Graphs, hypergraphs, and inductive logic programming

Hendrik Blockeel[1,2], Tijn Witsenburg[1], and Joost Kok[1]

[1] Leiden Institute of Advanced Computer Science, Universiteit Leiden
[2] Department of Computer Science, Katholieke Universiteit Leuven

Abstract. There are many connections between graph mining and inductive logic programming (ILP), or more generally relational learning. Up till now these connections have mostly been described or exploited in an informal way. A more careful and formal study of them may lead to new insights in the relationships between the different learning formalisms, and possibly new methods for solving certain problems that rely on translating problems or algorithms from one setting to another. In this paper we initiate an investigation into this.

1 Introduction

It has been observed many times that there are connections between graph mining and inductive logic programming (ILP), or more generally relational learning. So far, however, these connections have mainly been described in an informal way. Moreover, there are many possible ways of translating data / hypotheses from one representation framework to another.

A more careful study of such translations may give us additional insights in the relationship between different settings, and could directly lead to two ways of improving the current state of the art in graph mining. The first is "translation of algorithms": algorithmic techniques could be transferred from one framework to the other. The second is "translation of data": one might be able to solve problems by translating the data from one framework into the other, running a standard data mining algorithm for that framework, and then translating the results back to a proper solution for the original data. Several authors have already exploited similar connections (e.g., [5, 4]), but it seems that many more connections could be investigated.

In this paper we have a brief and non-exhaustive look at a number of connections between graphs, hypergraphs, and first order logic representations, and formulate a number of questions that this leads to. We hope that this discussion may be inspiring with respect to the development of new algorithms or unexpected application possibilities for existing algorithms.

2 Graphs, hypergraphs, ILP

A (hyper)graph is a structure (V, E) where V is a set of nodes and E is a set of edges. It is an **undirected graph** if $E \subseteq \binom{V}{2}$, with $\binom{V}{k} = \{S \subseteq V : |S| = k\}$; a **directed graph** if $E \subseteq V^2$; an **undirected hypergraph** if $E \subseteq 2^V$ contains (only nonempty) sets of nodes; a **directed hypergraph** (see, e.g., [3]) if E contains couples (X, Y) with X and Y non-empty and disjoint sets of nodes; and an **oriented hypergraph** if $E \subseteq \bigcup_{k=1}^{\infty} V^k$ contains tuples over V. Edges of a hypergraph are sometimes called hyperedges.

The set of nodes may be totally ordered, in which case we speak of an **ordered (hyper)graph**. Sometimes the set of edges is ordered in such a way that for each node, the edges in which it participates are totally ordered; we then speak of an **edge-ordered (hyper)graph**. A (hyper)graph is **node-labelled** if a label (from a given set of labels Σ) is associated with each node, and **edge-labelled** if a label is associated with each edge.

In the remainder of the text we focus on non-labelled, non-ordered structures, except where stated otherwise.

In inductive logic programming, the input data are sometimes represented as interpretations.[1] An interpretation is a set of ground facts; a ground fact is of the form $p(a_1, \dots, a_k)$ with the a_i denoting fixed objects (more precisely, elements of the Herbrand universe) and p a predicate symbol.

[1] For connections between learning from interpretations and other settings, see [1].

3 Translations

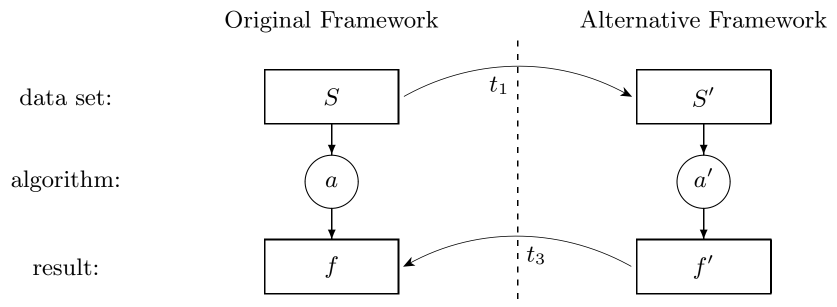

Fig. 1. Instead of using algorithm a to calculate result f it is also possible to use t_1 to translate data set S to S' in an alternative framework, run algorithm a' and use t_3 to translate result f' back to f.

A data mining algorithm can be seen as implementing a function $a : 2^X \to F$ that, given a training set $S \subseteq X$ returns an output f, which can be seen as a function from X to some result set R.[2] Let $A = F^{2^X}$ be the set of all such functions.

Assume we want to have a data mining algorithm a that given a data set S computes a result f. We could try to find a way to implement a directly. Alternatively, as visualized in Figure 1, we could translate the input data S to an alternative representational framework S', run an already existing algorithm a' on S', and translate the result f' back to the original f.

We can distinguish two separate problems. First, given a function a and a set of functions $\{a'_1, \ldots, a'_k\}$, we can look for translations t_1 and t_3 such that there exists an a'_i such that $t_3(a'_i(t_1(S))) = a(S)$. This amounts to reducing the original problem to a problem for which a solution is known.

Second, we can consider a framework (X, A, F) (e.g., hypergraph mining) and a second framework (X', A', F') (e.g, graph mining) and look for a triple of functions $t_1 : 2^X \to 2^{X'}$, $t_2 : A \to A'$ and $t_3 : F' \to F$ such that $\forall S \subseteq X, a \in A, f \in F$ and with $S' = t_1(S), a' = t_2(a), f = t_3(f')$ $: a(S) = f \Leftrightarrow a'(S') = f'$. This amounts to finding a translation that reduces all possible data mining problems in the first setting to equivalent problems in the second setting. We call this a universal translation.

Note that in this case it is crucial that the transformation t_1 preserves non-equivalence, that is, $\forall S_1, S_2 \subseteq X : (\exists a : a(S_1) \neq a(S_2) \Rightarrow t_1(S_1) \neq t_1(S_2))$. Indeed, if $t_1(S_1) = t_1(S_2))$ then $t_3(a'(t_1(S_1))) = t_3(a'(t_1(S_2)))$ which contradicts $t_3(a'(t_1(S_1))) = a(S_1) \neq a(S_2) = t_3(a'(t_1(S_1)))$.

Below we list a number of transformations of structures that can serve as the first component t_1 of the translation function t. We believe that all the proposed transformations have this property, and thus could be used in a universal translation, though for some this remains to be proven.

3.1 *V*-preserving

We first discuss transformations that preserve the set of nodes V. (E.g., a structure (V, E) is transformed into a structure (V', E') where $V' = V$.)

Undirected graphs can easily be represented as directed graphs by just including for each pair $\{x, y\}$ the couple (x, y) as well as (y, x). Similarly, undirected hypergraphs can be represented as oriented hypergraphs by including for each set $\{v_1, \ldots, v_k\}$ all its permutations $(p_i(v_1), p_i(v_2), \ldots, p_i(v_k))$, $i = 1 \ldots k!$.

All graphs are hypergraphs, and as such no transformation is necessary from graphs to hypergraphs. However, the extra expressiveness of hypergraphs can be exploited: a k-clique in the graph can be represented with one hyperedge instead of $\binom{k}{2}$ binary hyperedges. Reducing the amount of hyperedges by combining all maximal cliques can increase the comprehensibility, admittedly at

[2] This holds obviously for predictive modelling, but also clusters, itemset frequencies, ... can be seen as extensionally represented functions from X to some R.

high computational cost, since finding these hyperedges is NP-complete (equivalent to the maximum clique problem). But this also implies that problems that are NP-complete on the graph representation are not necessarily so on the hypergraph representation.

Hypergraphs cannot be transformed into graphs while preserving both V and non-equivalence, since for a given V with $|V| > 2$ there are more hypergraphs than graphs.

Given a k-ary predicate p/k and a universe U, a ground fact of p/k can be described as an oriented hyperedge over V with $V = U$. Given a set of predicates P and a universe U, a logical interpretation (set of facts) is therefore equivalent to an edge-labelled oriented hypergraph, where the label of each hyperedge is the name of the corresponding predicate. Similarly, a relational database corresponds to an edge-labelled oriented hypergraph.[3]

A logical non-ground atom a corresponds to a set of hyperedges, namely those hyperedges for which a ground variable substitution θ exists such that the ground atom $a\theta$ corresponds to the hyperedge.

Given an interpretation I, a conjunctive query Q succeeds in I if there exists a grounding substitution θ such that all atoms in $Q\theta$ have a corresponding hyperedge in I. With $Q\theta$ thus corresponds a subhypergraph of the hypergraph corresponding to I, and with Q itself a set of such subhypergraphs (which is empty if the query has no solutions).

Perhaps an interesting observation here is that ILP corresponds to mining *oriented* hypergraphs. Many graph or hypergraph mining algorithms work on undirected (hyper)graphs. The importance of this difference has not yet been determined. However, in the case of molecule mining, there is a certain amount of importance: bonds between atoms in a molecule are undirected; if we define a binary *bond* predicate in ILP (the standard approach), we need to define it symmetrically (i.e., intensionally or extensionally state that for each bond a-b there is also a bond b-a). Our personal experience is that this necessity affects the efficiency as well as the ease of use of ILP systems for this kind of problems.

3.2 Non-V-preserving

The previous section discussed transformations from (V, E) to (V, E') where the set of nodes V remains unchanged (in the case of logic: V is the universe U). More transformations are possible if we do not require this property.

A non-oriented hypergraph can easily be translated to a bipartite graph as follows: given the hypergraph $H = (V, E)$, define $G = (V \cup E, E')$ where E' contains $\{v, e\}$ iff $v \in e$. Figure 2 shows the result of this transformation. The labels are necessary to distinguish which nodes in G are the original nodes and which are the original edges. Alternatively, if G is a directed graph, the direction of the edges carries the same information, making the labeling redundant. If G was already edge-labelled, the edge labels can be transferred (as shown in Figure 2).

An oriented hypergraph can be translated into a graph as follows. The ordering of the elements of an edge can be indicated with edges representing the successor function. But since a node can occur in several hyperedges, we need to order *occurrences* of nodes in hyperedges, rather than the nodes themselves. We thus get a graph with three layers, with nodes in each layer representing (a) hyperedges, (b) occurrences of nodes in hyperedges, (c) the original nodes. Figure 2 illustrates this.

4 Questions

The above incomplete list of possible transformations already gives rise to a number of questions:

- How important is the difference between directed/oriented (hyper)graphs and their undirected counterparts, w.r.t. efficiency?
- What is the effect on efficiency of using hypergraphs instead of graphs?

[3] Edge-labelled oriented hypergraphs seem the most direct graph-representation of relational databases. For instance, relational calculus [2] directly translates to a corresponding calculus over nodes (domain calculus) or hyperedges (tuple calculus).

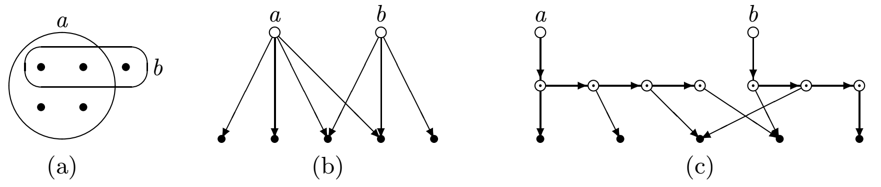

Fig. 2. (a) a hypergraph; (b) its representation as a graph; (c) representation as a graph of one particular oriented version of the hypergraph.

- How do implementation details influence efficiency? E.g., in ILP facts are looked up in the database using indexes, i.e., there is no direct link from a node to incident edges or adjacent nodes. In a (hyper)graph structure such links would be natural, and they may speed up certain operations significantly.
- How can the knowledge about refinement operators in ILP be transferred to the context of graphs and hypergraphs?
- What does, for instance, the (inverse) resolution operator from ILP look like when put in the oriented hypergraph framework?
- How does the concept of predicate invention translate to that setting?
- How does abduction of facts in ILP relate to link discovery (prediction of the existence of edges) in graphs? Can ILP lead to methods for hyperedge discovery?
- Could methods for frequent subgraph discovery be used for subhypergraph discovery (e.g., [4]) using the hypergraph-to-graph transformation described above? What would be the efficiency of this approach?
- Many more specific settings have been considered in graph mining, such as trees and sequences; how do these translate to the hypergraph mining setting?

Some of these questions may be easy to answer, others may present interesting challenges. We believe that the explicit study of transformations between different representation formalisms may generate many opportunities for advancing the state of the art in relational learning.

5 Conclusion

We have presented a non-exhaustive list of possible translations between learning tasks defined on graphs, hypergraphs, and logical representations, and presented a number of questions that this discussion gives rise to. We believe that further study of the relationship between graph mining, hypergraph mining and ILP, and defining perhaps other translations between these representational formalisms, may suggest many alternative approaches to mining from structured / relational data, yielding new and potentially useful insights as well as improved algorithms.

Acknowledgements

Hendrik Blockeel is a postdoctoral fellow of the Fund for Scientific Research of Flanders (FWO). This research is funded by the Dutch Science Foundation (NWO) through a VIDI grant. The authors thank Jan Van den Bussche for some very useful comments.

References

1. L. De Raedt. Logical settings for concept learning. *Artificial Intelligence*, 95:187–201, 1997.
2. R. Elmasri and S.B. Navathe. *Fundamentals of Database Systems*. Addison-Wesley, 4nd edition, 2004.
3. G Gallo, G Longo, S Pallottino, and Sang Nguyen. Directed hypergraphs and applications. *Discrete Applied Mathematics*, 42:177–201, 1993.
4. T. Horvath, B. Bringmann, and L. De Raedt. Frequent hypergraph mining. In *Proceedings on the International Workshop on Mining and Learning with Graphs*, pages 25–36, 2006.
5. T. Horvath and S. Wrobel. Towards discovery of deep and wide first-order structures: A case study in the domain of mutagenicity. In *Discovery Science 2001*, pages 100–112, 2001.