MINING TREE QUERIES IN A GRAPH

Bart Goethals [a] Eveline Hoekx [b] Jan Van den Bussche [b]

[a] *Universiteit Antwerpen, Belgium*
[b] *Universiteit Hasselt, Belgium*

The problem of mining patterns in graph-structured data has received considerable attention in recent years, as it has many interesting applications in such diverse areas as biology, the life sciences, the World Wide Web, or social sciences. Past work in this area falls in two major categories, not to be confused:

1. In the "transactional" category, e.g., (Dehaspe and Toivonen, 1999; Inokuchi et al., 2000; Kuramochi and Karypis, 2001; Zaki, 2002; Yan and Han, 2002; Huan et al., 2003), the dataset is a set of small graphs which we call transactions, and the task is to discover patterns that occur at least once in a sufficient number of transactions or examples. (Approaches from machine learning or inductive logic programming usually call the small graphs "examples" instead of transactions.)

2. In the "single graph" category, e.g., (Cook and Holder, 1994; Vanetik et al., 2002; Ghazizadeh and Chawathe, 2002; Kuramochi and Karypis, 2004; Jeh and Widom, 2004), the dataset is a single large graph, and the task is to discover patterns that occur sufficiently often in the dataset. Note that the transactional case can be simulated by the single-graph case, but the converse is not obvious.

The present work falls in the single-graph category. Past work in this category has mainly focused on patterns in the form of subgraphs that occur in sufficiently many edge-disjoint isomorphic copies in the graph that is being mined. We propose a rather different notion of pattern, which is still a graph, but with the following features:

- Patterns may have "existential" nodes: any occurrence of the pattern must have a copy of such a node, but existential nodes are not counted when determining the number of occurrences.

- Moreover, patterns may have "selected" nodes, labeled by constants, which must map to fixed designated nodes of the graph. Past work has dealt with node labels, but only with non-unique ones: such labels are easily simulated by constants, but the converse is not obvious. Also edge labels can be simulated using constants. (To simulate a node label a, add a special node a, and express that node x has label a by drawing an edge from x to a. For an edge $x \to y$ labeled b, introduce an intermediate node $x.y$ with $x \to x.y \to y$, and label node $x.y$ by b.)

- An "occurrence" of the pattern in G is defined as any homomorphism from the pattern in G. When counting the number of occurrences, two occurrences that differ only on existential nodes are identified.

A simple example of a pattern is shown in Figure 1; when applied to a food web ("who eats who"), it describes all organisms x that compete with organism #8 for some organism as food, that itself feeds on organism #0. This pattern has one existential node, two selected nodes, and one distinguished node x.

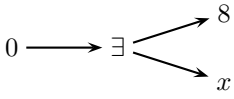

Figure 1: Simple example of a pattern

Effectively, what we want to mine are what is known in database research as *conjunctive queries* (Chandra and Merlin, 1977; Abiteboul et al., 1995); these are the queries we could pose to the graph (stored as a two-column table) in the core fragment of SQL where we do not use aggregates or subqueries, and use only conjunctions of equality comparisons as where-conditions. For example, the pattern of Figure 1 amounts to the following SQL query on a table `G(from,to)`:

```
select distinct G3.to as x
from G G1, G G2, G G3
where G1.from=0 and G1.to=G2.from
  and G2.to=8 and G3.from=G2.from
```

We will present an algorithm for mining conjunctive-query patterns that return sufficiently many answers on a given graph. Our algorithm has the following properties:

1. We restrict to patterns that are trees, such as the example in Figure 1. Tree patterns have formed an important special case in the transactional literature (Zaki, 2002; Chi et al., 2004), but have not yet received special attention in the single-graph literature. Note that the graph that is being mined is not restricted in any way.

2. The algorithm is incremental in the number of nodes of the pattern. We generate unordered, rooted trees of increasing sizes, avoiding the generation of isomorphic duplicates. It is well known how to do this efficiently (Scions, 1968; Li and Ruskey, 1999; Zaki, 2002; Chi et al., 2004). Note that this generation of trees is in no way "levelwise" (Mannila and Toivonen, 1997). Indeed, under the way we count pattern occurrences, a subgraph of a pattern might be less frequent than the pattern itself (this was already pointed out by Kuramochi and Karypis (2004)). So, our algorithm systematically considers ever larger trees, and can be stopped any time it has run long enough or has produced enough results. Our algorithm does not need any space beyond what is needed to store the mining results.

3. For each tree, all conjunctive queries based on that tree are generated. Here, we do work in a levelwise fashion. This aspect of our algorithm has clear similarities with "query flocks" (Tsur et al., 1998). A query flock is a user-specified conjunctive query, in which some constants are left unspecified and viewed as parameters. A levelwise algorithm was proposed for mining all instantiations of the parameters under which the resulting query returns enough answers. We push that approach further by also mining the query flocks themselves. Consequently, the specialization relation on queries used to guide the levelwise search is quite different in our approach.

4. A query based on some tree may be equivalent to a query based on a previously seen tree. Furthermore, two queries based on the same tree may be equivalent. We carefully and efficiently avoid the counting of equivalent queries, by using and adapting what is known from the theory of conjunctive database queries.

5. Last but not least, our algorithm naturally suggests a database-oriented implementation in SQL. This is useful for several reasons. First, the number of discovered patterns can be quite large, and it is important to keep them available in a persistent and structured manner, so that they can be browsed easily, and so that *association rules* can be derived efficiently. Moreover, we will show how the use of SQL allows us to generate and check large numbers of similar patterns in parallel, taking advantage of the query processing optimizations provided by modern relational database systems. Third, a database-oriented implementation does not require us to move the dataset out of the database before it can be mined. In classical itemset mining, database-oriented implementations have received serious attention (Tsur et al., 1998; Sarawagi et al., 2000), but less so in graph mining, a recent exception being an implementation in SQL of the seminal SUBDUE algorithm (Chakravarthy et al., 2004).

The primary purpose of this paper is to present our algorithm; concrete applications to discover new knowledge about real-life scientific datasets is a topic of planned future work. Yet, using a prototype implementation, we will already demonstrate here that our approach is feasible, by showing some concrete results mined from a food web, a protein interactions graph, and a citation graph. We also give performance figures on random graphs.