Proper Refinement of Datalog Clauses using Primary Keys

Siegfried Nijssen Joost N. Kok

LIACS, Leiden University,
Niels Bohrweg 1, 2333 CA, Leiden, The Netherlands
{snijssen,joost}@liacs.nl

Abstract

Inductive Logic Programming (ILP) is frequently used to data mine in multi-relational databases. However, most ILP algorithms disregard primary key information which is often available for such databases. This work demonstrates several disadvantages of the *mode refinement* operator that has been used in many multi-relational data mining algorithms in combination with both traditional subsumption and subsumption under Object Identity. We show how primary key information can be incorporated in this refinement operator and provide evidence that the resulting operator has several desirable properties in comparison with the traditional approaches. Especially, we will show that our refinement operator is *proper*.

1 Introduction

In multi-relational data mining research, much attention has been given to Inductive Logic Programming (ILP). A multi-relational database can be mapped to a Datalog database straightforwardly: relations are mapped to predicates and attributes to predicate arguments. ILP algorithms can be applied subsequently.

In this database-to-Datalog mapping, some information about databases is often disregarded: for most databases also *primary keys* and *foreign keys* of relations are defined. These keys describe some restrictions on a database. Our observation is that it is useless to query databases for information that cannot be stored according to the primary key information. Primary keys should therefore also be used to restrict the queries that an ILP algorithm considers. We will formalize this by defining a *downward refinement operator* which uses primary keys.

Every ILP algorithm traverses a search space of clauses of a certain language in some structured way using a so-called *refinement operator*. A *downward* refinement operator ρ is an operator that creates more specific clauses starting from very general clauses. Given a language of clauses and a *quasi-order* on these clauses, several desirable properties for refinement operators have been identified [6]:

(1) ρ should be locally finite: all refinements of a clause should be computable within finite time;

(2) ρ should be complete: given a clause, every clause in the language which is more specific according to the quasi-order should be obtainable by (repeatedly) applying the refinement operator;

(3) ρ should be proper: after refinement, according to the quasi-order the refined clause should always be more specific than the original clause (and therefore never equivalent).

If refinement operator ρ satisfies these properties, then this operator is called *ideal*.

In most ILP systems, the concepts of "more specific" and "general" are modeled using a quasi-order called θ-subsumption. This choice has a major drawback: if a refinement operator is finite and complete under θ-subsumption, it can be shown that this operator can never be proper; as a result, it is possible that a clause is infinitely refined without obtaining a more specific clause.

To face this problem, in [3] a different quasi-order based on subsumption was defined: subsumption under Object Identity. Under Object Identity a clause is evaluated in a different, more restricted way than is usual. Ideal refinement is possible under Object Identity. The additional restrictions however appear to be undesirable in many situations.

In this work, we will concentrate on a special downward refinement algorithm: refinement using *modes*. Mode refinement has been applied in several data mining algorithms [1, 2, 4, 5, 7, 8], and has shown its usefulness in these publications.

Our paper is organized as follows. In the second section, we will review both traditional subsumption and the concept of Object Identity, and we will introduce mode refinement. With several examples we will illustrate the problems which occur when either traditional or OI subsumption is used.

In the third section, we will introduce a new quasi-order and an enhanced mode refinement algorithm. As our new way of evaluating clauses is somewhere in the middle between evaluation under Object Identity and ordinary clause evaluation, we call our evaluation technique evaluation under *weak Object Identity*. Within our setup, the weak-OI quasi-order has several parameters, among which the primary keys. We will show that these parameters can be tuned in such a way that our quasi-order reduces to full Object Identity; in this way, our setup is a generalization of full OI. Using examples, we will show that one can also provide parameters for the refinement algorithm such that the resulting clauses do not suffer from the restrictions of full OI. Still, we will provide evidence that this refinement algorithm has exactly the same desirable properties as mode refinement using full Object Identity; more precisely, the refinement algorithm is finite and proper. Section four concludes.

2 Prerequisites and problem description

We will briefly review some terminology [6]. A (Datalog) *atom* $p(t_1, \ldots, t_n)$ consists of a relation symbol p of arity n followed by n terms t_i. A *term* is either a constant or a variable. A *substitution* θ is a set of the form $\{v_1/t_1, \ldots, v_n/t_n\}$ where v_i is a variable and t_i is a term. One can *apply a substitution* θ to an expression e,

yielding the expression $e\theta$, by simultaneously replacing all variables v_i by their corresponding terms t_i. An *atom set* is an unordered set of atoms; an ordered set of atoms is an *atom list*. A *clause* is an expression of the form $h \leftarrow S$, where h is an atom and S is an atom set. In this paper, without loss of generality, we consider the head h of clauses to be a fixed atom; we only consider the bodies of clauses. Previously, two kinds of subsumption have been defined:

- *Traditional θ-subsumption*: an atom set S_1 θ-subsumes an atom set S_2 ($S_2 \succeq S_1$) if there exists a substitution θ such that $S_1\theta \subseteq S_2$.
- *OI-subsumption*: an atom set S_1 OI-subsumes an atom set S_2 ($S_2 \succeq_{OI} S_1$) if there exists an injective substitution θ such that $S_1\theta \subseteq S_2$ and θ does not map any variable to a constant or variable already occurring in S_1.

Under traditional θ-subsumption, two atom sets S_1 and S_2 are considered to be equivalent (denoted by $S_1 \sim S_2$) iff $S_1 \succeq S_2$ and $S_2 \succeq S_1$. This is reasonable as one can show that: $(\forall S'(S' \succeq S_1 \rightarrow S' \succeq S_2) \wedge \forall S'(S' \succeq S_1 \rightarrow S' \succeq S_2)) \Leftrightarrow S_1 \sim S_2$; or, in words: if every possible set of atoms either subsumes two atom sets, or does not subsume any of these two, these atom sets are equivalent and must subsume each other.

We will illustrate these subsumption operators using predicates that encode directed, edge labeled graphs. The assumption is that we are interested in clauses with predicates $e(G, V_1, V_2, L)$ (which encodes that there is an edge from vertex V_1 to a vertex V_2 with label L in graph G) and $is(L, K)$ (which encodes that a label L is a label in the class K). So, our language consists of the set of predicates $\{e/4, is/2\}$; furthermore, we assume the set of constants $\{a, b\}$. The following clauses can be expressed in this language:

$$C_1 = p(G) \leftarrow e(G, V_1, V_2, L_1), is(L_1, a), \quad (1)$$
$$C_2 = p(G) \leftarrow e(G, V_1, V_2, L_1), is(L_1, a), e(G, V_3, V_4, L_2), \quad (2)$$
$$C_3 = p(G) \leftarrow e(G, V_1, V_2, L_1), is(L_1, a), e(G, V_3, V_4, L_2), e(G, V_4, V_5, L_3), \quad (3)$$
$$C_4 = p(G) \leftarrow e(G, V_1, V_2, L_1), is(L_1, a), e(G, V_4, V_5, L_3). \quad (4)$$

Clause C_1 states that a graph contains an edge of class a. Clause C_3 states that a graph contains an edge of class a and furthermore contains a vertex with at least one incoming and one outgoing edge, independent of the label. Under traditional subsumption, $C_1 \sim C_2 \sim C_4$ and $C_3 \succ C_1$.

We will now introduce the *bias* of the (traditional) mode refinement algorithm.

Definition 2.1 *A bias $\mathcal{B}$ is a tuple $(\mathcal{T}, \mathcal{C}, \mathcal{P}, \mathcal{M}, h)$, where $\mathcal{T}$ is a finite set of type symbols, $\mathcal{C}$ is a function that defines a finite set of constants for each type in $\mathcal{T}$; $\mathcal{P}$ is a finite set of declarations of the form $p(T_1, \ldots, T_n)$, where each $T_i \in \mathcal{T}$ and a predicate p occurs at most once. Set $\mathcal{M}$ defines* mode declarations*, which are declarations of the form $p(c_1, \ldots, c_n)$ and consist of a predicate symbol p with arguments c_i, each of which is either '+' (input), '-' (output) or '#' (constant). Set $\mathcal{M}$ may contain multiple modes for the same predicate symbol. h is an atom.*

Note that we use types in our bias; this is not common practice in most publications. The mode refinement algorithm and the bias define a search space of clauses, as follows.

Definition 2.2 *Given a bias $\mathcal{B}$ the mode refinement operator ρ recursively defines a search space $\mathcal{L}(\mathcal{B})$ as follows:*

- *'$h \leftarrow$' $\in \mathcal{L}(\mathcal{B})$;*
- *if $C =$'$h \leftarrow S$' $\in \mathcal{L}(\mathcal{B})$, then $\rho(C) \ni C' =$'$h \leftarrow S, A$' $\in \mathcal{L}(\mathcal{B})$, with $A = p(t_1, \ldots, t_n)$, iff there is a mode $M = p(c_1, \ldots, c_n) \in \mathcal{M}$ such that for every $1 \leq i \leq n$:*
  - *t_i is a variable in $var(C, T_i)$ and $c_i =$'+', or*
  - *t_i is a variable not in $\cup_j var(C, T_j)$ and $c_i =$'-', or*
  - *t_i is a constant in $\mathcal{C}(T_i)$ and $c_i =$'#'.*

  Here, T_i is the type of argument position i, as given by $\mathcal{P}$; $var(C, T)$ is the set of variables in C which occur at argument positions of type T.

An example bias is $\mathcal{B} = (\{G, V, L, K\}, \{K \rightarrow \{a, b\}\}, \{p(G), e(G, V, V, L), is(L, K)\}, \{e(+, -, -, -), e(+, +, -, -), is(+, \#)\}, p(G))$, which encodes a search space of labeled *forests*. One can show that $\{C_1, C_2, C_3, C_4\} \subseteq \mathcal{L}(\mathcal{B})$.

It is clear that this refinement algorithm does not generate all clauses that can be expressed using the given predicates and constants. Using traditional subsumption as quasi-order, one can show that the operator is complete within the sublanguage $\mathcal{L}(\mathcal{B})$. As an example, consider clause $C_3\{V_2/V_3\}$, which is a specialization of C_3. An equivalent clause, $C_3 \cup (C_3\{V_i/X_i | i \neq 2\}\{V_2/X_3\})$, can be constructed from C_3, where X_i are variables not occurring in C_3.

It is clear that for traditional subsumption, mode refinement is not proper either. By adding new atoms in two steps, C_3 can be obtained from C_1. In whatever order the last two atoms of C_3 are added, however, each intermediate clause is equivalent with C_1: $C_2 \sim C_1$ and $C_4 \sim C_1$. If one would decide not to allow a refinement from C_1 to C_2 or C_3, the operator would not be complete: one can show that C_1 cannot be refined to C_4 in that case.

If one applies OI-subsumption as quasi-order, the relations between clauses are different: $C_3 \succ_{OI} C_2 \succ_{OI} C_1$ and $C_3 \succ_{OI} C_4 \succ_{OI} C_1$. For example, to C_2 one may not apply $\theta = \{V_3/V_1, V_4/V_2, L_2/L_1\}$ to obtain C_1, as it maps variables to variables already occurring in C_2.

Under OI, mode refinement is always proper. This follows from the observation that under OI sets of atoms are always reduced [3]. Mode refinement is not complete. With the example bias, clause C_1 cannot be refined into $C_1 \cup \{e(G, V_3, V_1, L_2)\} \in \mathcal{L}(\mathcal{B})$.

A different way of defining OI is to define it using traditional θ-subsumption. We will follow this approach in this paper. Given a set of atoms S, we define $constr(S)$ to be the set of atoms

$$constr(S) = \{(t_1 \neq t_2) | t_1 \neq t_2, t_1, t_2 \in terms(S)\},$$

where $\neq$ is a binary predicate denoted in infix notation, and $terms(S)$ is the set of all terms occurring in atom set S. For example:

$$constr(\{is(L_1, a), is(L_1, K_1)\}) = \\ \{(L_1 \neq a), (L_1 \neq K_1), (a \neq L_1), (a \neq K_1), (K_1 \neq L_1), (K_1 \neq a)\}.$$

The OI-subsumption can then equivalently be defined as:

$$S_1 \succ_{OI} S_2 \Leftrightarrow S_1 \cup constr(S_1) \succ S_2 \cup constr(S_2).$$

When evaluating C_3, it is clear now that $\{(V_1 \neq V_2), (V_2 \neq V_3), (V_3 \neq V_4), (V_4 \neq V_5)\} \subset constr(C_3)$ and $\{(L_1 \neq L_2), (L_2 \neq L_3)\} \subset constr(C_3)$: the nodes must be different, and also all labels must be different.

Assume now that one still wishes to find a theory for predicate p that allows nodes to be equal, then this theory should contain several clauses under OI:

$$\begin{aligned}
p(G) &\leftarrow e(G, V_1, V_2, L_1), is(L_1, a), e(G, V_3, V_4, L_2), && (5)\\
p(G) &\leftarrow e(G, V_1, V_1, L_1), is(L_1, a), e(G, V_3, V_4, L_2), && (6)\\
p(G) &\leftarrow e(G, V_1, V_2, L_1), is(L_1, a), e(G, V_1, V_4, L_2), && (7)\\
p(G) &\leftarrow e(G, V_1, V_2, L_1), is(L_1, a), e(G, V_3, V_1, L_2), && (8)\\
&\vdots
\end{aligned}$$

for a total of 15 clauses, each of which reflects some case of variable equality. For C_3 even 52 clauses are required. One can show that the number of clauses grows exponentially in the number of variables. For theories in which one would like to allow equality, Object Identity can therefore be very impractical.

In some situations, there are ad-hoc solutions to solve problems caused by OI. Assume that one would like to express the following theory with only one clause:

$$\begin{aligned}
p(G) &\leftarrow e(G, V_1, V_2, L_1), is(L_1, a), e(G, V_2, V_3, L_2), && (9)\\
p(G) &\leftarrow e(G, V_1, V_2, L_1), is(L_1, a), e(G, V_2, V_3, L_1), && (10)
\end{aligned}$$

then one could choose to use another predicate language. Consider a language with predicate $e/4$ and a predicate $ea(G, V_1, V_2)$ which is defined in terms of e and is to express that there is an edge from V_1 to V_2 in label class a. The following clause can then be expressed:

$$p(G) \leftarrow ea(G, V_1, V_2), e(G, V_2, V_3, L_2);$$

in this clause L_2 can be the same label as the label between nodes V_1 and V_2. However, this representation has an unwanted side effect:

$$p(G) \leftarrow ea(G, V_1, V_2), e(G, V_1, V_2, L_1);$$

according to OI-subsumption, this clause is not equivalent to any smaller clause, but by the definition of ea we know that the last atom can be removed. We believe therefore that this construction is undesirable too.

From our point of view, the best solution would be to force Object Identity constraints only to some variables in a clause. The question is how this can be done without loosing the desirable, ideal properties of OI. In the next section we will provide an answer to this question.

3 Weak Object Identity using Primary Keys

We will first formally define the *bias* of a language with primary keys. Immediately after the definitions, we will illustrate their meaning using examples.

Definition 3.1 *A bias with primary keys $\mathcal{B}_K$ is a tuple $(\mathcal{T}, \mathcal{C}, \mathcal{P}, \mathcal{M}, h, \mathcal{K}, OI)$, where $\mathcal{B} = (\mathcal{T}, \mathcal{C}, \mathcal{P}, \mathcal{M}, h)$ is a simple bias as given in Definition 2.1 and $\mathcal{K}$ is a function which defines a set of primary keys for each predicate $p \in \mathcal{P}$. A primary key is a subset of $\{1, \ldots, arity(p)\}$. Set OI is a subset of the types, $OI \subseteq \mathcal{T}$.*

Definition 3.2 *An atom set S is* constrained by a primary key $K \in \mathcal{K}(p)$ *iff:*

$$\forall A_1 = p(t_{11}, \ldots, t_{1n}), A_2 = p(t_{21}, \ldots, t_{2n}) \in S : (\forall i \in K : t_{1i} = t_{2i}) \Rightarrow A_1 = A_2.$$

Definition 3.3 *A clause $C =$ '$h \leftarrow S$' is part of the language $\mathcal{L}_K(\mathcal{B}_K)$ defined by a bias $\mathcal{B}_K$ iff:*

- *$C \in \mathcal{L}(\mathcal{B})$, where $\mathcal{B}$ is the simple part of $\mathcal{B}_K$ and $\mathcal{L}(\mathcal{B})$ is defined according to Definition 2.2.*
- *S is constrained by each primary key in $\mathcal{K}(p) \cup K_t(p)$, for every predicate p. With $K_t(p)$ we denote the trivial key of a predicate p, $\{1, \ldots, arity(p)\}$.*

Furthermore C' is a key mode refinement of C, denoted by $C' \in \rho_K(C)$ (for $C \in \mathcal{L}_K(\mathcal{B}_K)$), iff $C' \in \rho(C)$ and C' is constrained by every primary key.

We will continue with our graph example (not restricted to trees). Assume that we know that in the database under consideration between each pair of nodes there is at most one edge in each direction, and that an edge always has exactly one label, then we can express this knowledge using a primary key:

$$\mathcal{K}(e) \rightarrow \{\{1, 2, 3\}\},$$

as this states that an edge can be identified uniquely by giving a graph and two vertices. If this primary key is part of a bias $\mathcal{B}_K$, then $\mathcal{L}(\mathcal{B}_K)$ does not contain the following clause in any case:

$$p(G) \leftarrow e(G, V_1, V_2, a), e(G, V_1, V_2, b);$$

this expression can never be true given our knowledge about the graph data. As we believe that for most relational databases primary key information is available, we believe that this restriction of a full clausal language is important. Because it reduces the number of clauses that an Inductive Logic Programming algorithm has to consider, we believe that this strategy could yield significant efficiency improvements in many algorithms.

Definition 3.4 *Given two clauses $C_1 =$ '$h_1 \leftarrow S_1$' $\in \mathcal{L}_K(\mathcal{B}_K)$, $C_2 =$'$h_2 \leftarrow S_2$' $\in \mathcal{L}_K(\mathcal{B}_K)$, S_1 $\mathcal{B}_K$-OI-subsumes S_2, denoted by $S_2 \succeq_{\mathcal{B}_K - OI} S_1$, iff $S_2 \cup constr_{\mathcal{B}_K}(S_2) \succeq S_1 \cup constr_{\mathcal{B}_K}(S_1)$, where $constr_{\mathcal{B}_K}(S) = \{(t_1 \neq t_2) | t_1, t_2 \in$ OI-terms$_{\mathcal{B}_K}(S), t_1 \neq t_2\}$ and OI-terms$_{\mathcal{B}_K}(S)$ is the set of terms occurring in S at argument positions i of predicates p for which $T_i \in OI \in \mathcal{B}_K$.*

The main difference with traditional OI-subsumption is that using types, OI constraints are only forced to some variables in a clause. As an example, consider the bias $\mathcal{B}_K = (\{G, V, L, K\}, \{K \rightarrow \{a, b\}\}, \{p(G), e(G, V, V, L), is(L, K)\}$, $\{e(+,-,-,-), e(+,+,-,-), e(+,-,+,-), e(+,-,-,+), is(+,\#)\}, p(G)$, $\{e \rightarrow \{\{1,2,3\}\}\}$, $\{G, V\})$. The following clauses are part of $\mathcal{L}_K(\mathcal{B}_K)$:

$$C_2 = p(G) \leftarrow e(G, V_1, V_2, L_1), is(L_1, a), e(G, V_3, V_4, L_2), \quad (11)$$

$$C_5 = p(G) \leftarrow e(G, V_1, V_2, L_1), is(L_1, a), e(G, V_3, V_4, L_1), \quad (12)$$

then $C_5 \succ_{\mathcal{B}_K - OI} C_2$ while $C_5 \not\succ_{OI} C_2$. In comparison with traditional OI, $(L_1 \neq L_2) \notin constr_{\mathcal{B}_K}(C_2)$.

A special case arises when $OI = \mathcal{T}$ and $\mathcal{K} = \emptyset$: all types are subject to Object Identity constraints such that our weak subsumption operator reduces to full Object Identity. Furthermore, by the absence of keys, no key constraints are effective. One can easily see that our refinement operator reduces to a traditional mode refinement operator with traditional OI as quasi-order. This shows that our formalism is a generalization of full OI.

Now temporarily assume that $is(+,-)$ would be part of $\mathcal{M}$ in $\mathcal{B}_K$. The following clauses would both be part of $\mathcal{L}_K(\mathcal{B}_K)$:

$$C_6 = p(G) \leftarrow e(G, V_1, V_2, L_1), is(L_1, K_1), \quad (13)$$

$$C_7 = p(G) \leftarrow e(G, V_1, V_2, L_1), is(L_1, K_1), is(L_1, K_2); \quad (14)$$

however, $C_6 \sim_{\mathcal{B}_K - OI} C_7$; our mode refinement operator would not be proper. If we however assume that $\{is \rightarrow \{\{1\}\}\} \subseteq \mathcal{K} \in \mathcal{B}_K$, then $C_7 \notin \mathcal{L}_K(\mathcal{B}_K)$, as the unproper refinement is not allowed.

Theorem 3.5 *Given a bias $\mathcal{B}_K$, mode refinement as given in Definition 3.3 is proper if for every $M = p(c_1, \ldots, c_n) \in \mathcal{M} \in \mathcal{B}_K$ there is a key $K = \{i_1, \ldots, i_n\} \in \mathcal{K}(p) \cup K_t(p)$ such that for every $i_j \in K$ one of the following holds:*

- *T_{i_j}, the type of the i_jth argument, is included into $OI \in \mathcal{B}_K$;*
- *$c_{i_j} =$ '+' or $c_{i_j} =$ '#'.*

Proof Outline We will provide a proof by contradiction. Assume that clause $C \sim_{\mathcal{B}_K - OI} C' \in \rho_K(C)$ and (without loss of generality) that C is not equivalent with any smaller clause. In this case, there is a weak OI substitution θ which maps one atom in C' onto another atom in C', resulting in C. Note that $|C'| = |C| + 1$; we may therefore assume that θ only affects one atom $A \in C'$. As a clause must be key constrained, at least one term in every primary key of this atom is different from the corresponding term in $A\theta$. As θ substitutes a variable with a term already occurring in C', each such different term must be a variable of a type not in $OI \in \mathcal{B}_K$. We may therefore conclude that a variable not in OI is part of every key, and that A is the only atom in which these variables occurs. At the other hand, our theorem states that a variable must be marked with '+' in a mode if it is not of a type in OI. According to the definition of '+', there must be another atom which contains this variable, so we derive a contradiction.

4 Conclusions

We have shown that mode refinement in combination with both traditional subsumption and Object Identity subsumption has undesirable properties. While for traditional subsumption no proper refinement operator exists, Object Identity restricts the expressiveness of single clauses too much to obtain properness. We propose to reduce the disadvantages of OI by only considering search spaces that do not violate primary key constraints. In most situations, this is a very desirable restriction as it restricts the full clausal language to expressions that make sense from a human user point of view. For these more restricted languages, we have given an outline of a proof which convinces us that, using a weak subsumption operator, it is not necessary to force Object Identity to all variables in order to obtain a proper mode refinement operator; this allows single clauses to express more interesting patterns.

We have implemented primary keys, weak Object Identity and mode refinement in our multi-relational data mining algorithm FARMER [8]. In experiments with a graph database, these features allowed us to restrict the search space to clauses that represent graphs with single labels on the edges. This reduced the number of clauses that FARMER had to consider, and resulted in significant speed-ups.

As our restricted language can be as large as a full clausal language —in this case our weak OI subsumption becomes full OI subsumption— our setup is a generalization of Object Identity. Weak subsumption is exactly in the middle between traditional subsumption and OI subsumption.

References

[1] H. Blockeel and L. De Raedt. *Top-down Induction of Logical Decision Trees.* 1997.

[2] L. Dehaspe and H. Toivonen. Discovery of frequent Datalog patterns. In: *Data Mining and Knowledge Discovery 3*, no. 1, pages 7–36, 1999.

[3] F. Esposito, N. Fanizzi, S. Ferilli and G. Semeraro. A generalization model based on OI-implication for ideal theory refinement. In: *Fundamenta Informaticae*, 47, pages 15-33, 2001.

[4] F.A. Lisi and D. Malerba. Towards Object-Relational Data Mining. In: *Atti dell'Undicesimo Convegno Nazionale su Sistemi Evoluti per Basi di Dati*, pages 269–280, Rubbettino Editory, Italy, 2003.

[5] S. Muggleton. Inverse entailment and Progol. In: *New Generation Computing*, 13, pages 245–286. 1995.

[6] S.-H. Nienhuys-Cheng and R. de Wolf. *Foundations of Inductive Logic Programming.* LNAI 1228. Springer, 1997.

[7] S. Nijssen and J.N. Kok. Faster Association Rules for Multiple Relations. In: *IJCAI-01*, pages 891–896, 2001.

[8] S. Nijssen and J.N. Kok. Efficient Frequent Query Discovery in Farmer. In: *PKDD-2003*, 2003.