A Method for Mining Discriminative Graph Patterns

Andrea Fuksová
Faculty of Electrical Engineering
Czech Technical University in Prague

Ondřej Kuželka
Dept. of Computer Science
KU Leuven

Andrea Szabóová
Faculty of Electrical Engineering
Czech Technical University in Prague

1 Introduction

In this paper, we are interested in methods for finding a small number of graphs which together cover (i.e. are homomorphic to) as many positive examples and as few negative examples as possible. For instance, if we have a dataset of protein structures described as labelled graphs, consisting of proteins possessing a specific function and proteins not having this function, we want to find a set of graphs which represent common substructures (under homomorphism) found frequently in the proteins with the function and not in the other group of proteins. Here, we present an efficient method based on graph products and properties of the k-consistency algorithm from CSP [1].

The work presented in this paper brings three main contributions. First, it translates a recent theoretically-justified method [4] from inductive logic programming into a graph-mining setting which is more accessible for most people from bioinformatics. Second, the implementation of the method presented here is more efficient (using some tricks which are beyond the scope of this paper). Third, the efficient implementation allows us to demonstrate relevancy of the method for computational biology by applying the method to a biological problem involving relatively large dense graphs with approximately 1000 edges per graph (which is more than the sizes of graphs appearing in typical molecular graph-mining datasets such as the well-known NCI datasets).

2 Preliminaries

Patterns and examples are assumed to be labelled directed graphs. In the rest of the paper, we will assume the graphs are directed and we will omit the word *directed* and speak e.g. only of *graphs* instead of *directed graphs*. The set of vertices of a graph G is denoted by $V(G)$, the set of edges of G is denoted by $E(G)$. The label of a vertex $v \in V(G)$ is denoted by $\lambda(v)$ and the label of an edge $e = (v, w) \in E(G)$ is denoted by $\lambda(e)$. A *homomorphism* from a graph G to a graph H is a mapping $\phi\colon V(G) \to V(H)$ of vertices of the graph G to the vertices of the graph H such that: (i) if $(v, w) \in E(G)$ then $(\phi(v), \phi(w)) \in E(H)$ (ii) $\lambda(v) = \lambda(\phi(v))$ for all $v \in V(G)$ and (iii) $\lambda(e) = \lambda(\phi(e))$ for all $e = (v, w) \in E(G)$. In other words, a homomorphism is a mapping which maps adjacent vertices to adjacent vertices and preserves labels of vertices and edges. If there is a homomorphism from G to H we denote this by $G \to H$. Two graphs G and H are said to be *homomorphically equivalent* if $G \to H$ and $H \to G$. If $\widehat{G}$ is a minimal graph homomorphically equivalent to G then $\widehat{G}$ is called a *core* of G. Deciding whether there is a homomorphism from a graph G to a graph H is an NP-complete problem. Finding cores of graphs is co-NP-complete. Tractable instances of these problems can be isolated by looking at *width of tree-decompositions* of the graphs, i.e. at so-called *treewidth*.

A tree decomposition of a graph $G = (V, E)$ is a tree T with nodes labelled by sets of vertices such that: (i) For every vertex $v \in V(G)$, there is a node of T with a label containing v, (ii) For every edge $(v, w) \in E(G)$, there is a node of T with a label containing v, w and (iii) for every $v \in V(G)$, the subgraph of T obtained by removing all its nodes not containing v (along with the associated edges) remains connected. The width of a tree decomposition T is the maximum cardinality of a label in T minus 1. The treewidth of a graph G is the smallest number k such that G has a tree decomposition of width k.

If a graph G has treewidth at most $k \in N$, the problem of deciding whether G is homomorphic to H can be solved in polynomial time, e.g. by so-called k-consistency algorithm [1]. If G has treewidth higher than k and if the k-consistency algorithm returns true then there may or may not be a homomorphism from G to H. However, if the k-consistency algorithm returns false then it is guaranteed that G is not homomorphic to H. If the k-consistency algorithm returns true for graphs G and H, we denote this by $G \rightarrow_k H$.

We will also need a binary operation on graphs known as graph-product. Given two graphs G and H, their product is a graph P which can be obtained as follows: (i) P has a vertex p_{vw} for each pair of vertices $v \in G$, $w \in H$ such that $\lambda(v) = \lambda(w)$, (ii) P has an edge $e = (p_{v,w}, p_{x,y})$ for a pair of vertices $p_{v,w}, p_{x,y} \in V(P)$ if and only if $e' = (v, x) \in E(G)$, $e'' = (w, y) \in E(H)$ and $\lambda(e') = \lambda(e'')$. The graph product of G and H will be denoted by $G \otimes H$. A useful property of graph product is that if $P = G \otimes H$ then $P \rightarrow G$ and $P \rightarrow H$. Moreover, for any other graph Q which is homomorphic to both G and H it holds $Q \rightarrow P$. The graph product of two graphs can be therefore seen as a kind of their least general generalization. Notice that the product of two graphs can have as many as $|V(G)| \cdot |V(H)|$ vertices. This number may often be reduced by computing a core of the product graph, but not always (consider e.g. the product of two directed cycles of lengths which are distinct prime numbers).

3 Bounded-Product-Homomorphism Method

In this section, we describe a method for finding discriminative graph patterns by using graph products for generalizing training examples. If we want to use graph products for generalizing training examples, we usually need to replace the computed graph products by their homomorphically equivalent cores. Otherwise, the size of the derived graphs would grow enormously. The problem is that computing cores of graph products is a computationally expensive operation since computing a core is a coNP-complete problem. Horváth et al. [2] resolved this problem by allowing only forest-structured learning examples. Forests have treewidth 1, therefore their cores can be computed in polynomial time. Moreover, a graph product of forests is always a forest and therefore if we limit ourselves to forest-structured learning examples then if we are deriving new patterns by graph products it is enough to use polynomial-time procedures for computing cores of forests and for checking homomorphisms between the derived patterns and the examples. The main disadvantage of the method of Horváth et al. is obviously that it requires the examples to be forests. In [4, 3], a relational learning approach was presented which generalizes the method of Horváth et al. in the following sense. It does not require learning examples to be forest-structured or to have bounded treewidth but still enables the reduction process to run in polynomial time. In the remainder of this section, we provide a brief description of this method, which was originally developed within inductive logic programming setting. Our description translates the method into a graph-mining setting which is, in our opinion, more accessible to a wider audience.

One of the key ideas of the presented pattern-search algorithm is to replace the exponential-time algorithm for computation of cores by the following polynomial-time (for a fixed k) algorithm called *vertex-elimination algorithm*.

Vertex-elimination algorithm ($\mathsf{vertexelim}_k(G)$)**:**

1. Given a graph G which should be reduced.
2. Select a vertex v s. t. if we remove v, it will hold $G \rightarrow_k G'$. If there is no such v, output G' and exit.
3. Set $G := G'$ and go to step 2.

The most important properties of the vertex-elimination algorithm regarding its use in combination with graph products are given by the next proposition.

Proposition 1 *Let $\mathcal{G} = \{G_1, G_2, \ldots, G_n\}$ be graphs and k be a natural number. Then the graph H obtained as $H = \mathsf{vertexelim}_k(G_n \otimes \mathsf{vertexelim}_k(G_{n-1} \otimes \mathsf{vertexelim}_k(G_{n-2} \otimes \ldots)))$ satisfies* *the following two conditions: (i) for all $G_i \in \mathcal{G}$, it holds $H \to G_i$ and (ii) if H' is a graph with treewidth at most k and $H' \to G_i$ for all $G_i \in \mathcal{G}$ then $H' \to H$ (i.e. H is more specific than any of the graphs with treewidth at most k which cover all graphs from $\mathcal{G}$).*

Proposition 1 ensures that if there is a graph H with treewidth at most k which is homomorphic to all examples from some subset $\mathcal{P} = \{G_1, G_2, \dots, G_n\}$ of positive examples and, for instance, to no negative examples then a graph H^* which is homomorphic to all positive examples from $\mathcal{P}$ and to no negative examples can be found as

$$H^* = \mathsf{vertexelim}_k(G_n \otimes \mathsf{vertexelim}_k(G_{n-1} \otimes \mathsf{vertexelim}_k(G_{n-2} \otimes \dots))).$$

The vertex-elimination algorithm together with graph-products can be used to search for graph patterns. However, we have not explained yet how to score graphs during the heuristic search. Computing homomorphisms is NP-complete for general graphs. Moreover, the graphs derived during the search do not typically have bounded treewidth. For the presented method, there is a trick by which we can avoid explicit computation of homomorphisms. We use the following procedure which obtains a graph pattern and a set of examples and returns another graph pattern together with the set of examples covered by the new graph pattern.

Coverage-computation algorithm ($\mathsf{vertexelim}_k(G)$)**:**

1. Given a graph H and a set of examples $\mathcal{E}$.
2. Set $\mathsf{Covered} := \{\}$.
3. Set $H' := H$.
4. Set $\mathsf{NewCovered} := \{G \in \mathcal{E} \setminus \mathsf{Covered} | H' \to_k G\}$.
5. Set $H' := \mathsf{vertexelim}_k(H' \otimes \mathsf{vertexelim}_k(G_{i_n} \otimes \mathsf{vertexelim}_k(G_{i_{n-1}} \otimes \dots)))$ where $G_{i_j} \in \mathsf{NewCovered}$.
6. Set $\mathsf{Covered} := \mathsf{Covered} \cup \mathsf{NewCovered}$.
7. If $\mathsf{NewCovered} \neq \emptyset$, go to step 4.
8. Return $\mathsf{Covered}$ and H'.

The most important property of this procedure is summarized in the next proposition.

Proposition 2 *Let a graph H and a set of examples $\mathcal{E}$ be input of the coverage-computation algorithm. Then the following holds for the output H' and $Covered$ of this algorithm:*

1. *$H' \to G$ for all $G \in \mathsf{Covered}$ and $H' \not\to G$ for all $G \in \mathcal{E} \setminus \mathsf{Covered}$.*
2. *Let $Covered'$ be a set of examples such that $\{G \in \mathcal{E} | H \to_k G\} \subsetneq Covered' \subsetneq Covered$ and H^* be a graph such that $H' \to H^* \to H$. If $H^* \to G$ for all $G \in Covered'$ and $H^* \not\to G$ for all $G \in \mathcal{E} \setminus Covered'$ then H^* has treewidth higher than k.*

The two procedures presented in this section can be combined into a best-first-search-like algorithm for searching for discriminative graph patterns which provides the following theoretical guarantee: When run with a k-consistency algorithm for a fixed k and confronted with a dataset of graphs containing positive and negative examples, it always finds a graph pattern with the score 'the number of covered positive minus the number of covered negative examples' at least as good as the score of the best graph pattern with treewidth at most k. This best-first-search-like algorithm never uses exponential-time procedures for computing homomorphism. It may still run for exponentially long but one can show that it is exponentially faster than an analogical algorithm based on graph products, homomorphisms and computations of cores.

4 Experiments and Discussion

We evaluated the method on the problem of predicting which protein domains are capable of binding hexoses and compared it to earlier approaches of Nassif et al. [5] and Santos et al. [6] which were both based on techniques from inductive logic programming. The dataset that we used contains 80 Hexose-binding protein domains and 80 non-Hexose-binding protein domains. We converted the spatial protein structures into graphs by adding one vertex labelled by the atom type (reflecting also its position in the amino acid) for every atom and one edge labelled by a discretized distance for

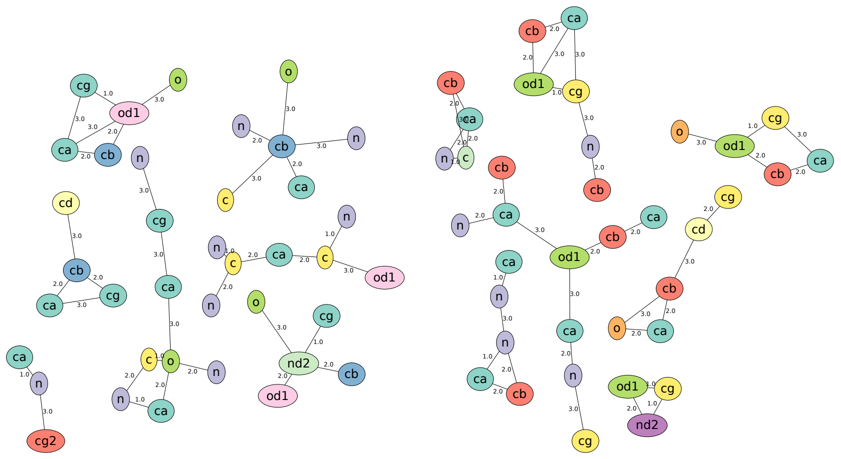
Figure 1: Two graph patterns found for the dataset of Hexose-binding protein domains.

all atoms in distances at most 4 Angstroms from each other. We then let our implementation of the algorithm (which uses several additional tricks not described in this paper due to lack of space) search for patterns which covered as many positive examples as possible while covering no negative examples. Examples of two structures found on the full dataset covering 39 positive examples and 0 negative examples. Interestingly, this is significantly higher number of positive examples covered by a pattern which does not cover any negative example than what was reported in [5, 6].

We also evaluated the efficiency of the method by 10-fold cross-validation and obtained accuracy 71.9 ± 5.3. This is slightly better than accuracy 67.5 ± 10.5 obtained in [5]. Note that while the difference between the cross-validated accuracy of our method and the method from [5] is not large, our method was able to find more complex patterns with significantly better performance on training data. It is likely that the superiority of our method would get more pronounced on bigger datasets where overfitting would not be such an issue. In [6], a slightly different task was considered in which the training and testing examples also contained coordinates of the binding center of the hexose therefore the results are not directly comparable to the results obtained by our method.

Acknowledgements: This work was supported by the Czech Science Foundation grant no. P202/12/2032. Part of this work was done while the second author was already with KU Leuven where he was supported by ERC Starting Grant 240186 ’MiGraNT: Mining Graphs and Networks, a Theory-based approach’.

References

[1] A. Atserias, A. Bulatov, and V. Dalmau. On the power of k-consistency. In *Proceedings of ICALP-2007*, pages 266–271, 2007.

[2] T. Horváth, G. Paass, F. Reichartz, and S. Wrobel. A logic-based approach to relation extraction from texts. In *Proceedings of the 19th international conference on Inductive logic programming*, ILP’09, pages 34–48, Berlin, Heidelberg, 2010. Springer-Verlag.

[3] O. Kuželka. *Fast Construction of Relational Features for Machine Learning*. PhD thesis, FEE, CTU in Prague, 2013.

[4] O. Kuželka, A. Szabóová, and F. Železný. Bounded least general generalization. In *ILP’12: Inductive Logic Programming*, 2013.

[5] H. Nassif, H. Al-Ali, S. Khuri, W. Keirouz, and D. Page. An Inductive Logic Programming approach to validate hexose biochemical knowledge. In *Proceedings of the 19th International Conference on ILP*, pages 149–165, Leuven, Belgium, 2009.

[6] J. Santos, H. Nassif, D. Page, S. Muggleton, and M. Sternberg. Automated identification of protein-ligand interaction features using inductive logic programming: a hexose binding case study. *BMC Bioinformatics*, 13(1):162, 2012.