# Local Query Mining in a Probabilistic Prolog

**Angelika Kimmig and Luc De Raedt**
Department of Computer Science, Katholieke Universiteit Leuven
Celestijnenlaan 200A - bus 2402, 3001 Heverlee, Belgium
{angelika.kimmig,luc.deraedt}@cs.kuleuven.be

## Abstract

Local pattern mining is concerned with finding the set of patterns that satisfy a constraint in a database. We study local pattern mining in the context of ProbLog, a probabilistic Prolog system, and introduce an approach for finding correlated patterns in the form of queries in such a Prolog system. The approach combines principles of inductive logic programming, data mining and statistical relational learning. Experiments on a challenging biological network mining task provide evidence for the interestingness of the approach.

## 1 Introduction

The traditional *local pattern mining* task is that of identifying those elements in a language of patterns that satisfy the constraints imposed by a *selection predicate* w.r.t. a given database [Mannila and Toivonen, 1997]. Numerous works have been devoted to local pattern mining since the introduction of item-set mining, cf. [Agrawal *et al.*, 1996]. They can be distinguished alongst three main dimensions. The first is concerned with the type of pattern considered, that is whether one mines for item-sets, episodes and strings, graphs or relational patterns. The second is concerned with the nature of the selection predicate applied [Ng *et al.*, 1998], such as whether one employs frequency, confidence, significance, lift, closedness, freeness, and so forth. Finally, the third distinguishes two types of desired answer sets: all patterns covered by the selection predicate in the case of frequent pattern mining [Agrawal *et al.*, 1996], or the best $k$ patterns in the case of correlated pattern mining [Morishita and Sese, 2000].

In many domains, probabilistic data arises quite naturally, especially in scientific data or in areas such as robotics, and transferring local pattern mining techniques to *probabilistic databases* allows to take the uncertainty attached to such data into account. The main difference in this setting is the nature of the membership relation determining when a query matches a tuple: it changes from deterministic to probabilistic. So, rather than having a 0-1 criterion, there is now a probability of a tuple belonging to a relation.

This paper investigates the mining of local relational patterns in probabilistic databases. To apply and evaluate our work we employ a large biological database [Sevon *et al.*, 2006; Kimmig *et al.*, 2007; De Raedt *et al.*, 2007]. This database contains probabilistic information about known or predicted relationships for various types of objects, such as genes, proteins, and molecular functions.

Throughout the paper we employ ProbLog [De Raedt *et al.*, 2007] to represent the database. ProbLog is a recently introduced extension of Prolog and Datalog, and the underlying distribution semantics is identical to that of many proposals in the database and artificial intelligence literature, cf. [Dalvi and Suciu, 2004; Fuhr, 2000; Sato and Kameya, 2001]. Even though we employ ProbLog in the present paper, the techniques and principles are directly transferable to other probabilistic formalisms allowing to calculate probabilities for queries. ProbLog makes two key assumptions: 1) tuples in the database are assigned a probability value, and 2) these probabilities are mutually independent. As ProbLog is a direct extension of Prolog, our formulation and implementation of local pattern mining builds upon existing work in multi-relational data mining and inductive logic programming, where local pattern mining is known under the names of *query mining* [Dehaspe *et al.*, 1998] and *query flocks* [Tsur *et al.*, 1998]. Queries in a (probabilistic) relational database form a very general type of pattern that can emulate many other pattern domains, such as item-sets, sequences, trees and, as in our motivating application, graphs. Using ProbLog, we define *probabilistic query mining* in a general way and then focus on correlated pattern mining, where the goal is to find the top $k$ queries. We develop a branch-and-bound algorithm to solve this task. The key difference with standard local pattern mining is the use of a probabilistic membership function. To this aim, we consider two measures: a kind of likelihood criterion based on a set of positive and negative examples, as well as an adaptation of the frequency measure along the lines of [Chui *et al.*, 2007], who use this only in the context of frequent item-set mining. Our work generalizes that of [Chui *et al.*, 2007] in two ways: we use *relational* databases, and we consider correlated pattern mining.

This paper is organised as follows: We discuss query mining in Section 2, review the basics of ProbLog in Section 3 and introduce probabilistic query mining in Section 4. Section 5 describes our implementation used in the experiments presented in Section 6. Finally, in Section 7 we conclude and touch upon related work.

## 2 Query Mining

Query mining upgrades traditional local pattern mining to the representations of multi-relational databases [Dehaspe *et al.*, 1998]. We use Datalog to represent databases and queries, abbreviating vectors of variables as $\vec{X}$. We assume a designated relation $ID$ containing the set of tuples or examples to be characterized using queries, and restrict the language $\mathcal{L}$ of patterns to the set of conjunctive queries $r(\vec{X})$ defined as

$$r(\vec{X}) : -ID(\vec{X}), l_1, ..., l_n \quad (1)$$

where the $l_i$ are positive atoms. Additional syntactic or semantic restrictions, called *bias*, can be imposed on the form of queries by explicitly specifying the language $\mathcal{L}$, cf. [Tsur *et al.*, 1998; De Raedt and Ramon, 2004; Dehaspe *et al.*, 1998].

*Query Mining* aims at finding all queries satisfying a selection predicate $\phi$. It can be formulated as follows, cf. [Dehaspe *et al.*, 1998; De Raedt and Ramon, 2004]:

**Given** a language $\mathcal{L}$ containing queries of the form (1), a database $\mathcal{D}$ including the designated relation $ID$, and a selection predicate $\phi$

**Find** all queries $q \in \mathcal{L}$ such that $\phi(q, \mathcal{D}) = true$.

The most prominent selection predicate is minimum frequency, an anti-monotonic predicate, requiring a minimum number of tuples covered. Anti-monotonicity is based on a generality relation between patterns. We employ $OI$-subsumption [Esposito *et al.*, 2000], as the corresponding notion of subgraph isomorphism is favorable within the intended application in network mining. More formally, a conjunctive query $q_1$ represented as a set of literals $OI$-subsumes a conjunctive query $q_2$, notation $q_1 \preceq q_2$, if and only if there exists a substitution $\theta = \{V_1/t_1, \cdots, V_n/t_n\}$ such that $q_1\theta \subseteq q_2$ and the $t_i$ are different constants not occurring in $q_1$.

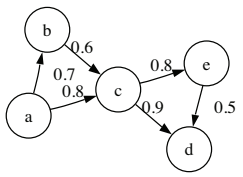
**Example 2.1** *For queries Q1 to Q3 below, query Q1 $OI$-subsumes Q2, but neither Q1 nor Q2 $OI$-subsumes Q3: for Q1, $Y$ and $Z$ cannot both be mapped to $c$; for Q2, $Y$ cannot be mapped to $c$ as $Q2$ already uses $c$.*

$$(Q1) \qquad q(X) : -ID(X), edge(X, Y), edge(Y, Z).$$
$$(Q2) \qquad q(X) : -ID(X), edge(X, Y), edge(Y, c).$$
$$(Q3) \qquad q(X) : -ID(X), edge(X, c), edge(c, c).$$

*Correlated Pattern Mining* [Morishita and Sese, 2000] uses both *positive* and *negative* examples, specified as two designated relations $ID^+$ and $ID^-$ of the same arity, to find the top $k$ patterns, that is, the $k$ patterns scoring best w.r.t. a function $\psi$. The function $\psi$ employed is convex, e.g. measuring a statistical significance criterion such as $\chi^2$, cf. [Morishita and Sese, 2000], and measures the degree to which the pattern is statistically significant or unexpected. Thus correlated pattern mining corresponds to setting

$$\phi(q, \mathcal{D}) = q \in \arg_k \max_{q \in \mathcal{L}} \psi(q, \mathcal{D})$$

**Example 2.2** *Consider the database in Figure 1 (ignoring probability labels) with $ID^+ = \{a, c\}$ and $ID^- = \{d, e\}$. A simple correlation function is $\psi(q, \mathcal{D}) = \mathsf{COUNT}(\mathsf{q}^+(*)) - \mathsf{COUNT}(\mathsf{q}^-(*))$, where $\mathsf{COUNT}(\mathsf{q}(*))$ is the number of different provable ground instances of $q$ and $q^x$ denotes query $q$ restricted to $ID^x$. We obtain $\psi(Q4, \mathcal{D}) = 2 - 0 = 2$ and $\psi(Q5, \mathcal{D}) = 1 - 1 = 0$ for queries*




```
0.8::edge(a,c).
0.7::edge(a,b).
0.6::edge(b,c).
0.9::edge(c,d).
0.8::edge(c,e).
0.5::edge(e,d).
```

Figure 1: Example: probabilistic database.

$$(Q4) \qquad q(X) : -ID(X), edge(X, Y), edge(Y, Z).$$
$$(Q5) \qquad q(X) : -ID(X), edge(X, d).$$

*Multi-relational query miners* such as [Dehaspe *et al.*, 1998; De Raedt and Ramon, 2004] often follow a levelwise approach for frequent query mining [Mannila and Toivonen, 1997], where at each level new candidate queries are generated from the frequent queries found on the previous level. In contrast to Apriori, instead of a "joining" operation, they employ a refinement operator $\rho$ to compute more specific queries, and also manage a set of infrequent queries to take into account the specific language requirements imposed by $\mathcal{L}$. To search for all solutions, it is essential that the refinement operator is optimal w.r.t. $\mathcal{L}$, i.e. ensures that there is exactly one path from the most general query to every query in the search space. This can be achieved by restricting the refinement operator to generate queries in a canonical form, cf. [De Raedt and Ramon, 2004].

Morishita and Sese [2000] adapt Apriori for finding the top $k$ patterns w.r.t. a boundable function $\psi$, i.e. for the case where there exists a function $u$ (different from a global maximum) such that $\forall g, s \in \mathcal{L} : g \preceq s \rightarrow \psi(s) \leq u(g)$.

**Example 2.3** *The function $\psi(q, \mathcal{D}) = \mathsf{COUNT}(\mathsf{q}^+(*)) - \mathsf{COUNT}(\mathsf{q}^-(*))$ introduced in Example 2.2 is upper-boundable using $u(q, \mathcal{D}) = \mathsf{COUNT}(\mathsf{q}^+(*))$. For any $g \preceq s$, $\psi(s) \leq \mathsf{COUNT}(\mathsf{s}^+(*)) \leq \mathsf{COUNT}(\mathsf{g}^+(*))$, as $\mathsf{COUNT}(\mathsf{s}^-(*)) \geq 0$ and $\mathsf{COUNT}$ is anti-monotonic.*

Again, at each level candidate queries are obtained from those queries generated at the previous level that qualify for refinement, which now means they either belong to the current $k$ best queries, or are still promising as their upper-bound is higher than the value of the current $k$-th best query.

**Example 2.4** *Assume we mine for the 3 best correlated queries in Example 2.2. Table 1 shows counts on $ID^+$ and $ID^-$ and $\psi$-values obtained during the first level of mining. The highest score achieved is 1. Queries 1, 3, 4, 6, 8, 9 are the current best queries and will thus be refined on the next level. Queries 5 and 7 have lower scores, but upper bound $c^+ = 1$, implying that their refinements may still belong to the best queries and have to be considered on the next level as well. The remining queries are pruned, as they all have an upper bound $c^+ = 0 < 1$, i.e. all their refinements are already known to score lower than the current best queries.*

The framework for query mining as outlined above can directly be adapted towards *probabilistic* databases. The key changes involved are 1) that the database $\mathcal{D}$ is *probabilistic*,

| | query | $c^+$ | $c^-$ | $\psi$ |
|---|---|---|---|---|
| 1 | `ID(X),edge(X,Y)` | **2** | 1 | **1** |
| 2 | `ID(X),edge(X,a)` | 0 | 0 | 0 |
| 3 | `ID(X),edge(X,b)` | **1** | 0 | **1** |
| 4 | `ID(X),edge(X,c)` | **1** | 0 | **1** |
| 5 | `ID(X),edge(X,d)` | **1** | 1 | 0 |
| 6 | `ID(X),edge(X,e)` | **1** | 0 | **1** |
| 7 | `ID(X),edge(Y,X)` | **1** | 2 | - 1 |
| 8 | `ID(X),edge(a,X)` | **1** | 0 | **1** |
| 9 | `ID(X),edge(b,X)` | **1** | 0 | **1** |
| 10 | `ID(X),edge(c,X)` | 0 | 2 | -2 |
| 11 | `ID(X),edge(d,X)` | 0 | 0 | 0 |
| 12 | `ID(X),edge(e,X)` | 0 | 1 | -1 |

Table 1: Counts on $ID^+$ and $ID^-$ and $\psi$-values obtained during the first level of mining in Example 2.4. The current minimal score for best queries is 1, i.e. only queries with $\psi \geq 1$ or $c^+ \geq 1$ will be refined on the next level.

and 2) that the selection predicate $\phi$ or the correlation measure $\psi$ is based on the probabilities of queries. Throughout this paper we shall employ a recently developed extension of Prolog, called ProbLog [De Raedt *et al.*, 2007], as it is a very simple logic that has already been applied to challenging tasks in biological network mining.

## 3 ProbLog: A Probabilistic Prolog

A *ProbLog program* $T$ consists of a set of labeled facts $p_i :: c_i$ together with a set of definite clauses encoding *background knowledge* (BK). Each ground instance of such a fact $c_i$ is true with probability $p_i$, where all probabilities are assumed mutually independent. The program thus naturally defines a probability distribution $P(L|T) = \prod_{c_i \in L} p_i \prod_{c_i \in L_T \setminus L}(1-p_i)$ over logic programs $L \subseteq L_T = \{c_1, \cdots, c_n\}$. It can be used to specify two types of query probabilities:

$$P_s(q|T) = \sum_{L \subseteq L_T} P(q|L) \cdot P(L|T) \quad (2)$$

$$P_x(q|T) = \max_{e \in E(q)} P(e|T) = \max_{e \in E(q)} \prod_{c_i \in e} p_i \quad (3)$$

where $P(q|L) = 1$ if there exists a $\theta$ such that $L \cup BK \models q\theta$, $P(q|L) = 0$ otherwise, and $E(q) = \{L \subseteq L_T | P(q|L) = 1 \wedge \forall M \subset L : P(q|M) = 0\}$ is the set of all explanations or proofs for query $q$ in the logic program $L_T \cup BK$ [Kimmig *et al.*, 2007]. The *success probability* $P_s(q|T)$ thus corresponds to the probability that $q$ is *provable* in a randomly sampled logic program, the *explanation probability* $P_x(q|T)$ to that of sampling all clauses needed in the most likely proof.

**Example 3.1** *We extend our example (Figure 1) with background knowledge defining a path:*

```
path(X,Y) :- edge(X,Y).
path(X,Y) :- edge(X,Z),path(Z,Y).
```

*Now, the success probability of query* `path(a,c)` *is obtained as the sum over all subgraphs having a path from* `a` *to* `c`*, i.e. where either* `edge(a,c)` *is true or* `edge(a,c)` *is false while both* `edge(a,b)` *and* `edge(b,c)` *are true:* $P_s(\texttt{path(a,c)}) = 0.8 + (1-0.8)\cdot 0.7 \cdot 0.6 = 0.884$. *The explanation probability is* $P_x(\texttt{path(a,c)}) = 0.8$*, as the query has possible explanations* `edge(a,c)` *with probability* $0.8$ *and* `edge(a,b),edge(b,c)` *with probability* $0.7 \cdot 0.6 = 0.42$.

Evaluating $P_s$ is computationally hard due to the disjoint-sum problem, cf. [De Raedt *et al.*, 2007], who also propose an approximation algorithm that repeatedly computes bounds on $P_s$ until their difference becomes sufficiently small. $P_x$ can easily be calculated using a best-first search.

## 4 Probabilistic Query Mining

To mine for probabilistic queries, we will adapt the local pattern mining algorithms sketched in Section 2 by changing the selection predicate $\phi$ or correlation measure $\psi$ to work with probabilistic databases. The key idea is to employ a probabilistic membership function. In non-probabilistic frequent query mining, every tuple in the relation $ID$ either satisfies the query or not. So, for a conjunctive query $q$ and a 0-1 membership function $M(t|q, \mathcal{D})$, we can explicitly write the counting function underlying frequency as a sum:

$$freq(q, \mathcal{D}) = \sum_{t \in ID} M(t|q, \mathcal{D})$$

On a more general level, this type of function can be seen as *aggregate* of the membership function $M(t|q, \mathcal{D})$.

To apply the algorithms sketched in Section 2 with a probabilistic database $\mathcal{D}$, it suffices to replace the deterministic membership function $M(t|q, \mathcal{D})$ with a probabilistic variant. Possible choices for such a probabilistic membership function $P(t|q, \mathcal{D})$ include the success probability $P_s(q(t)|\mathcal{D})$ or the explanation probability $P_x(q(t)|\mathcal{D})$ as introduced for ProbLog in Equations (2) and (3). Note that using such query probabilities as probabilistic membership function is anti-monotonic, that is, if $q_1 \preceq q_2$ then $P(t|q_1, \mathcal{D}) \geq P(t|q_2, \mathcal{D})$. Again, a natural choice of selection predicate $\phi$ is the combination of a minimum threshold with an aggregated probabilistic membership function:

$$agg(q, \mathcal{D}) = \mathbf{AGG}_{t \in ID}\, P(t|q, \mathcal{D}). \quad (4)$$

Here, **AGG** denotes an aggregate function such as $\sum$, min, max or $\prod$, which is to be taken over all tuples $t$ in the relation $ID$. Choosing $\sum$ with a deterministic membership relation corresponds to the traditional frequency function, whereas $\prod$ computes a kind of *likelihood* of the data. Note that whenever the membership function $P$ is anti-monotone, selection predicates of the form $agg(q, \mathcal{D}) > c$ (with $agg \in \{\sum, \min, \max, \prod\}$) are anti-monotonic with regard to $OI$-subsumption, which is crucial to enable pruning.

When working with both positive and negative examples, the main focus lies on finding queries with a high aggregated score on the positives and a low aggregated score on the negatives. Note that using unclassified instances $ID$ corresponds to the special case where $ID^+ = ID$ and $ID^- = \emptyset$. In the following, we will therefore consider instances of the selection function (4) for the case of classified examples $ID^+$ and $ID^-$ only. Choosing sum as aggregation function results in a *probabilistic frequency* $pf$ (5) also employed by [Chui *et al.*, 2007] in the context of item-set mining, whereas product defines a kind of *likelihood* $LL$ (6). Notice that using the product in combination with a non-zero threshold implies that

*all* positive examples must be covered with non-zero probability. We therefore introduce a softened version $LL_n$ (7) of the likelihood, where $n < |ID^+|$ examples have to be covered with non-zero probability. This is achieved by restricting the set of tuples in the product to the $n$ highest scoring tuples in $ID^+$, thus integrating a deterministic (anti-monotonic) selection predicate into the probabilistic one. More formally, the three functions used are defined as follows:

$$pf(q,\mathcal{D}) = \sum_{t\in ID^+} P(t|q,\mathcal{D}) - \sum_{t\in ID^-} P(t|q,\mathcal{D}) \quad (5)$$

$$LL(q,\mathcal{D}) = \prod_{t\in ID^+} P(t|q,\mathcal{D}) \cdot \prod_{t\in ID^-} (1-P(t|q,\mathcal{D})) \quad (6)$$

$$LL_n(q,\mathcal{D}) = \prod_{t\in ID_n^+} P(t|q,\mathcal{D}) \cdot \prod_{t\in ID^-} (1-P(t|q,\mathcal{D})) \quad (7)$$

Here, $ID_n^+$ contains the $n$ highest scoring tuples in $ID^+$. In correlated query mining, we obtain an upper bound on each of these functions by omitting the scores of negative examples, i.e. the aggregation over $ID^-$.

**Example 4.1** *Consider the database of Example 2.2, now with probabilities. Using $P_x$ as probabilistic membership function, the query $q(X) : -ID(X), edge(X,Y)$ gets probabilistic frequency $pf(q,\mathcal{D}) = P_x(a|q,\mathcal{D}) + P_x(c|q,\mathcal{D}) - (P_x(d|q,\mathcal{D}) + P_x(e|q,\mathcal{D})) = 0.8 + 0.9 - (0 + 0.5) = 1.2$ (with upper bound $0.8 + 0.9 = 1.7$), likelihood $LL(q,\mathcal{D}) = 0.8 \cdot 0.9 \cdot (1-0) \cdot (1-0.5) = 0.36$ (with upper bound $0.8 \cdot 0.9 = 0.72$), and softened likelihood $LL_1(q,\mathcal{D}) = 0.9 \cdot (1-0) \cdot (1-0.5) = 0.9$ (with upper bound $0.9$).*

## 5 Implementation

Our implementation of correlated query mining is built upon and extends the public version of ProbLog as well as the public domain implementation of the (non-probabilistic) frequent query mining system *c-armr* of [De Raedt and Ramon, 2004].

As in *c-armr*, the language bias can be defined using type and mode restrictions as well as background knowledge. This reduces the number of queries generated by taking advantage of general knowledge about the domain of interest. As each query is evaluated on all training examples in turn, we prune query evaluation as soon as the current upper bound of its aggregated score falls below the threshold, where we include maximum estimates for positive examples not processed yet.

For correlated query mining we further modified *c-armr*. First, to deal with positive and negative examples, we keep track of queries that are infrequent, but cannot be pruned as their upper bound is still promising. Second, we modified the search strategy to dynamically adapt the threshold to the score of the $k$th best query whenever the set of $k$ best queries found so far changes, thereby allowing for more pruning.

## 6 Experiments

To evaluate our work, we report on experiments in the context of the probabilistic biological database of [Sevon *et al.*, 2006] containing a large number of biological entities (such as genes, proteins, tissues) as well as probabilistic information on various kinds of relationships. It has been used before in the context of ProbLog, cf. [De Raedt *et al.*, 2007; Kimmig *et al.*, 2007]. Even though the database is very large, at any point in time, a biologist will typically focus on a particular phenomenon for which only a limited number of nodes is known to be relevant. As a test-case to be studied, we therefore use the 142 genes known to be related to Alzheimer disease contained in our database. We set up experiments to answer the following questions about correlated query mining:

| | nodes | edges | all genes | AD genes |
|---|---|---|---|---|
| G0 | 936641 | 5967842 | 181026 | 142 |
| G1 | 658 | 3544 | 37 | 17 |
| G2 | 3364 | 17666 | 185 | 130 |

Table 2: Graph characteristics: numbers of nodes, edges, gene nodes, and genes annotated with AD.

**Q1** How do $P_s$ and $P_x$ differ in performance?

**Q2** Can the top queries discriminate unseen positive and negative examples?

**Q3** Does the correlated query miner prune effectively?

**Q4** Can the correlated query miner use the full network?

To answer these questions, we used three graphs: the full network G0 of roughly 6 million edges as well as two connected subgraphs around the phenotype representing Alzheimer disease. To obtain the nodes of interest, we searched Entrez for human genes with the relevant annotation (AD). Weights were assigned to edges as described in [Sevon *et al.*, 2006]. We used a fixed number of genes to extract the subgraphs by taking all acyclic paths with maximum length of 5, with a probability of at least 0.01, between the phenotype and any of these genes. We excluded articles since they would dominate the subgraphs. Table 2 gives a summary of the properties of all three graphs, including information on the number of gene nodes and their annotation status w.r.t. AD. Graph G1 was obtained using 17 randomly chosen Alzheimer disease genes. G2 was extracted starting with all 142 annotated Alzheimer disease genes, but as we only use the connected component around the phenotype, some of them do not occur in G2. Each graph is represented using one probabilistic relation describing edges in terms of two nodes and a label (e.g. `contains`), and a deterministic relation assigning a label (e.g. `'Gene'`) to each node. The language bias employed allows to add literals of the form `edge(X,Y,e)`, `edge(X,Y)` (as a shortcut of `edge(X,Y,_)`) and `node(X,n)` where `X` and `Y` are variables of type node name, `X` already appears in the query (thereby ensuring linkage), and `e` and `n` are constants denoting an edge and a node label appearing in the graph respectively. Note that in contrast to the running example used for illustration, we do not allow node names as constants in the query language here, as this would entail prohibitively many possible refinements for each query. The bias further states that labels are mutually exclusive, that `edge(X,Y,e)` implies `edge(X,Y)`, and how to invert labels when using edges backwards. This ensures that edges in queries map to database entries independently of direction. We use a query reordering function greedily moving literals containing constants to the left where possible.

Training examples are gene nodes annotated (positive) resp. not annotated with AD (negative) randomly picked from

| | $LL$ | $LL_n$ | $pf$ |
|---|---|---|---|
| $P_s$ | .72/.45/.33 | .27/.02/.00 | .13/.03/.00 |
| $P_x$ | 1/1/.45 | 1/1/1 | 1/1/1 |

Table 3: Fraction of cases where mining for $k = 1/5/20$ best queries successfully terminated within 30 minutes.

| | $LL^s$ | $LL^s_n$ | $pf^s$ | $LL^x$ | $LL^x_n$ | $pf^x$ |
|---|---|---|---|---|---|---|
| precision | 0.76 | 0.95 | 0.92 | 0.77 | 0.93 | 0.93 |
| recall | 0.94 | 0.79 | 0.81 | 1.00 | 0.85 | 0.86 |
| F-measure | 0.84 | 0.85 | 0.86 | 0.87 | 0.88 | 0.88 |

Table 4: Using query obtained with $k = 1$ to reason by analogy: Overall precision, recall and F-measure, averaged over cases with mining time at most 30 minutes.

| | $LL^s$ | $LL^s_n$ | $pf^s$ | $LL^x$ | $LL^x_n$ | $pf^x$ |
|---|---|---|---|---|---|---|
| (a) | 1/.99/.95 | 1/1/1 | 1/1/1 | 1/1/1 | 1/1/1 | 1/1/1 |
| (b) | 0.18 | 0.67 | 0.33 | 0.93 | 0.83 | 0.86 |

Table 5: Using query obtained with $k = 1$ to reason by analogy: (a) Precision among the first $n = 1/10/20$ ranked examples, (b) fraction of positives ranked before the first negative, averaged over cases with mining time at most 30 minutes.

| k | 1 | 5 | 10 | 20 | 50 |
|---|---|---|---|---|---|
| (a) | 225 | 626 | 983 | 1814 | 4057 |
| (b) | 1.25 | 6.44 | 32.46 | 98.37 | 1523.26 |

Table 6: Mining on G2: (a) average number of queries tested for datasets of size 10, (b) average runtimes (sec).

G1. We create ten sequences of size ten, i.e. containing ten examples of each class. For each such sequence, we use the first $i$ examples of each class to obtain datasets of varying size, leading to a total of 100 datasets. To avoid one trivial refinement step, `q(X):-id(X),node(X,'Gene')` is used as most general query.

As probabilistic membership function $P(t|q, \mathcal{D})$ we employ either the explanation probability $P_x$ or the lower bound of the exact probability $P_s$ obtained by the approximation algorithm with interval width $\delta = 0.1$ and a timelimit of 60 sec for the evaluation of each individual bound. As aggregation functions to obtain probabilistic selection predicates, we use the likelihood $LL$ (6), the probabilistic frequency $pf$ (5) and the softened likelihood $LL_n$ (7) with $n = \lceil m/2 \rceil$ for $m$ examples of each class, indicating the probability function used by superscripts where needed. All experiments are performed on 2.4 GHz 4GB machines running Linux.

To answer questions **Q1** and **Q2**, we mine on G1 for $k = 1, 5, 20$ with a timelimit of 30 minutes per run, using the 60 datasets with at least 5 examples of each class. Table 3 illustrates the performance in terms of the fraction of successful runs. In the case of $P_x$, the timelimit is only reached for $k = 20$ with $LL$, i.e. when a higher number of queries covering all positive examples is desired, whereas $P_s$ crosses the limit frequently in all settings. These results clearly suggest that $P_x$ is more favorable in terms of runtimes. To compare the two choices of probabilistic membership function $P$ in terms of their results, we use the highest scored query (omitting `id(X)`; note that scores are independent of $k$) to retrieve covered examples from the larger graph G2, and rank those using the corresponding $P$. In case of equally likely queries, we choose the most specific one. Note that due to the form of the most general query employed in mining, this will return nodes of type gene only. Training examples are excluded from the rankings. The fraction of annotated genes and thus positive examples among the possible answers is 0.76.

We calculate overall precision (percentage of ranked genes that are positive), recall (percentage of positive genes included in the ranking), and F-measure ($2 \cdot prec \cdot rec/(prec + rec)$) for all rankings. Table 4 gives the averages over all successful cases. Using $LL$ results in high recall and low precision (close to the fraction of positives among unseen genes), as queries covering all positive training examples are very general and therefore often cover all negative examples as well. Reducing the number of positives included in the score, i.e. using $LL_n$, and using probabilistic frequency both increase precision at the expense of lower recall. This tendency is confirmed by the F-measure, which balances precision and recall. Combining resource requirements and results, the answer to question **Q1** is thus that using the explanation probability is more favorable.

To study the predictive performance of best queries in more detail, we examine the top regions of the rankings used above. Table 5 shows the precision among the top ranked examples. All queries return several positive examples first, only in few cases, negatives occur within the first twenty positions. Furthermore, especially for the case of $P_x$, the queries mined return a large fraction of all positive examples before the first negative one. Together, these results show that the best queries are indeed able to distinguish unseen positive and negative examples, thus answering **Q2** affirmatively.

Comparing the different aggregation functions in terms of both runtime and results confirms that both $LL_n$ and $pf$ clearly outperform $LL$. As probabilistic frequency has the advantage of not requiring an extra parameter, we restrict ourselves to probabilistic frequency using $P_x$ in the following.

To compare the number of queries examined for various $k$ and thus answer **Q3**, we mine on the larger graph G2 with 17666 edges, again with $pf^x$ as selection predicate. Table 6 shows the number of queries tested and runtimes for various values of $k$, averaged over all datasets of size ten. The query language $\mathcal{L}_{G2}$ already contains roughly 50K elements of length at most 3. The size of the search space explored nicely scales with $k$, focussing on very small fractions of the entire search space, thereby answering **Q3** affirmatively.

Finally, as both the tasks of query mining and probabilistic reasoning in relational databases are computationally hard and are combined here, we also tested the algorithm with $pf^x$ on the entire network G0 with around 6M edges, using the ten largest datasets and a time limit of one hour. For $k = 1$, runtimes vary from 626 to 1578 seconds, with an average of 865. For $k = 2$, the seven runs finishing within the time limit take between 1701 and 3145 seconds, with an average of 2610. Runtimes for higher $k$ exceed the limit. These results indicate that although probabilistic relational query mining is computationally challenging, it is in principle possible to run

the algorithm on large scale networks for small values of $k$. Thus the answer to **Q4** is positive as well, although improving the efficiency of the probabilistic reasoning engine would help to further increase scalability.

## 7 Conclusions and Related Work

We have extended the frequent pattern mining paradigm towards a probabilistic Prolog and introduced a correlated query mining algorithm. We have also introduced various scoring functions aggregating probabilities. The resulting general techniques have been evaluated on challenging biological network mining tasks, where queries emulate subgraphs. The results have clearly shown that correlated query mining using $P_x$ is most effective in terms of both function and efficiency. The discovered correlated queries can be used to retrieve similar instances with high accuracy. Furthermore, the correlated query miner using $P_x$ scales well and is able to run on a very large biological network. Nevertheless, further work on improving the efficiency of this approach (and the underlying probabilistic Prolog) would be useful.

The probabilistic correlated query mining system introduced here extends existing multi-relational data mining systems such as Warmr [Dehaspe and Toivonen, 1999] and c-armr [De Raedt and Ramon, 2004] to deal with probabilistic data. This is useful when mining large network data. As far as we are aware, approaches to mining probabilistic data addressed only item-sets [Chui *et al.*, 2007; Zhang *et al.*, 2008]. The notion of expected support used to find frequent item-sets in [Chui *et al.*, 2007] corresponds to the probabilistic frequency of Equation (5). The focus of [Zhang *et al.*, 2008] lies on mining single items that are likely to be frequent in a random possible world. However, both approaches consider neither relational data nor correlated pattern mining. As a consequence, their algorithms are tailored to a specific type of queries, whereas relational query mining as considered here has to deal with a much broader class of possible queries that are more complex to evaluate.

Probabilistic explanation based learning (PEBL) [Kimmig *et al.*, 2007] is a related approach in that it also results in a set of patterns. However, it is also significantly different: patterns are obtained by generalizing the logical structure of proofs of the examples w.r.t. a domain theory defining a target predicate. PEBL thus searches a highly constrained space of possible patterns, and hence, it is more efficient to use, but also more restricted.

## Acknowledgments

A. Kimmig is supported by the Research Foundation-Flanders (FWO-Vlaanderen). This work is partially supported by the GOA project 2008/08 Probabilistic Logic Learning and the European Commission FP7 project BISON.

## References

[Agrawal *et al.*, 1996] R. Agrawal, H. Mannila, R. Srikant, H. Toivonen, and A. Inkeri Verkamo. Fast discovery of association rules. In *Advances in Knowledge Discovery and Data Mining*, pages 307–328. AAAI/MIT Press, 1996.

[Chui *et al.*, 2007] C. Kit Chui, B. Kao, and E. Hung. Mining frequent itemsets from uncertain data. In *PAKDD*, pages 47–58, 2007.

[Dalvi and Suciu, 2004] N. N. Dalvi and D. Suciu. Efficient query evaluation on probabilistic databases. In *VLDB*, pages 864–875, 2004.

[De Raedt and Ramon, 2004] L. De Raedt and J. Ramon. Condensed representations for inductive logic programming. In *KR*, pages 438–446, 2004.

[De Raedt *et al.*, 2007] L. De Raedt, A. Kimmig, and H. Toivonen. ProbLog: A probabilistic Prolog and its application in link discovery. In *IJCAI*, pages 2462–2467, 2007.

[Dehaspe and Toivonen, 1999] L. Dehaspe and H. Toivonen. Discovery of frequent datalog patterns. *Data Min. Knowl. Discov.*, 3(1):7–36, 1999.

[Dehaspe *et al.*, 1998] L. Dehaspe, H. Toivonen, and R. D. King. Finding frequent substructures in chemical compounds. In *KDD*, pages 30–36, 1998.

[Esposito *et al.*, 2000] F. Esposito, N. Fanizzi, S. Ferilli, and G. Semeraro. Ideal theory refinement under object identity. In *ICML*, pages 263–270, 2000.

[Fuhr, 2000] N. Fuhr. Probabilistic Datalog: Implementing logical information retrieval for advanced applications. *JASIS*, 51(2):95–110, 2000.

[Kimmig *et al.*, 2007] A. Kimmig, L. De Raedt, and H. Toivonen. Probabilistic explanation based learning. In *ECML*, pages 176–187, 2007.

[Mannila and Toivonen, 1997] H. Mannila and H. Toivonen. Levelwise search and borders of theories in knowledge discovery. *Data Min. Knowl. Discov.*, 1(3):241–258, 1997.

[Morishita and Sese, 2000] S. Morishita and J. Sese. Traversing itemset lattice with statistical metric pruning. In *PODS*, pages 226–236, 2000.

[Ng *et al.*, 1998] R. T. Ng, L. V. S. Lakshmanan, J. Han, and A. Pang. Exploratory mining and pruning optimizations of constrained association rules. In *SIGMOD*, pages 13–24, 1998.

[Sato and Kameya, 2001] T. Sato and Y. Kameya. Parameter learning of logic programs for symbolic-statistical modeling. *J. Artif. Intell. Res. (JAIR)*, 15:391–454, 2001.

[Sevon *et al.*, 2006] P. Sevon, L. Eronen, P. Hintsanen, K. Kulovesi, and H. Toivonen. Link discovery in graphs derived from biological databases. In *DILS*, pages 35–49, 2006.

[Tsur *et al.*, 1998] S. Tsur, J. D. Ullman, S. Abiteboul, C. Clifton, R. Motwani, S. Nestorov, and A. Rosenthal. Query flocks: A generalization of association-rule mining. In *SIGMOD*, pages 1–12, 1998.

[Zhang *et al.*, 2008] Q. Zhang, F. Li, and K. Yi. Finding frequent items in probabilistic data. In *SIGMOD*, pages 819–832, 2008.