Mining for Tree-Query Associations in a Graph

Eveline Hoekx and Jan Van den Bussche
Hasselt University and transnational University of Limburg
Agoralaan D, 3590 Diepenbeek, Belgium
{eveline.hoekx, jan.vandenbussche}@uhasselt.be

Abstract

New applications of data mining, such as in biology, bioinformatics, or sociology, are faced with large datasets structured as graphs. We present an efficient algorithm for mining associations between tree queries in a large graph. Tree queries are powerful tree-shaped patterns featuring existential variables and data constants. Our algorithm applies the theory of conjunctive database queries to make the generation of association rules efficient. We propose a practical, database-oriented implementation in SQL, and show that the approach works in practice through experiments on data about food webs, protein interactions, and citation analysis.

1 Introduction

The problem of mining graph-structured data has received considerable attention in recent years, as it has applications in such diverse areas as biology, the life sciences, the World Wide Web, or social sciences. At KDD 2005, we presented an efficient algorithm for mining *tree queries* in a large graph [12], but we considered only a trivial form of associations between such queries. In the present paper, we present an efficient algorithm to mine fully fledged tree-query associations in a large graph.

Tree queries are powerful tree-shaped patterns, inspired by conjunctive database queries [28]. In comparison to most other graph mining approaches, tree queries have two powerful extra features in that they allow variables in the pattern to be *existential* or *parameterized*. Existential variables must be matched in the graph just like any other variable, but their different matchings are not counted as contributing towards the overall frequency of the pattern. So, only the matchings of the non-existential variables are counted. Parameterized variables, on the other hand, can be matched only by one specific data constant (node in the data graph).

By mining for tree-query associations we can discover quite subtle properties of the graph. Figure 1(a) shows a very simple example of an association that our algorithm might find in a social network: a graph of persons where there is an edge $x \to y$ if x considers y to be a close friend. The tree query on the left matches all pairs (x_1, x_2) of "co-friends": persons that are friends of a common person (represented by an existential variable). The query on the right matches all co-friends x_1 of person #5 (represented by a parameterized variable), and pairs all those co-friends to 5. Now were the association from the left to the right to be discovered with a confidence of c, with $0 \leq c \leq 1$, then this would mean that the pairs retrieved by the right query actually constitute a fraction of c of all pairs retrieved by the left query, which indicates (for nonnegligible c) that 5 plays a special role in the network.[1]

Figure 1(b) shows quite a different, but again simple, example of a tree-query association that our algorithm might discover in a food web: a graph of organisms, where there is an edge $x \to y$ if y feeds on x. With confidence c, this association means that of all organisms that are not on top of the food chain (i.e., they are fed upon by some other organism), a fraction of c is actually at least two down in the food chain.

The two examples we just saw are didactical examples, but in Section 11 we will see examples of associations mined in real-life datasets.

The three main features of our algorithm are the following.

1. As in classical association rules over itemsets [3], our association rule generation phase comes after the generation of frequent patterns and does not require access to the original dataset. In our case, however, all

[1]Note that this does not just mean that 5 has many co-friends; if we only wanted to express that, just a frequent pattern in the form of the right query would suffice. For instance, imagine a graph consisting of n disjoint 2-cliques (pairs of persons who have each other as a friend), where additionally all these persons also consider 5 to be an extra friend (but not vice versa). In such a graph, 5 is a co-friend of everybody, and the association has a rather high confidence of more than $2/7$. If, however, we would now add to the graph a separate n-clique, then still 2/3rds of all persons are a co-friend of 5, which is still a lot, but the confidence drops to below $2/n$.

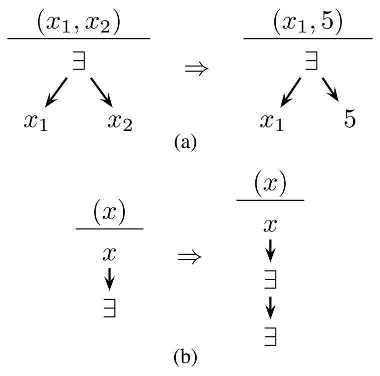

Figure 1. Simple examples of association rules over tree queries.

frequent tree queries are stored in structured form in a relational database. The associations can be obtained from this database in an automatic way using carefully constructed, yet simple, SQL queries, as we will show. For every potential association rule containing parameterized variables, one single SQL query retrieves, in parallel, all possible instantiations of the parameters by data constants that yield a confident association. Thanks to this feature, it is easy to develop a very fast association browsing tool.

2. We apply the theory of conjunctive database queries [28] to formally define and to correctly generate association rules over tree queries. The conjunctive-query approach to pattern matching allows for an efficiently checkable notion of frequency, whereas in the subgraph-based approach, determining whether a pattern is frequent is NP-complete (in that approach the frequency of a pattern is the maximal number of disjoint subgraphs isomorphic to the pattern [20]).

3. A fundamental notion in our approach is that of *containment* among conjunctive queries, which in general is NP-complete, but which again is efficiently checkable here, thanks to the restriction to tree shapes. This not only allows us to generate associations efficiently, but also to efficiently avoid the generation of *duplicates*, i.e., associations equivalent to a previously generated association. We can actually solve only part of the duplicate detection problem efficiently, but we prove that the general problem is as hard as general graph isomorphism, even under our restriction to tree shapes.

The primary purpose of this paper is to present our algorithm. Concrete applications to discover new knowledge about scientific datasets are the topic of planned future research. Yet, the algorithm is already fully implemented, and we can already show that our approach works in practice, by showing some concrete results mined from a food web, a protein interactions graph, and a citation graph. We will also give performance results on random graphs (as a worst-case scenario) which show that the generation of associations is very fast.

2 Related Work

Approaches to graph mining, especially mining for frequent patterns or association rules, can be divided in two major categories which are not to be confused. In transactional graph mining, e.g., [8, 14, 15, 16, 19, 30, 31], the dataset consists of many small graphs which we call transactions, and the task is to discover patterns that occur at least once in a sufficient number of transactions. (Approaches from machine learning or inductive logic programming usually call the small graphs "examples" instead of transactions.) In contrast, in single-graph mining, e.g., [7, 11, 17, 20, 29], the dataset is a single large graph, and the task is to discover patterns that occur sufficiently often in the dataset. Our approach falls squarely within the single-graph category. Note that single-graph mining is more difficult than transactional mining, in the sense that transactional graph mining can be simulated by single-graph mining, but the converse is not obvious.

Within single-graph mining, not much previous work exists on association rules. Jeh and Widom [17] consider patterns that are, like our tree queries, inspired by conjunctive database queries, and they also emphasize the tree-shaped case. A severe restriction, however, is that their patterns can be matched by single nodes only, rather than by tuples of nodes. Moreover, they mention association rules only in passing. Their work is still interesting in that it presents a rather nonstandard approach to graph mining, quite different from our own incremental, levelwise approach, and in that it incorporates ranking.

The related work that was most influential for us is Warmr [8, 9]. Based on inductive logic programming, patterns in Warmr also feature existential variables and parameters. While not restricted to tree shapes, the queries in Warmr are restricted in another sense so that only transactional mining can be supported. Association rules in Warmr are defined in a naive manner through pattern extension, rather than being founded upon the theory of conjunctive query containment. The Warmr system is also Prolog-oriented, rather than database-oriented, which we believe is fundamental to mining of single large graphs, and which allows a more uniform and parallel treatment of parameter instantiations, as we will show in this paper. Finally, Warmr does not seriously attempt to avoid the generation of duplicates. Yet, Warmr remains a pathbreaking work, which did not receive sufficient follow-up in the data mining community at large. We hope our present work represents an improvement in this respect. Many of the improvements we make to Warmr were already envisaged (but without concrete algorithms) in 2002 by Goethals and the second author [13].

Finally, we note that parameterized conjunctive database queries have been used in data mining quite early, e.g., [27, 26], but then in the setting of "data mining query languages", where a *single* such query serves to specify a family of patterns to be mined or queried for, rather than the mining for such queries themselves, let alone associations among them.

3 Mining for Association Rules over Tree Queries

In this section we define the problem formally. We basically assume a set U of *data constants* from which the nodes of the graph to be mined will be taken. Graphs are always directed, so basically, for the purposes of the present paper, a graph is simply a finite set of ordered pairs of elements from U. We assume familiarity with the notion of a *tree* as a special kind of graph, and with standard graph-theoretic concepts as supplied by any algorithms textbook.

Tree Patterns A *tree pattern* P is a tree whose nodes are called *variables*, and where additionally:

- Some variables may be marked as being *existential*;
- Some other variables may be marked as *parameters*;
- The variables of P that are neither existential nor parameters are called *distinguished*.

We will denote the set of existential variables by Π, and the set of parameters by Σ. To make clear that these sets belong to some tree pattern P we will use a subscript as in Π_P or Σ_P.

A *parameter assignment* α, for a tree pattern P, is a mapping $\Sigma \to U$ which assigns data constants to the parameters.

An *instantiated* tree pattern is a pair (P, α), with P a tree pattern and α a parameter assignment for P. We will also denote this by P^α.

When depicting tree patterns, existential nodes are indicated by labeling them with the symbol '$\exists$' and parameters are indicated by labeling them with the symbol 'σ'. When depicting instantiated tree patterns, parameters are indicated by directly writing down their parameter assignment.

Figure 2 shows an illustration.

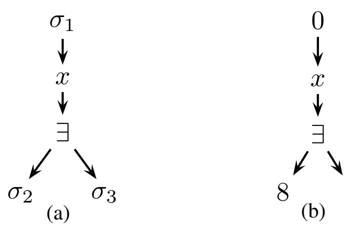

Figure 2. (a) is a tree pattern, and (b) is an instantiation of (a).

Matching Recall that a *homomorphism* from a graph G_1 to a graph G_2 is a mapping f from the nodes of G_1 to the nodes of G_2 that preserves edges, i.e., if $(i, j) \in G_1$ then $(f(i), f(j)) \in G_2$. We now define a *matching* of an instantiated tree pattern P^α in a graph G as a homomorphism f from the underlying tree of P to G, with the constraint that for any parameter σ, if $\alpha(\sigma) = a$, then $f(\sigma)$ must be the node a.

Frequency of a tree pattern The *frequency* of an instantiated tree pattern P^α in a graph G, is defined as the number of matchings of P^α in G, *where we identify any two matchings that agree on the distinguished variables.* For a given threshold k (a natural number) we say that P^α is *k-frequent* if its frequency is at least k. Often the threshold is understood implicitly, and then we talk simply about "frequent" patterns and denote the threshold by *minsup*.

Tree Queries A *tree query* Q is a pair (H, P) where:

1. P is a tree pattern, called the *body* of Q;
2. H is a tuple of distinguished variables and parameters coming from P. All distinguished variables of P must appear at least once in H. We call H the *head* of Q.

A parameter assignment for Q is simply a parameter assignment for its body, and an *instantiated* tree query is then again a pair (Q, α) with Q a tree query and α a parameter assignment for Q. We will again also denote this by Q^α.

When depicting tree queries, the head is given above a horizontal line, and the body below it. Two illustrations are given in Figure 3.

Containment of tree queries The final step towards our formal definition of tree-query association is the notion of *containment* among queries.

First, we define the *answer set* of an instantiated tree query Q^α, with $Q = (H, P)$, in a graph G as follows:

$$Q^\alpha(G) := \{f(H) \mid f \text{ is a matching of } P^\alpha \text{ in } G\}$$

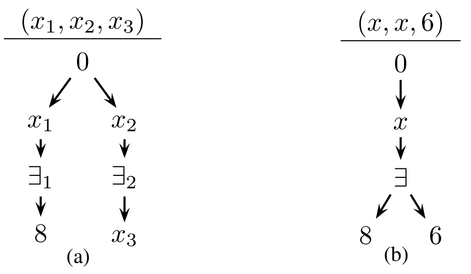

Figure 3. Simple examples of instantiated tree queries. Query (b) is contained in query (a)

We then say that an instantiated tree query $Q_2^{\alpha_2}$ is *contained* in an instantiated tree query $Q_1^{\alpha_1}$, if $Q_2^{\alpha_2}(G) \subseteq Q_1^{\alpha_1}(G)$ for all graphs G. In shorthand notation we write this as $Q_2^{\alpha_2} \subseteq Q_1^{\alpha_1}$.

Containment as just defined is a semantical property, referring to all possible graphs, and it is not immediately clear how one could decide this property syntactically. The required syntactical notion for this is that of *containment mapping*, which we next define in several steps. Consider again two instantiated tree queries $Q_1^{\alpha_1}$ and $Q_2^{\alpha_2}$, with $Q_i = (H_i, P_i)$ for $i = 1, 2$.

1. A containment mapping from P_1 to P_2 is a homomorphism f from the underlying tree of P_1 to the underlying tree of P_2, with the property that f maps the distinguished nodes of P_1 to distinguished nodes or parameters of P_2, and the parameters of P_1 to parameters of P_2.

2. A containment mapping from $P_1^{\alpha_1}$ to $P_2^{\alpha_2}$ is a containment mapping f from P_1 to P_2 that respects the parameter assignments, i.e., for any parameter σ of P_1, we have $\alpha_2(f(\sigma)) = \alpha_1(\sigma)$.

3. Finally, a containment mapping from $Q_1^{\alpha_1}$ to $Q_2^{\alpha_2}$ is a containment mapping f from $P_1^{\alpha_1}$ to $P_2^{\alpha_2}$ such that $f(H_1) = H_2$.

A classical result [5, 28, 2] now states that $Q_2^{\alpha_2}$ is contained in $Q_1^{\alpha_1}$ precisely when there exists a containment mapping from $Q_1^{\alpha_1}$ to $Q_2^{\alpha_2}$. Checking for a containment mapping is evidently computable, and although the problem for general database conjunctive queries is NP-complete, our restriction to tree shapes allows for efficient checking, as we will see later.

Example. Consider the instantiated tree queries shown in Figure 3. In the example graph shown in Figure 4(a), the frequency of query (a) is 10 and that of query (b) is 2. A moment's reflection should convince the reader that (b) is contained in (a), and indeed a containment mapping from (a) to (b) can be found as follows: $0 \mapsto 0$; $x_1 \mapsto x$; $x_2 \mapsto x$; $\exists_1 \mapsto \exists$; $\exists_2 \mapsto \exists$; $8 \mapsto 8$; $x_3 \mapsto 6$. For a good understanding, note that were we to change the head of (b) to $(x, x, 8)$, then this new query (b) would still be contained in (a), because we can alternatively map $x_3 \mapsto 8$ and still have a containment mapping from the body of (a) to the body of (b).

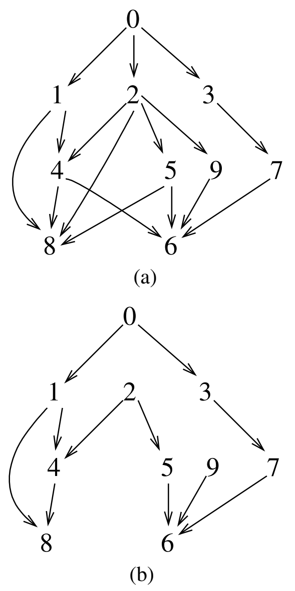

Figure 4. Two graphs.

Association Rules A potential *association rule (AR)* is of the form $Q_1^{\alpha_1} \Rightarrow Q_2^{\alpha_2}$, with $Q_1^{\alpha_1}$ and $Q_2^{\alpha_2}$ instantiated tree queries. The AR is *legal* if $Q_2^{\alpha_2} \subseteq Q_1^{\alpha_1}$. We call $Q_1^{\alpha_1}$ the left-hand side (lhs), and $Q_2^{\alpha_2}$ the right-hand side (rhs).

The *confidence* of the AR in a graph G is defined as the frequency of the rhs in G, divided by the frequency of the lhs in G. If the AR is legal, we know that the answer set of the rhs is a subset of the answer set of the lhs, and hence the confidence equals precisely the proportion that the rhs answer set takes up in the lhs answer set. Thus, our notion of legal AR and confidence is very intuitive and natural.

For a given threshold c (a rational number, $0 \leq c \leq 1$) we say that the AR is *c-confident* in G if its confidence in G is at least c. Often the threshold is understood implicitly, and then we talk simply about "confident" ARs and denote the threshold by *minconf*.

Furthermore, the AR is called *frequent* in G if the body of the rhs is frequent in G. Note that if the AR is legal and frequent, then also the body of the lhs is frequent, since the rhs is contained in the lhs.

Example. Continuing the previous example, we can see that we can form a legal AR from the queries of Figure 3, with (a) the lhs and (b) the rhs. The confidence of this AR in the graph of Figure 4(a) is $2/10$. Many more examples of ARs are given in the next Section.

Association Rule Mining We are finally ready to formulate the problem we want to solve:

Input: A graph G; a threshold *minsup*; an instantiated tree query $(Q_{\text{left}}, \alpha_{\text{left}})$ that is frequent in G; a threshold *minconf*.

Output: All instantiated tree queries Q^{α} such that $Q_{\text{left}}^{\alpha_{\text{left}}} \Rightarrow Q^{\alpha}$ is a legal and confident association rule in G.

In theory, however, there are infinitely many legal and confident association rules for a fixed lhs, and even if we set an upper bound on the size of the rhs, there may be exponentially many. Hence, in practice, we want an algorithm that runs incrementally, and that can be stopped any time it has run long enough or has produced enough results.

4 Problem Reduction

In this section, we show that it is not necessary to attack the problem in its full generality. We will show that, without loss of generality, we can focus on the case where the given lhs query Q_{left} is "pure" in a sense that we will make precise. We will also show that this restriction cannot be imposed on the rhs queries to be output. We also make a remark regarding "free constants" in the head of a query.

Pure tree queries To define this formally, assume that all possible variables (nodes of tree patterns) have been arranged in some arbitrary but fixed order. We then call a tree query $Q = (H, P)$ *pure* when H consists of the enumeration, in order and without repetitions, of all the distinguished variables of P. In particular, H cannot contain parameters. As an illustration, the lhs of rule (a) of Figure 5 is impure, while the lhs of rule (b) is pure.

An AR with an impure lhs can always be rewritten to an equivalent AR with a pure lhs, with the same confidence and frequency. Indeed, take a legal AR $Q_1^{\alpha_1} \Rightarrow Q_2^{\alpha_2}$, with Q_1 not pure. We know that Q_1's head is mapped to Q_2's head by some containment mapping. Hence, we can purify Q_1 by removing all constants and repetitions of distinguished variables from Q_1's head, sort the head by the order on the variables, and perform the corresponding actions on Q_2's head as prescribed by the containment mapping. An illustration is given in Figure 5.

We can conclude that it is sufficient to only consider ARs with pure lhs's. The rhs, however, need not be pure; impure rhs's are in fact interesting, as we will demonstrate next.

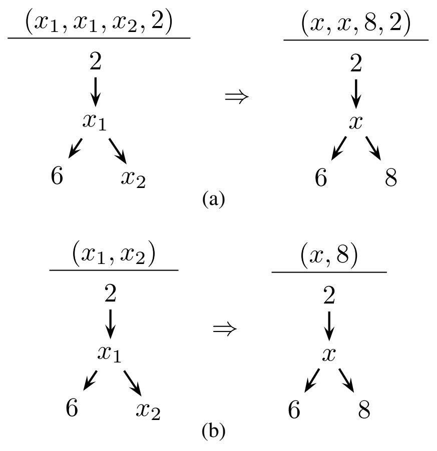

Figure 5. Rule (a) has a non-pure lhs. Rule (b) is the purification of rule (a), and expresses precisely the same information.

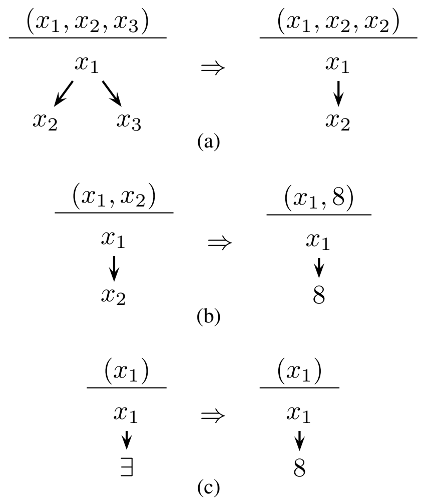

Figure 6. (a) and (b) are ARs with impure rhs. (c) is an ill-advised attempt to purify (b) on the rhs.

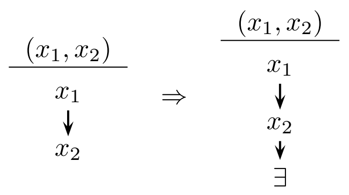

Figure 7. An AR with a pure rhs.

Impure rhs's Consider the AR in Figure 6(a). The rhs is impure since x_2 appears twice in the head. The AR expresses that a sufficient proportion of the matchings of the lhs pattern, are also matchings of the rhs pattern, which is the same as the lhs pattern except that x_2 is equal to x_3. The confidence of this AR is $m / \sum_x \deg^2 x$, where m is the number of edges, x ranges over the nodes in the graph, and $\deg x$ is the outdegree of (number of edges leaving) x. Since $m = \sum_x \deg x$, an easy calculation shows that this confidence is much larger than $1/m$. Hence, the sparser the graph (with the number of nodes remaining the same), the higher the confidence, and thus the AR is interesting in that it tells us something about the sparsity of the graph. As an illustration, on the graph of Figure 4(a) the confidence is 0.4, but on the the graph of Figure 4(b), it is 0.6.

Also consider the AR in Figure 6(b). Again the rhs is impure since its head contains a parameter. With confidence c, the AR expresses that a fraction of c of all edges point to node 8, which again would be an interesting property of the graph.

The knowledge expressed by the above two example ARs cannot be expressed using ARs with pure rhs's. To illustrate, the AR of Figure 6(c) may at first seem equivalent (and has a pure rhs) to that of Figure 6(b). On second thought, however, it says nothing about the proportion of edges pointing to 8, but only about the proportion of nodes with an edge to 8.

Of course, we are not implying that ARs with pure rhs's are uninteresting. But all they can express are statements about the proportion of matchings of the lhs that can be specialized or extended to a matching of the rhs (another example is in Figure 7, which says something about the proportion of edges that can be extended); they cannot say anything about the proportion of matchings of the lhs that satisfy certain equalities in the distinguished variables.

Free Constants Most treatments of conjunctive database queries [2, 28] allow arbitrary constants in the head. In our treatment, a constant can only appear in the head as the value of a parameter. This restriction is justified, since we can show that for the sake of legal association rules among conjunctive queries, constants appearing in the head but not in the body do not buy us anything. We defer the easy argument (based on the existence of a containment mapping) to the full paper.

5 Overall Approach

Given the inputs: G, $Q_{\text{left}}^{\alpha_{\text{left}}} = ((H_{\text{left}}, P_{\text{left}}), \alpha_{\text{left}})$, and *minconf* and *minsup*, an outline of our algorithm for the association rule mining problem is that of four nested loops:

1. Generate, incrementally, all possible trees of increasing sizes. Avoid trees that are isomorphic to previously generated ones. The height of the generated trees must be at least the height of the tree underlying P_{left}. (When enough trees have been generated, this loop can be terminated.)

2. For each new generated tree T, generate all frequent instantiated tree patterns P^α based on that tree, in a way that is "levelwise" in the sense of Mannila and Toivonen [22].

3. For each tree pattern P, generate all containment mappings f from P_{left} to P, ignoring parameter assignments.

4. For each f, generate all instantiated tree queries $Q^\alpha = ((f(H_{\text{left}}), P), \alpha)$ such that $f : Q_{\text{left}}^{\alpha_{\text{left}}} \to Q^\alpha$ respects the parameter assignments; Q^α is frequent; and the confidence of $Q_{\text{left}}^{\alpha_{\text{left}}} \Rightarrow Q^\alpha$ exceeds *minconf*. The generation of all these α's happens in a parallel fashion.

This approach is complete, i.e., it will output everything that must be output. In proof, consider a legal, frequent and confident AR $Q_{\text{left}}^{\alpha_{\text{left}}} \Rightarrow Q^\beta$, with $Q = (H_Q, P_Q)$. The tree T is the underlying tree of P_Q; the tree pattern P in loop 2 is P_Q; the containment mapping f in loop 3 is a containment mapping from $Q_{\text{left}}^{\alpha_{\text{left}}}$ to Q^α; and α in loop 4 is β.

As to loop 1, it is already well known how to efficiently generate all trees uniquely up to isomorphism, in increasing number of nodes [6, 21, 25, 31]. We present loops 2, 3 and 4 in detail in Sections 6, 7 and 8.

The reader may wonder whether loop 3 cannot be organized in a levelwise fashion as well as loop 2. This is not obvious, however, since any two queries of the form $((f_1(H_{\text{left}}), P), \alpha)$ and $((f_2(H_{\text{left}}), P), \alpha)$ have exactly the same frequency, namely that of P^α. Loop 4, however, is levelwise because it is based on loop 2 which is levelwise.

In Section 9, we will show how our overall approach must be refined so that the generation of equivalent association rules is avoided.

6 The levelwise generation of tree patterns

Loop 2 of our algorithm, the generation of all frequent instantiated tree patterns P^α based on a fixed tree, in a levelwise fashion, has already been solved in our earlier work [12]. We recall here the details that are needed further on.[2]

The levelwise search is based on a natural specialization relation that is suggested by an alternative notation for the instantiated tree patterns under consideration. Concretely,

[2]We warn the reader that what we earlier called a "tree query" [12] is here called an "instantiated tree pattern"; what we here call a "tree query" was not yet studied in our earlier work.

since the underlying tree is fixed, any tree pattern is characterized by two parameters: the set Π of existential nodes, and the set Σ of parameters. Thus, an instantiated tree pattern P^α, with $P = (\Pi, \Sigma)$ is completely characterized by the triple (Π, Σ, α).

We now say that $P_1^{\alpha_1} = (\Pi_1, \Sigma_1, \alpha_1)$ *specializes* $P_2^{\alpha_2} = (\Pi_2, \Sigma_2, \alpha_2)$ if $\Pi_1 \supseteq \Pi_2$; $\Sigma_1 \supseteq \Sigma_2$; and α_1 agrees with α_2 on Σ_2. We also say that $P_2^{\alpha_2}$ *generalizes* $P_1^{\alpha_1}$.

Clearly, if $P_1^{\alpha_1}$ specializes $P_2^{\alpha_2}$, then the frequency of $P_1^{\alpha_1}$ is at most that of $P_2^{\alpha_2}$. Furthermore if $Q_1^{\alpha_1} = ((H_1, P_1), \alpha_1)$ and $Q_2^{\alpha_2} = ((H_2, P_2), \alpha_2)$ are instantiated tree queries such that AR1: $Q_{\text{left}}^{\alpha_{\text{left}}} \Rightarrow Q_1^{\alpha_1}$ and AR2: $Q_{\text{left}}^{\alpha_{\text{left}}} \Rightarrow Q_2^{\alpha_2}$ are legal ARs, then the confidence of AR1 will be at most that of AR2. So we can use this relation to guide a levelwise search for the frequent and confident association rules.

Our algorithm outputs the frequent patterns in the form of *frequency tables*, which are defined as follows:

$$\mathit{FreqTab}_{\Pi,\Sigma} = \{(\alpha, k) \mid (\Pi, \Sigma, \alpha) \text{ is frequent with frequency } k\}$$

So, a frequency table $\mathit{FreqTab}_{\Pi,\Sigma}$ contains all parameter assignments α for which P^α, with $P = (\Pi, \Sigma)$, is a frequent instantiated tree pattern.

Technically, the table has columns for the different parameters, plus a column `freq`. Note that when $\Sigma = \emptyset$, i.e., P has no parameters, this is a single-column, single-row table containing just the frequency of P. Of course in practice, all frequency tables for parameterless patterns can be combined into a single table. All frequency tables are kept in a relational database.

7 Generation of containment mappings

In this section, we discuss loop 3, the generation of all containment mappings from P_{left} to P. So, we need to solve the following problem: Given two tree patterns P_1 and P_2, find all containment mappings from P_1 to P_2.

Since the patterns are typically small, a naive algorithm suffices. For a node x_1 of P_1 and a node x_2 of P_2, we say that x_1 "matches" x_2 if there is a containment mapping f from the subtree pattern of P_1 rooted at x_1 to the subtree pattern of P_2 rooted at x_2 such that $f(x_1) = x_2$. In a first phase, we determine for every node y of P_2 separately whether the root r of P_1 matches y. While doing so, we also determine for every other node x_1 of P_1, and every node x_2 below y at the same distance as x_1 is from r, whether x_1 matches x_2. We store all these boolean values in a two-dimensional matrix.

This first phase compares every possible pair (x_1, x_2), with x_1 a node in P_1 and x_2 a node in P_2, at most once. Indeed, if x_1 is at distance d from r, then x_1 will be compared to x_2 only during the matching of r with the node y that is d steps above x_2 in P_2 (if existing). We thus have an $O(n_1 \times n_2)$ algorithm, where n_1 (n_2) is the number of nodes in P_1 (P_2).

In a second phase, we output all containment mappings. Initially, by a synchronous preorder traversal of P_1 and P_2, we map each node of P_1 to the first matching node of P_2. In each subsequent step, we look for the last node x_1 (in preorder) of P_1, currently matched to some node x_2, with the property that x_1 can also be matched to a right sibling x_3 of x_2, and now map x_1 to the first such x_3. The mappings of all nodes of P_1 coming after x_1 are reinitialized. Every such step takes time that is linear in n_1 and independent of n_2. Of course, the total number of different containment mappings may well be exponential in n_1.

We can thus easily generate all containment mappings f from P_{left} to P as required for loop 3 of our overall algorithm. Note, however, that in loop 4 these mappings are used to produce the head $f(H_{\text{left}})$ of query Q. For Q to be a legal query, this head must contain all distinguished variables of P. Hence, we only pass to loop 4 those f whose image contains all distinguished variables of P.

8 Generation of parameter assignments

In loop 4, our task is the following. Given a containment mapping $f : P_{\text{left}} \to P$, generate all instantiated tree queries $Q^\alpha = ((f(H_{\text{left}}), P), \alpha)$ such that $f : Q_{\text{left}}^{\alpha_{\text{left}}} \to Q^\alpha$ respects the parameter assignments; Q^α is frequent; and the confidence of $Q_{\text{left}}^{\alpha_{\text{left}}} \Rightarrow Q^\alpha$ exceeds *minconf*.

Since $Q = (f(H_{\text{left}}), P)$ is determined, the only problem is to generate the parameter assignments, α. This happens in a parallel database-oriented fashion.

Indeed, recall from Section 6 that the frequency tables for P_{left} and P are available in a relational database. Our crucial observation is that we can compute precisely the required set of parameter assignments α, together with the frequency and confidence of the corresponding association rules, by a single relational algebra expression.[3] This expression has the following form:

$$\boldsymbol{\pi}_{\mathit{plist}}\ \boldsymbol{\sigma}_{\frac{\mathit{FreqTab}_P.\text{freq}}{\mathit{FreqTab}_{P_{\text{left}}}.\text{freq}} \geq \mathit{minconf}} \\ (\boldsymbol{\sigma}_{\theta_{\text{left}}}(\mathit{FreqTab}_{P_{\text{left}}}) \bowtie_\theta \mathit{FreqTab}_P)$$

Here, $\boldsymbol{\pi}$ denotes projection, $\boldsymbol{\sigma}$ denotes selection, and $\bowtie$ denotes join. The join condition θ, the selection condition θ_{left}, and the projection list *plist* are defined as follows. Let Σ_{left} be the set of parameters of P_{left}. Then θ is the conjunction:

$$\bigwedge_{\sigma \in \Sigma_{\text{left}}} \mathit{FreqTab}_{P_{\text{left}}}.\sigma = \mathit{FreqTab}_P.f(\sigma)$$

[3]The relational algebra is the basic query language for relational databases; see any database textbook.

The selection condition θ_{left} is defined as the conjunction:

$$\bigwedge_{\sigma\in\Sigma_{\text{left}}} \mathit{FreqTab}_{P_{\text{left}}}.\sigma = \alpha_{\text{left}}(\sigma)$$

Furthermore, *plist* consists of all attributes $P.\sigma$, with $\sigma \in \Sigma$, together with the attributes $\mathit{FreqTab}_P$.freq and $\mathit{FreqTab}_P$.freq$/\mathit{FreqTab}_{P_{\text{left}}}$.freq.

Referring back to our overall algorithm (Section 5), we thus generate, for each pattern P generated in loop 2 and each containment mapping f in loop 3, all association rules with the given $Q_{\text{left}}^{\alpha_{\text{left}}}$ as lhs in parallel, by one relational database query (which can be implemented by a simple SQL select-statement).

Moreover, we now see that *we not actually have to limit ourselves to one given instantiation α_{left} of Q_{left}!* Indeed, simply by omitting the selection $\boldsymbol{\sigma}_{\theta_{\text{left}}}$ on $\mathit{FreqTab}_{P_{\text{left}}}$, and by adding the parameters of P_{left} to the projection list, we obtain in parallel all legal and confident association rules for all possible instantiations of Q_{left} as lhs.

Example. Consider Q_{left} and P as shown in Figures 8(a) and Figure 8(b). We have $\Sigma_{\text{left}} = \{x_1, x_4\}$ and $\Pi_{\text{left}} = \{x_3, x_6\}$, and $\Sigma_P = \{x_1, x_4, x_5\}$ and $\Pi_P = \{x_3\}$. Take the following containment mapping f from P_{left} to P: $x_1 \mapsto x_1$; $x_2 \mapsto x_2$; $x_3 \mapsto x_3$; $x_4 \mapsto x_4$; $x_5 \mapsto x_2$; $x_6 \mapsto x_3$; $x_7 \mapsto x_4$. Then the rhs query Q equals $((x_2, x_2, \sigma_4), P)$, and the relational algebra expression for computing all parameter assignments α for all instantiations α_{left} of Q_{left} looks as follows:

$$\boldsymbol{\pi}_{\mathit{plist}}\ \boldsymbol{\sigma}_{\frac{\mathit{FreqTab}_P.\text{freq}}{\mathit{FreqTab}_{P_{\text{left}}}.\text{freq}} \geq \mathit{minconf}}(\mathit{FreqTab}_{P_{\text{left}}} \bowtie_\theta \mathit{FreqTab}_P)$$

with *plist* equal to

$$\mathit{FreqTab}_P.x_1,\ \mathit{FreqTab}_P.x_4, \mathit{FreqTab}_P.x_5,$$
$$\mathit{FreqTab}_P.\text{freq},\ \mathit{FreqTab}_P.\text{freq}/\mathit{FreqTab}_{P_{\text{left}}}.\text{freq}$$

and θ equal to

$$\mathit{FreqTab}_P.x_1 = \mathit{FreqTab}_{P_{\text{left}}}.x_1$$
$$\wedge\ \mathit{FreqTab}_P.x_4 = \mathit{FreqTab}_{P_{\text{left}}}.x_4$$

In SQL, we get:

```
SELECT freqP.x1, freqP.x4,
       freqP.x5, freqP.freq,
       freqP.freq/freqQleft.freq
FROM freqP, freqQleft
WHERE freqQleft.x1= freqP.x1
  AND freqQleft.x4=freqP.x4
  AND freqP.freq/freqQleft.freq >= minconf
```

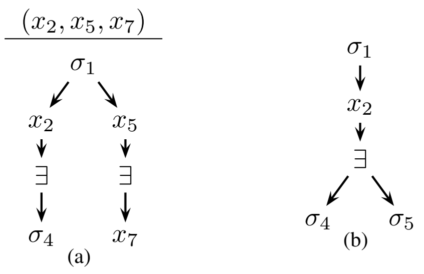

Figure 8. Example Q_{left} and P.

9 Equivalent association rules

In this section, we make a number of modifications to the algorithm described so far, so as to avoid duplicate work on equivalent queries.

From our previous work [12] we already know how to make sure that the tree patterns that are generated in loop 2 of the overall approach (Section 5) are never equivalent to a previously generated one. Thus, we can focus on ARs AR1: $Q_{\text{left}} \Rightarrow Q_1$ and AR2: $Q_{\text{left}} \Rightarrow Q_2$, with $Q_1 = (f_1(H_{\text{left}}), P)$ and $Q_2 = (f_2(H_{\text{left}}), P)$ and f_1 and f_2 containment mappings from P_{left} to P, and we want to know when these two ARs are equivalent.

A tricky example of two ARs that convey precisely the same information and should thus be considered equivalent, is shown in Figure 9. We formalize equivalence as follows: the structures $(P_{\text{left}}, P, f_1)$ and $(P_{\text{left}}, P, f_2)$ are isomorphic. Specifically, there must exist isomorphisms (actually automorphisms) $g : P_{\text{left}} \rightarrow P_{\text{left}}$ and $h : P \rightarrow P$ such that $f_2 \circ g = h \circ f_1$. In the figure, where f_1 (for (a)) and f_2 (for (b)) can be read out from the heads of the rhs's, h swaps x_2 and x_3, and g is the cyclic permutation $x_2 \mapsto x_3 \mapsto x_4 \mapsto x_2$.

So, using graph isomorphism (to be precise, edge-colored graph isomorphism, where we use different colors for the edges in P_{left}, the edges in P, and the pairs in f_1 or f_2), we can test for equivalence. Since our patterns are not very large, fast heuristics for graph isomorphism can be used [23]. This works well in practice, but theoretically this situation is not entirely satisfying, as graph isomorphism is not known to be efficiently (polynomial-time) solvable in general. By a reduction from the isomorphism problem for bipartite graphs, we can actually show that isomorphism of our structures is really as hard as the general graph isomorphism problem (proof deferred to the full paper). But as we show next, we can still capture an important special case in polynomial time, so that the general graph isomorphism heuristics only have to be applied on instances not captured by the special case.

The special efficient case is to check whether $(P_{\text{left}}, P, f_1)$ and $(P_{\text{left}}, P, f_2)$ are already isomorphic with g

$$\begin{tikzcd} & \underline{(x_1, x_2, x_3, x_4)} & & & & \underline{(x_1, x_2, x_2, x_3)} & \\ & x_1 \arrow[dl] \arrow[d] \arrow[dr] & & \Rightarrow & & x_1 \arrow[dl] \arrow[dr] & \\ x_2 & x_3 & x_4 & & x_2 & & x_3 \end{tikzcd}$$

(a)

$$\begin{tikzcd} & \underline{(x_1, x_2, x_3, x_4)} & & & & \underline{(x_1, x_2, x_3, x_3)} & \\ & x_1 \arrow[dl] \arrow[d] \arrow[dr] & & \Rightarrow & & x_1 \arrow[dl] \arrow[dr] & \\ x_2 & x_3 & x_4 & & x_2 & & x_3 \end{tikzcd}$$

(b)

Figure 9. Equivalent association rules.

the identity, i.e., whether the structures (P, f_1) and (P, f_2) are already isomorphic. So, we look for an automorphism h of P such that $f_2 = h \circ f_1$. This can be solved efficiently by a reduction to node-labeled tree isomorphism. As explained in Section 6, if we know the tree T underlying P, then P is characterized by the pair (Π, Σ), and thus (P, f) is characterized by (Π, Σ, f). We can view this triple as a labeling of T, as follows. We label every node y of P with a triple $(b_\Pi, b_\Sigma, f^{-1}(y))$, where b_Π is a bit that is 1 iff $y \in \Pi$; b_Σ is a bit that is defined likewise; and $f^{-1}(y)$ is the set of nodes of P_{left} that are mapped by f to y. Then (P, f_1) and (P, f_2) are isomorphic if and only if the corresponding node-labeled trees are isomorphic, and the latter can be checked in linear time using canonical ordering [4, 6].

We are now in a position to describe how our general algorithm must be modified to avoid equivalent association rules. There is only extra checking to be done in loop 3. For each new containment mapping f from P_{left} to P, we canonize the corresponding node-labeled tree and we check if the canonical form is identical to an earlier generated canonical form; if so, f is dismissed. We can keep track of the canonical forms seen so far efficiently using a trie data structure. If the canonical form was not yet seen, we can either let f through to loop 4, if the presence of duplicates in the output is tolerable for the application at hand, or we can default to an edge-colored graph isomorphism check with the containment mappings previously seen, to be absolutely sure we will not generate a duplicate.

10 Browsing association rules, and performance

As already explained in Section 6, in loop 2 we build up a structured database containing all frequency tables for all trees generated in loop 1. These two first loops should be regarded as a preprocessing step; once built up, the database is an ideal platform for an interactive tool by which the user can repeatedly specify lhs's, after which the tool only needs to run loops 3 and 4 to produce rhs's that form an association with the given lhs.

In a typical usage scenario, the user draws a tree shape, marks some nodes as existential, marks some others as parameters, instantiates some parameters by constants, but possibly also leaves some parameters open. The browser then returns, by consulting the appropriate frequency table in the database, all instantiations of the parameters that make the pattern frequent, together with the frequency. The user can then select one of these instantiations, set a minconf value, and ask the browser to return all rhs's that form a confident association with the selected pure tree query as lhs. Also, instead of letting the browser return all association rules, the user can already suggest a rhs by drawing a tree shape, possibly with some nodes already marked as parameter or existential, and let the browser return all rhs's of the prescribed form.

The preprocessing step, i.e., the building up of the database with frequent patterns, is of course a hugely intensive task, first because the large data graph must be accessed intensively, and second because the number of frequent patterns is huge. Nevertheless, in our previous work [12] we already presented detailed experimental results showing that this can be implemented with satisfactory performance. Also, in scientific discovery applications it is no problem, indeed typical, if a preprocessing step takes a few hours, as long as after that the interactive exploration of association rules can happen very fast.

And indeed, we found the actual generation of association rules (i.e., loops 3 and 4) to be very fast. For instance, Figure 10 shows the performance of generating association rules for two different (absolute) values of minconf, against a frequency table database built up for a random graph with 33 nodes and 113 edges, an absolute minsup of 25, and all trees up to size 7. We see that associations are generated with constant overhead, i.e., in linear-output time. The coefficient is larger for the larger minconf, because in this experiment we have counted instantiated rhs's, and per rhs query less instantiations satisfy the confidence threshold for larger such thresholds. Had we simply counted rhs's regardless of the number of confident instantiations, the two lines would have had the same slope.

The experiments were performed on a Pentium IV (3.6GHz) architecture with 2GB of internal memory, running under Linux 2.6. The program was written in C++ with embedded SQL, with DB2 UDB 8.2 as the relational database system.

11 Experimental results

While the application of our algorithm to serious scientific data mining is planned future work, in this section, we still report on some preliminary experiments performed using our prototype implementation applied to a food web, a

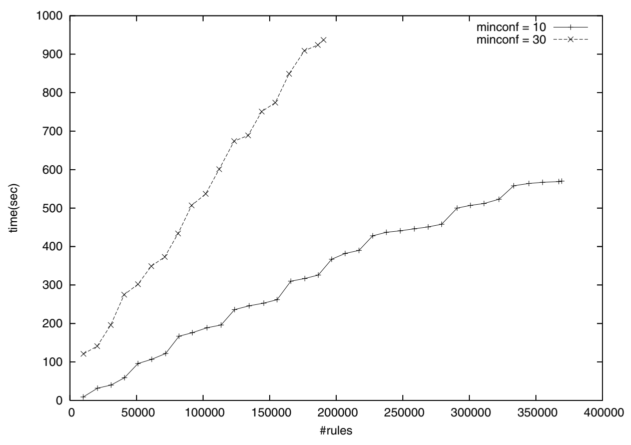

Figure 10. Performance in terms of number of discovered rules.

protein interactions graph, and a citation graph. These results show that our approach is workable.

For each dataset we built up a frequency table database using the following parameters:

	#nodes	#edges	minsup	max tree size
food web	154	370	100	6
proteins	2114	4480	100	5
citations	2500	350000	100	4

The **food web** [24] comprises 154 organisms that live on the Scotch Broom (a common kind of shrub). Here are two associations we discovered:

$$\begin{tikzcd}
\underline{(x_1,x_2,x_3,x_4,x_5)} & & \underline{(0,x_2,x_3,x_4,x_5)} \\
x_1 \arrow[d] & & 0 \arrow[d] \\
x_2 \arrow[d] & 45\% & x_2 \arrow[d] \\
x_3 \arrow[d] & \Rightarrow & x_3 \arrow[d] \\
x_4 \arrow[d] & & x_4 \arrow[d] \\
x_5 & & x_5
\end{tikzcd}$$

$$\begin{tikzcd}
 & & \underline{(x_1,x_2,x_3,x_4,x_5)} \\
\underline{(x_1,x_2,x_3,x_4,x_5)} & & 0 \arrow[d] \\
x_1 \arrow[d] & & x_1 \arrow[d] \\
x_2 \arrow[d] & 55\% & x_2 \arrow[d] \\
x_3 \arrow[d] & \Rightarrow & x_3 \arrow[d] \\
x_4 \arrow[d] & & x_4 \arrow[d] \\
x_5 & & x_5
\end{tikzcd}$$

Since $45\% + 55\% = 100\%$, these rules together say that each path of length 5 either starts in 0, or one beneath 0. This tells us that the depth of the food web equals 6. Constant 0 turns out to denote the Scotch Broom itself, which is the root of the food web.

Another rule we mined, just to give a rather arbitrary example of the kind of rules we find with our algorithm, is the following:

$$\begin{tikzcd}
 & \underline{(x_1,x_2,x_3,x_4,x_5)} & & & & \underline{(x_1,x_2,x_4,x_2,x_5)} & \\
 & x_1 \arrow[dl] \arrow[dr] & & 11\% & & x_1 \arrow[d] & \\
x_2 \arrow[d] & & x_4 \arrow[d] & \Rightarrow & & x_2 \arrow[dl] \arrow[d] \arrow[dr] & \\
x_3 & & x_5 & & 101 & x_4 & x_5
\end{tikzcd}$$

The **protein interactions graph** [18] comprises molecular interactions (symmetric) among 1870 proteins occurring in the yeast *Saccharomyces cervisae*. We found the following rule:

$$\begin{tikzcd}
\underline{(x_1,x_2)} & & & \underline{(x_1,x_2)} & \\
x_1 \arrow[d] & 90\% & & x_1 \arrow[d] & \\
x_2 \arrow[d] & \Rightarrow & & x_2 \arrow[dl] \arrow[dr] & \\
746 & & 746 & & 376
\end{tikzcd}$$

This rule expresses that almost all interactions that link to protein 746 also link to protein 376, which unveils a close relationship between these two proteins.

The **citation graph** comes from the KDD cup 2003, and contains around 2500 papers about high-energy physics taken from arXiv.org, with around 350 000 citations among these papers. One of the discovered rules is the following:

$$\begin{tikzcd}
 & \underline{(x_1,x_2)} & & & \underline{(x_1,x_2)} \\
 & x_1 \arrow[dl] \arrow[dr] & & 15\% & x_1 \arrow[d] \\
\exists \arrow[d] & & x_2 & \Rightarrow & x_2 \arrow[d] \\
\exists & & & & 9503124
\end{tikzcd}$$

This rule shows that paper 9503124 is an important paper. In 15% of all "non-trivial" citations (meaning that the citing paper cites at least one paper that also cites a paper), the cited paper cites 9503124.

Acknowledgment

We thank Bart Goethals, Jan Hidders, and Dries Van Dyck for fruitful discussions.

References

[1] *Proceedings of the 2002 IEEE International Conference on Data Mining (ICDM 2002), 9-12 December 2002, Maebashi City, Japan*. IEEE Computer Society Press, 2002.

[2] S. Abiteboul, R. Hull, and V. Vianu. *Foundations of Databases*. Addison-Wesley, 1995.

[3] R. Agrawal, H. Mannila, R. Srikant, H. Toivonen, and A. Verkamo. Fast discovery of association rules. In Fayyad et al. [10], pages 307–328.

[4] A. Aho, J. Hopcroft, and J. Ullman. *The Design and Analysis of Computer Algorithms*. Addison-Wesley, 1974.

[5] A. Chandra and P. Merlin. Optimal implementation of conjunctive queries in relational data bases. In *Proceedings 9th ACM Symposium on the Theory of Computing*, pages 77–90. ACM Press, 1977.

[6] Y. Chi, Y. Yang, and R. R. Muntz. Canonical forms for labelled trees and their applications in frequent subtree mining. *Knowl. Inf. Syst.*, 8(2):203–234, 2005.

[7] D. Cook and L. Holder. Substructure discovery using minimum description length and background knowledge. *Journal of Artificial Intelligence Research*, 1:231–255, 1994.

[8] L. Dehaspe and H. Toivonen. Discovery of frequent Datalog patterns. *Data Mining and Knowledge Discovery*, 3(1):7–36, 1999.

[9] L. Dehaspe and H. Toivonen. Discovery of relational association rules. In S. Dzeroski and N. Lavrac, editors, *Relational Data Mining*, pages 189–212. Springer-Verlag, 2001.

[10] U. Fayyad, G. Piatetsky-Shapiro, P. Smyth, and R. Uthurusamy, editors. *Advances in Knowledge Discovery and Data Mining*. MIT Press, 1996.

[11] S. Ghazizadeh and S. Chawathe. SEuS: Structure extraction using summaries. In S. Lange, K. Satoh, and C. Smith, editors, *Discovery Science*, volume 2534 of *Lecture Notes in Computer Science*, pages 71–85. Springer, 2002.

[12] B. Goethals, E. Hoekx, and J. Van den Bussche. Mining tree queries in a graph. In R. L. Grossman, R. Bayardo, K. Bennett, and J. Vaidya, editors, *Proceedings of the Eleventh ACM SIGKDD International Conference on Knowledge Discovery and Data Mining.*, pages 61–69. ACM, 2005.

[13] B. Goethals and J. Van den Bussche. Relational association rules: getting warmer. In D. Hand, R. Bolton, and N. Adams, editors, *Proceedings of the ESF Exploratory Workshop on Pattern Detection and Discovery in Data Mining*, volume 2447 of *LNCS*, pages 125–139. Springer-Verlag, 2002.

[14] T. Horvath, J. Ramon, and S. Wrobel. Frequent subgraph mining in outerplanar graphs. KDD 2006, to appear.

[15] J. Huan, W. Wang, and J. Prins. Efficient mining of frequent subgraphs in the presence of isomorphism. In *Proceedings of the 3rd IEEE International Conference on Data Mining (ICDM 2003)*, pages 549–552. IEEE Computer Society Press, 2003.

[16] A. Inokuchi, T. Washio, and H. Motoda. An Apriori-based algorithm for mining frequent substructures from graph data. In D. Zighed, H. Komorowski, and J. Zytkow, editors, *PKDD*, volume 1910 of *Lecture Notes in Computer Science*, pages 13–23. Springer, 2000.

[17] G. Jeh and J. Widom. Mining the space of graph properties. In W. Kim, R. Kohavi, J. Gehrke, and W. DuMouchel, editors, *Proceedings of the Tenth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 187–196. ACM Press, 2004.

[18] H. Jeong, S. Mason, et al. Lethality and centrality in protein networks. *Nature*, 411(3 May 2001).

[19] M. Kuramochi and G. Karypis. Frequent subgraph discovery. In N. Cercone, T. Lin, and X. Wu, editors, *Proceedings of the 2001 IEEE International Conference on Data Mining (ICDM 2001)*, pages 313–320. IEEE Computer Society Press, 2001.

[20] M. Kuramochi and G. Karypis. Finding frequent patterns in a large sparse graph*. *Data Mining and Knowledge Discovery*, 11(3):243–271, 2005.

[21] G. Li and F. Ruskey. The advantages of forward thinking in generating rooted and free trees. In *Proceedings 10th Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 939–940, 1999.

[22] H. Mannila and H. Toivonen. Levelwise search and borders of theories in knowledge discovery. *Data Mining and Knowledge Discovery*, 1(3):241–258, 1997.

[23] B. D. McKay. Practical graph isomorphism. *Congressus Numerantium*, 30:45–87, 1981.

[24] J. Memmott, N. Martinez, and J. Cohen. Predators, parasites and pathogens: species richness, trophic generality, and body sizes in a natural food web. *Journal of Animal Ecology*, 69:1–15, 2000.

[25] H. Scions. Placing trees in lexicographic order. In D. Michie, editor, *Machine Intelligence 3*, pages 43–62. Edinburgh University Press, 1968.

[26] W.-M. Shen, K. Ong, B. Mitbander, and C. Zaniolo. Metaqueries for data mining. In Fayyad et al. [10], pages 375–398.

[27] S. Tsur, J. Ullman, et al. Query flocks: A generalization of association-rule mining. In *Proceedings of the 1998 ACM SIGMOD International Conference on Management of Data*, volume 27:2 of *SIGMOD Record*, pages 1–12. ACM Press, 1998.

[28] J. Ullman. *Principles of Database and Knowledge-Base Systems*, volume II. Computer Science Press, 1989.

[29] N. Vanetik, E. Gudes, and S. Shimony. Computing frequent graph patterns from semistructured data. In *Proceedings of the 2002 IEEE International Conference on Data Mining (ICDM 2002)* [1], pages 458–465.

[30] X. Yan and J. Han. gSpan: Graph-based substructure pattern mining. In *Proceedings of the 2002 IEEE International Conference on Data Mining (ICDM 2002)* [1], pages 721–724.

[31] M. J. Zaki. Efficiently mining frequent trees in a forest: Algorithms and applications. *IEEE Transactions on Knowledge and Data Engineering*, 17(8):1021–1035, 2005.