h. trees ($k = 2$)	65.26 ± 7.80	66.79 ± 7.75	66.75 ± 7.31	67.53 ± 7.26	67.84 ± 6.53
NCI1	s.g.i. graphs	80.59 ± 2.28	85.97 ± 1.89	88.32 ± 1.11	89.20 ± 0.96	89.33 ± 1.13
	s.g.i. trees	80.59 ± 2.28	85.65 ± 1.83	88.08 ± 1.06	88.93 ± 1.03	89.14 ± 1.19
	p.i.h. trees ($k = 4$)	80.59 ± 2.28	86.41 ± 1.53	87.86 ± 1.08	88.35 ± 1.17	88.30 ± 1.29
	p.i.h. trees ($k = 3$)	81.74 ± 2.10	86.26 ± 1.42	87.85 ± 1.08	88.65 ± 1.23	88.77 ± 1.32
	p.i.h. trees ($k = 2$)	80.28 ± 2.51	84.45 ± 1.97	86.37 ± 1.58	87.27 ± 1.52	87.68 ± 1.35
NCI109	s.g.i. graphs	81.09 ± 1.89	85.82 ± 1.77	88.27 ± 1.14	88.53 ± 1.60	88.55 ± 1.77
	s.g.i. trees	81.09 ± 1.89	85.75 ± 1.78	88.01 ± 1.20	88.29 ± 1.54	88.37 ± 1.72
	p.i.h. trees ($k = 4$)	81.09 ± 1.89	86.38 ± 1.50	87.63 ± 1.73	87.64 ± 1.87	87.54 ± 2.03
	p.i.h. trees ($k = 3$)	81.24 ± 0.74	86.21 ± 1.46	87.60 ± 1.22	87.72 ± 1.89	87.77 ± 2.02
	p.i.h. trees ($k = 2$)	80.15 ± 2.07	84.43 ± 1.47	86.42 ± 1.01	86.79 ± 1.13	86.72 ± 1.66

Table 1: Prediction measures stated as AUC values in % for different tree-width choices k in contrast to freq. subgraphs and subtrees (s.g.i.: subgraph isomorphism, p.i.h.: partially injective homomorphism). Values are calculated by a SVM on feature vectors w.r.t. pattern graphs of size at most $|V|$.

homomorphism. In summary, a fairly low degree of injectivity already suffices to considerably outperform the predictive power of ordinary homomorphism.

5.2 Predictive Performance

We also evaluate the predictive performance of the frequent maximally constrained PIHom tree patterns generated by our algorithm. We compare their predictive power to that achieved by the set of (ordinary) frequent subtrees, as well as frequent *subgraphs* w.r.t. subgraph isomorphism. The binary feature vectors for the target graphs are calculated in the same way as described in the previous section for the patterns generated by our algorithm, for ordinary frequent subtrees, and for ordinary frequent subgraphs.

Table 1 shows the predictive performance of our approach for different pattern sizes (cf. the last five columns). The full sets of frequent subgraphs perform slightly better than its subset formed by frequent subtrees. Interestingly, already for tree-width $k = 3$, the absolute difference (in AUC) to frequent subgraphs (cf. row "s.g.i. graphs") and to ordinary frequent subtrees (cf. row "s.g.i. trees") is marginal for all cases, but one (MUTAG with $|V| = 5$). In all cases, the differences are statistically insignificant (for $p = 0.05$). Hence, our approach offers an attractive trade-off between runtime (depending on k) and predictive power.

	MUTAG	PTC	NCI1	NCI109	Erdős-Rényi random graphs							
					$\lvert\Sigma\rvert = 1$				$\lvert\Sigma\rvert = 2$			
					$q = 1.0$	$q = 1.5$	$q = 2.0$	$q = 3.0$	$q = 1.0$	$q = 1.5$	$q = 2.0$	$q = 3.0$
GASTON (EL)	0.1	0.2	2.6	2.7	2.8	54.6	*mem err*	*mem err*	1.8	20.1	1168.9	56464.6
GASTON (RE)	0.3	1.0	8.5	8.6	5.0	39.5	1163.0	31061.4	5.7	44.9	1120.2	24778.0
FSG	0.7	4.1	30.2	29.9	194.2	10584.9	10888.4	10852.5	19.9	82.8	2375.7	58816.2
PIH Miner ($k = 3$)	0.8	3.4	27.3	24.1	0.3	1.3	2.8	8.4	5.3	16.3	149.9	622.8

Table 2: Runtimes (in sec.) of our algorithm in comparison to GASTON and FSG on molecular and artificial datasets. Σ denotes the set of vertex labels. The cases in which GASTON ran out of the necessary memory are marked by "*mem err*". The maximum size of patterns to be found was set to 10 and the minimum support was set to 5% in all cases. Note that FSG does not restrict its search to trees but considers general graphs, in contrast to all other algorithms.

5.3 Runtimes

We measured the runtimes of our algorithm and compared them to those achieved with the state-of-the-art graph miners GASTON [11] (using the graph counting methods *embedding lists* (EL) and *recomputed embeddings* (RE)) and FSG [5] on real-world and artificial datasets. While molecules, having roughly as many edges as vertices (i.e., having $q \approx 1.0$), q is up to 3.0 in the artificial datasets.

Table 2 shows the running times for each algorithm and dataset. As expected, GASTON and FSG perform very well on molecular graphs, compared to our algorithm. However, for slightly more complex structures (i.e. $q \geq 2.0$) the two traditional graph miners become quickly infeasible for the correlation between the number of embeddings and runtime. Our approach (referred to as *PIH Miner* in Table 2) clearly outperforms FSG and GASTON on both artificial datasets for $q \geq 2.0$.

6 Concluding Remarks

To bridge the gap between homomorphisms (ILP) and subgraph isomorphisms (graph mining), we proposed partially injective homomorphisms, a new kind of *dynamic* pattern matching operator. We considered the efficiently enumerable fragments of frequent maximally constrained PIHom tree patterns w.r.t. bounded tree-width and showed on benchmark molecular graph datasets that their predictive performance is close to that of ordinary frequent subtrees (and hence, to that of subgraphs as well). Since our algorithm does not assume any structural properties on the input graphs, it is effectively applicable also to such transaction graphs where most state-of-the-art pattern mining algorithms become infeasible.

Our approach raises several questions. Perhaps the most interesting one is the extension to more general relational vocabularies, bridging another gap between graph mining and ILP in the richness of the relational vocabularies. To formulate this generalization, we note that any PIHom$(H, G, \mathcal{C})$ problem can polynomially be reduced to θ-subsumption between DATALOG goal clauses (or equivalently, Boolean conjunctive queries) as follows: Using a relational vocabulary Σ consisting of binary and unary predicate symbols only, represent $(H, \mathcal{C})$ as a DATALOG goal clause Q_H with a body composed of a set of literals for H and another set of literals of a distinguished binary predicate for the injectivity constraints in $\mathcal{C}$. Each vertex of H is represented by a unique variable in Q_H. For a target graph G, represent the PIHom pattern $(G, [V(G)]^2)$ in a similar way by a clause Q_G over the same Σ. It holds that Q_H θ-subsumes Q_G iff $H \underset{\mathcal{C}}{\rightarrow} G$. From this reduction and our empirical results it is immediate that the predictive performance of DATALOG goal clauses as patterns can also be improved by adding injectivity constraints (i.e., literals of a distinguished binary predicate) to their bodies. Considering our approach developed for graphs from the above logical viewpoint, it would be interesting to generalize it to relational vocabularies containing predicate symbols of *arbitrary* arities by considering such fragments of Boolean conjunctive queries for which θ-subsumption is in P (cf. [7]).

References

1. H. L. Bodlaender. A linear time algorithm for finding tree-decompositions of small treewidth. In *Proc. of the 21st Annual ACM Symposium on Theory of Computing*, (STOC), pp. 226–234. ACM, 1993.
2. A. K. Chandra and P. M. Merlin. Optimal implementation of conjunctive queries in relational data bases. In *Proc. of the 9th Annual ACM Symposium on Theory of Computing*, (STOC), pp. 77–90. ACM Press, 1977.
3. C.-C. Chang and C.-J. Lin. Libsvm: a library for support vector machines. *ACM Transactions on Intelligent Systems and Technology*, 2(3):27, 2011.
4. V. Dalmau, P. G. Kolaitis, and M. Y. Vardi. Constraint satisfaction, bounded treewidth, and finite-variable logics. In *Proc. of the 8th Int. Conf. on Principles and Practice of Constraint Programming*, pp. 310–326, 2002.
5. M. Deshpande, M. Kuramochi, N. Wale, , and G. Karypis. Frequent substructure-based approaches for classifying chemical compounds. *Knowledge and Data Engineering*, 17(8):1036–1050, 2005.
6. R. Diestel. *Graph theory.* Springer, Berlin, 2000.
7. G. Gottlob, N. Leone, and F. Scarcello. Hypertree decompositions and tractable queries. *J. Computer and System Sciences*, 64(3):579–627, 2002.
8. T. Horváth and G. Turán. Learning logic programs with structured background knowledge. *Artificial Intelligence*, 128(1-2):31–97, 2001.
9. T. Horváth and J. Ramon. Efficient frequent connected subgraph mining in graphs of bounded tree-width. *Theoretical Computer Science*, 411(31):2784 – 2797, 2010.
10. S. Nienhuys-Cheng and R. de Wolf, editors. *Foundations of Inductive Logic Programming*, vol. 1228 of *Lecture Notes in Computer Science.* Springer, Berlin, 1997.
11. S. Nijssen and J. N. Kok. The gaston tool for frequent subgraph mining. *Electronic Notes in Theoretical Computer Science*, 127(1):77–87, 2005.
12. N. Robertson and P. D. Seymour. Graph minors. II. algorithmic aspects of tree-width. *J. Algorithms*, 7(3):309–322, 1986.
13. D. J. Rose. On simple characterizations of k-trees. *Discrete Mathematics*, 7(3-4):317–322, 1974.
14. P. Welke, T. Horváth, and S. Wrobel. Probabilistic frequent subtrees for efficient graph classification and retrieval. *Machine Learning*, 2017. (to appear).