Mining Tree Patterns with Partially Injective Homomorphisms

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

[1]Dept. of Computer Science, University of Bonn, Bonn, Germany
[2]Fraunhofer IAIS, Schloss Birlinghoven, Sankt Augustin, Germany

Abstract. One of the main differences between ILP and graph mining is that while pattern matching in ILP is mainly defined by homomorphism (subsumption), it is the subgraph isomorphism in graph mining. Using that subgraph isomorphisms are injective homomorphisms, we bridge the gap between the two pattern matching operators with *partially injective homomorphisms*, which are homomorphisms requiring the injectivity constraint only for subsets of the vertex pairs in the pattern. Utilizing positive complexity results on the efficiency of homomorphisms from bounded tree-width graphs, we sketch the main ingredients of an algorithm mining frequent trees with respect to partially injective homomorphisms. Our experimental results on benchmark datasets show that the predictive performance of the patterns obtained is comparable to that of ordinary frequent patterns. Thus, our approach provides a trade-off between predictive power and tractability, while bridging the gap between ILP and graph mining.

1 Introduction

Despite the fact that graphs can be considered as relational structures, *inductive logic programming* (ILP) and *graph mining* establish themselves as two independent research fields. One of the main reasons for this separation lies in the relative simplicity of the vocabularies corresponding to graphs as relational structures: They contain only unary and binary predicate symbols representing vertex and edge labels, respectively. Another important difference is that while the pattern matching operator in ILP is defined by *first-order implication*, it is mainly the *subgraph isomorphism* in graph mining. Since implication between first-order clauses is undecidable [10], a common approach in ILP is to use *subsumption* instead of implication (cf. [11]). On the one hand, subsumption between first-order clauses is decidable. On the other hand, it is a sound, but incomplete pattern matching operator w.r.t. implication; the conditions for their equivalence are investigated in [5]. For function-free clauses, subsumptions are actually homomorphisms between relational structures [8]. Thus, from the point of view of the pattern matching operators, ILP is (mainly) concerned with *relational homomorphisms*, while graph mining with *subgraph isomorphisms*.

Our goal is to propose a binary feature space spanned by frequent patterns for predictive tasks. Using frequent patterns w.r.t. homomorphism results, on

the one hand, in a loss of predictive performance compared to subgraph isomorphism, due to the lack of injectivity. On the other hand, however, homomorphism can be decided in polynomial time for a broad class of pattern graphs for which subgraph isomorphism is NP-complete. 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 such graph classes. The main idea for this work is that the gap between homomorphisms and subgraph isomorphisms can be bridged by requiring injectivity only for subsets of the pattern graph's vertex pairs, i.e., by employing *partially injective homomorphisms* for pattern matching. Such partially injective homomorphisms can polynomially be reduced to ordinary homomorphisms by adding new edges to the pattern and text graphs, corresponding to the injectivity constraints, and coloring the original edges in both graphs by blue and all new edges by red.

By applying partially injective homomorphisms, we can resolve the complexity limitation mentioned above for a broad class of patterns. We consider the particular case that the patterns are restricted to trees (i.e., the blue edges in the pattern form a tree) and the red edges (i.e., the injectivity constraints) of the pattern, together with the blue ones, form a graph of bounded *tree-width* [12]. From graphs of bounded tree-width, homomorphism can be decided in polynomial time [2]. To put our approach into context, we note that ordinary subgraph isomorphism from trees is NP-complete even for very simple text graphs [1].

Using the idea above, we propose an algorithm mining *frequent trees* w.r.t. partially injective homomorphisms. The rationale behind the choice of tree patterns is that experimental results on benchmark graph datasets clearly indicate that the predictive performance of frequent trees compares well with that of frequent connected subgraphs [16]. Utilizing that *k-trees* have an algorithmic definition (see, e.g., [13]) and that a graph has tree-width at most k iff it is a *partial k-tree* (see, e.g., [15]), we arrive at a very natural refinement operator.

We have experimentally evaluated our approach on several molecular benchmark datasets. The empirical results show that already for tree-width 3, the predictive performance of frequent subtrees generated w.r.t. partially injective homomorphisms is close to that of ordinary frequent subtrees. Thus, frequent trees w.r.t. partially injective homomorphisms are good candidates to overcome the computational intractability of mining frequent trees in *arbitrary* graphs [7].

2 Partially Injective Homomorphisms

In this section we sketch our algorithm mining frequent tree patterns w.r.t. partially injective homomorphisms. We first recall some notions from graph theory (see, e.g., [4]) and fix the notation. For a set S, $[S]^2$ denotes the family of 2-subsets of S (i.e., $[S]^2 = \{X \subseteq S : |X| = 2\}$). The set $\{u, v\} \in [S]^2$ is designated by uv. The set of vertices (resp. edges) of a graph G is denoted by $V(G)$ (resp. $E(G)$). We consider *undirected* graphs only (i.e., $E(G) \subseteq [V(G)]^2$). A *homomorphism* from a graph H (the *pattern*) into a graph G (the *text*) is a

mapping $\varphi : V(H) \to V(G)$ that preserves the edges (i.e., $uv \in E(H)$ implies $\varphi(u)\varphi(v) \in E(G)$ for all $u, v \in V(H)$). When H and G are labeled graphs, φ is required to preserve all vertex and edge labels of H as well. A *subgraph isomorphism* from H to G is an *injective* homomorphism from H to G.

Deciding whether there exists a homomorphism or a subgraph isomorphism from H to G is, in general, NP-complete. On the one hand, the injectivity condition results in a restriction of the pattern classes from which homomorphism can be decided in polynomial time. For example, while homomorphism from graphs of bounded tree-width can be decided in polynomial time [2], subgraph isomorphism is NP-complete even for the very simple case that the pattern graph is a path (which is of tree-width 1). On the other hand, a simple argument on unlabeled graphs clearly shows the superiority of the discriminative performance of frequent patterns generated w.r.t. subgraph isomorphism to those generated with homomorphism. As a trade-off between complexity and predictive performance, we consider such graph morphisms that are between homomorphisms and subgraph isomorphisms. These graph morphisms can naturally be defined by requiring the injectivity constraint only for subsets of the set of vertex pairs in the pattern graph. More precisely, let $\mathcal{C} \subseteq [V(H)]^2 \setminus E(H)$. Then a *partially injective homomorphism* from H to G satisfying the injectivity constraints in $\mathcal{C}$ is a homomorphism φ from H to G such that $\varphi(u) \neq \varphi(v)$ for all $uv \in \mathcal{C}$. We denote by $H \underset{\mathcal{C}}{\to} G$ if such a partially injective homomorphism exists and will define pattern matching by the Partially Injective Homomorphism problem (PIHom$(H, G, \mathcal{C})$): *Given* H, G, and $\mathcal{C}$ as above, *decide* whether $H \underset{\mathcal{C}}{\to} G$.

When H is a tree, there are $2^{\binom{n-1}{2}}$ PIHom problems from H to G, where $n = |V(H)|$. Furthermore, there is a natural partial order $\preceq$ on the set $\mathcal{L}$ of all PIHom problems from H to G: For all $\mathcal{C}_1, \mathcal{C}_2 \subseteq [V(H)]^2 \setminus E(H)$, PIHom$(H, G, \mathcal{C}_1) \preceq$ PIHom$(H, G, \mathcal{C}_2)$ iff $\mathcal{C}_1 \subseteq \mathcal{C}_2$. Clearly, $(\mathcal{L}, \preceq)$ is a lattice and its bottom (i.e., PIHom$(H, G, \emptyset)$) and top (i.e., PIHom$(H, G, [V(H)]^2 \setminus E(H))$) elements correspond to the ordinary homomorphism and subgraph isomorphism problems from H to G, respectively. It follows that $(\mathcal{L}, \preceq)$ depends only on H and is closed downwards, i.e., $H \underset{\mathcal{C}_1}{\longrightarrow} G$ and $\mathcal{C}_2 \subseteq \mathcal{C}_1$ implies $H \underset{\mathcal{C}_2}{\longrightarrow} G$.

Another important property of PIHom problems is that they can *polynomially* be reduced to ordinary homomorphism problems. Indeed, for H, G, and $\mathcal{C}$ as above, let H' be the graph with $V(H') = V(H)$ and $E(H') = E(H) \cup \mathcal{C}$, and color the original edges from $E(H)$ by *blue* and those corresponding to $\mathcal{C}$ by *red*. Furthermore, color all edges of G by blue and connect all pairs of its vertices by a red edge. Let G' be the graph obtained. It holds that $H \underset{\mathcal{C}}{\to} G$ iff $H' \underset{\emptyset}{\to} G'$. (Recall that homomorphisms preserve all edge colors.)

As already noted, we are interested in *maximally* constrained PIHom problems that lie in P. For this purpose, we utilize the concept of *tree-width* (see, e.g., [4]), as homomorphisms from bounded tree-width graphs can be decided in polynomial time [2]. More precisely, we restrict the pattern class to trees and consider for a particular tree H a *maximal* subset $\mathcal{C}$ of $[V(H)]^2 \setminus E(H)$ such that the graph with vertex set $V(H)$ and edge set $E(H) \cup \mathcal{C}$ has tree-width k, where

k is some constant. Notice that this is equivalent to considering the *positive border* [9] of the lattice $(\mathcal{L}, \preceq)$ formed by the set of PIHOM problems from H, where the interestingness predicate is defined by tree-width.

Our mining algorithm generates frequent *trees* w.r.t. partially injective homomorphisms using levelwise search [9]. Its new feature, compared to all traditional frequent pattern mining algorithms, is that the pattern matching operator is not static (i.e. fixed in advance), but *dynamic*, depending on the underlying graph pattern. Since the number of PIHOM problems for a pattern H is exponential in H's size, we keep only a subset of the output patterns for each level. Due to space limitations, we omit these technical details and focus rather on the *refinement operator*. It is based on the algorithmic definition of *k-trees* [13]: (i) A clique of $k+1$ vertices is a k-tree and (ii) given a k-tree T_k with n vertices, a k-tree with $n+1$ vertices is obtained from T_k by adding a new vertex v to T_k and connecting v to all vertices of a k-clique of T_k. The tree-width of a k-tree is always k and all k-trees are maximal graphs in the sense that adding any new edge to them would result in graphs of tree-width $k+1$. The definition above gives rise to the following natural refinement operator (for simplicity, we regard the PIHOM problems as graphs with blue and red edges as defined in their reduction to ordinary homomorphism problems): A pattern P' with n vertices is in the refinements of a pattern P with $n-1$ vertices if P' can be obtained from P by (i) introducing a new vertex v, (ii) connecting v to all vertices of P if $n \leq k$; o/w connecting v with all vertices of a k-clique in P, and (iii) coloring one of the new edges with blue (i.e., it will be an edge of the tree) and all other new edges by red (i.e., they all will define injectivity constraints).

3 Experiments

We empirically evaluated our approach by performing classification tasks on the well-established molecular datasets NCI1, NCI109, MUTAG, and PTC (see, e.g., [14]). We contrast the performance of our approach with that of frequent subgraphs and subtrees obtained by FSG [3]. The prediction performance was measured in terms of the area under the ROC-curve (AUC), using support vector machines (SVM) with the RBF kernel on points representing the transaction graphs in the binary feature space spanned by the frequent patterns generated. The AUC values were averaged over a 3-fold cross-validation. The parameters of the SVM were fixed throughout all classification tasks for a given database to avoid overfitting by chance.

The size of the tree patterns was limited to 9 (i.e., to 8 blue edges)[1] and the tree-width of the graphs corresponding to the PIHOM problems was bounded by 4. Table 1 shows that the predictive performance of our approach compares well with that of the frequent subtree based predictors. For tree-width 3, the absolute difference is marginal for all cases but one (MUTAG with $|E| = 4$). In 6 out of the

[1] Patterns with more than 8 edges did not significantly increase the prediction performance. In fact, in most cases the AUC-values even slightly decreased for graphs with more than 8 edges

Dataset	Frequent Patterns	$\|E\| = 4$	$\|E\| = 5$	$\|E\| = 6$	$\|E\| = 7$	$\|E\| = 8$
NCI1	s.g.i. graphs	79.30	84.36	86.48	87.17	87.18
	s.g.i. trees	79.30	84.10	86.16	86.83	86.92
	p.i.h. trees ($k = 2$)	78.94	83.18	85.02	85.41	86.07
	p.i.h. trees ($k = 3$)	80.51	84.53	85.83	86.64	86.68
	p.i.h. trees ($k = 4$)	79.30	84.66	86.02	86.39	86.11
NCI109	s.g.i. graphs	80.04	84.64	86.50	87.04	87.29
	s.g.i. trees	80.04	84.51	86.39	86.79	87.09
	p.i.h. trees ($k = 2$)	79.17	83.42	85.27	85.31	85.41
	p.i.h. trees ($k = 3$)	81.21	85.08	85.97	86.34	86.63
	p.i.h. trees ($k = 4$)	80.04	85.18	85.89	86.27	86.28
PTC	s.g.i. graphs	63.46	66.42	70.73	73.98	73.11
	s.g.i. trees	63.46	66.42	70.77	74.03	73.18
	p.i.h. trees ($k = 2$)	64.12	65.54	67.45	69.60	70.66
	p.i.h. trees ($k = 3$)	63.39	67.15	71.31	73.02	72.93
	p.i.h. trees ($k = 4$)	63.50	67.67	72.56	72.68	72.14
MUTAG	s.g.i. graphs	72.71	85.28	89.27	89.89	90.29
	s.g.i. trees	72.71	83.05	87.23	88.50	89.49
	p.i.h. trees ($k = 2$)	59.02	65.60	77.99	69.92	74.69
	p.i.h. trees ($k = 3$)	63.03	82.95	86.88	88.41	89.05
	p.i.h. trees ($k = 4$)	71.42	82.44	85.31	88.53	88.93

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 with at most $|E|$ edges. All mining processes applied a frequency threshold of t=5%.

20 prediction tasks, our approach even outperforms frequent subtrees. Frequent subgraphs perform slightly better than frequent subtrees. Still, we are able to top this method in 6 out of 20 cases when choosing $k = 3$, which is surprising considering that our approach cannot contain cyclic patterns. We also note that the predictive performance with $k = 2$ is consistently outperformed by higher tree-width values. However, $k = 4$ does not improve the performance over $k = 3$. This suggests that our embedding method offers an attractive trade-off between the computational complexity and degree of injectivity of the pattern embedding operator for classification tasks.

4 Remark

Our approach can naturally be generalized to first-order clauses in the following way: Let C and D be function-free first-order clauses (i.e., only variables and constants are allowed). A *substructure isomorphism* from C to D is a subsumption $\theta = \{x_1/t_1, \ldots, x_k/t_k\}$ such that $C\theta \subseteq D$ and the t_is are pairwise different (i.e., θ is injective). Let $\mathrm{Var}(C)$ denote the set of variables in C and let $\mathcal{C} \subseteq [\mathrm{Var}(C)]^2$. A *partial substructure isomorphism* from C to D satisfying the injectivity constraints in $\mathcal{C}$ is a substitution θ such that $C\theta \subseteq D$ and for all $xy \in \mathcal{C}$, $x\theta \neq y\theta$. The problem of deciding whether such a partial substructure isomorphism from

C into D exists can be reduced to the ordinary subsumption problem in the following way: Introduce a new binary predicate, say NEQ, and (i) add a literal $\mathrm{NEQ}(x, y)$ to C for all $xy \in \mathcal{C}$ and (ii) the literals $\mathrm{NEQ}(t_i, t_j), \mathrm{NEQ}(t_j, t_i)$ for all different terms t_i, t_j in D. Denote the clauses obtained by C' and D'. It is easy to check that there is a partial substructure isomorphism from C to D satisfying the injectivity constraints in $\mathcal{C}$ if and only if C subsumes D. Similarly to the case of bounded tree-width graphs, one can consider different *tractable* fragments of the subsumption problem between first-order clauses, define the corresponding lattice, and calculate (subsets of) its positive border. Some interesting examples of tractable fragments of conjunctive queries are presented, e.g., in [6].

References

1. T. Akutsu. A polynomial time algorithm for finding a largest common subgraph of almost trees of bounded degree. *IEICE Trans. on Fundamentals of Electronics, Communications and Computer Sciences*, 76(9):1488–1493, 1993.
2. 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*, pages 310–326, 2002.
3. 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.
4. R. Diestel. *Graph theory*. Springer-Verlag, New York, 3 edition, 2005.
5. G. Gottlob. Subsumption and implication. *Information Processing Letters*, 24(2):109–111, 1987.
6. G. Gottlob, N. Leone, and F. Scarcello. Hypertree decompositions and tractable queries. *J. Computer and System Sciences*, 64(3):579–627, 2002.
7. T. Horváth, B. Bringmann, and L. De Raedt. Frequent hypergraph mining. In *Inductive Logic Programming*, pages 244–259. Springer, Berlin, Heidelberg, 2007.
8. T. Horváth and G. Turán. Learning logic programs with structured background knowledge. *Artificial Intelligence*, 128(1-2):31–97, 2001.
9. H. Mannila and H. Toivonen. Levelwise search and borders of theories in knowledge discovery. *Data Mining and Knowledge Discovery*, 1(3):241–258, 1997.
10. J. Marcinkowski and L. Pacholski. Undecidability of the Hornclause implication problem. In *Proc. of the 33rd Annual IEEE Symposium on Foundations of Computer Science (FOCS-92)*, pages 354–362, 1992.
11. G. Plotkin. A note on inductive generalization. In *Machine Intelligence*, volume 5, pages 153–163. Edinburgh University Press, 1970.
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. N. Shervashidze, P. Schweitzer, J. van Leeuwen, K. Mehlhorn, , and K. M. Borgwardt. Weisfeiler-lehman graph kernels. *The J. of Machine Learning Research*, 12:2539–2561, 2011.
15. J. van Leeuwen. Graph algorithms. In *Handbook of Theoretical Computer Science, Volume A: Algorithms and Complexity (A)*, pages 525–631. 1990.
16. P. Welke, T. Horváth, and S. Wrobel. Probabilistic subtree kernels. In *New Frontiers in Mining Complex Patterns*. Springer, Berlin, Heidelberg, 2015.