Scalability and efficiency in multi-relational data mining

Hendrik Blockeel
Department of Computer Science
Katholieke Universiteit Leuven
Celestijnenlaan 200A, B-3001 Leuven, Belgium
hendrik.blockeel@cs.kuleuven.ac.be

Michèle Sebag
LRI, Université Paris-Sud
91405 Orsay, France
michele.sebag@lri.fr

ABSTRACT

Efficiency and scalability have always been important concerns in the field of data mining, and are even more so in the multi-relational context, which is inherently more complex. The issue has been receiving an increasing amount of attention during the last few years, and quite a number of theoretical results, algorithms and implementations have been presented that explicitly aim at improving the efficiency and scalability of multi-relational data mining approaches. With this article we attempt to present a structured overview.

1. INTRODUCTION

Efficiency and scalability have always been important concerns in the field of data mining. They are even more so when we focus on multi-relational data mining. The increased complexity of the task calls for algorithms that are inherently more expensive, computation-wise: larger hypothesis spaces are searched and evaluation of a single hypothesis becomes more complex. For instance, in database terminology, evaluation of a single hypothesis might involve one or more joins between tables, which is not the case for classical data mining methods.

In this article, we attempt to give an overview of recent evolutions in multi-relational data mining that have influenced the efficiency and scalability of certain approaches. We do not aim at giving an exhaustive survey of existing techniques, but rather try to create a structured context in which they can be placed. Many of the techniques and ideas we discuss here originate in inductive logic programming (ILP), but most of them carry over to the general context of relational databases, as we will repeatedly point out.

Typically, ILP techniques perform a search through some large hypothesis space, during which many hypotheses are generated and evaluated. There are two obvious ways in which this process can be made more efficient: by reducing the number of hypotheses evaluated, and by making the evaluation process itself more efficient. This is a first classification suitable for many (though not all) optimization techniques.

We can further distinguish techniques that increase efficiency at the cost of correctness, and techniques that preserve correctness. Correctness in this context should be understood as: yielding the same results as some reference algorithm that does not employ the technique. An algorithm that does not preserve correctness, should still give results that are with sufficiently high probability sufficiently similar to the reference results. Ideally this probability and similarity are formally defined, and are parameters of the algorithm.

Some efficiency gain can be obtained by changing the representation of the data. While the original work on ILP considers the given knowledge base to be monolithic, more recent approaches exploit a certain kind of locality of relevant knowledge. This influences both the efficiency with which hypotheses can be evaluated, and the ability to process data sets without loading them entirely into main memory. Other optimizations related to changes of knowledge representation are those that pre-compute and materialize certain information that will often be needed; these include so-called propositionalization approaches, where the multi-relational problem is cast into a single-relational form.

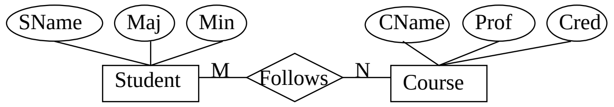
The above considerations are reflected in the structure of this article. We start (Section 2) with looking at aspects of representation: the way in which data and knowledge are represented influences the efficiency with which they can be processed. We continue by discussing methods for reducing the number of hypotheses that need to be evaluated, in Section 3, and discuss the computational complexity of hypothesis evaluation (and how to improve it) in Section 4. In Section 5 we discuss a class of methods that are more or less in the intersection of the former categories: these are methods that exploit similarities in data and hypotheses by processing data and hypotheses in such a way that computations can be shared. Scalability with respect to memory is discussed in Section 6. In Section 7 we present some related work that does not really belong to any of the previous categories. Finally, we illustrate the scalability of current state-of-the-art relational data mining systems with a number of concrete cases (Section 8), and conclude in Section 9.

2. REPRESENTATIONAL ASPECTS

A distinction is sometimes made between two paradigms in ILP: learning from entailment, and learning from interpretations [19]. Which one is used, has an effect on efficiency and scalability. This is mainly because they differ with respect to assumptions of locality of relevant information.

We do not go into technical details here, but illustrate the issues on an example. Consider a relational database that has information on students and the courses they follow. There are three relations: Student, Course, and Follows; the latter represents an M-to-N relationship between students and courses.

Several tasks can be defined on this database. We might



Student

SName	Maj	Min
joe	phil.	math
jane	math	phil.

Course

CName	Prof	Cred
calculus	Jones	4
algebra	Smith	3
history	Miller	4

Follows

SName	CName
joe	calculus
jane	algebra
jane	calculus

```
student(joe, phil, math).
student(jane, math, phil).
course(calculus, jones, 4).
course(algebra, smith, 3).
course(history, miller, 4).
follows(joe, calculus).
follows(jane, algebra).
follows(jane, calculus).
```

Figure 1: A toy database with information on students and courses.

want to classify students into specific classes, or cluster them. Similarly, we might want to classify or cluster courses (for instance, popular and non-popular courses), or tuples of the Follows relation (e.g., the target concept to characterize is which students follow which courses). Note that a natural join between the three relations leads to a universal relation for which the tuples can be mapped one-to-one with the Follows tuples, but many-to-one with Student or Course tuples. This implies that data mining tasks on the Follows relation are inherently propositional, while tasks on Student or Course are inherently relational.

Let us focus on the setting of classifying students. "Multi-relational" in this context refers to the fact that for a given student, information in different tuples in different relations is relevant. This information is typically linked to from a single tuple in the Student relation, via foreign keys. Thus, the classification of a student is based on information in a *subdatabase* of the original database, that is, a database with the same database schema as the original one but a subset of its tuples. We illustrate this with the following example.[1]

Example 1. *Consider the toy database shown in Figure 1. The figure shows an entity-relationship diagram describing the database structure, and a possible instance of the database. The instance is shown in a relational as well as a first order logic format; ILP systems would typically use the latter format. Given a specific student, say, Jane, we can identify that part of the database that is somehow connected to Jane and therefore possibly relevant for her classification. We call this the subdatabase describing Jane. It is shown in Figure 2, again in both relational and first order logic format.*

[1]More explanations and illustrations are given by De Raedt et al. [20] and a constructive definition of this subdatabase is given by Blockeel [11], p. 77–79.

Student

SName	Maj	Min
jane	math	phil.

Course

CName	Prof	Cred
calculus	Jones	4
algebra	Smith	3

Follows

SName	CName
jane	algebra
jane	calculus

```
student(jane, math, phil).
course(calculus, jones, 4).
course(algebra, smith, 3).
follows(jane, algebra).
follows(jane, calculus).
```

Figure 2: A partial database, containing the information relevant for classifying a single tuple of the Student relation.

While classically a distinction is made between propositional data mining (find patterns within the tuples of a single relation) and multi-relational data mining (find patterns that extend over different tuples of different relations), by introducing the notion of "individuals" we can make an other classification: methods that find patterns within the description of an individual, and those that find patterns that extend over individuals.

We can then distinguish three settings for data mining:

1. finding patterns within individuals that are represented as tuples
2. finding patterns within individuals represented as sets of tuples (that is, each individual is a sub-database of the original one)
3. finding patterns within the whole database

For instance, we could look for patterns that indicate which minors are often chosen with which majors. Such patterns can be found by looking only at Student tuples; hence this is setting 1. We could be interested in patterns regarding the behavior of students, e.g., which combinations of courses student tend to choose ("a student who takes course A will probably also take course B"). Note that information on the courses followed by a particular student is spread over several tuples, but the number of such tuples is limited (as a particular student follows a limited number of courses). Therefore, this task belongs to setting 2. Finally, finding patterns concerning multiple students ("courses followed by student A are often also followed by student B") belongs to setting 3, because the information relevant for such a pattern may be spread throughout the whole database.

Settings 2 and 3 are multi-relational. Setting 2 degenerates into Setting 3 if the subdatabases that are constructed turn out to be the whole database. In many cases, however, there is a natural notion of "individual" which causes the subdatabases to be much smaller than the original database. For instance, when mining molecular databases, patterns are sought within single molecules. The subdatabase then corresponds to the description of a single molecule, which is much smaller than the whole database.

Assuming that there is indeed a clear notion of "individual", there are two options: one is to mine the database in its original format, the other is to reformat the database, explicating the subdatabases. The latter option is used by some ILP systems that learn from interpretations, such as ACE [15], or those that use a term-based representation [26]. Following Flach and Lachiche's terminology [26], we call these representations "individual-centered", as opposed to the original "predicate-centered" representation.

The use of individual-centered representations has a number of advantages. First, it has a positive effect on the theoretical learnability of concepts. De Raedt and Džeroski [21] have obtained positive PAC-learnability results for this setting, and this is mainly due to the assumption that patterns are searched within individuals and that the description of individuals in the database is complete (that is, all relevant information on an individual is given).

Second, the individual-centered approach is more similar to the propositional mining setting, in that there is a clear notion of individual examples. Because of this, techniques from propositional learning can more easily be copied. For instance, sampling becomes easier (taking a subsample of individuals is difficult if it is not obvious which information in the database is relevant for which individuals), and so do techniques for processing data one example at a time (thus avoiding the need to have all data in main memory simultaneously; we will return to this in Section 6).

An obvious disadvantage of the individual-centered representation is that its format depends on the notion of "individual". In those cases where there are several natural individuals, a separate representation has to be formed for each of them. In the students and courses example, both students and courses may be natural individuals, and when we want to classify students we would need to use a different representation than when we want to classify courses.

A more extensive comparison of representations in the relational setting is made by Lachiche and Flach [48]. We conclude here by remarking that the difference between the individual-centered and the predicate-centered representations is to some extent similar to the difference between object-oriented and relational databases. In object-oriented databases, information on an individual is directly linked to the object representing the individual, whereas in a relational database, the information is spread throughout several tables and has to be looked up via indexes. It may well be that for multi-relational data mining, the object-oriented paradigm will turn out to be more suitable than the relational database paradigm.

3. MULTI-RELATIONAL DATA MINING AS SEARCH

Multi-relational data mining can be formalized either as a constraint satisfaction problem (e.g., find all clauses covering more than ε students) or as an optimization problem (find the most discriminant clauses, e.g., discriminating East Coast from West Coast students). In both cases, the task can be formulated as a search process. Given a hypothesis space H and some real-valued (respectively boolean) criterion c, find the clauses h in H such that they maximize c (resp. such that $c(h)$ holds).

Clearly, these goals can be formalized along Mannila and Toivonen's framework [52].

DEFINITION 1. *Multi-relational Data Mining is the process of finding all clauses h, aka hypotheses, in a language H, that satisfy a predicate c with respect to a database, or set of examples $\mathcal{E}$.*

$$\textit{Find } TH(h, H, r) = \{h|\ h \in H, h \textit{ satisfies } c(h, \mathcal{E})\}$$

Predicate c is most often related to the coverage of clause h, or a numerical expression thereof, which must either be greater than a user-fixed threshold, or reach an optimum value.

As exhaustive search is usually intractable because of the size of H, several approaches have been proposed to enforce an efficient search procedure. These approaches are based on different kinds of inductive biases: syntactical biases, search biases, and validation biases. In particular, one can:

- Identify a subspace H' of H in which the solution is; here, one uses prior knowledge or user's requirements to define syntactical biases (ILP) or pattern languages (MRDM).
- Identify rules for pruning the search space; this in a sense corresponds to dynamically adapting the hypothesis space during the search, cutting away parts of H for which it has become clear during the search that they cannot contain a solution. For instance, monotonic constraints naturally induce pruning rules [53]: if a clause covers less than ε students, any specialization of this clause will cover even fewer students.
- Weaken the task into finding T instead of the true solution set TH, where T contains all h such that $c(h)$ holds with a certain probability, or $c(h)$ is close to the optimum. This relaxation can be achieved by sampling the hypothesis space H (stochastic search biases), or by reconsidering the assessment of hypotheses (validation biases will be considered in more detail in Section 4.2).

3.1 Syntactical biases and typed logic

We distinguish the hypothesis space H and the search space, which is the subset of H actually evaluated during the search process.

ILP methods typically reduce the size of H by specifying as specifically as possible the form of potentially interesting hypotheses. This is done through a "language bias", or pattern language, which typically imposes syntactical constraints on the format of a hypothesis.

Types and input/output modes are often used in ILP [59]. By using typed arguments of predicates, certain nonsensical hypotheses are avoided; for instance, it does not make sense to say that X is the number of courses followed by a student and then test whether X attends course Y. Input/output modes tell the system which predicates generate certain information, and which consume this information. For instance, the *age* predicate returns for a certain person the age of that person, whereas the $<$ predicate compares two variables but cannot instantiate a variable to a specific number. In this respect, modes can be viewed as constraints on hypotheses or queries, enforcing their utility.

Also schemas are popular for defining hypothesis spaces. These schemas provide a more strict syntactical format for hypotheses, typically specifying which predicates have to occur in which order, but making some of them optional or leaving the variables that should occur in certain positions unspecified. Examples are DLAB (Dehaspe and De Raedt, 1996) or the schemata used, for instance, by RDT (Kietz and Wrobel, 1992). A more complete overview of language biases that have been used in ILP is given by Nedellec et al. [61].

While much work on declarative bias specifications uses logic programming terminology, several specification languages have been proposed that are much more in line with relational databases. For instance, Wrobel's Midos system [85] uses the notion of foreign links in its pattern language specification, and Knobbe et al. (2000) propose to use UML models to define a pattern language, where the patterns are graphical query representations ("selection graphs"). The use of selection graphs as patterns has since then been adopted by several other authors [6; 5].

Syntactical biases are often explicitly enforced through search operators (see below). An alternative is to include type constraints into the definition of H [49; 39], and make no restriction about the search operators.

3.2 Search biases and pruning rules

As mentioned in the introduction, ILP systems perform a search through a hypothesis space, generating and evaluating many candidate solutions and using the result of these evaluations to generate new candidates. The search usually stops at the first candidate solution meeting the requirements (on coverage, generality, etc.) [66], or it might continue until no better solution can provably be found, for instance using an A^* algorithm [59].

In each step, some candidate hypotheses are generated from the current hypotheses using so-called refinement operators. For instance, the construction of L_{k+1} candidates from the L_k ones in Apriori [1], constitutes a refinement operator. Along the same lines, many refinement operators in ILP proceed by adding or removing a literal from the current hypothesis.

Besides limiting the hypothesis space through syntactical biases, the actually traversed search space can be reduced further by introducing rules, or search biases, limiting the generation of candidate hypotheses (e.g., guiding the choice of the literals to be added or removed from the hypothesis). Some of these rules are related to the properties of refinement operators *per se*; these properties have been studied extensively in ILP. For instance, there is no point in generating a given candidate hypothesis more than once (non-redundancy property [4]). Conversely, no potentially relevant hypothesis should be skipped (completeness property). Nienhuys-Cheng and De Wolf [62] provide theoretical foundations for ILP in which refinement operators play an important role.

Other pruning rules are related to the properties of refinement operators in connection with the search criterion. Typically, the Apriori algorithm uses the anti-monotonicity of coverage to prune the candidates in L_{k+1}. Along the same principle, the systems Progol [59] or Aleph[2] typically perform an A^*-search which soundly cuts branches of the search tree without giving up the guarantee of finding the optimal hypothesis. Such pruning rules are based on the monotonic properties of the search criteria: typically, complexity can only increase as a hypothesis undergoes specialization; in the meanwhile, its coverage can only decrease; etc.

Ideally, the search criteria and refinement operators should be designed together, in such a way that i) any solution hypothesis can be obtained by refining an admissible candidate hypothesis; and ii) any refinement of a non-admissible hypothesis is non-admissible either.

When this is the case, the search space is said to be convex with respect to the criteria and the refinement operators [80]. Interestingly, monotonic and anti-monotonic constraints define a convex search space for level-wise algorithms [52], which explains their computational efficiency.

3.3 Search strategies and stochastic biases

Many data mining methods (such as decision tree induction or rule induction) use heuristics to guide their search, and multi-relational data mining methods are no exception to this rule. These methods attempt to find a good hypothesis, but usually do not guarantee that it is optimal, nor that it is "probably close to optimal" with a maximal deviation and minimal probability chosen by the user.

In practice, the exploration and pruning of candidate hypotheses can follow several search strategies. Among the most efficient strategies is depth-first exploration, retaining and refining only the current best hypothesis; this strategy is implemented in Foil [67] and Tilde [13] among others.

The price to pay for this efficiency is the myopia of greedy optimization. In the worst case (see Section 4.3), greedy search is trapped into some local optima of poor predictive quality. In other cases, it might happen that depth first exploration leads to unnecessarily specific hypotheses (see [63]) due to perturbations of the search criteria caused by the amount of data.

Beam search is another search strategy; it avoids the limitations of greedy myopic search, by retaining and refining a limited number of the best current hypotheses [9]. The computational cost varies linearly with the beam width. The advantage is that a better learning robustness is obtained through beam search, though there are still no guarantees of optimality.

A third strategy implements the stochastic, population-based exploration of the hypothesis space. This strategy is that of evolutionary computation and genetic algorithms (GAs) [31; 7], which crudely mimic the Darwinian principle of *survival of the fittest*. During each generation, candidate hypotheses are generated by randomly perturbing the current pool of hypotheses; the resulting hypotheses are thereafter evaluated, and the worst candidate hypotheses are filtered out.

An important point is that these perturbations might indifferently generalize or specialize the hypotheses, which makes it easier to escape from local optima.

GA-based relational learning, such as implemented in Regal [28], Dogma [34] or G-Net [3], usually provides very accurate and predictively efficient hypotheses, at a high computational cost; a few hundred of generations is routinely achieved, generating a few hundred candidate hypotheses each. As genetic search is intrinsically parallel (hypotheses are assessed independently of each other), the computational cost was an incentive to develop parallel implementations of GA-based relational learning [2].

The search space explored by GA-based relational learning

[2] http://web.comlab.ox.ac.uk/oucl/research/areas/machlearn/Aleph/

is usually defined from a template selected by the expert, in the line of DLab-like specifications; as could be expected, the choice of the template is critical to the success of learning [28].

More flexible search is permitted by variable-length evolutionary computation, more precisely *Genetic Programming* [43]. Genetic Programming extends the principles of genetic algorithms to tree-structured search space, and was specifically designed for optimization in program spaces [44; 8]. It has been used to explore Horn clauses and context-free grammar spaces [84; 68]. It also allows for direct exploration of higher order logic languages, such as Escher [49; 39].

Interestingly, efficient solutions are found on some problems after a few generations (three or four), indicating that pure random search might be sufficient to solve the learning problem. This statement is corroborated by recent results obtained with Monte-Carlo exploration, randomly sampling a set of hypotheses and returning the best one [77]. In spite of its simplicity, this Monte-Carlo relational learner yields results comparable with those of other learners on some application domains. Further research is concerned with elaborating truly uniform sampling mechanisms on complex relational hypothesis spaces.

4. EVALUATING CANDIDATE SOLUTIONS

As mentioned in the previous section, the search for solutions is interleaved with evaluating the current candidate solutions with respect to the database. These evaluations involve matching the condition part of the hypotheses to specific examples. Therefore, the efficiency of this matching procedure is critical to relational data mining.

In this section, we first introduce the matching procedure most used in ILP (θ-subsumption [65]), and compare it with logical querying. In the general case, logical queries and θ-subsumption test are equivalent to NP-hard constraint satisfaction problems. For this reason, several optimization heuristics have been developed and will be presented. Last, a theoretical study of θ-subsumption, based on the phase transition paradigm [36] has been achieved [29] and its impact on the scalability of ILP has been examined on artificial problems. These results are briefly summarized and discussed.

4.1 Logical queries and θ-subsumption

A (candidate) solution is most often of the kind *all instances satisfying condition A also satisfy condition B*, where condition B usually is a very simple one (e.g., membership of some class). The focus therefore is on optimizing the process of collecting the examples satisfying condition A, where A corresponds to a conjunctive query.

More generally in ILP, a candidate solution is a (set of) clause(s); matching a clause with an example boils down to searching a variable instantiation such that the body of the clause is true given the example.

For efficiency reasons,[3] the relational matching test used in the ILP literature [58; 60; 25] is the theta-subsumption test defined by Plotkin [65]; it amounts to finding a variable instantiation for the clause body (resp. head) such that this body (resp. head) becomes a subset of all facts in the example. In the particular case where clauses are equivalent to conjunctive queries (i.e., their head is empty), θ-subsumption is equivalent to query containment.

[3]Logical implication is not decidable in the general case [69]. For this reason, the ILP literature uses a weaker covering test, correct but not complete.

DEFINITION 2 (θ-SUBSUMPTION). *Clause C_1 θ-subsumes clause C_2 if and only if there exists a substitution θ mapping the variables in C_1 onto the variables/constants in C_2 such that all literals in $body(C_1)\theta$ appear in $body(C_2)$ and $head(C_1)\theta = head(C_2)$.*

EXAMPLE 2. *Let C be defined as the conjunctive query $? - student(X, \_, \_)$, $student(Y, \_, \_)$, $follows(X, calculus)$, $follows(Y, algebra)$. C subsumes the clause given by the conjunction of all facts in the toy database, Fig. 1. Indeed, C subsumes the database given in Fig. 1 according to two possible substitutions $\theta_1 = \{X/joe,\ Y/jane\}$ and $\theta_2 = \{X/jane,\ Y/jane\}$.*

In other words, clause matching corresponds to executing a logical query. The theory of first order logic and of logic programming (see, e.g., Lloyd, 1987) provides a large number of theorems and techniques that can be used to reformulate these queries, making them simpler or more efficient to execute.

Inspiration for improving the efficiency of matching is also provided by the database community. Reordering of relational algebra operations is a well-known method for improving the efficiency of a computation. For instance, when applying consecutive selections it is useful to apply the most selective ones first [38]. Similar techniques can be used to improve the efficiency of clause-example matching [78].

Note, however, an important difference between query execution in relational or deductive databases and in Prolog systems: queries in a database are normally executed bottom-up, whereas the Prolog execution mechanism works top-down. This is natural from the point of view that databases aim at computing sets of results, whereas Prolog aims at confirming or denying the existence of at least one solution. This difference, however, influences the optimization techniques.

4.2 Optimization heuristics

As mentioned earlier on, theta-subsumption testing is NP-complete due to the fact that the literals in the clause and example need to be matched to each other, and the number of possible matchings grows combinatorially in the number of literals.

Several heuristics have been considered in the literature to keep the complexity under control and reduce the number of possible matchings as much as possible. These heuristics can be grouped into three categories: i) ad hoc heuristics operating on particular kinds of clauses; ii) optimization heuristics; iii) relaxed, stochastic, heuristics.

4.2.1 *Exploiting particular clause structures*

The simplest way of reducing the θ-subsumption cost is to consider only short hypotheses, if at all possible. Incidentally, this heuristic (also known as Occam's razor or simplicity bias) is built-in in most ILP systems.

Another heuristic relies on the decomposition of the hypothesis into independent (sets of) literals. As this is not possible in general (ILP systems look for connected clauses), a relaxed version of decomposability known as k-locality has been defined [41]; the idea is to take advantage of the fact that sets of literals are independent, after the instantiation of some variables has been defined.

EXAMPLE 3. *Consider the conjunctive query* $?-p(X,Y)$, $q(Y,Z)$, $r(Y,U)$. $q(Y,Z)$ *and* $r(Y,U)$ *are dependent, in the sense that success of the query* $?-q(Y,Z),r(Y,U)$ *cannot be determined by checking whether* $?-q(Y,Z)$ *succeeds independently of checking whether* $?-r(Y,U)$ *succeeds. However, given a fixed instantiation for* Y*, they are independent: for example, the query* $?-q(a,Z),r(a,U)$ *succeeds if and only if both* $?-q(a,Z)$ *and* $?-r(a,U)$ *succeed. If we know that the Prolog execution mechanism upon calling* $p(X,Y)$ *instantiates* Y*,*[4] *then the remainder of the query can be decomposed into independent parts. The query is transformed into* $?-p(X,Y),once(q(Y,Z)),r(Y,U)$. *The meta-predicate* `once` *indicates that only one solution for* $q(Y,Z)$ *need be generated. After finding a solution for* $q(Y,Z)$*, if no solution for* $r(Y,U)$ *is found, the normal Prolog execution mechanism would backtrack and try to generate different* Z *that make* $q(Y,Z)$ *true, but because of the independence property we know this is pointless. By introducing the* `once` *meta-predicate (which has a simple definition in Prolog) in the clause, this can be avoided, without changing the normal Prolog execution mechanism.*

Scheffer et al. [71] propose an improved implementation of k-locality, based on the construction of an intermediate data structure, the substitution graph.

Along the same lines, Santos Costa et al. [70] present a number of query transformations that can speed up query execution considerably. These are based on identifying parts of a query that can be checked independently of each other, which influences the efficiency in two ways. First, the standard backtracking process of Prolog has no notion of such independencies and may therefore perform unnecessary backtracking. This can be avoided by reordering literals into groups that succeed or fail independently of each other and placing cuts between these groups. (A more advanced version of this transformation applies the same principle recursively.) Second, in a typical ILP search process, part of the query is known to succeed for certain examples (because the clause is obtained by extending a previously encountered clause that has been evaluated already). Any part of the clause that succeeded previously and is independent from the extension of the clause, will certainly succeed and need not be tested again.

Another line of research examines the case of acyclic conjunctive queries. Following Gottlob [32], Horvath and Wrobel [37] discuss how efficiency gains can be obtained by considering only acyclic conjunctive queries, a relatively general subclass of queries for which the matching problem is tractable. Such classes of queries/hypotheses are particularly representative in tree-structured application domains, such as XML data.

4.2.2 Optimization heuristics

In the general, intractable, case, the procedures developed for constraint satisfaction problems (see Tsang [81] for a comprehensive presentation) can be exploited to enhance the θ-subsumption test efficiency. A specific rewriting of the matching clause problem into a binary CSP has been proposed [51], making it possible to employ standard arc consistency and constraint propagation procedures, and reduce the subsumption cost by some orders of magnitude.

4.2.3 Relaxing θ-subsumption

In particular regions of the search space, which will be further detailed in the next section, it makes sense to replace θ-subsumption by a stochastic estimate [73; 74]. Stochastic subsumption proceeds by uniformly sampling the set of substitutions matching the candidate solution with the example, and deciding whether the solution subsumes the example based on this sample only. Note that stochastic subsumption is correct, but not complete; if clause C stochastically subsumes example E, C subsumes E, but the converse does not hold. Hence, clause assessment based on stochastic subsumption is biased towards overly general solutions, although this bias might be countered by using again stochastic matching when applying the clause.

A last possibility is to evaluate clauses on a subsample of the available learning set. Sub-sampling of the learning set has been extensively investigated in propositional learning (e.g., [63; 72]), with considerable efficiency gains in computational cost and little loss if any in predictive accuracy in general. In the relational context, Srinivasan (1999) explores two sampling techniques (selecting sub-samples randomly, or through "windowing"; in the latter case examples with erroneous predictions are added to the sub-sample), with similar results.

4.3 The phase transition barrier

As mentioned earlier on, θ-subsumption testing and logical querying are equivalent to constraint satisfaction problems [33].

In CSPs, another framework for analyzing the computational complexity has appeared in the nineties [17]. Contrasting with average- and worst-case analysis, this novel framework handles complexity as a random variable depending on the order parameters of complexity (e.g., constraint density and tightness).

This framework has been adapted for analyzing the complexity of the θ-subsumption test by Giordana and Saitta [29]. Considering the θ-subsumption of example e by clause (hypothesis) h, four order parameters are considered:

- the number n of variables in h,
- the number m of predicate symbols in h,
- the number L of constants in e (assuming that e is a grounded clause)
- the number N of literals in e per predicate symbol in[5] h.

For the sake of simplicity, it is assumed that all predicates are binary, and that example e contains exactly N literals built on each predicate symbol in h.

For fixed values of parameters n and N, one thousand pairs (h,e) have been generated for each value of (m,L), where h is a conjunction of m literals built on m distinct predicate symbols involving n variables, and e is the conjunction of $m \times N$ ground literals, the arguments of which are uniformly selected without replacement in a set of L constants

[4]Such knowledge is often available in practice, and especially in the context of relational databases it is typically true because tuples in a relation do not contain free variables.

[5]Predicate symbols in e that do not appear in h do not play any role in the subsumption test.

$\{a_1, \ldots, a_L\}$.
For each value pair (m, L) one measures the average computational cost[6] of θ-subsumption (Fig. 3) and the percentage of success of the θ-subsumption test (Fig. 4).

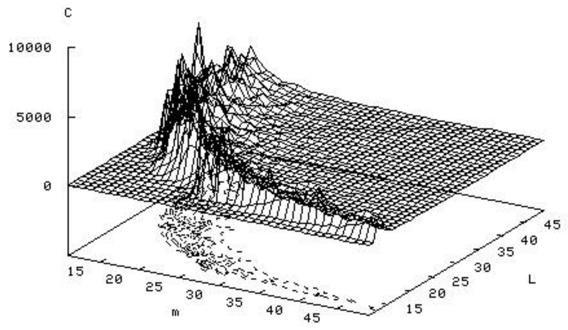


Figure 3: Computational cost of θ-subsumption in plane (m, L), averaged on 1,000 pairs h, e, for $N = 100$ and $n = 10$. (Reproduced with kind permission from A. Giordana and L. Saitta.)

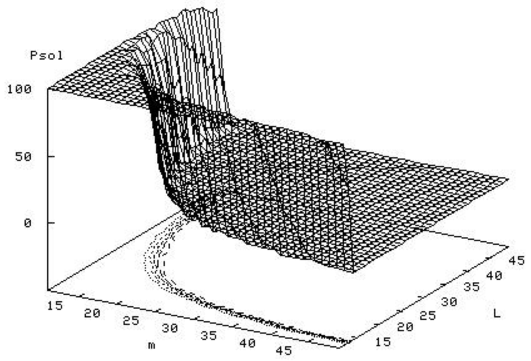
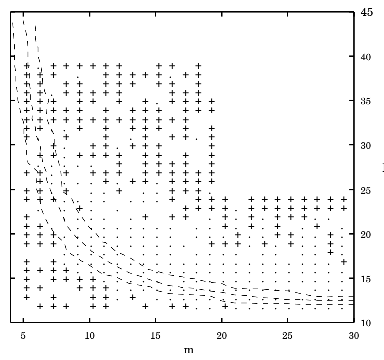


Figure 4: Percentage of successful θ-subsumption tests in plane (m, L) over 1,000 pairs h, e, for $N = 100$ and $n = 10$

This experiment confirms the findings of CSPs [35]. The effective complexity landscape depicted in Fig. 3 shows that the θ-subsumption cost is almost always negligible, except in a narrow region termed the *phase transition* region. The average complexity reaches its maximum in this region, where the probability of success of the θ-subsumption test abruptly decreases from almost 1 (in the high plateau on the left of Fig. 4, called *satisfiable* region) to almost 0 (in the low plateau on the right, called *insatisfiable* region).

EXAMPLE 4. *It is important to see that the (satisfiable and insatisfiable) regions characterize clauses and examples with respect to one another. By abuse of language, as the examples are fixed from the context in ILP and MRDM, one often says that a candidate clause belongs to the satisfiable or insatisfiable region.*
For instance, with respect to the toy database (examples) given in Fig 1, clauses such as $follows(X, Y), course(Y, \_, \_)$ belong to the satisfiable region (the examples contain many students following many courses).
In contrast, clauses such as $follows(X, Y)$, $course(Y, \_, \_)$, $follows(X', Y)$, $follows(X'', Y)$, $X \neq X', X \neq X'', X' \neq X''$ would belong to the insatisfiable region, as the toy database contains at most two students following the same course. (One sees that this clause would belong to the satisfiable region if a larger database had been considered).

This landscape describes the behavior of hypotheses and examples with respect to each other. When examples are fixed, as is the case in machine learning and data mining, the satisfiable region would correspond to overly general hypotheses (almost surely subsuming all examples), and the insatisfiable region to overly specific hypotheses (almost surely subsuming no examples).
The phase transition phenomenon that is observed for the θ-subsumption test, has far reaching effects on the behavior of relational learners [30]. Comprehensive experiments on artificial learning problems first show that most learners tend to select hypotheses lying in the phase transition. In retrospect, this should have been expected since this region concentrates the hypotheses separating the examples.
These experiments also demonstrate that the greedy optimization of coverage-related criteria is misleading, when dealing with long examples with poor background knowledge. Fig. 5 displays the competence map associated to FOIL [66], i.e. the region where FOIL succeeds in learning (the predictive accuracy on the test examples being greater than 80% for all problems indicated with a +), while the failure region is indicated with a $\cdot$.



Legend	+	predictive accuracy on test set > 80 %
	$\cdot$	predictive accuracy on test set $\approx$ 50 %

Figure 5: Competence map of FOIL in plane (m, L). The phase transition region is situated between the dotted curves.

Typically, when the learning search starts in the satisfiable (or in the insatisfiable) region, the coverage criterion just misleads the learner since it hardly makes any relevant difference between the hypotheses[7]. The reader is referred to Giordana et al. [30] for more details.

[6]The conjecture done in CSP is that the height of the complexity peak depends on the algorithm used to solve the CSP problem, but the location of the peak is independent of this algorithm [36].

[7]Unexpectedly, learning becomes easier as the sought target concept is longer, and the learning problem is farther away from the PT region. A tentative explanation offered for this fact is that the number of generalizations of the target concept in the PT, which are acceptable solutions, exponentially grows with the size of the target concept.

These experiments on "Needle-in-the-Haystack"-like problems suggest that novel heuristics are required to learn long target concepts [75; 10].

However, these results must be taken with care, for two reasons. First of all, phase transition depicts a global behavior, and does not say anything on a particular case (meaning that simple problems can be met within the phase transition, and hard problems can be encountered in the middle of the satisfiable or insatisfiable region). Secondly, the extensive study done by [30] relies on artificial problems; it can be the case that real-world problems involve typical features (e.g., the existence of large cliques, very unlikely in artificial problems), which might in turn significantly facilitate (or hinder) learning. For instance, in most real-world problems it is expected that the phase transition is larger and smoother, for examples will have different sizes (number of constants, number of literals per predicate).

5. SHARING COMPUTATIONS

The methods that we describe in this section, aim at improving the efficiency of the matching procedure, just like the previous ones. They are different in the sense that here we look at the matching process in a context where *many similar clauses are matched to the same examples.* In this context, some computations may be repeated over and over again, and it makes sense to try to store intermediate results instead of recomputing them. We distinguish three approaches: materialization of features, pre-computation of statistics, and reorganizing the search in such a way that intermediate results can be reused without effectively storing them.

5.1 Materialization of Features

Repeated execution of the same or similar queries can give rise to a lot of redundant computation. For instance, in the context of mining a database containing molecular structures, consider a predicate $benzene(L)$ that instantiates L to a list of atoms that form a benzene ring. If a molecule is represented by listing its atoms and the bonds between them, finding all benzene rings in a molecule involves a relatively expensive search process. If no special measures are taken, this computation is repeated each time a call to $benzene$ occurs.

In this case it is clearly more efficient to materialize the $benzene$ predicate, that is, to compute for each molecule only once which benzene rings occur in it and store this information explicitly with the description of the molecule. Of course this increases the memory requirements of the database. In general, this option is desirable in those cases where the computation is complex and the number of results to be stored is limited.

This process can be automated: an ILP system could easily materialize all background predicates and add the results to its database. As this may increase the size of the database considerably, it should happen in a controlled fashion, that is, the predicates to be materialized should be selected carefully. Currently this decision, and usually also the materialization itself, are left to the user. The development of heuristics for automatically selecting predicates to be materialized seems quite feasible and might have a considerable effect on the ease with which ILP systems can be applied in practice.

We add that current Prolog technology includes methods for storing intermediate results at the fact level (instead of the predicate level): this is known as tabling [18]. Tabling can be considered a lazy version of the materialization mentioned above, and might in some cases be preferable over materialization of complete predicates. The main problem remains the choice for which predicates tabling should be employed.

Propositionalization, as proposed by several authors (for an overview, see Kramer et al. [46]), is also an instance of precomputation of features. In this case, precomputed features are not added to a relational description of the examples, but to a propositional description, so that a propositional learner can be run afterwards. The efficiency gain obtained by running a propositional learner is obvious, but the transformation to propositional format is usually not lossless: the information content of the propositional representation is not equivalent to that of the original relational representation.

5.2 Pre-computation of Statistics

In the previous subsection we discussed how features of individual instances can be pre-computed. Pre-computation is also possible at the level of the database as a whole. In this case, statistics that describe the database and that will be needed several times during the induction process, are computed in advance and stored.

Moore and Lee [56] present a good example of this approach. They argue in favor of precomputing sufficient statistics for induction procedures. More precisely, given a table with n attributes, they propose to count and store the frequency of every combination of attribute values occurring in the database. Note that this amounts to estimating the full joint probability distribution of the domain of the table. This full distribution contains all necessary information to compute for instance class entropy (for decision tree induction), conditional probabilities (for Bayesian approaches), etc. They propose a data structure called *AD-tree* to efficiently store the distribution. Note that the memory consumption of an n-dimensional table representing the full joint probability distribution is proportional to 2^n. AD-trees are a sparse data structure that stores only the non-zero frequencies explicitly; thus the size of this representation is bounded linearly by the size of the data set. A number of other tricks are used to minimize the memory consumption of the data structure. Nevertheless, such AD-trees can still be very large.

Pavlov et al. [64] look at the so-called query selectivity estimation problem, where the task is to estimate the size of the result set of a query in a database. Note that this is exactly the kind of query that is frequently generated by data mining systems. They compare several approaches to approximating the joint probability distribution of a relation; these approaches include AD-trees, models based on independence of all attributes, maximum entropy modeling (which models some dependencies between combinations of attributes), and more. They compare the efficiency and accuracy of these approaches, and conclude that counts can often be estimated quite accurately and efficiently with models of reasonable size. In other words, even when AD-trees are infeasible because of their size, accurate estimates of statistics can be made efficiently using other techniques.

All of the previous work is set in a propositional setting, but it is obvious that similar approaches could be employed in a relational setting, with equally large efficiency gains to be expected. Some work in this category is presented by Getoor et al. [27]. They define stochastic relational models, which form a probabilistic description of a relational database based on relational Bayesian networks. Further work in this direction seems very promising.

5.3 Reorganizing Computations to Reuse Intermediate Results Without Storing Them

The previous methods are based on storing intermediate results for later use. This principle is applicable only when there is little risk of running out of memory because of this storage. An alternative approach that avoids this risk, is to reorganize the computations in such a way that intermediate results are used immediately after they have been produced, so that they do not have to be stored for a long time. This implies that all consumers of these results should be run shortly after the results have been produced, which often makes some kind of parallelism necessary (that is, algorithms are conceptually run in parallel, not necessarily on parallel hardware).

One approach in this category is the work on query flocks by Tsur et al. [82]. A query flock is a set of queries where all queries have the same structure but differ with respect to specific constants that are filled in certain positions. An example of such a flock, taken from Tsur et al., is

```
answer(P) :- exhibits(P, $s), treatments(P, $m),
             diagnoses(P, D), not causes(D, $s).
```

The idea is that if the user is interested in all couples ($s,$m) that co-occur at least c times in a database,[8] instead of computing the count for each ($s,$m) combination consecutively, it is better to run a single query (the flock) through which all the counts are simultaneously computed.

A similar approach is proposed by Blockeel et al. [15]. In this case, the set of queries that is evaluated consists of queries that share part of their structure, but not all of it. The queries are structured into a kind of tree, called a query pack, so that the common part of the queries is represented only once. Such a tree can be defined in Prolog and executed by any standard Prolog engine, but to execute it in a maximally efficient way, changes at the level of the Prolog interpreter are necessary. ILPROLOG is a dedicated Prolog system for data mining that provides such a pack execution mechanism [15; 83].

EXAMPLE 5. *Consider the set of queries*

```
?- p(X), I = 1.
?- p(X), q(X,a), I = 2.
?- p(X), q(X,b), I = 3.
?- p(X), q(X,Y), t(X), I = 4.
?- p(X), q(X,Y), t(X), r(Y,1), I = 5.
```

The task is to find out for which queries succeed for which X. We will use the variable I as a query identifier; that is, a solution $X = a$, $I = 3$ implies that query 3 succeeds for $X = a$. The set can be structured into a pack as follows (the `or` *operator is similar to the Prolog disjunction but has slightly different operational semantics, see Blockeel et al. [15]):*

[8]Such a couple represents medication $m that often has some side effect $s.

```
?- p(X), (I=1 or q(X,a), I=2 or q(X,b), I=3 or q(X,Y),
          t(X), (I=4 or r(Y,1), I=5))
```

When running the pack, the common parts of the queries are executed less frequently than when running all queries consecutively. For instance, finding all instantiations of X for which $p(X)$ holds, is done only once in the pack, but five times if all queries are executed independently.

An interesting open problem is how the use of query packs can be combined with individual query optimization techniques, such as the ones mentioned in Section 4.2. The combination is non-trivial because restructuring individual queries may destroy the structure of the pack as a whole.

6. MEMORY-WISE SCALABILITY

As long as external storage devices have more storage capacity than internal memory, it will remain useful to devise algorithms that can handle data that are stored on disk. As external memory access is relatively slow, the amount of such access should be kept low, and this may require changes to induction algorithms. A number of techniques follow this approach. Alternatively, memory-wise scalability can be improved by storing data in internal memory as efficiently as possible.

6.1 Processing Data on Disk

Blockeel et al. [14] describe a version of the first order decision tree induction algorithm Tilde that processes an ILP knowledge base without loading it entirely into main memory. The approach is based on the level-wise tree building approach proposed by Mehta et al. [55] for propositional trees. It loads the database one example at a time and needs a single scan of the full database for a single level of the decision tree. To achieve this, two loops in the standard decision tree induction algorithm are switched. The standard description of decision tree induction involves computation of the quality of all possible splits for a certain data set; this involves iterating over all tests and evaluating their quality. The latter involves computing the split the test generates by evaluting the test on each relevant example in the database, which is done with a second loop. By making the example loop the outer loop, the quality of all tests has to be computed incrementally, which increases administrative work, but ensures that for a given level of the tree only one scan through the database is needed.

This approach was proposed for the learning from interpretations setting, and indeed some notion of locality of relevant information is crucial for it to work. There is an assumption that all information about a single example can be localized and it is possible to load just this information into main memory. In the learning from interpretations setting this information consists of the interpretation describing the example together with the background knowledge.

Note that the "switching the loops" principle is to some extent exploited implicitly in the query packs approach (running a query pack on an example is equivalent to running all queries contained in it on that example). Consequently, the latter can easily be adopted when processing data on disk.

6.2 Compact Representations of Data

A running thread through this and the previous section is the trade-off that has to be made between storing information explicitly and computing it on demand. In certain application domains, explicit storage of all information creates a lot of redundancy as far as storage is concerned, while on-demand computation creates redundancy with respect to computations and hence makes the process computationally more expensive.

Consider for instance knowledge bases that describe game sessions of a board game such as Go. A game session is a list of consecutive board states and the move made in that state. In principle, each board state is determined by the previous state and the move taken, so storing just a list of moves is sufficient to describe the whole game. In practice, the data mining system needs explicit representations of the states, and computation of the next state involves quite a bit of work (in Go, adding a stone may result in a set of stones being removed from the board and computing this set is non-trivial). The question is then whether it is possible to first compute explicit representations of all states and then store them in a more compact way but without making recomputation of all information necessary, e.g., by making the representations share certain data structures. For instance, instead of storing the move, one could store the set of stones that are added and removed during a game turn. This yields a representation that is much more compact than storing full board positions and still avoids most of the computational effort that is needed to compute the following state from the current one.

A general solution to this problem is proposed by Struyf et al. [79]. Their approach is set in the learning from interpretations setting. They introduce two operators for defining an interpretation in terms of other interpretations; one (Diff) defines the difference between the interpretation and a previous one, the second (Comb) combines two or more interpretations into a new one. Examples can be described either explicitly, by listing the interpretation, or implicitly, by referring to other examples or structures and using the above operators. As materializing an interpretation may require to materialize other interpretations first, the question arises how the database should be navigated in order to minimize these computations. Struyf et al. use a graph representation for the knowledge base and follow a planning approach to navigate through it.

7. OTHER APPROACHES

The suitability of ILP to mine relational databases has been recognized early on in the history of ILP, and some research in ILP has explicitly focused on the relational database viewpoint [86; 12; 42]. As ILP uses a logical representation, which is different from but largely equivalent to a relational database representation, a natural question is how ILP systems could be adapted to work directly with data that are stored in a relational database. Some research effort was spent on this question. It was also hoped that if such a link could be made, the ILP system would profit from the query optimizers and efficient query execution procedures that characterize RDBMSs.

Blockeel and De Raedt [12] describe a number of different levels at which ILP algorithms can be coupled with RDB systems, ranging from loose to tight integration. Morik and Brockhausen [57] present an implementation called RDT/DB where logical queries are transformed into SQL and these SQL queries are run by a RDBMS. The results of these efforts, in general, were somewhat disappointing. By now it has become clear that the complexity of data mining lies not only in the size of the database but also in the number of queries, and in order to make the mining process as efficient as possible, single query optimization techniques are insufficient. It is essential to exploit the fact that many similar queries are run on the same data, in other words, to optimize sets of queries. Both approaches are not necessarily mutually exclusive: some of the work on single query optimization in databases can probably be combined with the multi-query optimization approach. Exactly how this should be done, is an open problem.

Finally, we repeat that this paper does not aim at an exhaustive survey of all approaches beyond the propositional ones in machine learning and data mining. For instance, we did not detail the Multiple Instance Problem paradigm introduced by Dietterich et al [24; 54] which has been analyzed as intermediate between propositional and fully relational settings [19]. Description logics [16] also constitutes a setting most relevant to data mining in particular relational domains, e.g. XML data, beyond the scope of our paper.

8. SOME CONCRETE EXAMPLES

Several concrete applications confirm that nowadays multi-relational data mining systems can indeed process relatively large databases. We list a few.

Kramer et al. [45] used a multi-relational data mining approach to find structural properties in a set of over 40000 molecules, where each molecule has a relatively complex description. A small percentage of these molecules is known to be active. Kramer et al. look for substructures appearing frequently in the active molecules and infrequently in the inactive ones, using an approach that can be seen as an extension of Apriori that uses both maximum and minimum frequency constraints, and that is tuned for finding chains of specific atoms (e.g., Cl-C:C:C:C-O with '-' denoting a single and ':' an aromatic bond) in first-order descriptions of molecules. Their approach exploits the techniques mentioned here, in that they constrain the search space in a clever way, using the frequency constraints, and that they look for patterns for which the matching procedure is cheap (linear patterns).

In a project on mining UK traffic data, the ILP system ACE-ilProlog [15] was applied to a relational database containing over five million examples, using samples of 100,000 to 1,000,000 instances [47]. First order logic decision trees [13] were built and converted into predictive rules, and trend analysis was performed on frequent relational patterns found by the ACE-ilProlog implementation of the first order association rule algorithm Warmr [23]. ACE-ilProlog incorporates many of the techniques mentioned here (query packs, data sampling, ...).

In a different investigation [79], the same ACE-ilProlog system was applied to a 194MB database of Go games containing 172411 examples of game boards and moves made. These data were analysed in order to find a good heuristic for predicting the quality of a move in certain situations in Go. To handle a data set of this size, the authors applied the compact representation techniques mentioned before. Typical runtimes in this case were under five minutes. The experiment also indicated that the compact representation did not cause any significant loss of computational efficiency.

9. CONCLUSIONS

In recent years, techniques for ILP and multi-relational data mining have undergone a significant evolution in the direction of more scalable and more efficient systems. We have attempted to give an overview of this evolution. In our opinion, major steps towards better performance are the incorporation of stochastic techniques in ILP systems, a shift towards individual-centered methods (which brings this research more in line with other data mining research), and methods for precomputing features and statistics. Especially in the latter area, work seems to have just started, and further advancements can be expected. In addition, an increased understanding of complexity issues in relational data mining (from, e.g., the results on phase transitions) may yield important novel approaches to optimization in the future.

10. ACKNOWLEDGEMENTS

Hendrik Blockeel is a post-doctoral fellow of the Fund for Scientific Research of Flanders, Belgium. The authors thank the reviewers of this article for their useful suggestions, and Jan Struyf for proofreading the final version.

11. REFERENCES

[1] R. Agrawal, H. Mannila, R. Srikant, H. Toivonen, and A.I. Verkamo. Fast discovery of association rules. In U. Fayyad, G. Piatetsky-Shapiro, P. Smyth, and R. Uthurusamy, editors, *Advances in Knowledge Discovery and Data Mining*, pages 307–328. The MIT Press, 1996.

[2] C. Anglano, A. Giordana, and G. Lo Bello. High-performance data mining on networks of workstations. In *10th International Symposium on Methodologies for Intelligent Systems, ISMIS'99*, pages 520–528. Springer Verlag, 1999.

[3] C. Anglano, A. Giordana, G. Lo Bello, and L. Saitta. An experimental evaluation of coevolutive concept learning. In J. Shavlik, editor, *Proceedings of the 15th International Conference on Machine Learning*, pages 19–27. Morgan Kaufmann, 1998.

[4] S. Anthony and A. M. Frisch. Cautious induction in inductive logic programming. In N. Lavrač and S. Džeroski, editors, *Proceedings of ILP-97*, pages 45–60. Springer Verlag, LNCS 1297, 1997.

[5] A. Appice, M. Ceci, and D. Malerba. Mining model trees: a multi-relational approach. In *Proceedings of the 13th International Conference on Inductive Logic Programming*. Springer-Verlag, 2003. To appear.

[6] A. Atramentov, H. Leiva, and V. Honavar. A multi-relational decision tree learning algorithm: Implementation and experiments. In *Proceedings of the 13th International Conference on Inductive Logic Programming*. Springer-Verlag, 2003. To appear.

[7] T. Bäck. *Evolutionary Algorithms in theory and practice*. New-York:Oxford University Press, 1995.

[8] W. Banzhaf, P. Nordin, R.E. Keller, and F.D. Francone. *Genetic Programming — An Introduction On the Automatic Evolution of Computer Programs and Its Applications*. Morgan Kaufmann, 1998.

[9] F. Bergadano, R. Gemello, A. Giordana, and L. Saitta. ML-SMART: a Problem Solver for Learning from Examples. *Fundamenta Informaticae*, XII:29–50, 1989.

[10] J. Ales Bianchetti, C. Rouveirol, and M. Sebag. Constraint-based learning of long relational concepts. In C. Sammut, editor, *Proceedings of the 19th International Conference on Machine Learning*, pages 35–42. Morgan Kaufmann, 2002.

[11] H. Blockeel. *Top-down induction of first order logical decision trees*. PhD thesis, Department of Computer Science, Katholieke Universiteit Leuven, 1998. http://www.cs.kuleuven.ac.be/~ml/PS/blockeel98:phd.ps.gz.

[12] H. Blockeel and L. De Raedt. Relational knowledge discovery in databases. In *Proceedings of the Sixth International Workshop on Inductive Logic Programming*, volume 1314 of *Lecture Notes in Artificial Intelligence*, pages 199–212. Springer-Verlag, 1996.

[13] H. Blockeel and L. De Raedt. Top-down induction of first order logical decision trees. *Artificial Intelligence*, 101(1-2):285–297, June 1998.

[14] H. Blockeel, L. De Raedt, N. Jacobs, and B. Demoen. Scaling up inductive logic programming by learning from interpretations. *Data Mining and Knowledge Discovery*, 3(1):59–93, 1999.

[15] H. Blockeel, L. Dehaspe, B. Demoen, G. Janssens, J. Ramon, and H. Vandecasteele. Improving the efficiency of inductive logic programming through the use of query packs. *Journal of Artificial Intelligence Research*, 16:135–166, 2002.

[16] A. Borgida. On the relative expressiveness of description logics and predicate logics. *Artificial Intelligence*, 1996.

[17] P. Cheeseman, B. Kanefsky, and W.M. Taylor. Where the really hard problems are. In *Proc. of IJCAI'91*, pages 331–337. Morgan Kaufmann, 1991.

[18] W. Chen and D. S. Warren. Tabled evaluation with delaying for general logic programs. *Journal of the ACM*, 43(1):20–74, January 1996. http://www.cs.sunysb.edu/~sbprolog.

[19] L. De Raedt. Logical settings for concept learning. *Artificial Intelligence*, 95:187–201, 1997.

[20] L. De Raedt, H. Blockeel, L. Dehaspe, and W. Van Laer. Three companions for data mining in first order logic. In S. Džeroski and N. Lavrač, editors, *Relational Data Mining*, pages 105–139. Springer-Verlag, 2001.

[21] L. De Raedt and S. Džeroski. First order jk-clausal theories are PAC-learnable. *Artificial Intelligence*, 70:375–392, 1994.

[22] L. Dehaspe and L. De Raedt. DLAB: A declarative language bias formalism. In *Proceedings of the International Symposium on Methodologies for Intelligent Systems*, volume 1079 of *Lecture Notes in Artificial Intelligence*, pages 613–622. Springer-Verlag, 1996.

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

[24] T. G. Dietterich, R. H. Lathrop, and T. Lozano-Pérez. Solving the multiple-instance problem with axis-parallel rectangles. *Artificial Intelligence*, 89(1-2):31–71, 1997.

[25] S. Džeroski and N. Lavrač, editors. *Relational Data Mining*. Springer-Verlag, 2001.

[26] P.A. Flach and N. Lachiche. 1BC: a first order Bayesian classifier. In D. Page, editor, *Proceedings of the Ninth International Workshop on Inductive Logic Programming*, volume 1634, pages 93–103. Springer-Verlag, 1999.

[27] L. Getoor, D. Koller, and B. Taskar. Statistical models for relational data. In *Proc. of the KDD-02 Workshop on Multi-Relational Data Mining*, 2002.

[28] A. Giordana and L. Saitta. REGAL: An integrated system for learning relations using genetic algorithms. In *Proceedings of the 2nd International Workshop on Multistrategy Learning*, pages 234–249, 1993.

[29] A. Giordana and L. Saitta. Phase transitions in relational learning. *Machine Learning Journal*, 41:217–251, 2000.

[30] A. Giordana, L. Saitta, M. Sebag, and M. Botta. Analyzing relational learning in the phase transition framework. In P. Langley, editor, *Proceedings of the 17^{th} International Conference on Machine Learning*, pages 311–318. Morgan Kaufmann, 2000.

[31] D. E. Goldberg. *Genetic algorithms in search, optimization and machine learning*. Addison Wesley, 1989.

[32] G. Gottlob, N. Leone, and F. Scarcello. The complexity of acyclic conjunctive queries. In *39th Annual Symposium on Foundations of Computer Science, FOCS '98, November 8-11, 1998, Palo Alto, California, USA.*, pages 706–715. IEEE Computer Society, 1998.

[33] G. Gottlob, N. Leone, and F. Scarcello. On tractable queries and constraints. In *Proceedings of the 10th International Conference on Database and Expert Systems Applications.*, pages 1–15, 1999.

[34] J. Hekanaho. Dogma: A ga-based relational learner. In *Proceedings of the Eighth International Conference on Inductive Logic Programming*, pages 205–214, 1998.

[35] T. Hogg. Refining the phase transition in combinatorial search. *Artificial Intelligence*, 81(1–2):127–154, 1996.

[36] T. Hogg, B.A. Huberman, and C.P. Williams, editors. *Artificial Intelligence: Special Issue on Frontiers in Problem Solving: Phase Transitions and Complexity*, volume 81(1-2). Elsevier, 1996.

[37] T. Horvath and S. Wrobel. Towards discovery of deep and wide first-order structures: A case study in the domain of mutagenicity. In *Discovery Science 2001*, volume 2226 of *Lecture Notes in Artificial Intelligence*, pages 100–112, 2001.

[38] M. Jarke and J. Koch. Query optimization in database systems. *ACM Computing Surveys*, 16(2), 1984.

[39] C. J. Kennedy and C. Giraud-Carrier. A depth controlling strategy for strongly typed evolutionary programming. In *GECCO 1999: Proceedings of the First Annual Conference*, pages 1–6. Morgan Kaufmann, 1999.

[40] J-U. Kietz and S. Wrobel. Controlling the complexity of learning in logic through syntactic and task-oriented models. In S. Muggleton, editor, *Inductive Logic Programming*, pages 335–359. Academic Press, 1992.

[41] J.U. Kietz and M. Lübbe. An efficient subsumption algorithm for inductive logic programming. In *Proceedings of the 11th International Conference on Machine Learning*, pages 130–138. Morgan Kaufmann, 1994.

[42] A.J. Knobbe, A. Siebes, H. Blockeel, and D. van der Wallen. Multi-relational data mining, using UML for ILP. In *Proceedings of PKDD-2000 - The Fourth European Conference on Principles and Practice of Knowledge Discovery in Databases*, volume 1910 of *Lecture Notes in Artificial Intelligence*, pages 1–12, Lyon, France, 2000. Springer.

[43] J. R. Koza. *Genetic Programming: On the Programming of Computers by means of Natural Evolution*. MIT Press, Massachusetts, 1992.

[44] J. R. Koza. *Genetic Programming II: Automatic Discovery of Reusable Programs*. MIT Press, Massachussetts, 1994.

[45] S. Kramer, L. De Raedt, and C. Helma. Molecular feature mining in HIV data. In *Proceedings of the Seventh ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD-01)*, pages 136–143. ACM Press, 2001.

[46] S. Kramer, N. Lavrač, and P. Flach. Propositionalization approaches to relational data mining. In S. Džeroski and N. Lavrač, editors, *Relational Data Mining*, pages 262–291. Springer-Verlag, 2001.

[47] D. Krzywania, J. Struyf, and H. Blockeel. Mining the UK traffic database. Internal SolEUNet report, 2002.

[48] N. Lachiche and P. Flach. A first-order representation for knowledge discovery and bayesian classification on relational data. In *PKDD2000 workshop on Data Mining, Decision Support, Meta-learning and ILP : Forum for Practical Problem Presentation and Prospective Solutions*, pages 49–60, 2000.

[49] J. W. Lloyd. Declarative programming in Escher. Technical Report CSTR-95-013, University of Bristol, June, 1995.

[50] J.W. Lloyd. *Foundations of Logic Programming*. Springer-Verlag, 2nd edition, 1987.

[51] J. Maloberti and M. Sebag. Theta-subsumption in a constraint satisfaction perspective. In C. Rouveirol and M. Sebag, editors, *Proceedings of Inductive Logic Programming*, LNAI 2157, pages 164–178. Springer Verlag, 2001.

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

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

[54] O. Maron and T. Lozano-Perez. A framework for multiple-instance learning. In *Advances in Neural Information Processing Systems*, volume 10, pages 570–576. Mit Press, 1998.

[55] M. Mehta, R. Agrawal, and J. Rissanen. SLIQ: A fast scalable classifier for data mining. In *Proceedings of the Fifth International Conference on Extending Database Technology*, volume 1057 of *Lecture Notes in Computer Science*. Springer-Verlag, 1996.

[56] A. W. Moore and M. S. Lee. Cached sufficient statistics for efficient machine learning with large datasets. *Journal of Artificial Intelligence Research*, 8:67–91, 1998.

[57] K. Morik and P. Brockhausen. A multistrategy approach to relational discovery in databases. *Machine Learning*, 27(3):287–312, 1997.

[58] S. Muggleton. Inductive logic programming. In S. Muggelton, editor, *Inductive Logic Programming.* Academic Press, 1992.

[59] S. Muggleton. Inverse entailment and Progol. *New Generation Computing, Special issue on Inductive Logic Programming*, 13(3-4):245–286, 1995.

[60] S. Muggleton and L. De Raedt. Inductive logic programming: Theory and methods. *Journal of Logic Programming*, 19:629–679, 1994.

[61] C. Nédellec, H. Adé, F. Bergadano, and B. Tausend. Declarative bias in ILP. In L. De Raedt, editor, *Advances in Inductive Logic Programming*, volume 32 of *Frontiers in Artificial Intelligence and Applications*, pages 82–103. IOS Press, 1996.

[62] S.-H. Nienhuys-Cheng and R. De Wolf. *Foundations of Inductive Logic Programming*, volume 1228 of *Lecture Notes in Computer Science and Lecture Notes in Artificial Intelligence*. Springer-Verlag, New York, NY, USA, 1997.

[63] T. Oates and D. Jensen. Large datasets lead to overly complex models: An explanation and a solution. In *Knowledge Discovery and Data Mining*, pages 294–298, 1998.

[64] D. Pavlov, H. Mannila, and P. Smyth. Beyond independence: Probabilistic models for query approximation on binary transaction data. *IEEE Transactions on Knowledge and Data Engineering*, 2003. To appear.

[65] G. Plotkin. A note on inductive generalization. In B. Meltzer and D. Michie, editors, *Machine Intelligence*, volume 5, pages 153–163. Edinburgh University Press, 1970.

[66] J.R. Quinlan. Learning logical definition from relations. *Machine Learning*, 5:239–266, 1990.

[67] J.R. Quinlan. FOIL: A midterm report. In P. Brazdil, editor, *Proceedings of the 6th European Conference on Machine Learning*, Lecture Notes in Artificial Intelligence. Springer-Verlag, 1993.

[68] A. Ratle and M. Sebag. Genetic programming with domain knowledge for machine discovery. In *Proceedings of the 12th International Conference on Inductive Logic Programming.* Springer-Verlag, 2002.

[69] J.A. Robinson. A machine-oriented logic based on the resolution principle. *Journal of the ACM*, 12:23–41, 1965.

[70] 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. In press.

[71] T. Scheffer, R. Herbrich, and F. Wysotzki. Efficient theta-subsumption based on graph algorithms. In *Inductive Logic Programming, 6th International Workshop, Proceedings*, volume 1314 of *Lecture Notes in Artificial Intelligence*, pages 212–228, 1996.

[72] T. Scheffer and S. Wrobel. Finding the most interesting patterns in a database quickly by using sequential sampling. *Journal of Artificial Intelligence Research*, 3:833–862, 2002.

[73] M. Sebag and C. Rouveirol. Tractable Induction and Classification in First-Order Logic via Stochastic Matching. In *Proceedings of the 15th International Joint Conference on Artificial Intelligence*, pages 888–893. Morgan Kaufmann, 1997.

[74] M. Sebag and C. Rouveirol. Any-time relational reasoning: Resource-bounded induction and deduction through stochastic matching. *Machine Learning*, 38(1–2):41–62, 2000.

[75] A. Serra, A. Giordana, and L. Saitta. Learning on the phase transition edge. In *Proc. of IJCAI 2001*, pages 921–926. Morgan Kaufmann, 2001.

[76] A. Srinivasan. A study of two sampling methods for analysing large datasets with ILP. *Data Mining and Knowledge Discovery*, 3(1):95–123, 1999.

[77] A. Srinivasan. A study of two probabilistic methods for searching large spaces with ILP. Technical Report PRG-TR-16-00, Oxford University Computing Laboratory, 2000.

[78] J. Struyf and H. Blockeel. Query optimization in inductive logic programming by reordering literals. In *Proceedings of the 13th International Conference on Inductive Logic Programming*, Lecture Notes in Artificial Intelligence. Springer-Verlag, 2003. To appear.

[79] J. Struyf, J. Ramon, and H. Blockeel. Compact representation of knowledge bases in ILP. In *Proceedings of the 12th International Conference on Inductive Logic Programming*, volume 2583 of *Lecture Notes in Artificial Intelligence*, pages 254–269. Springer-Verlag, 2002.

[80] F. Torre and C. Rouveirol. Natural ideal operators in inductive logic programming. In *Proceedings of the 9th European Conference on Machine Learning*, pages 274–289, 1997.

[81] E. Tsang. *Foundations of Constraint Satisfaction*. Academic Press, 1993.

[82] D. Tsur, J.D. Ullman, S. Abiteboul, C. Clifton, R. Motwani, S. Nestorov, and A. Rosenthal. Query flocks: A generalization of association-rule mining. In *Proceedings of the ACM SIGMOD International Conference on Management of Data (SIGMOD-98)*, volume 27,2 of *ACM SIGMOD Record*, pages 1–12, New York, June 1–4 1998. ACM Press.

[83] H. Vandecasteele, B. Demoen, and G. Janssens. Compiling large disjunctions. In *First International Conference on Computational Logic : Workshop on Parallelism and Implementation Technology for (Constraint) Logic Programming Languages*, 2000. Also available as Technical Report CW 295, `http://www.cs.kuleuven.ac.be/publicaties/rapporten/cw/CW295.ps.gz`.

[84] M.L. Wong and K.S. Leung. Combining genetic programming and inductive logic programming using logic grammars. In D. B. Fogel, editor, *Proceedings of the Second IEEE International Conference on Evolutionary Computation*, pages 733–736. IEEE Press, 1995.

[85] S. Wrobel. An algorithm for multi-relational discovery of subgroups. In J. Komorowski and J. Zytkow, editors, *Proceedings of the First European Symposium on Principles of Data Mining and Knowledge Discovery (PKDD '97)*, pages 78 – 87. Springer-Verlag, 1997.

[86] S. Wrobel, D. Wettschereck, E. Sommer, and W. Emde. Extensibility in data mining systems. In *Proceedings of the Second International Conference on Knowledge Discovery and Data Mining (KDD-96)*. AAAI Press, 1996.