Ideal Refinement of Datalog Clauses using Primary Keys

Siegfried Nijssen and **Joost N. Kok** [1]

Abstract. Inductive Logic Programming (ILP) algorithms are frequently used to data mine multi-relational databases. However, in many ILP algorithms the use of primary key constraints is limited. We show how primary key constraints can be incorporated in a downward refinement operator. This refinement operator is proved to be finite, complete, proper and therefore ideal for clausal languages defined by primary keys. As part of our setup, we introduce a weak Object Identity subsumption relation between clauses which generalizes over traditional, full Object Identity. We find that the restrictions on the language and the subsumption relation are not very restrictive. We demonstrate the feasibility of our setup by showing how the refinement operator can be incorporated in the refinement strategy of common ILP algorithms.

1 INTRODUCTION

In multi-relational data mining research, much attention has been given to Inductive Logic Programming. Inductive Logic Programming provides a well-founded framework on which machine learning algorithms can be built which extend beyond the classical setting of attribute-value learning. Such situations typically arise when one has to deal with learning tasks for databases that consist of multiple, related tables. In this setup, it is common practice to map the relations of a database to predicates and the rows of tables in a database to facts (atoms of a predicate for which all parameters are constants). The close relationship between logic and relational databases has already been studied and led to the development of Datalog. While Datalog is a restriction of the full mathematical logic to function-free clauses on the one hand, at the other hand Datalog extends relational databases with recursive queries.

However, as far as we are aware of, until now the relation between databases and Datalog for machine learning tasks has not been given much attention. More precisely, we observe that in the development of many relational machine learning algorithms typical properties of databases have been disregarded: for most databases also *primary keys* and *foreign keys* of relations are defined. These keys describe constraints on a database. One could wonder in which ways this information can be used in ILP algorithms. One possibility is to use the keys to restrict the search space of queries; clearly, queries that express relations that are known to be forbidden by primary keys should not be considered by a learning algorithm. We will formalize this by integrating the primary key constraints in a *downward refinement operator*, and will show that this new refinement operator has several desirable properties in comparison with traditional operators.

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 clauses in this language several desirable properties for refinement operators have been identified [7]:

(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);
(4) ρ should be ideal: the combination of (1)-(3).

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, 4] 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. In the second section of this paper, we will review both traditional subsumption and the concept of Object Identity; we will illustrate with several examples that the additional constraints of OI introduce several new problems.

In the third section, we will introduce a new quasi-order and a corresponding downward refinement operator which promises to solve the problems that we illustrated in the second section for function-free (Datalog) clauses. The key feature of the refinement operator is that it exploits primary key information. Our new way of evaluating clauses is somewhere in the middle between evaluation under Object Identity and ordinary clause evaluation; we therefore call our evaluation technique evaluation under *weak Object Identity*. The fourth section provides hints on the incorporation of weak OI refinement in ILP algorithms. Section five concludes.

2 PREREQUISITES AND PROBLEM DESCRIPTION

We will briefly review some terminology [7] and introduce some new concepts. 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 on an expression e, yielding the expression $e\theta$, by simultaneously

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

replacing all variables v_i by their corresponding terms t_i. An *atom set* is a 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: given two atom sets S_1 and S_2, if it is not possible that an atom set subsumes S_1 but does not subsume S_2 (or vice versa), then S_1 and S_2 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) 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:

$$\begin{aligned}
C_1 = p(G) &\leftarrow e(G, V_1, V_2, L_1), is(L_1, a),\\
C_2 = p(G) &\leftarrow e(G, V_1, V_2, L_1), is(L_1, a), e(G, V_3, V_4, L_2),\\
C_3 = p(G) &\leftarrow e(G, V_1, V_2, L_1), is(L_1, a), e(G, V_4, V_5, L_3),\\
C_4 = p(G) &\leftarrow e(G, V_1, V_2, L_1), is(L_1, a), e(G, V_3, V_4, L_2),\\
&\quad\ e(G, V_4, V_5, L_3),
\end{aligned}$$

Clause C_1 states that a graph contains an edge of class a. Clause C_4 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, $C_1 \sim C_2 \sim C_3$ and $C_4 \succ C_1$.

A well-known refinement operator [7] for traditional clauses applies the following refinements to a clause C:

- unify any pair of different variables in C by applying a substitution that maps one variable onto another;
- substitute each variable with all possible constants in the language;
- add a new atom different from every atom in C.

By adding new atoms in two steps, C_4 can be obtained from C_1. In whatever order the last two atoms of C_4 are added, however, each intermediate clause is equivalent with C_1: $C_2 \sim C_1$ and $C_3 \sim C_1$. This refinement operator is therefore not proper. 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: $C_4 \succ_{OI} C_2 \succ_{OI} C_1$ and $C_4 \succ_{OI} C_3 \succ_{OI} C_1$. 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. For OI-subsumption one can obtain an ideal refinement operator by limiting the above three refinement operators: a pair of variables may never be unified; constants may only be substituted with constants that do not already occur in a clause. More details can be found in [3].

A different way of defining OI is to define it using traditional θ-subsumption. We will follow this practice 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:

$$\begin{aligned}
constr(\{is(L_1, a), is(L_1, K_1)\}) = \\
&\{(L_1 \neq a), (L_1 \neq K_1), (a \neq L_1),\\
&(a \neq K_1), (K_1 \neq L_1), (K_1 \neq a)\}.
\end{aligned}$$

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_4, it is clear now that $\{(V_1 \neq V_2), (V_2 \neq V_3), (V_3 \neq V_4), (V_4 \neq V_5)\} \subset constr(C_4)$ and $\{(L_1 \neq L_2), (L_2 \neq L_3)\} \subset constr(C_4)$: 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:

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

for a total of 15 clauses, each of which reflects some case of variable equality. For C_4 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:

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

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 with 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 A bias $\mathcal{B}$ is a tuple $(\mathcal{P}, \mathcal{C}, \mathcal{K}, OI, h)$, where $\mathcal{P}$ is a finite set of predicate symbols (each of which has a unique arity $arity(p)$), $\mathcal{C}$ a finite set of constants, $\mathcal{K}$ is a function which defines a set of primary keys for each predicate $p \in \mathcal{P}$; a primary key is a subset of $\{1, \ldots, arity(p)\}$. OI is a function which for each $p \in \mathcal{P}$ defines a subset of $\{1, \ldots, arity(n)\}$ and thus partitions the arguments of each predicate into OI *arguments* (arguments which are part of $OI(p)$), and OI*-free arguments* (arguments which are not part of $OI(p)$); h is an atom.

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

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

The intuition is that in an atom set there may be no two different atoms of the same predicate which are equal in the terms that are part of the primary key.

We will continue with our graph example. Assume that we know that in one direction there is at most one edge from one vertex to another, and that an edge always has exactly one label, then we can express this knowledge using one primary key for the predicate e:

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

this key states that an edge can be identified uniquely by giving a graph and two vertices. Following common practice in database theory, we allow for multiple primary keys for each predicate; therefore, the function $\mathcal{K}$ defines a set of primary keys. The following atom set is not constrained by the single primary key in $\mathcal{K}(e)$:

$$\{e(G, V_1, V_2, a), e(G, V_1, V_2, b)\}$$

By $OI(e) \rightarrow \{1, 2, 3\}$ we define that the first three arguments of e are OI arguments. Terms that are used as OI arguments, in the example G, V_1 and V_2, we will refer to as *OI terms*. *OI-free terms* are terms that are used in OI-free arguments, in the example constant b.

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

- all predicate symbols in S are part of $\mathcal{P}$, with the correct arity;
- all constants in S are part of $\mathcal{C}$;
- S is constrained by each primary key in $\mathcal{K}$;
- in S no single term is both OI and OI-free;
- there is an order L of atoms in S (called the *proper order*) such that for every atom $p(t_1, \ldots, t_n) \in S$ there is at least one primary key $K \in \mathcal{K}(p)$ such that for every term t_i in that key ($i \in K$):
  - t_i is either a constant,
  - or t_i is a variable and $i \in OI(p)$,
  - or t_i is a variable and t_i occurs in L before it occurs in A.

  We call above three constraints on one key the *properness constraints*.

In our example assume furthermore that the primary key of *is* is defined by $\mathcal{K}(is) \rightarrow \{\{1, 2\}\}$ and that *is* has no OI arguments, $OI(is) \rightarrow \emptyset$; then the following clause is *not* in $\mathcal{L}(\mathcal{B})$:

$$p(G) \leftarrow e(G, V_1, V_2, L_1), is(L_1, C_1);$$

in the last atom C_1 is new and does not occur at an OI position, while there is no key which does not include C_1. For the other order of atoms, the same problem remains. Using bias $\mathcal{B}$, is atoms may only have constants as second argument. If either $OI(is) \rightarrow \{2\}$ or $\mathcal{K}(is) \rightarrow \{\{1\}\}$ would be contained in $\mathcal{B}$, the clause is part of $\mathcal{L}(\mathcal{B})$. The technical reasons for the atom order constraint will become clear after we have defined weak OI-subsumption. The idea behind weak Object Identity is however that some variables, although they are not forced to have an own 'identity' through Object Identity, will always have a distinctive identity through their relation to other variables.

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

The main difference with traditional OI-subsumption is that OI constraints are only forced to some variables in a clause. Consider the following clauses which are part of $\mathcal{L}(\mathcal{B})$:

$$C_2 = p(G) \leftarrow e(G, V_1, V_2, L_1), is(L_1, a), e(G, V_3, V_4, L_2),$$
$$C_5 = p(G) \leftarrow e(G, V_1, V_2, L_1), is(L_1, a), e(G, V_3, V_4, L_1),$$

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

Now consider the following refinement operator ρ.

Definition Given a clause $C \in \mathcal{L}(\mathcal{B})$, $C' \in \rho(C)$ iff C' is constrained by $\mathcal{K}$ and either:

- $C' = C\theta$, where $\theta = \{t_1/t_2\}$, t_1 is a variable in $terms(C)$, and t_2 is one of the following possibilities:
  - t_2 is a constant in $\mathcal{C}$, $t_2 \notin$ *OI-terms*(C).
  - if $t_1 \notin$ *OI-terms*$_{\mathcal{B}}(C)$, t_2 is a variable in $terms(C)$, $t_2 \notin$ *OI-terms*$_{\mathcal{B}}(C)$.
- or, $C' = C \cup A$, where $A = p(t_1, \ldots, t_n)$ and there is at least one primary key $K \in \mathcal{K}(p)$ such that for every argument t_i in that key ($i \in K$):
  - t_i is either a constant,
  - or t_i is a variable and $i \in OI(p)$,
  - or t_i is a variable and t_i occurs in C.

In our example, $C_2 \in \rho(C_1)$, $C_3 \in \rho(C_2)$, $C_3 \in \rho^*(C_1)$ and $C_5 \in \rho(C_2)$. Our claim is now:

Theorem 1 *Refinement operator ρ for language $\mathcal{L}(\mathcal{B})$ with quasi-order $\succeq_{\mathcal{B}-OI}$ is finite, complete and proper, and therefore ideal.*

Proof *Finiteness* Clearly, the number of substitutions is finite (the number of variables is finite, as well as the number of constants in the language). Also the number of possible new atoms is finite, assuming that of each possible new clause only one alphabetic variant is considered (for example, by numbering new variables in the new atom in order of occurrence, instead of giving them names).

Properness We distinguish two cases:

- refinement by substitution. One can show that $(C \cup constr_{\mathcal{B}}(C))\theta = C\theta \cup constr_{\mathcal{B}}(C\theta)$. Furthermore, one can show that refinement by above substitutions yields a more specific clause under traditional subsumption. Under traditional subsumption $(C \cup constr_{\mathcal{B}}(C))\theta$ is more specific than $C \cup constr_{\mathcal{B}}(C)$, consequently also $C\theta \not\sim_{\mathcal{B}-OI} C$.
- refinement by adding an atom. Assume that $[C, A] \sim_{\mathcal{B}-OI} C$. Then there is a substitution such that exactly one atom $A_1 \in [C, A]$ is mapped to exactly one atom $A_2 \in [C, A]$, $A_1\theta = A_2$, while at least $A_1 = A$ or $A_2 = A$. No more atoms may be affected, as otherwise $[C, A]\theta$ would yield a clause smaller than C. Furthermore, as $[C, A]$ is a refinement of C, A_1 must be different from A_2 in all primary keys. Every primary key in A_1 must contain an OI-free variable that is substituted by θ.
  As C is in $\mathcal{L}(\mathcal{B})$, and the added atom A is constrained to satisfy the properness constraints, $[C, A]$ must be a proper atom order. In this order, A_1 must satisfy the properness constraints, and there must be a primary key in A_1 which contains an OI-free variable that occurs in another atom before A_1 in L. This contradicts our observation that a substitution may only affect one atom.

Completeness We first note that any subset of a clause in $\mathcal{L}(\mathcal{B})$ also obeys the primary key constraints. One can safely —and should— always remove clauses that disobey primary key constraints.

Given are two clauses $C_1, C_2 \in \mathcal{L}(\mathcal{B})$, $C_2 \succ_{\mathcal{B}-OI} C_1$; θ is the substitution involved in this weak subsumption. Without loss of generality, we assume that there is no alphabetic variant of C_2 such that the substitution involved is smaller. In this case substitution θ only maps from variables in C_1 to variables in C_1 or to constants. Furthermore, only for OI-free variables θ maps from variables to variables; it can only map to constants not in *OI-terms*(C_1).

We claim that refinement operator ρ can perform all substitutions in θ in some order; therefore $\rho^*(C_1) \ni C_3 = C_1\theta \subseteq C_2$. As $C_2 \in \mathcal{L}(\mathcal{B})$, there is an order L_2 of atoms in C_2 which obeys the properness constraints.

We claim furthermore that ρ can incrementally extend C_3 with all atoms in $C_2 - C_3$, in an order such that in each intermediate step the atom set obeys the properness constraints. Consider the following list of atoms: $L_2' = [L_3, L_4]$, where L_3 is a proper order of atoms in C_3 and L_4 is the list of atoms in $C_2 - C_3$ in order of occurrence in L_2. Given an atom A in a list L, let $L(A)$ denote the set of atoms occurring before A in that list (not including A). We observe that for each $A \in C_2 - C_3$, $L_2(A) \subseteq L_2'(A)$. For each atom in L_4 the properness constraints are therefore obeyed; L_2' is also a proper order of C_2 and for each atom $A \in L_4$ also $[L_2'(A), A] \in \rho(L_2'(A))$.

This construction shows that every specialization of a clause C can be constructed using operations in ρ.

4 APPLICATIONS IN ILP ALGORITHMS

Our theoretical observations of the previous section can easily be incorporated into practical systems and refinement algorithms. In this section, we apply our observations to the refinement of clauses using *modes*. Mode refinement is a strategy taken in several ILP algorithms [1, 2, 5, 8, 9]. A mode is a declaration of the form

$$m = p(c_1, \ldots, c_n),$$

where c_i is either $+$, $-$ or $\#$. Clauses are refined only by adding new atoms according to these declarations[2]. Given a clause C, atom $A = p(t_1, \ldots, t_n)$ may only be added to a clause C if a mode has been specified such that for every $1 \leq i \leq n$:

- t_i is a constant and c_i is $\#$;
- t_i is a variable not occurring in C and c_i is $-$;
- t_i is a variable occurring in C and c_i is $+$.

The question is in which cases this refinement operator is proper if we take weak Object Identity as quasi-order. According to our theory, a refined clause should satisfy the properness and the primary key constraints, while OI terms and OI-free terms must be disjoint. A check for primary key and disjunction constraints should be performed by the refinement algorithm while refining clauses according to the modes. The properness constraint can however be checked at beforehand. It suffices to check every mode: for at least one primary key $K \in \mathcal{K}(p)$ in every mode m, for every $i \in K$, c_i should be either $\#$ or $+$, or if c_i is $-$, $i \in OI(p)$.

Continuing our graph example where we left it, the following modes yield a proper refinement operator:

$$\mathcal{M} = \{e(+,-,-,-), e(+,+,-,-), is(+,\#)\}.$$

Example clauses $C_1, \ldots, C_5$ can be constructed using these modes. The refinement operator is proper, although the last argument of e and the first argument of is are OI-free.

In many ILP algorithms, the mode principle is extended with the notion of types. Using a description language similar to the mode definition language, in these algorithms every argument of each predicate is given a type; during query construction these types forbid queries in which the same variable is used in arguments with different types. In many cases, this is a useful restriction, as it makes queries impossible such as

$$p(G) \leftarrow e(G, V_1, V_2, L_1), e(G, L_1, L_1, L_2),$$

which would otherwise be generated by mode set $\mathcal{M}$. As one sees here, our mechanism of OI-arguments and OI-free arguments is a special case in which only two mode types are used. Our approach can however easily be extended to a situation with multiple types. In that case, instead of defining OI and OI-free arguments, one has to specify which types are considered to be OI types and which types are OI-free types. The argument types then force the same subdivision between OI and OI-free arguments as we discussed here, and force some additional restrictions on top of that.

In order to gain some further insight in the benefits and drawbacks of weak Object Identity, we implemented this mode refinement with types under weak Object Identity in the multi-relational frequent query data mining algorithm FARMER [9, 10]; more information about frequent query miners can also be found in [2, 11]. Contrary to most other ILP algorithms which perform a heuristic search [1, 5] these algorithms perform an exhaustive search in which *all* queries are found that are satisfied by a large enough number of examples in a database, where 'large enough' is defined by a threshold value. Similar to other algorithms, they perform this task in an

[2] The refinement operator is therefore certainly not complete.

iterative two-phase procedure which consists of a query generation phase first, and a query evaluation phase next. To reduce the number of queries that needs to be evaluated FARMER uses a refinement algorithm which, as much as possible at least, joins two previously evaluated queries in stead of extending one single query.

Our testcase is the molecular PTE [6] dataset which consists of 320 molecules. We conceive the molecules as undirected graphs with labels on both the edges and the nodes. In our first experiment we are interested in finding frequent induced subgraphs, which can be expressed using modes $\{bond(+,+,-,\#), bond(+,+,+,\#), atom(+,+,\#), atom(+,-,\#)\}$ and keys $\{\mathcal{K}(bond) \rightarrow \{\{1,2,3\}\}, \mathcal{K}(atom) \rightarrow \{\{1,2\}\}\}$. A query within the search space is for example:

$$p(M) \leftarrow atom(M,A_1,c), bond(M,A_1,A_2,single), bond(M,A_2,A_1,single), atom(M,A_2,c),$$

and denotes two carbon atoms connected by a single bond. Please note that for this particular problem setting we do not strictly require an ILP algorithm. There are several frequent graph miners which can also be applied to this setting [12, 13]. Furthermore, within the framework of full OI De Raedt et al. also recently proposed [11] to use additional constraints to restrict the search space of frequent query miners, including the possibility of using primary keys.

We first evaluate all types under Object Identity. Thus, the effect of introducing primary keys is isolated. Results are shown in Table 1; as one sees here, the relative number of avoided queries is low; the source of this phenomenon is the effective refinement algorithm of FARMER: if no primary keys are specified, infrequent combinations of atoms are still quickly discovered and used to avoid the further generation of useless combinations.

Absolute threshold	Number of avoided queries	Total number of evaluated queries	Number of frequent queries
30	4749	141881	1065
20	9741	311343	2343
10	96447	4325893	22786

Table 1. Overview of query evaluation savings using primary keys.

For the PTE dataset, it is also interesting to consider different kinds of frequent patterns. For example, it is known that when a limited number of carbon atoms is replaced by nitrogen atoms in a molecule, its chemical properties do not change significantly. Nitrogens and carbons can therefore be put into an equivalence class. In an ILP algorithm one can easily formalize this by adding a predicate $nc(T)$ to the knowledge base, where $nc(c)$ is true for the elements c and n, and false for all other elements; the bias would become $\{bond(+,+,-,\#), bond(+,+,+,\#), atom(+,+,\#), atom(+,-,\#), atom(+,+,-), atom(+,-,-), nc(+)\}$. Consider the following query:

$$p(M) \leftarrow atom(M,A_1,T), nc(T), bond(M,A_1,A_2,single), bond(M,A_2,A_1,single), atom(M,A_2,c).$$

Under full OI, T would be restricted to nitrogen! Within our framework, however, it is possible to remove the OI constraint from T very naturally, after which the semantics of the query are as expected. Table 2 gives a short overview of the resulting numbers of queries. Only when one adds the nc predicate with weak OI, the number of frequent queries increases exactly as desired.

Absolute threshold	Number of frequent queries without nc	with nc, OI	with nc, weak OI
150	61	63	1852
125	65	74	1873
100	71	137	3648
75	163	262	8521

Table 2. Number of frequent queries discovered.

5 CONCLUSIONS

We have shown that both traditional subsumption and Object Identity subsumption have undesirable properties. While for traditional subsumption no ideal refinement operator exists, Object Identity restricts the expressiveness of single clauses too much. We propose a solution by restricting full clausal languages to languages 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 a proof which convinces us that, using a weak subsumption operator, it is not necessary to force Object Identity on all variables to obtain an ideal refinement operator; this allows single clauses to express more interesting patterns.

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 traditional Object Identity. Weak subsumption is exactly in the middle between traditional subsumption and OI subsumption.

Thus, we have shown how concepts from relational database theory can also be used for other useful purposes than the reduction of the number of evaluated queries. Several questions have however been left unanswered. On the theoretical side, it would be interesting to investigate how other constraints, such as *foreign keys* and *participation constraints*, but also more general constraints [11], can be exploited further. Similar to primary keys, these constraints can also restrict the search space of ILP algorithms; the relations between these constraints and refinement operators deserve further study. On the practical side, we have evaluated weak OI here in the context of frequent query mining and concluded that the capabilities of primary keys to limit the search space may be limited. We would expect different results in learners that use more traditional refinement operators, but further experiments would have to confirm this.

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. *Fundamenta Informaticae*, 47:1533, 2001.

[4] F.A. Lisi, S. Ferilli and N. Fanizzi. Object Identity as Search Bias for Pattern Spaces. In: *Proceedings of the Fifteenth European Conference on Artificial Intelligence (ECAI-2002)*, pages 375–379, Lyon, 2002.

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

[6] `http://web.comlab.ox.ac.uk/oucl/research/areas/machlearn/PTE/`

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

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

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

[10] `http://hms.liacs.nl`

[11] L. De Raedt and J. Ramon. Condensed Representations for Inductive Logic Programming. In: *KR-2004*, 2004.

[12] T. Washio and H. Motoda. State of the Art of Graph-Based Data Mining. In: *ACM SIGKDD Explorations*, pages 59–68, 2003.

[13] X. Yan and J. Han. gSpan: Graph-Based Substructure Pattern Mining. In: *Proceedings of the 2002 International Conference on Data Mining (ICDM2002)*, 2002.