# Condensed Representations for Inductive Logic Programming*

Luc De Raedt† and Jan Ramon‡

† Institute for Computer Science, Machine Learning Lab
Albert-Ludwigs-University, Georges-Köhler-Allee, Gebäude 079,
D-79110 Freiburg i. Brg., Germany
‡ Department of Computer Science, Katholieke Universiteit Leuven,
Celestijnenlaan 200A, B-3001 Heverlee, Belgium

**Abstract.** When mining frequent Datalog queries, many queries will be equivalent in the light of an implicit or explicit background knowledge. To alleviate the problem, we introduce various types of condensed representations: clauses that are semantically free or closed w.r.t. a user specified background theory, as well as $\delta$-free and closed clauses. A novel algorithm that employs these representations is also presented and experimentally evaluated.

## 1 Introduction

One of the central tasks in data mining is that of finding all patterns that are frequent in a given database. The inductive logic programming instantiation of this task considers patterns that are logical queries or clauses and data in the form of a Datalog (or Prolog) knowledge base [5, 6]. This problem is known under the name of frequent Datalog query mining and has received quite some attention in the literature, cf. [14, 13, 11, 19]. Nevertheless, there are several difficulties that arise when applying these techniques, including: the computational resources needed, the vast number of queries that are discovered, and the difficulties in guiding the induction process using declarative knowledge. The standard way of guiding an inductive logic programming system is by specifying syntactic restrictions that the clauses should satisfy (the so-called language bias). Syntactic restrictions are not really declarative. Furthermore, when analyzing the discovered patterns in the light of background knowledge, it turns out that many of the patterns discovered will be semantically equivalent, and hence, redundant. These redundancies do not only lead to unnecessary inefficiencies but also impose an unnecessary burden on the user.

In this paper, we address these problems by introducing various types of condensed representations for frequent Datalog query mining. Condensed representations are well studied for simple pattern domains such as item sets [21,

* This paper is an extended abstract of a paper that appeared in the Proceedings of the 9th International Conference on the Principles and Practice of Knowledge Representation, KR 2004. Please use this reference.

3] and have recently also been applied in the context of graph mining [20]. The key idea underlying condensed representations (such as closed and free sets) is that one aims at finding a representative for each equivalence class of frequent patterns (the so-called closed set) instead of finding all frequent patterns. In this context, we first introduce the novel concept of *semantic* freeness and closedness, where the term *semantic* refers to the use of a declarative background knowledge about the domain of interest that is specified by the user. Under this notion, two clauses $c_1$ and $c_2$ are semantically equivalent if and only if $KB \models c_1 \leftrightarrow c_2$. Employing such a background knowledge and semantics is an elegant alternative to specifying complex and often ad hoc language bias constraints (such as those for dealing with symmetric predicates). In addition, it provides the user with a powerful and declarative tool for guiding the mining process. Secondly, we upgrade the already existing notions of freeness and closedness for use in an inductive logic programming setting, and show how these traditional notions are related to the new notions of semantic condensed representations. Thirdly, a novel frequent Datalog clause engine, called *c-armr*, that works with condensed representations is presented and experimentally validated.

## 2 Problem Setting

We assume some familiarity with Datalog and Prolog. The task of frequent query mining was first formulated in [5, 6]:

- **Given**
  - a language of clauses $\mathcal{L}$
  - a database $\mathcal{D}$
  - a predicate $key/1$ belonging to $\mathcal{D}$
  - a frequency threshold $t$
- **Find** all clauses $c \in \mathcal{L}$ such that $freq(c, \mathcal{D}) \geq t$.

The database $\mathcal{D}$ is a knowledge base written in Prolog or Datalog; it contains the data to be mined. In this extended abstract, we will assume that all queries posed to the database terminate and also that all clauses are range-restricted (i.e. all variables in the conclusion part of the clause also occur in the condition part). The first requirement can be guaranteed by employing Datalog; in the full version of the paper, we will address complications that may arise when these assumptions do not hold. The database $\mathcal{D}$ is assumed to contain a special predicate $key/1$ which determines the entities of interest and what is being counted. The language of clauses $\mathcal{L}$ defines the set of well-formed clauses. All clauses in $\mathcal{L}$ are assumed to be of the form $p(K) \leftarrow key(K), q_1, ..., q_n$ where the $q_i$ are different literals. Within inductive logic programming, $\mathcal{L}$ typically imposes syntactic restrictions on the clauses to be used as patterns, most notably, type and mode restrictions. Finally, the frequency of a clause $c$ with head $key(K)$ in database $\mathcal{D}$ is defined as

$$freq(c, \mathcal{D}) = \mid \{\theta \mid \mathcal{D} \cup c \models p(K)\theta\} \mid \quad (1)$$

So, the frequency of a clause is the number of instances of $p(K)$ that are logically entailed by the database and the clause.

*Example 1.* Consider the database consisting of the following facts: drinks(jan,duvel), drinks(hendrik,calvados), drinks(luc,hoegaarden), beer(duvel), beer(hoegaarden), key(jan), key(hendrik), key(luc).
The frequency of the clause $h(X) \leftarrow key(X), drinks(X, B), beer(B)$ is two.

Observe that the constraint $freq(c, \mathcal{D}) \geq t$ is anti-monotonic. A constraint $c$ on patterns is anti-monotonic if and only if $c(p)$ and $q \preceq p$ implies $c(q)$, where, $q \preceq p$ denotes that $q$ subsumes (i.e. generalizes) $p$.

An important problem with the traditional setting for frequent pattern mining in inductive logic programming is that – due to the expressiveness of clausal logic – the number of patterns that is considered and generated is extremely large. This does not only cause problems when interpreting the results but also leads to computational problems. The key contribution of this paper is that we address this problem by introducing condensed representations for inductive logic programming.

## 3 Knowledge for Condensed Representations

It is often argued that one of the advantages of inductive logic programming is the ease with which one can employ background knowledge in the mining process. Background knowledge typically takes the form of dividing the database $\mathcal{D} = KB \cup D$ into two components: the background knowledge $KB$ and the data $D$, where $KB$ constitutes the *intentional* and $D$ the *extensional* part of the database. The idea is then that both intentional and extensional predicates are used in the hypotheses and that their nature is transparent to the user. With only a few exceptions, most inductive logic programming systems do not employ the background theory during hypothesis generation[1] but only while computing the coverage or frequency of candidate clauses. These systems typically search the $\theta$-subsumption [15] or $OI$-subsumption lattice [11] starting at the empty clause and by repeatedly applying a refinement operator. Because typical refinement operators do not employ background knowledge, they generate many clauses that are semantically equivalent. As the inductive logic programming system is unaware of this, unnecessary work is being performed and the search space is blown up resulting in severe inefficiencies.

*Example 2.* Consider that we have the predicates $p$ and $q$ and that we know that $p \leftarrow q$. A typical refinement operator – working with a canonical form[2]–

[1] Perhaps, the only exception is Progol [12], which employs the background knowledge to generate a most specific clause in $\mathcal{L}$ that covers a specified example, and older heuristic systems that employ the background knowledge during refinement using a resolution based operator, cf. [1]).

[2] A canonical form is used to avoid generating the same clause more than once, e.g. $h \leftarrow k, p, q$ and $h \leftarrow k, q, p$. Refinement operators working with a canonical form are sometimes called *optimal*, cf. Section 5.

will generate $h \leftarrow k, p$; $h \leftarrow k, p, q$ and $h \leftarrow k, q$. However, given $p \leftarrow q$ the first two generated clauses are equivalent.

To alleviate this problem, we introduce, as the first contribution of this paper, the notions of semantically closed and semantically free clauses. These novel concepts assume that a background theory $KB$ is given in the form of a set of Horn-clauses. At this point, we wish to stress that the background theory used should not contain information about specific examples and also that the theory used here might well be different than the one implicitly used in $\mathcal{D}$. So, $KB$ here denotes a set of properties about the domain of interest.

**Definition 1.** *A clause $h \leftarrow k, q_1, ..., q_n$ is* semantically free*, or* s-free*, w.r.t. the background knowledge $KB$ if and only if there is no clause $p_0 \leftarrow k, p_1, ..., p_m$, where each $p_i$ corresponds to a single $q_j$, for which $KB \models p_0 \leftarrow p_1, ..., p_m$*[3]*.*

*A clause $h \leftarrow k, q_1, ..., q_n$ is* semantically closed*, or* s-closed*, w.r.t. the background knowledge $KB$ if and only if $\{k\theta, q_1\theta, q_2\theta, ..., q_n\theta\}$ is the least Herbrand model of $KB \cup \{k\theta, q_1\theta, ..., q_n\theta\}$ where $\theta$ is a skolem substitution for $h \leftarrow k, q_1, ..., q_n$.*[4]

*A clause $h \leftarrow k, q_1, ..., q_n$ is* consistent *if and only if $KB \cup \{k\theta, q_1\theta, ..., q_n\theta\} \not\models \square$ where $\theta$ is a skolem substitution.*

Intuitively, a clause is s-free if it is not possible to delete literals without affecting the semantics; it is s-closed if it is not possible to add literals without affecting the semantics. In Example 2, the clause $h \leftarrow k, p, q$ is s-closed, the other clauses are s-free, and all clauses are consistent. However, if we would have included the horn clause $false \leftarrow p, q$ in the background theory, then the clauses $h \leftarrow k, p, q$ and $h \leftarrow k, q$ would be inconsistent. Horn-clauses of this type act as constraints on the clauses.

The reader familiar with the theory of inductive logic programming might observe that s-closed clauses are closely related to Progol's bottom clauses [12] as well as to Buntine's notion of generalized subsumption [4]. Indeed, the two clauses $h \leftarrow k, q$ and $h \leftarrow k, p, q$ in Example 2, are equivalent under generalized subsumption. Observe also that each s-free clause has a unique s-closed clause, the *s-closure*, that is equivalent to it and that several s-free clauses may have the same s-closure.

From the above considerations, it follows that it would be beneficial if the search could be restricted to generate only s-closed clauses. Unfortunately, the constraint s-closed is not anti-monotonic. However, it turns out that s-freeness is an anti-monotonic constraint, which therefore can be integrated in traditional frequent query mining algorithms. This integration will be discussed in Section 5. Once the s-free clauses have been found, their s-closures can be computed and filtered to eliminate doubles. The s-closure of a clause $h \leftarrow k, q_1, ..., q_m$ w.r.t.

[3] A more general but less operational definition requires that there is no clause $p_0 \leftarrow k, p_1, ..., p_m$ such that $p_0, ..., p_m$ subsumes $q_1, ..., q_n$.

[4] Here it is assumed that the least Herbrand model is finite, which can be enforced when using Datalog and range-restriction.

the background theory can be computed as follows: compute a skolemization substitution $\theta$ for the clause, then compute the least Herbrand model $\{r_1, ..., r_n\}$ of $KB \cup \{k\theta, q_1\theta, ..., q_m\theta\}$, and then deskolemize the clause $h\theta \leftarrow k\theta, r_1, ..., r_n$ and return the result.

Observe that the constraint of s-closedness provides the user with a powerful means to influence the results of the mining process. Indeed, adding or removing clauses from the background theory will strongly influence the number as well as the nature of the discovered patterns. Basically, adding a clause of the form $h \leftarrow p, q$ has the effect of ignoring $h$ in clauses where $p$ and $q$ are already present. Therefore, the user may also desire to declare clauses in the background theory that do not possess a 100 per cent confidence.

## 4 Discovering Associations

Specifying all clauses that hold in the domain may be cumbersome and the question arises as to whether the data mining system may not be able to discover the clausal regularities that hold among the data. For item sets, sequential patterns and even graphs [20, 3, 21], techniques have been developed that discover high confidence association rules during the mining process and once discovered, employ them to prune the search. To this aim, we upgrade the notions of $\delta$-free and closed item sets [3] to clausal logic:

**Definition 2.** *A clause $h \leftarrow k, q_1, ..., q_n$ is* $\delta$-free *(with $\delta$ being a small positive integer), if and only if there exists no clause $c$ of the form $h \leftarrow k, p_1, ..., p_m, not\ p_0$, where each $p_i$ corresponds to a single $q_j$, for which $freq(c, \mathcal{D}) \leq \delta$.*

To understand this concept, first consider the clauses $h \leftarrow k, p_1, ..., p_m, not\ p_0$ and the case that $\delta = 0$. If such a clause has frequency 0, this implies that the association rule $p_0 \leftarrow k, p_1, ..., p_m$ has a confidence of 100 per cent. The negative literal *not* $p_0$ is used because we need to know that the rule holds *for all* substitutions. Now, the definition of 0-freeness is analogous to that of s-freeness except that these association rules are not specified in the background theory but rather implicit properties of the data. Indeed,

**Theorem 1.** *For $\delta = 0$, if $KB$ would contain all 100 per cent confidence association rules then a clause is $\delta$-free if and only if it is s-free w.r.t. $KB$.*

Now consider the case that $\delta \neq 0$. Then rather than requiring association rules to be perfect, a (small) number of exceptions are allowed in each association rule. Observe that as $\delta$ increases, the number of $\delta$-free clauses will decrease.

For $\delta = 0$, we can define a corresponding notion of closedness.

**Definition 3.** *A clause $h \leftarrow k, q_1, ..., q_n$ is* closed *if and only if there exists no clause $c$ of the form $h \leftarrow k, p_1, ..., p_m, not\ p$,where each $p_i$ (but not $p$) corresponds to a single $q_j$, for which $freq(c, \mathcal{D}) = 0$.*

**Theorem 2.** *If $KB$ contains all 100 per cent confidence association rules then a clause is closed if and only if it is s-closed w.r.t. $KB$.*

We have not defined a corresponding notion of $\delta$-closedness because traditional deduction rules do not hold any more. Indeed, from the fact that the association rules $p \leftarrow q$ and $q \leftarrow r$ have at most $\delta$ exceptions, one may not conclude that $p \leftarrow r$ has only $\delta$ exceptions.

Again, it is easy to see that $\delta$-freeness is an anti-monotonic property, which will be useful when developing algorithms.

One of the interesting properties of $\delta$-free clauses is that they can be used to closely approximate the frequencies of any frequent clause, cf. [3]. The following theorem follows from a corresponding result for item sets due to [3].

**Theorem 3.** *Let $\mathcal{D}$ be a database. Let $S$ be a set of $\delta$-free clauses (w.r.t. $\mathcal{D}$). Let $c_1 : p(K) \leftarrow x_1 \ldots x_m$ and $c_2 : p(K) \leftarrow x_1 \ldots x_n$ be clauses with $n > m$ such that $\forall i \in \{m \ldots n-1\} : \exists (h \leftarrow b) \in S, \exists \theta : h\theta = x_{i+1} \wedge b\theta \subset \{x_1, \ldots, x_i\}$. Then, $freq(c_1, \mathcal{D}) \geq freq(c_2, \mathcal{D}) \geq freq(c_1, \mathcal{D}) - \delta.n$.*

## 5 An Algorithm for Mining Frequent Clauses

Algorithms for finding frequent Datalog queries are similar in spirit to those traditionally employed in frequent item set mining. A high-level algorithm for mining frequent clauses along these lines is shown in Algorithm 1. It searches the subsumption lattice breadth-first, it repeatedly generates candidate clauses $C_i$ that are potentially frequent, and tests for their frequency. To generate candidates, a refinement operator $\rho$ is applied. To employ a minimum frequency threshold in Algorithm 1 one must set $con = (freq(f, \mathcal{D}) \leq t)$. Using the abstract constraint $con$, it is also possible to employ other types of anti-monotonic constraints in Algorithm 1. Despite the similarity with traditional frequent pattern mining algorithms, there are also some important differences. First, traditional frequent pattern mining approaches assume that the language $\mathcal{L}$ is anti-monotonic. For clausal logic, a language is anti-monotonic when for all clauses $h \leftarrow k, p_1, ..., p_m \in \mathcal{L}$, all generalizations of the form $h \leftarrow k, p_1, ..., p_{i-1}, p_{i+1}, ..., p_m \in \mathcal{L}$. Even though this assumption holds for expressive pattern languages such as those involving trees or graphs, cf. [10], it is typically invalid in the case of inductive logic programming because of the mode and type restrictions. Indeed, consider e.g. the clause $p(K) \leftarrow key(K), benzene(K, S), member(A, S), atom(K, A, c)$. Even though this clause will typically be well-formed, its generalization $p(K) \leftarrow key(K), member(A, S)$ will typically not. Because the language $\mathcal{L}$ employed in inductive logic programming is not anti-monotonic, one must not only keep track of the frequent clauses, but also of the (maximally general) infrequent ones. Furthermore, when a new candidate is generated, it is tested whether the candidate is not subsumed by an already known infrequent one. (Our implementation uses an index to realize this efficiently). Second, in order to search efficiently for solutions, it is important that each relevant pattern is generated at most once. Early implementations [6] of frequent pattern mining systems in inductive logic programming were inefficient because they generated several variants of the same clause (w.r.t. $\theta$-subsumption) and had to filter these away using computationally

expensive subsumption tests. One approach to alleviating is to define a canonical form for clauses and to employ a so-called optimal refinement operator that only generates clauses that are in this canonical form, cf. [14, 13]. More formally, the canonical form we employ is defined as follows:

**Definition 4.** *A clause $h \leftarrow k, p_1, ..., p_m$ with variables $V_1, ..., V_n$ (ordered according to their first occurrence from left to right) is in canonical form if and only if $(h \leftarrow k, p_1, ..., p_m)\theta$ where $\theta = \{V_1 \leftarrow 1, ..., V_n \leftarrow n\}$ is the smallest clause according to the standard lexicographic order on clauses that can be obtained when changing the order of the literals $p_i$ in the clause.*

If one furthermore requires that all variables in a pattern are instantiated to a different term (as in $OI$-identity [11]) when determining whether a clause covers an example it is possible to define an optimal refinement operator. A refinement operator $\rho$ is optimal for a language $\mathcal{L}$ if and only if for all clauses $c \in \mathcal{L}$ there is exactly one sequence of clauses $c_0, ..., c_n$ such that $c_0 = \top$ and $c_n = c$ for which $c_i \in \rho(c_{i+1})$.

This approach is related to those employed when searching for frequent subgraphs, cf. [20, 10], and is largely adapted in our implementation. (The details of the refinement operator working with this canonical form will be further elaborated in the longer version of this paper.) Third, computing the frequency of a clause is computationally expensive as one evaluates a query against the whole database. Several optimizations have been proposed in this context, cf. [2].

In our implementation, we employ the smartcall introduced in [16] in combination with a heuristic for speeding up the computation of a coverage test (i.e. a test whether a clause covers a specific example). The heuristic orders the literals in a clause according to the number of answers it has. A more detailed description of the effect of such an optimisation is given in [17]. We also store the identifiers of the covered example which each frequent clause, which allows to reduce the number of coverage tests needed as well as further optimizations.

One further feature of our implementation is worth mentioning. It was a design goal to produce a *light* Prolog implementation that would be small but still reasonably efficient. In this regard, because many Prolog systems lack intelligent garbage collection and have indexing problems for huge amounts of data, it turned out crucial that the main memory required by Prolog is kept as small as possible. This was realized by writing the sets $F_i$ to files at level $i$, and then reading the frequent clauses $c$ again at level $i+1$ in order to compute the refinements $\rho(c)$ potentially belonging to $C_{i+1}$. Similarly, the $I_i$ are written to a file and indexed after each level. We plan to release the code of the system to the public domain, once the paper is published.

## 6 Adaptations for Mining Free Clauses

Let us now discuss how to adapt the previously introduced algorithm for mining free sets.

First, concerning the s-free clauses, we have made the following enhancements:

**Algorithm 1** Computing all clauses that satisfy an anti-monotonic constraint *con*.

$C_0 := \{h(K) \leftarrow key(K)\}$
$i := 0; F_0 := \emptyset; I_0 := \emptyset$
**while** $C_i \neq \emptyset$ **do**
  $F_i := \{h \in C_i \mid con(h) = true\}$
  $I_i := C_i - F_i$
  $C_{i+1} := \{h \mid h \in \rho(h'), h' \in C_i\}$
  $i := i + 1$
  $C_i := \{h \mid h \in C_i \text{ and } \neg\exists s \in \bigcup_j I_j : s \preceq h\}$
**end while**

- use the constraint $(freq(c, \mathcal{D}) \geq t) \wedge s - free(c, KB)$ instead of only the minimum frequency threshold; this constraint is also anti-monotonic, hence the algorithm can directly be applied;
- those candidates that do not satisfy the s-freeness constraint are simply added to the appropriate $I_i$
- finally, before testing whether a candidate clause $c$ is subsumed by an already known infrequent clause, replace $c$ by its completion under $KB$; this will allow further pruning to take place.

Second, for what concerns the $\delta$-free clauses, we employ the first two enhancements. Observe that it is possible as well as desirable to employ the constraint $(freq(c, \mathcal{D}) \geq t) \wedge \delta - free(c, \mathcal{D}) \wedge s - free(c, KB)$. Then one does not only use the already available knowledge in the background but also tries to discover new knowledge. One interesting alternative for adding the non-$\delta$-free candidates to the $I_i$ is to simply add them to the background theory. Doing so results in propagating the effects of the discovered association rules by combining their conclusions with those already in the background theory. When $\delta$=0, this will always yield correct results. However, when $\delta \neq 0$, this might lead – in some cases – to some unsound conclusions (because deduction using $\delta$-free is not sound as illustrated in Section 4).

## 7 Conclusions and Related Work

We have introduced various types of condensed representations for use in inductive logic programming and we have demonstrated that this reduces the number of patterns searched for, the number of solutions as well as the time needed for the discovery task. Although condensed representations have been used in the context of simpler pattern languages (such as item sets, sequences and graphs), it is the first time that they have been employed within an inductive logic programming setting. The notions $\delta$-free and closed clauses are a direct upgrade of the corresponding notions for item sets. However, the semantic notions are novel and could easily be applied to the simpler pattern domains as well. The semantic notions are somewhat related to the goal of deriving a non-redundant theory in

the clausal discovery engines by [9, 7]. In these engines, one was computing a set $H$ of 100 per cent confidence association rules in the form of clauses such that no clause in $H$ was logically redundant. Employing our semantic notions has a similar effect as working with Buntine's [4] generalized subsumption notion.

At this point, we wish to stress that working with semantically free and/or closed clauses is not only useful when mining for frequent patterns, but can also be beneficial when mining for other types of patterns, such as for classification rules. This could – in an inductive logic programming setting – easily be implemented by employing a semantic refinement operator.

## Acknowledgements

The first author was partly supported by the EU FET project cInQ (Consortium on Inductive Querying). The authors would like to thank Jean-Francois Boulicaut and Jan Struyf for interesting discussions about this work, and Andreas Karwath for his comments on an early version of this paper.

## References

1. Francesco Bergadano, Attilio Giordana, Lorenza Saitta: Biasing Induction by Using a Domain Theory: An Experimental Evaluation. *European Conference on AI*, 1990: 84-89.
2. Hendrik Blockeel, Luc Dehaspe, Bart Demoen, Gerda Janssens, Jan Ramon, Henk Vandecasteele: Improving the Efficiency of Inductive Logic Programming Through the Use of Query Packs. *Journal of Artificial Intelligence Research* 16: 135-166 (2002)
3. J-F. Boulicaut, A. Bykowski, C. Rigotti. Free-sets: a condensed representation of boolean data for the approximation of frequency queries. *Data Mining and Knowledge Discovery journal*, 7(1) 2003. pp. 5-22.
4. W. Buntine. Generalized subsumption and its application to induction and redundancy. *Artificial Intelligence*, 36:375–399, 1988.
5. L. Dehaspe and L. De Raedt. Mining Association Rules in Multiple Relations. In S. Dzeroski and N. Lavrac, editors, *Proceedings of the 7th International Workshop on Inductive Logic Programming,* volume 1297 of Lecture Notes in Artificial Intelligence, pages 125–132. Springer-Verlag, 1997.
6. Luc Dehaspe, Hannu Toivonen: Discovery of Frequent DATALOG Patterns. *Data Mining and Knowledge Discovery* 3(1): 7-36 (1999)
7. Luc De Raedt, Luc Dehaspe: Clausal Discovery. *Machine Learning* 26(2-3): 99-146 (1997).
8. F. Esposito, N. Fanizzi, S. Ferilli, and G. Semeraro. Ideal Refinement under Object Identity. In P. Langley, editor, *Proceedings of the 17th International Conference on Machine Learning,* pages 263–270. Morgan Kaufmann, August 2000.
9. N. Helft. Induction as nonmonotonic inference. In *Proceedings of the 1st International Conference on Principles of Knowledge Representation and Reasoning,* pages 149–156. Morgan Kaufmann, 1989.
10. Akihiro Inokuchi; Takashi Washio; Hiroshi Motoda. Complete Mining of Frequent Patterns from Graphs: Mining Graph Data. *Machine Learning* 50(3): 321-354; 2003.

11. D. Malerba and F. A. Lisi. Discovering Associations between Sapatial Objects: An ILP Application. In Cline Rouveirol and Michle Sebag, editors, Proceedings of the 11th International Conference on Inductive Logic Programming, volume 2157 of Lecture Notes in Artificial Intelligence, pages 156–163. Springer-Verlag, September 2001.
12. S. Muggleton. Inverse Entailment and Progol. New Generation Computing, Special issue on Inductive Logic Programming, 13(3-4):245–286, 1995.
13. Siegfried Nijssen, Joost N. Kok: Efficient Frequent Query Discovery in FARMER. in Nada Lavrac, Dragan Gamberger, Hendrik Blockeel, Ljupco Todorovski (Eds.): *Proceedings of the 7th European Conference on Principles and Practice of Knowledge Discovery in Databases, Proceedings.* Lecture Notes in Computer Science 2838 Springer 2003.
14. Siegfried Nijssen, Joost N. Kok: Faster Association Rules for Multiple Relations. IJCAI 2001: 891-896 Bernhard Nebel (Ed.): *Proceedings of the Seventeenth International Joint Conference on Artificial Intelligence, IJCAI 2001,* Seattle, Washington, USA, August 4-10, 2001. Morgan Kaufmann 2001.
15. G.D. Plotkin. A note on inductive generalization. In *Machine Intelligence,* volume 5, pages 153–163. Edinburgh University Press, 1970.
16. V. Santos Costa, A. Srinivasan, R. Camacho, H. Blockeel, B. Demoen, G. Janssens, J. Struyf, H. Vandecasteele, and W. Van Laer, Query transformations for improving the efficiency of ILP systems, *Journal of Machine Learning Research* (2002).
17. J. Struyf, and H. Blockeel, Query optimization in Inductive Logic Programming by reordering literals,(Horvath, T. and Yamamoto, A., eds.) *Proceedings of the 13th International Conference,* Lecture Notes in Computer Science, vol 2835, pp. 329-346, 2003.
18. Ashwin Srinivasan, Stephen Muggleton, Michael J. E. Sternberg, Ross D. King: Theories for Mutagenicity: A Study in First-Order and Feature-Based Induction. *Artificial Intelligence* 85(1-2): 277-299 (1996)
19. I. Weber. Discovery of First-Order Regularities in a Relational Database Using Offline Candidate Determination. In S. Dzeroski and N. Lavrac, editors, *Proceedings of the 7th International Workshop on Inductive Logic Programming,* volume 1297 of Lecture Notes in Artificial Intelligence, pages 288–295. Springer-Verlag, 1997.
20. Xifeng Yan, Jiawei Han: gSpan: Graph-Based Substructure Pattern Mining. in *Proceedings of the 2002 IEEE International Conference on Data Mining (ICDM 2002),* Japan. IEEE Computer Society 2002.
21. Mohammed Javeed Zaki: Generating non-redundant association rules. *KDD* 2000: 34-43