Machine Learning, 42, 61–95, 2001

Confirmation-Guided Discovery of First-Order Rules with `Tertius`

PETER A. FLACH flach@cs.bris.ac.uk
NICOLAS LACHICHE lachiche@cs.bris.ac.uk
Department of Computer Science, University of Bristol, United Kingdom

Editor: Douglas H. Fisher

Abstract. This paper deals with learning first-order logic rules from data lacking an explicit classification predicate. Consequently, the learned rules are not restricted to predicate definitions as in supervised inductive logic programming. First-order logic offers the ability to deal with structured, multi-relational knowledge. Possible applications include first-order knowledge discovery, induction of integrity constraints in databases, multiple predicate learning, and learning mixed theories of predicate definitions and integrity constraints. One of the contributions of our work is a heuristic measure of confirmation, trading off novelty and satisfaction of the rule. The approach has been implemented in the `Tertius` system. The system performs an optimal best-first search, finding the k most confirmed hypotheses, and includes a non-redundant refinement operator to avoid duplicates in the search. `Tertius` can be adapted to many different domains by tuning its parameters, and it can deal either with individual-based representations by upgrading propositional representations to first-order, or with general logical rules. We describe a number of experiments demonstrating the feasibility and flexibility of our approach.

Keywords: first-order inductive learning, inductive logic programming, knowledge discovery

1. Introduction

This paper deals with unsupervised discovery of rules in first-order logic. We define a statistically well-founded confirmation measure to induce rules that are in some sense unusual or interesting. We then describe a complete best-first search algorithm that uses an optimistic estimate of the best confirmation of possible refinements of a rule to prune the search, and an non-redundant refinement operator that is guaranteed to generate each rule not more than once during search. The algorithm works on a function-free first order Prolog representation, which enables application to a wide range of structured domains, and to include background knowledge as part of the heuristic evaluation.

1.1. First-order unsupervised learning

In our perspective, there are four main paradigms in rule learning (Table 1). Along one dimension, rule learning can be either supervised or unsupervised. Supervised learning includes concept learning, classification, and inductive logic programming (ILP). It is distinguished by the use of a single predicate that appears in the head of rules. Unsupervised learning does not have such a single focus of inference. This may be either because there

Table 1. The four paradigms in rule learning. Each cell lists a typical task addressed in that paradigm.

	Supervised	*Unsupervised*
Individual-based	Concept learning	Rule discovery
Other	Program synthesis	Multiple predicate learning

are several, inter-dependent predicates to be learned, as in multiple predicate learning, or because the main direction of inference is not known a priori, as in association rule learning or database dependency discovery. A second, major dimension distinguishes between representations that are essentially individual-based, and other representations. Propositional representations are inherently individual-based, although there is usually no explicit representation of individuals in either examples or rules due to the single representation trick. It is simply assumed that each example concerns a single individual, and that rules generalise over all individuals. First-order representations offer significantly more flexibility in representing a domain. While initial work in ILP tended to focus on tasks like Prolog program synthesis, which is less obviously individual-based, many recent ILP systems adopt a more structured representation based on an underlying notion of individual. The `Tertius` system described in this paper was designed to handle unsupervised learning in first-order representations of both kinds.

1.2. The learning task

In general, inductive learning is concerned with finding plausible generalisations (hypotheses) from the evidence. This is often formalised by assuming a particular syntactic form of evidence and hypotheses, such as pre-classified instances or non-instances of a concept, and classification rules defining that concept. In this paper we make no such assumptions, and therefore resort to the more abstract framework of confirmatory induction.

Definition 1 (Confirmatory induction). Given a first-order language L, a *confirmation relation* is a binary relation $\mid\!\sim \subseteq 2^L \times L$; if $E \mid\!\sim H$ we say that evidence E *confirms* hypothesis H. A confirmation function is a partial function $c : 2^L \times L \to [0, 1]$; we say that evidence E confirms hypothesis H *to degree* $c(E, H)$ if $c(E, H)$ is defined. A confirmation relation is *categorical* if evidence and all confirmed hypotheses are consistent with each other; a confirmation function is categorical if it is defined only for such evidence and hypothesis.

A simple example of a categorical confirmation relation can be defined when the evidence E is given as a set of ground (i.e., variable-free) facts, from which we can construct a model $m(E)$ (i.e., a truthvalue assignment). In this case we can define $E \mid\!\sim H$ iff H is true in $m(E)$; it follows that $m(E)$ is a model of the set of confirmed hypotheses. In order to ensure generalisation, in categorical confirmatory induction we are typically interested in some intensional representation of the set of all confirmed hypotheses, i.e., a compact axiomatisation. This is roughly the approach followed in the `Claudien` system (De Raedt & Dehaspe, 1997).

An alternative way of defining a confirmation relation is on top of a confirmation function, e.g., $E \mid\!\sim H$ iff $c(E, H) \geq c_0$ for a fixed threshold c_0. In general the resulting confirmation relation will be non-categorical, i.e., some hypotheses confirmed by the same evidence may be mutually inconsistent. In such a case the confirmation relation does not contain sufficient information to single out a consistent set of hypotheses. On the other hand, the use of a confirmation function has the advantage of being able to rank the hypotheses.

The system Tertius presented in this paper was designed to address both categorical and non-categorical confirmatory induction tasks. The system searches for the k most confirmed hypotheses; if a consistent set of hypotheses is requested, the additional requirement of the evidence forming a model of the hypothesis is enforced. Our research goals were, first, to devise a suitable confirmation function that is theoretically and empirically well-founded; and secondly, to implement a first-order knowledge discovery system that employs this confirmation function. A major consideration when implementing the system was efficiency: specifically, much attention was paid to employing the confirmation heuristic for pruning the search, and to efficiently traversing the search space.

1.3. Outline of the paper

The outline of the paper is as follows. In Section 2 we present the preliminary material necessary to understand the rest of the paper. Section 3 defines the confirmation measure used throughout the paper. It is a modified X^2 statistic defined on a sample of grounding substitutions for the rule under consideration. We present a thorough analysis of this measure, including an optimistic estimate for use in our A* algorithm, and a discussion of how to adapt the measure in order to incorporate logical background knowledge. In Section 4 we present the Tertius system and give implementation details, including our approach to avoid generating redundant hypotheses during search. The system has many options, allowing the user to customise it to the domain. These options include the use of individual-based and structural representations, and the use of a frequency threshold as in association rule learning. Section 5 describes our experience with applying Tertius to a range of domains. The tasks we consider are classification, learning in problem solving, database dependency discovery, and multiple predicate learning. In Section 6 we discuss related work, and Section 7 concludes.

2. Preliminaries

This section explicates the logical and statistical background and notation employed in the paper.

2.1. Logical setting

Throughout the paper, the hypothesis language is a function-free first-order language. The *signature* of such a language is a set of domain-specific predicates and constants. A *literal* consists of a predicate applied to variables and/or constants. A *ground literal* does

not contain any variables. Literals can be combined into formulae by means of the usual logical connectives (negation $\neg$, conjunction $\wedge$, disjunction $\vee$, implication $\rightarrow$ or $\leftarrow$, and equivalence $\leftrightarrow$). A variable in a formula can be *bound* by a universal quantifier $\forall$ or an existential quantifier $\exists$. A *free variable* is a variable that is not bound by a quantifier. A *closed formula* is a formula without free variables. If F is a formula, $\forall(F)$ denotes the *universal closure* of F, i.e. a closed formula in which all free variables in F are bound by universal variables.

Although our approach is generally applicable to first-order languages, the `Tertius` system employs a normal form variant called *clausal logic*. A *clause* is a disjunction of possibly negated literals, e.g., $H_1 \vee H_2 \vee \neg B_1 \vee \neg B_2$. Such a rule is usually written as an implication $H_1 \vee H_2 \leftarrow B_1 \wedge B_2$, or `H1;H2:-B1,B2` in Prolog notation. All variables in a clause are universally quantified (the quantifiers are conventionally left implicit). In Section 3 we mostly employ a general first-order language, whereas in Sections 4 and 5 we use clausal logic.

We assume the standard Tarskian semantics for first-order logic, and write $F \models G$ if all models of F are also models of G, where a model is an interpretation in which the formula is true. We say that F *entails* G, or that G is a *specialisation* of F. In the case of clausal logic, specialisations G of a clause F can be obtained by a syntactic process called *refinement*. Briefly, there are three types of refinement operations: adding a literal to a clause, unifying two variables, or instantiating a variable with a term (i.e., a constant in function-free logic). The refinement operator employed by `Tertius` will be discussed in Section 4.2.

The evidence that is input to `Tertius` consists essentially of ground literals, possibly enhanced with intensional background knowledge. However, in the theoretical setting of Section 3 we will make minimal assumptions about the logical form of the evidence, only assuming availability of the following procedures:

GS: a procedure returning a finite set of grounding substitutions for the free variables in a given formula;
TV: a procedure assigning truthvalues to closed formulae according to the evidence.

These procedures allow us to leave the form in which the evidence is given unspecified. For instance, if the evidence is partitioned as in learning from interpretations (De Raedt, 1997) **GS** will not return grounding substitutions that mix terms from different interpretations. Since the procedures **GS** and **TV** provide the interface to a given body of evidence, we will normally not explicitly refer to the evidence in our formalisation, and simply say 'the degree of confirmation of hypothesis H is $c(H)$'.

2.2. Contingency tables

Our main statistical tool in this paper will be the contingency table, which is an important tool in the analysis of categorical data (Wickens, 1989). Suppose we have a population of unspecified entities, and two attributes X and Y that can each take on two possible values for each entity. We want to know whether these two attributes are dependent with respect to the population. To this end, we draw a sample from the population, and record the

Table 2. A contingency table. n_{ij} denotes observed frequency of a joint event, and n_{i*} and n_{*j} denote marginal frequencies.

	Y = y1	Y = y2	*total*
X=x1	n_{11}	n_{12}	n_{1*}
X=x2	n_{21}	n_{22}	n_{2*}
total	n_{*1}	n_{*2}	N

frequencies of possible outcomes in a two by two table, which is called a *contingency table* (Table 2).

Suppose the two attributes are independent (the *null hypothesis*). In that case, the expected frequency for e.g. n_{11} would be $\mu_{11} = n_{*1}n_{1*}/N$, and similarly for the other cells. We would expect the observed frequencies n_{ij} to be close to these μ_{ij}. To check this, we compare the two sets of numbers by calculating the Pearson statistic X^2:

$$X^2 = \sum_{ij} \frac{(n_{ij} - \mu_{ij})^2}{\mu_{ij}} = \sum_{ij} \frac{n_{ij}^2}{\mu_{ij}} - N \tag{1}$$

If one of the μ_{ij}'s is 0, we set $X^2 = 0$ (see below).

Large values of X^2 indicate big discrepancies between observed and expected values, while small values indicate a close fit. Because we observe only part of the population an exact fit is unlikely, but we can assess the value of X^2 by referring it to the sampling distribution χ^2 with one degree of freedom (an n by m table has $(n-1)(m-1)$ degrees of freedom). For instance, the critical χ^2 value at the 5% level is 3.84 — if our X^2 value is larger than that, the probability is less than 5% that discrepancies this large are attributable to chance, and we are led to reject the null hypothesis of independence.

For a two by two table we can rewrite Eq. (1) as follows:

$$X^2 = \frac{(n_{11}n_{22} - n_{12}n_{21})^2}{n_{1*}n_{2*}n_{*1}n_{*2}} N \tag{2}$$

From this equation we see that X^2 is proportional to the sample size N, which is typical for a test statistic (with increasing sample size we can be more certain of our decision to accept or reject the null hypothesis). Dividing X^2 by N we obtain a measure of the strength of the dependency between X and Y:

$$\Phi^2 = X^2/N = \sum_{ij} \frac{(n_{ij} - \mu_{ij})^2}{N\mu_{ij}} = \frac{(n_{11}n_{22} - n_{12}n_{21})^2}{n_{1*}n_{2*}n_{*1}n_{*2}} \tag{3}$$

Φ^2 ranges from 0 (total independence) to 1 (total dependence) for a two by two table. Total independence occurs iff the products of the observed frequencies on the diagonals are identical (a special case is when there are 2 zeroes in the same row or column, i.e.,

when one of the marginals is zero). Total dependence is obtained iff a table has 2 zeroes in different rows and columns. A single zero in the table typically leads to a relatively high Φ^2, especially when it is accompanied by a low value in the diagonally opposite cell.

Φ^2 assesses the strength of the dependency between two variables, based on a single sample of events. This can be generalised to several samples and/or more than two variables. There are various ways of combining information from several contingency tables into a single Φ^2. In the simplest case one combines the tables into a single contingency table by summing the corresponding cells, and then calculates Φ^2 of the combined table. Alternatively, one can calculate Φ^2 for each table separately and average those. This is preferable if each sample can be meaningfully interpreted as a separate experiment, otherwise the first approach is better. `Tertius` can use partitioned data, and uses the first summative method of combination unless explicitly instructed by the user to use the averaging method.

A contingency table with more than two variables is called a *multiway* table. For instance, in a three-way table there are eight observed frequencies n_{ijk}. Furthermore, there are three *one-way* marginals n_{i**}, n_{*j*} and n_{**k}; and three *two-way* marginals n_{ij*}, n_{i*k} and n_{*jk}. In general an m-way contingency table has $m-1$ different kinds of marginals. The expected frequencies can sum up to all of these marginals, or only to a subset, as determined by the null hypothesis. For instance, under the null hypothesis of *complete independence* of the set of variables, the expected frequencies are products of one-way marginals, as before. In this case, they sum up only to the one-way marginals, not necessarily to the others. Multiway contingency tables do not play a major role in this paper, except for indicating how logical background knowledge can be incorporated into the confirmation measure.

3. Confirmation

This section presents the main theoretical contribution of the paper, which is a confirmation measure for first-order formulae. In Section 3.1 we derive and justify this confirmation measure for the special case of universally quantified implications. In Section 3.2 we obtain an upper bound on the degree of confirmation that can be obtained by specialising a formula, and prove its correctness for a specific kind of specialisations called *admissible* specialisations. This upper bound is exploited in `Tertius`' A* search algorithm to prune without losing completeness. Finally, Section 3.3 discusses an extension of the confirmation function which takes logical background knowledge into account.

3.1. Confirmation of rules

In this section we only consider rules of the form $\forall(H \leftarrow B)$, where H and B are arbitrary formulae with some free variables, and $\forall(\cdot)$ denotes universal closure over the free variables. H is called the *head* of the rule, and B is called its *body*. We do not place further restrictions on the syntactic form of H and B, although frequently the rule is a clause, i.e., H is a disjunction and B is a conjunction of atoms.

The idea is to measure how strongly the rule is confirmed by the evidence by determining the number of (counter-)instances of H and B and the number of (counter-)instances of $\forall(H \leftarrow B)$. We consider a sample of N grounding substitutions θ of $H \leftarrow B$, returned

Table 3. The contingency table used to evaluate a rule $\forall(H \leftarrow B)$.

	B	$\bar{B}$	
H	n_{HB} (μ_{HB})	$n_{H\bar{B}}$ $(\mu_{H\bar{B}})$	n_H
$\bar{H}$	$n_{\bar{H}B}$ $(\mu_{\bar{H}B})$	$n_{\overline{HB}}$ $(\mu_{\overline{HB}})$	$n_{\bar{H}}$
	n_B	$n_{\bar{B}}$	N

by **GS**, and tabulate the corresponding truthvalues of $H\theta$ and $B\theta$ as returned by **TV** in a contingency table (Table 3). For instance, n_{HB} denotes the number of grounding substitutions that satisfy both head and body of the rule; similarly, $n_{\bar{H}B}$ denotes the number of substitutions that satisfy the body but falsify the head, also referred to as *counterinstances* of the rule $\forall(H \leftarrow B)$.

A second set of frequencies μ_{ij} is obtained from the row and column marginals under some null hypothesis. They are called *expected frequencies*—in particular, $\mu_{\bar{H}B}$ is the expected frequency of counterinstances of the rule $\forall(H \leftarrow B)$. Notice that expected frequencies add up to the same marginal frequencies as the observed frequencies—they represent a different distribution of mass over the four cells of the contingency table. The more the observed frequencies n_{ij} differ from the expected frequencies μ_{ij}, the more information is contained in the observed frequencies that cannot be predicted from the marginal frequencies. It will often be convenient to switch to relative frequencies p_{ij} (observed) and π_{ij} (expected).[1]

In the simplest case, expected frequencies are given by the product of the corresponding row and column marginals divided by N.

Definition 2 (Simple expected frequency). $\mu_{ij} = \frac{n_{i*}n_{*j}}{N}$.

For instance, $\mu_{\bar{H}B} = \frac{n_{\bar{H}}n_B}{N}$. Most of what follows holds for arbitrary null hypotheses (i.e., arbitrary ways of computing expected frequencies), but we will occasionally resort to Definition 2. An alternative definition of expected frequencies will be considered in Section 3.3.

Our objective is to define a confirmation function for rules of the form $\forall(H \leftarrow B)$ in terms of the frequencies in its associated contingency table. A similar problem is considered by Piatetsky-Shapiro (1991). He considers confirmation functions defined in terms of n_{HB}, n_H, and n_B. Specifically, the n_{HB} substitutions satisfying both H and B are considered the *confirming instances* of $\forall(H \leftarrow B)$. The problem with this approach is that the rules $\forall(H \leftarrow B)$ and $\forall(B \leftarrow H)$ will always have the same number of confirming instances, which is clearly undesirable in learning tasks where H and B can be chosen freely. We solve this problem by focusing instead on the quantity $n_{\bar{H}B}$, which is the number of *counterinstances* of the rule $\forall(H \leftarrow B)$. All grounding substitutions that are not counterinstances are considered to be confirming instances. That is, we do not treat $\forall(H \leftarrow B)$ as a classification rule, which has only n_{HB} confirming instances, but rather as a logical formula.

Piatetsky-Shapiro proposes three principles that should be satisfied by confirmation functions. Rewritten in terms of counterinstances, these principles are as follows:

P1: the confirmation should be 0 if $p_{\bar{H}B} = \pi_{\bar{H}B}$;
P2: the confirmation should monotonically decrease with $p_{\bar{H}B}$, all other parameters remaining the same;
P3: the confirmation should monotonically increase with $p_{\bar{H}}$ (or p_B), all other parameters remaining the same.

As Piatetsky-Shapiro notes, the simplest measure satisfying P1-3 is $\pi_{\bar{H}B} - p_{\bar{H}B}$. It is easy to show that this is equal to $p_{HB} - \pi_{HB}$, and thus symmetric in H and B. On the other hand, this measure has a certain interest: e.g., under simple expected frequency it equals $p(B)[p(H \mid B) - p(H)]$, which has been called *weighted relative accuracy* in a classification context (Lavrač, Flach, & Zupan, 1999). Here, we call it the *novelty* associated with the rule, because it measures the novel information expressed by the rule that cannot be inferred from the evidence and the null hypothesis alone.

Definition 3. The *novelty* of a rule $\forall(H \leftarrow B)$ is defined as $\Delta_{\bar{H}B} = \pi_{\bar{H}B} - p_{\bar{H}B}$.

Note that $-0.25 \leq \Delta_{\bar{H}B} \leq 0.25$.

To exclude measures that are symmetric in H and B, we add the following principle:

P4: two rules $\forall(H \leftarrow B)$ and $\forall(B \leftarrow H)$ should get equal confirmation only if they have the same number of counterinstances.

We thus need a second measure to distinguish between $\forall(H \leftarrow B)$ and $\forall(B \leftarrow H)$, selecting the one that is most satisfied (has the least number of counterinstances).

Definition 4. The *satisfaction* of a rule $\forall(H \leftarrow B)$ is defined as $\sigma_{\bar{H}B} = \frac{\pi_{\bar{H}B} - p_{\bar{H}B}}{\pi_{\bar{H}B}}$.

That is, satisfaction is defined as the fraction of expected counterinstances that are not observed, i.e., the relative difference between expected error and observed error. Satisfaction is a version of rule accuracy $p(H \mid B)$ that takes the whole contingency table into account. To see this, notice that $\sigma_{\bar{H}B}$ can be re-expressed as $\frac{p_{HB} - \pi_{HB}}{p_B - \pi_{HB}}$, which under simple expected frequency is equal to $\frac{p(H \mid B) - p(H)}{1 - p(H)}$. Thus, satisfaction increases linearly with rule accuracy, reaching its maximum of 1 iff $p(H \mid B) = 1$, but becoming negative as soon as the rule accuracy drops under the 'default' accuracy $p(H)$.

Naturally, we want high novelty and high satisfaction. The question is then how to combine them in a single measure. Note that the product of novelty and satisfaction, i.e., $\frac{(\pi_{\bar{H}B} - p_{\bar{H}B})^2}{\pi_{\bar{H}B}}$, is equal to the term in the sum for Φ^2 associated with the cell $\bar{H}B$. If we want to assess the contribution of a particular cell, i.e., a particular pair of observed and expected frequencies, we ask the question: What would be the *lowest* Φ^2 obtainable if we could vary everything except that particular cell and the population size? The answer is given by the following result.

Theorem 1. *For any contingency table with given $\pi_{\bar{H}B}$ and $p_{\bar{H}B}$:*

$$\Phi^2 \geq \Phi^2_{\bar{H}B} = \left(\frac{\pi_{\bar{H}B} - p_{\bar{H}B}}{\sqrt{\pi_{\bar{H}B}} - \pi_{\bar{H}B}} \right)^2$$

This minimum is reached in the following table:

	B	$\bar{B}$	
H	$\sqrt{\pi_{\bar{H}B}} - p_{\bar{H}B}$	$1 + p_{\bar{H}B} - 2\sqrt{\pi_{\bar{H}B}}$	$1 - \sqrt{\pi_{\bar{H}B}}$
	$p_{\bar{H}B}$	$\sqrt{\pi_{\bar{H}B}} - p_{\bar{H}B}$	$\sqrt{\pi_{\bar{H}B}}$
$\overline{H}$	$\sqrt{\pi_{\bar{H}B}}$	$1 - \sqrt{\pi_{\bar{H}B}}$	1

Proof: Φ^2 can be expressed in $p_{\bar{H}B}$, $\pi_{\bar{H}B}$, and (say) $p_{\bar{H}}$. The resulting expression is quadratic in $p_{\bar{H}}$ and thus has a single minimum. $\Phi^2_{\bar{H}B}$ is then calculated by setting the $p_{\bar{H}}$-derivative to 0 and solving for $p_{\bar{H}}$. □

Notice the equality of $p_{\bar{H}}$ and p_B in this table. More generally, we have $\Phi^2 = \Phi^2_{\bar{H}B}$ iff $p_{\bar{H}} = p_B = \sqrt{\pi_{\bar{H}B}}$.

Putting everything together, we are now ready to define the degree of confirmation of a rule.

Definition 5. The *degree of confirmation* of a rule $\forall(H \leftarrow B)$ is defined as

$$\Phi_{\bar{H}B} = \pm\sqrt{\Phi^2_{\bar{H}B}} = \frac{\pi_{\bar{H}B} - p_{\bar{H}B}}{\sqrt{\pi_{\bar{H}B}} - \pi_{\bar{H}B}}$$

Thus, given the observed and expected relative frequencies of counterinstances of a rule $\forall(H \leftarrow B)$, we define its degree of confirmation as the minimal Φ that results from those relative frequencies, by building the "virtual" contingency table displayed in Theorem 1. In this way we cancel out the contributions of other associations that can possibly exist between H and B, for instance the association expressed by $\forall(B \leftarrow H)$.

Clearly $\Phi_{\bar{H}B}$, being the square root of the "virtual" Φ^2, is normalised between -1 and $+1$. We have that $\Phi_{\bar{H}B} = 1$ iff $p_{\bar{H}B} = 0$ and $p_{\bar{H}} = p_B = .5$, $\Phi_{\bar{H}B} = 0$ iff $p_{\bar{H}B} = \pi_{\bar{H}B}$ (P1), and $\Phi_{\bar{H}B} = -1$ iff $p_{\bar{H}B} = \sqrt{\pi_{\bar{H}B}}$. In general, $\Phi_{\bar{H}B}$ increases with decreasing number of observed counterinstances and increasing number of expected counterinstances (P2-3). Finally, P4 is satisfied since $\Phi_{\bar{H}B}$ is not symmetric in H and B.

Example. Consider the following contingency table, associated with the rule `∀XY : female(X) ← mother(X, Y)`:

	mother(X,Y)	$\overline{\texttt{mother(X,Y)}}$	
female(X)	11 (5.8)	179 (184.2)	190
$\overline{\texttt{female(X)}}$	0 (5.2)	171 (165.8)	171
	11	350	361

We have $\pi_{\bar{H}B} = .014$, and $\Phi_{\bar{H}B} = .137$. This corresponds to $\sqrt{\Phi^2}$ of the following virtual table:

	mother(X,Y)	$\overline{\texttt{mother(X,Y)}}$	
female(X)	43.4 (38.2)	274.2 (279.4)	317.6
$\overline{\texttt{female(X)}}$	0 (5.2)	43.4 (38.2)	43.4
	43.4	317.6	361

Notice how both observed and expected frequencies of the lower left-hand cell stay the same, but the values got evenly distributed on the other diagonal, also increasing the upper right-hand cell.

On the other hand, the rule $\forall XY : \texttt{mother(X,Y)} \leftarrow \texttt{female(X)}$ has $p_{\bar{H}B} = .496$, $\pi_{\bar{H}B} = .510$, and $\Phi_{\bar{H}B} = .070$. □

3.2. Optimistic estimate of confirmation

The degree of confirmation of a rule, as defined in the previous section, is a measure for ranking given rules. In order to use it as a search heuristic, we need a function that estimates the extent to which the degree of confirmation of a given rule can be improved by specialising it. If this function never under- estimates the possible improvement, it is called an *optimistic* estimate. The significance of such an optimistic estimate is that it guarantees completeness of A* search, i.e., we can prune without fear of losing good solutions. Clearly, in order to prune as much as possible we would like the estimate to be as tight as possible, without becoming pessimistic.

In clausal logic, specialising a clause means adding a literal or applying a substitution. The latter kind of specialisation reduces the number of variables, thereby changing the sample of grounding substitutions. We discuss this special case later, and for the moment assume that the only possible specialisations consist in making the body harder to satisfy and/or the head easier to satisfy.

Definition 6. An *admissible specialisation* of a rule $\forall(H \leftarrow B)$ is a rule $\forall(H' \leftarrow B')$ such that $\forall(H \leftarrow B) \models \forall(H' \leftarrow B')$, $n_{\bar{H}'} \leq n_{\bar{H}}$, and $n_{B'} \leq n_B$.

Thus, any admissible specialisation (possibly) decreases $n_{\bar{H}B}$ and (possibly) increases $n_{H\bar{B}}$, while the other two cells can either increase or decrease. Note, however, that in general

Φ is highest for tables with low numbers in $\bar{H}B$ and $H\bar{B}$, and high numbers in the other two cells. That is, the most promising admissible specialisations are those that move all substitutions from $\bar{H}B$ to HB and $\overline{HB}$, leaving $H\bar{B}$ untouched. Furthermore, $n_{HB} = n_{\overline{HB}}$ prevents redistribution of substitutions in the virtual table of Theorem 1.

We then arrive at the following optimistic estimate, expressed as a function of $p_{H\bar{B}}$.

Theorem 2. *Under simple expected frequencies, if* $\forall(H' \leftarrow B')$ *is an admissible specialisation of* $\forall(H \leftarrow B)$*, then*

$$\Phi_{\bar{H}'B'} \leq \frac{1 - p_{H\bar{B}}}{1 + p_{H\bar{B}}}$$

Proof: We have argued above that the most confirmed admissible specialisation $\forall(H' \leftarrow B')$ has $p_{\bar{H}'B'} = 0$, $p_{H'B'} = p_{\overline{H'B'}}$, and $p_{H'\bar{B}'} = p_{H\bar{B}}$. It follows that $p_{\bar{H}'} = p_{B'}$, and thus $\sqrt{\pi_{\bar{H}'B'}} = (p_{\bar{H}'} + p_{B'})/2$. Then

$$\begin{aligned}
\Phi_{\bar{H}'B'} &= \frac{\pi_{\bar{H}'B'} - p_{\bar{H}'B'}}{\sqrt{\pi_{\bar{H}'B'}} - \pi_{\bar{H}'B'}} \\
&= \frac{\pi_{\bar{H}'B'}}{\sqrt{\pi_{\bar{H}'B'}} - \pi_{\bar{H}'B'}} \\
&= \frac{\sqrt{\pi_{\bar{H}'B'}}}{1 - \sqrt{\pi_{\bar{H}'B'}}} \\
&= \frac{p_{\bar{H}'} + p_{B'}}{2 - p_{\bar{H}'} - p_{B'}} \\
&= \frac{1 - (p_{H'\bar{B}'} - p_{\bar{H}'B'})}{1 + (p_{H'\bar{B}'} - p_{\bar{H}'B'})} \\
&= \frac{1 - p_{H\bar{B}}}{1 + p_{H\bar{B}}}
\end{aligned}$$

□

Such a specialisation decreases the expected relative frequency of counterinstances to $(p_{\bar{H}} + p_B - p_{\bar{H}B})^2/4$ (assuming that the specialised rule is satisfied, the degree of confirmation of the specialised rule will however increase).

Example. Consider the contingency table on the left. As $p_{\bar{H}B} > \pi_{\bar{H}B} = .2$, this table has negative confirmation. The best we can hope to achieve by specialisation is a confirmation of $\frac{10-4}{10+4} = .43$, corresponding to the table on the right.

	B	$\bar{B}$	
H	2	4	6
$\bar{H}$	3	1	4
	5	5	10

	B'	$\bar{B}'$	
H'	3	4	7
$\bar{H}'$	0	3	3
	3	7	10

This table has $\pi_{\bar{H}'B'} = (.4 + .5 - .3)^2/4 = .09$. □

We can make the estimate a bit tighter by noting that the optimum cannot be reached if $p_{\bar{H}B} < p_B - p_{\bar{H}}$ (or $p_{\bar{H}B} < p_{\bar{H}} - p_B$). In such cases, we can get no more than $p_{\bar{H}}(p_B - p_{\bar{H}B})$ (or $p_B(p_{\bar{H}} - p_{\bar{H}B})$) expected counterinstances in any specialisation.

The optimistic estimate just derived is correct for admissible specialisations, that make the head easier and/or the body harder to satisfy. It is thus correct for any specialisation that adds a literal without altering the variables in the rule. It is also correct for specialisations that add a literal with new variables to the head (body), since this cannot increase the relative frequency of satisfying (falsifying) substitutions. A problem occurs with specialisations that decrease the number of variables, i.e., substitutions. The problem is that under substitutions $\pi_{\bar{H}B}$ can *increase*, counter to the assumptions in the above proof.

Example. Consider a domain of 20 animals, each belonging to one of 4 classes. 8 of the 20 animals are birds, all of them having feathers. The expected relative frequency of counterinstances of the rule $\forall \mathtt{AC} : \mathtt{class}(\mathtt{A}, \mathtt{C}) \leftarrow \mathtt{feathered}(\mathtt{A})$ is $\frac{80-20}{80} \times \frac{8}{20} = .3$. The expected relative frequency of counterinstances of the rule $\forall \mathtt{A} : \mathtt{class}(\mathtt{A}, \mathtt{bird}) \leftarrow \mathtt{feathered}(\mathtt{A})$ is $\frac{20-4}{20} \times \frac{8}{20} = .32$. The optimistic estimate associated with the first rule will under-estimate this number and therefore the confirmation of the second rule, since it has no counterinstances. □

There are basically two solutions to this problem. The first is to represent substitutions explicitly in both head and body of a rule. In this way we make sure that also substitutions are admissible specialisations, since they don't remove any variables. E.g., the second rule in the example becomes

$$\forall \mathtt{AC} : \mathtt{class}(\mathtt{A}, \mathtt{C}) \vee \mathtt{C} \neq \mathtt{bird} \leftarrow \mathtt{feathered}(\mathtt{A}) \wedge \mathtt{C} = \mathtt{bird}$$

The expected relative frequency of counterinstances of this clause is $\frac{16}{80} \times \frac{8}{80} = .02$.

The second solution to the substitution problem is to restrict counting of satisfying and falsifying substitutions to a fixed set of variables. E.g., in the first rule above we would restrict counting to the variable A, and the expected relative frequency of counterinstances becomes $\frac{20}{20} \times \frac{8}{20} = .4$. In effect, the rule would be interpreted as

$$\forall \mathtt{A} : (\forall \mathtt{C} : \mathtt{class}(\mathtt{A}, \mathtt{C})) \leftarrow \mathtt{feathered}(\mathtt{A})$$

i.e., variable C becomes *local* to the head of the rule. The negation of the head is now $\exists \mathtt{C} : \neg \mathtt{class}(\mathtt{A}, \mathtt{C})$, which is true for any value of A.

Definition 7. In the *individual-based representation*, the domains of grounding substitutions are restricted to a fixed set of *individual variables*. Other variables occurring in a rule are called *auxiliary variables*; they are always local to the body or head in which they occur. Admissible specialisations can only instantiate or unify auxiliary variables.

This representation is called individual-based because the individual variables quantify over the individuals in the domain, be they animals, trains, or molecules. The individual-based first-order representation is a natural upgrade of the propositional attribute-value

representation (Flach, Giraud-Carrier, & Lloyd, 1998; Flach & Lachiche, 1999a). In the Tertius system (Section 4) the user can choose between an individual-based representation and explicit substitutions if she wants to restrict to admissible specialisations. The unrestricted setting is also available, but as indicated above this renders the confirmation estimate non-optimistic and the A* search potentially incomplete.

3.3. Including background knowledge in confirmation

Until now we have assumed simple expected frequencies under the null hypothesis of statistical independence of head and body of a rule, as defined by Definition 2. In this section we describe a more sophisticated null hypothesis which offers two distinct advantages:

1. it assigns the same confirmation to logically equivalent rules;
2. it allows the incorporation of logical background knowledge.

Although the approach is fully implemented in the Tertius system, there are still some open issues that need to be resolved. This section is therefore somewhat speculative, and can be skipped on first reading.

To illustrate the first point, consider the two logically equivalent rules $\forall x : P(x) \leftarrow Q(x) \wedge R(x)$ and $\forall x : P(x) \vee \neg R(x) \leftarrow Q(x)$. Although these rules have the same number of observed counterinstances, they have different heads and bodies and therefore different marginal distributions and expected frequencies. This means that under simple expected frequency we treat $\leftarrow$ non-classically.[2] As long as we have some means of deciding what goes in the head and what in the body, for instance because we are doing classification, this is not really a problem. If we do not have a reason for distinguishing between logically equivalent but syntactically different rules, it makes sense to consider multiway contingency tables, which are capable of assessing dependencies between more than two objects.

In Table 4 a three-way contingency table is depicted for the three-literal clause $\forall(H \leftarrow B_1 \wedge B_2)$. There are eight observed frequencies, $n_{HB_1B_2}$ etc.; also depicted are the one-way marginals for H, B_1 and B_2. In addition there are three two-way marginals for HB_1, HB_2 and B_1B_2 which are not explicitly included in the table, although they can easily be calculated from the observed frequencies. As stated before, the expected frequencies can sum up to all of these marginals, or only to a subset, as determined by the null hypothesis. We will be using a general iterative algorithm to approximate expected frequencies, and therefore are able to use arbitrary null hypotheses of complete or partial independence. To simplify the

Table 4. The three-way contingency table for the clause $\forall(H \leftarrow B_1 \wedge B_2)$.

	B_1/n_{B_1}	$\bar{B_1}/n_{\bar{B_1}}$	B_1	$\bar{B_1}$	
H	$n_{HB_1B_2}$	$n_{H\bar{B_1}B_2}$	$n_{HB_1\bar{B_2}}$	$n_{H\overline{B_1B_2}}$	n_H
$\bar{H}$	$n_{\bar{H}B_1B_2}$	$n_{\overline{HB_1}B_2}$	$n_{\bar{H}B_1\bar{B_2}}$	$n_{\overline{HB_1B_2}}$	$n_{\bar{H}}$
	B_2/n_{B_2}		$\bar{B_2}/n_{\bar{B_2}}$		N

discussion, we will however only discuss the null hypothesis of complete independence, and show how to adapt it to include background knowledge.

Example. Consider the following three-way contingency table:

	$B_1/6$	$\bar{B}_1/14$	B_1	$\bar{B}_1$	
H	3	3	0	3	9
$\bar{H}$	3	4	0	4	11
	$B_2/13$		$\bar{B}_2/7$		20

The expected number of counterinstances of the clause $\forall(H \leftarrow B_1 \wedge B_2)$ under complete independence of the 3 literals is $\frac{11 \times 6 \times 13}{20^2} = 2.15$. Similarly, $\mu_{\bar{H}B_1\bar{B}_2} = \frac{11 \times 6 \times 7}{20^2} = 1.15$. Notice that the clause gets negative confirmation, as the actual number of counterinstances is 3. □

Another way to calculate the 8 expected frequencies is to initialise them all to 1, and then proportionally change them to fit each of the one-way marginals in turn. For instance, since the first row of four 1's needs to add up to 9, we multiply all by 2.25; similarly, the second row gets multiplied by 2.75. We then continue to fit the B_1 marginal in the same way, yielding 1.35 - 3.15 - 1.35 - 3.15 for the first row, and 1.65 - 3.85 - 1.65 - 3.85 for the second row. Finally, after fitting the B_2 marginal the table contains the correct expected frequencies. For further details the reader is referred to Dahl (1999).

This iterative fitting algorithm provides the clue for incorporating background knowledge. Suppose that the clause $\forall(B_2 \leftarrow B_1)$ is part of the background knowledge. That is, the two zeroes in the above table could have been predicted without consulting the data. What we would like to do now is to adjust the expected frequencies to take this knowledge into account. That is, we would like to find a set of expected frequencies that fit the one-way marginals **and** obey $\mu_{HB_1\bar{B}_2} + \mu_{\bar{H}B_1\bar{B}_2} = 0$. This is easily achieved by using the iterative fitting algorithm just explained, and to initialise all cells to 1 except the indicated ones, which are set to 0. Since the fitting algorithm only makes multiplicative changes, it will converge to a solution that fits the one-way marginals **and** includes the desired zeroes.

Example. We continue the previous example. Taking the required zeroes into account, the iterative fitting algorithm calculates the following expected frequencies:

	$B_1/6$	$\bar{B}_1/14$	B_1	$\bar{B}_1$	
H	2.7	3.15	0	3.15	9
$\bar{H}$	3.3	3.85	0	3.85	11
	$B_2/13$		$\bar{B}_2/7$		20

Due to the lower expected frequencies in the $B_1\bar{B}_2$ column, the frequencies in the B_1B_2 column get increased, and the clause $\forall(H \leftarrow B_1 \wedge B_2)$ will get positive confirmation. □

The approach has been implemented in the Tertius system (Section 4). Before calculating the expected values in the multiway table, each of the cells is tested for satisfiability against the background knowledge by asking the appropriate Prolog query. Clearly, some open issues remain. For instance, multiway contingency tables are exponential, e.g., for an m-way table 2^m Prolog queries are required. Also, interfacing with Prolog can be slow, as the rest of the system is implemented in C. More generally, the results obtained by means of two-way analysis (including the optimistic estimate) may no longer apply and need to be adapted. However, we do believe to have indicated a novel and promising way in which statistical and logical information can be combined in a general-purpose discovery system.

4. Tertius

The Tertius system implements a top-down rule discovery system employing the confirmation measure defined in the previous section. The system is implemented in approximately 7500 lines of GNU C and is freely available for academic purposes (Flach & Lachiche, 1999b). In this section we describe the Tertius system and some of its options in more detail. From this point on, the representation language is first-order clausal logic in Prolog notation. A clause consists of a head H and a body B, and is denoted $H :- B$. The head is a disjunction of literals separated by semicolons. The body is a conjunction of literals separated by commas. Constant names start with a lowercase letter or digit, whereas variable names start with a capital.

4.1. Representation

Tertius uses a first-order logic representation. Such a representation allows it to deal with several kinds of data and, moreover, allows the user to choose the most convenient or the most comprehensible representation among several possible representations.

Tertius is able to deal with extensional knowledge, either with explicit negation or under the Closed-World Assumption. In the first case, the truthvalue of all ground facts is given. For example, `mutagenic(d101)` indicates that molecule `d101` is mutagenic, and if molecule `d102` is not mutagenic, it should be specified by the explicit negative ground fact ¬ `mutagenic(d102)`. No assumption is made when the truthvalue of a ground fact is not given. Given a third molecule `d103` whose mutagenicity is unknown, molecules `d102` and `d103` are not counted as molecules satisfying `mutagenic(X)` and molecules `d101` and `d103` are not counted as molecules satisfying ¬ `mutagenic(X)`.

For calculating the degree of confirmation, only the substitutions satisfying explicitly the body of the rule (respectively the negation of the head, the conjunction of the body and the negation of the head) are taken into account. The remaining substitutions are counted in the other cells and marginals. This approach is conservative with respect to the counterinstances, since only those that are explicitly given are counted. From a three-valued logic perspective,

the undefined value of the body (respectively the negation of the head) is merged with the substitutions not satisfying the body (resp. the negation of the head).

Under the Closed-World Assumption, only true ground facts are given and all the other ground facts are supposed to be false. For example, given `molecule(d101)`, `molecule(d102)`, `mutagenic(d101)`, the missing ground fact `mutagenic(d102)` is implicitly assumed to be false. The degree of confirmation is calculated from true ground facts only, which means that in this case the approach is conservative regarding substitutions satisfying the body, but greedy when it comes to counting substitutions falsifying the head.

The previous representations require the truthvalue of all ground facts to be given either explicitly or implicitly by the Closed-World Assumption. A main advantage of a first-order representation is, however, the ability to use background knowledge. `Tertius` can use intensional knowledge where the truthvalues of ground facts are not given in extension but in intension. In this case, the truthvalues are derived from the available knowledge using either the Prolog inference mechanism if the background knowledge consists of Horn clauses, or using a theorem prover otherwise. For instance, since a bond between two atoms in a molecule is symmetric, given `bond(d101_1,d101_2,7)` the symmetric fact `bond(d101_2,d101_1,7)` can be deduced from the background knowledge `bond(A1,A2,K):-bond(A2,A1,K)`.[3]

`Tertius` assumes that predicates are weakly typed, i.e., each argument of a predicate belongs to a named type.[4] For instance, arguments of the predicate `bond(A1,A2,B)` are two atoms `A1` and `A2`, and the kind of bond `B`. The set of values of the argument found in the data defines the domain of the associated type. For example, the domain of the kind of bond is the set {`1,2,3,4,5,7`} of all constants appearing in the data as the third argument of `bond(A1,A2,B)`.

Some arguments of a predicate are meaningful only when they are instantiated. For instance, the predicate `atomel(Atom,Element)` states that the atom `Atom` is an `Element`. Obviously, an atom is always of some element. It cannot be used as a condition in a rule since it is always satisfied. Therefore the variable `Element` must always be instantiated with an element constant. This kind of argument is called a *parameter* in `Tertius`. Parameters are similar to non-boolean attributes in attribute-value learning. Moreover, if a domain is continuous, `Tertius` allows discretisation into a user-specified number of intervals, each one standard deviation wide, and centred around the mean (figure 1).[5]

4.2. Non-redundant refinement operator

`Tertius` uses a top-down best-first search. In a top-down search, shorter and more general rules are considered first. θ-subsumption (Plotkin, 1970, 1971) is used to compare the

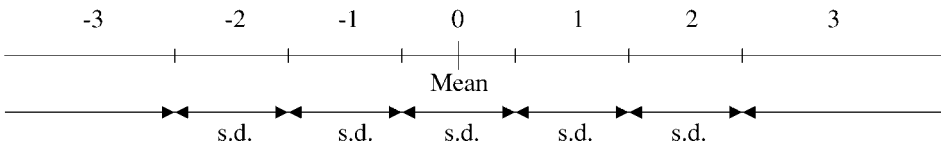

Figure 1. Discretisation of continuous domains.

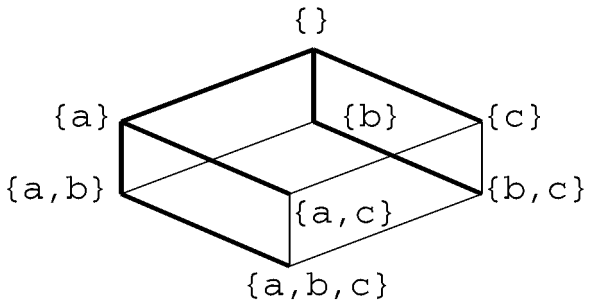

Figure 2. A lattice and a covering tree.

generality of rules. The search starts with the empty rule and then refines it iteratively. The refinement operations used in Tertius are: adding a literal, unifying two variables, and instantiating a variable with a constant from its domain. The search space is a lattice: there may be several paths from the empty rule to a hypothesis. Considering the same clauses several times would mean generating all their refinements several times, therefore it would decrease the efficiency of the search. The aim is to find a tree that covers the lattice. A typical example is the enumeration of subsets of a set (Rymon, 1992). Figure 2 shows the lattice of subsets of the set a, b, c and a covering tree in bold lines. The search is constrained along a covering tree by ordering the refinement operations and allowing an operation to be applied only if all the refinement operations applied until then precede it in the ordering.

Refinement operations in Tertius obey the following order: adding a new literal, unifying two variables, and instantiating a variable with a constant from its domain. For instance, adding a literal to a clause is not allowed once two variables have been unified. Furthermore, each of these refinement operations can generate a lattice of hypotheses, and thus has to be constrained in a similar way.

Predicates are ordered following the order in which they are declared (see Section 4.3). A literal containing predicate P can only be added to a clause not containing predicates succeeding P in the ordering. For instance, if predicate `atomel(Atom,c)` is declared before predicate `atomty(Atom,22)`, then it is possible to refine the rule `mutagenic(M) :- atm(M,A), atomel(A,c)` into the rule `mutagenic(M) :- atm(M,A), atomel(A,c), atomty(A,22)`, but the latter rule cannot be obtained as a refinement of `mutagenic(M) :- atm(M,A), atomty(A,22)`.

Variables of a rule are also ordered according to their order in the clause (right to left). A variable at position i can be unified with a variable at position $j > i$ iff (1) no variable at position $k > i$ has been unified so far, and (2) the variable at position i has not been unified with a variable at position $l > j$ before. For instance, the clause `p(Z,Y,X) :- q(Y,X)` can be refined into either of

```
p(Z,X,X) :- q(X,X)
p(X,Y,X) :- q(Y,X)
p(Y,Y,X) :- q(Y,X)
```

The first refinement can be further refined to `p(X,X,X) :- q(X,X)`. But the second refinement cannot be further refined by unification of two variables, because the first variable `X` has been unified with the third one `Z`, i.e., `X` can no longer be unified with the second variable `Y` (2), and `Y` is never allowed to be unified with a variable preceding it in the ordering. The third refinement cannot be further refined by unification of two variables either, because since the second variable `Y` has been unified already, `X` can no longer be unified with any variable (1).

Finally, the instantiation of variables with a constant of their domain is constrained in a similar fashion. Constants are ordered following the order in which they appear in the data. Then the mechanism is similar to the one of unification: a variable at position i can be instantiated with a constant at position j from its domain iff no variable at position $k > i$ has been instantiated so far, and the variable at position i has not been instantiated with a constant at position $l > j$ before. For instance, some possible refinements of the rule `mutagenic(M):- atm(M,A), atomel(A,B), atomty(A,C)` are:

```
mutagenic(M) :- atm(M,A), atomel(A,c), atomty(A,C)
mutagenic(M) :- atm(M,A), atomel(A,B), atomty(A,22)
```

The first refinement can in turn be refined into `mutagenic(M) :- atm(M,A), atomel(A,c), atomty(A,22)`. But the second refinement cannot be further refined by instantiation of the variable `B`, since it has been obtained by instantiating the variable `C` which succeeds `B` in the ordering.

In conclusion, we note that these constraints are similar to the constraints established independently by Badea and Stanciu in order to prove that refinement operators can be weakly perfect (Badea & Stanciu, 1999).

4.3. Inductive bias

Inductive biases are auxiliary pieces of knowledge used to produce, in a deductive way, candidate inductive hypotheses from the data. They can be either language biases or search biases.

Several language biases are used in `Tertius` to define and restrict the search space. The search is always restricted to a maximum number of literals and of variables, specified to `Tertius` as command-line arguments. In addition, the user can restrict the number of occurrences of individual predicates. The hypothesis language considered in `Tertius` is the set of clausal rules. If the user so chooses, it can be restricted to Horn clauses only (option `-b horn`). Moreover the predicate in the head can be fixed for classification or for subgroup discovery (options `-b class` for positive and negative classification rules, `-b pos_class` for positive classification rules only, and `-b horn_pos_class` for positive classification rules without negation in the body).

Additional language biases are best explained by considering an example of the declarations required by `Tertius`. These declarations concern a possible representation of the mutagenesis problem discussed in Section 5.1.

```
--INDIVIDUAL
mol 1 mol cwa
--STRUCTURAL
atm 2 1:mol *:atom * cwa
--PROPERTIES
mutagenic 1 mol cwa
atomel 2 atom #letter cwa
atomty 2 atom #number cwa
atomch 2 atom #charge cwa
bond 3 atom atom #type cwa
```

In this domain we use an individual-based representation. The individual is defined by a type (or a cartesian product of types) by means of a dummy predicate. In the example above, the first two lines define a single individual variable of type `mol`.

Moreover, we distinguish between *structural predicates* referring to parts of individuals, and *properties* of either individuals or their parts. Structural predicates are binary predicates used to represent the link between a complex type and one of its components. They are used to introduce new variables in rules. Lines 3–4 in the declaration above define a structural predicate `atm(M,A)`, referring to an atom `A` of the molecule `M`. It is a one-to-many relation, indicating that one molecule may contain many atoms, but each atom belongs to exactly one molecule. This functions as a language bias, since rules like `mutagenic(M):-atm(M,A),atm(B,A)` will not be considered. Furthermore, with a structural representation like the above, every auxiliary variable must be introduced by a structural predicate in the body of the rule.

In an individual-based representation, properties are the predicates that are not structural. They cannot introduce any new variable, they just state properties of the individual or of its parts represented by the variables introduced so far. The remaining lines of the example above indicate that there are five properties, a unary property `mutagenic` of molecules, three binary properties of atoms (`atomel`, `atomty` and `atomch`), and a ternary property representing a `bond` between two atoms. The last argument's types of the last properties are preceded by #, indicating that they are *parameters* which should always be instantiated in rules. Notice that we use the Closed-World Assumption for all predicates here (`cwa`).

These language biases are similar to mode declarations, albeit on the predicate level rather than the argument level. Other ILP systems, such as `Progol` (Muggleton, 1995) or WARMR (Dehaspe & De Raedt, 1997), use mode declarations such as `inside (-Object1, +Object2)` indicating that the first object is the output argument, and that the second object `Object2` is an input argument and should use a variable already occurring previously in the current hypothesis. They however do not distinguish between structural predicates and properties, as their hypothesis language is not individual-based.

Several search biases are used in `Tertius`. When only admissible specialisations are considered (Definition 6), refining a rule can only lead to a new body satisfied by fewer substitutions, and similarly for the negation of the new head. Therefore, if less than 10% of the individuals satisfy, for instance, the body of a rule, then less than 10% of the individuals will satisfy the body of any of its refinements. Since it is desirable that a rule covers a

significant proportion of the individuals, a threshold f can be set on the proportion of individuals satisfying the body (resp. the negation of the head) of a rule (option `-f` f). This *frequency threshold* is one of the search biases used in `Tertius`, and is similar to the pruning rule used in the search of frequent subsets by Dehaspe and Toivonen (1999).

The optimistic estimate (Section 3.2) is also used to constrain the search. That is, a threshold c can be set on the confirmation, and the refinement of a rule can be stopped as soon as the optimistic estimate of the confirmation of its refinements is lower than the threshold (option `-c` c). Alternatively, the k best confirmations found so far can be stored and the lowest of the current k best confirmations can be used as a dynamic threshold on the confirmation (option `-k` k; $k = 10$ by default).

If a rule is added to the current k best rules, it may or may not be further refined. The reason for refining it is that one of its refinements may have an even higher confirmation. However, if the rule has no counterinstances, its confirmation cannot be increased by specialisation—in this case we do not consider its refinements. This stopping criterion can be relaxed by allowing a small percentage n of counterinstances, typically referred to as the *noise level* (option `-n` n). Another `Tertius` option is to restrict output to satisfied clauses only (option `-sat`); with this option, `Tertius` performs categorical confirmatory induction (see Definition 1). Combining the `-sat` and `-n` n options, `Tertius` produces 'almost satisfied' clauses.

5. Experiments

In this section we describe our experience with applying `Tertius` to a variety of discovery tasks. In Section 5.1 we use `Tertius` to discover classification rules in the mutagenesis domain. Section 5.2 describes our experiments in a problem solving and planning domain. In Section 5.3 `Tertius` discovers functional dependencies in a database relation, and in Section 5.4 the system is applied to a multiple predicate learning task.

5.1. Classification

Although not specifically designed for classification tasks, `Tertius` can be used to learn rules predicting a given predicate. We consider in this section the problem of identifying mutagenic compounds from a dataset of 188 molecules (Srinivasan et al., 1994). `Tertius` was run with an individual-based hypothesis language, restricted to clauses with the target predicate `mutagenic(A)` in the head and positive literals in the body. The rules it found are reproduced below. The first column of numbers indicates the degree of confirmation $\Phi_{\bar{H}B}$ of the rule, while the second column indicates the relative frequency of counterinstances $p_{\bar{H}B}$. The number of hypotheses explored and the average number of ground literals that have been evaluated for each rule are reported after the list of rules. Notice that the latter number is roughly the number of possible instantiations of the rule times its number of literals. The CPU-time on a Sun Ultra 10 is also given.

```
tertius -b hornposclass 3/2 mutagenesis
t1. /* 0.364484 0.058511 */ mutagenic(A) :- atm(A,B), atomty(B,27).
```

```
t2. /* 0.231743 0.063830 */ mutagenic(A) :- atm(A,B), atomty(B,29).
t3. /* 0.210764 0.000000 */ mutagenic(A) :- atm(A,B), atomty(B,28).
t4. /* 0.198029 0.005319 */ mutagenic(A) :- atm(A,B), atomch(B,0.142).
t5. /* 0.198029 0.005319 */ mutagenic(A) :- atm(A,B), atomch(B,-0.118).
t6. /* 0.187607 0.000000 */ mutagenic(A) :- atm(A,B), atomch(B,0.812).
t7. /* 0.182032 0.005319 */ mutagenic(A) :- atm(A,B), atomch(B,0.145).
t8. /* 0.182032 0.005319 */ mutagenic(A) :- atm(A,B), atomch(B,0.012).
t9. /* 0.179557 0.000000 */ mutagenic(A) :- atm(A,B), atomch(B,0.141).
t10. /* 0.173767 0.005319 */ mutagenic(A) :- atm(A,B), atomch(B,-0.388).
t11. /* 0.171306 0.000000 */ mutagenic(A) :- atm(A,B), atomty(B,195).
t12. /* 0.154081 0.000000 */ mutagenic(A) :- atm(A,B), atomch(B,-0.085).
Number of hypotheses explored: 498
Average number of ground queries to the database: 14479
Time: 46.7s
```

The rules found by Tertius are compared with the following rules found with Progol[6] using information on atoms and bonds only (Srinivasan, King, & Muggleton, 1996) (to facilitate comparison, we calculated $\Phi_{\bar{H}B}$ and $p_{\bar{H}B}$ for these rules):

```
p1. /* 0.126 0.000 */ mutagenic(A):-atm(A,B),atm(A,C),atm(A,D),
                                    atomel(B,c),atomty(B,29),
                                    bond(C,B,7),bond(D,C,1).
p2. /* -0.01 0.027 */ mutagenic(A):-atm(A,B),atm(A,D),atomel(B,c),
                                    atomty(B,22),atomch(B,C),C=<0.202,
                                    atomel(D,c),atomty(D,29).
p3. /* 0.241 0.021 */ mutagenic(A):-atm(A,B),atm(A,C),atm(A,D),
                                    atomel(B,c),atomty(B,27),
                                    bond(C,B,7),bond(D,C,1).
p4. /* 0.389 0.000 */ mutagenic(A):-atm(A,B),atomel(B,c),
                                    atomty(B,27),atomch(B,C),C=<-0.085.
p5. /* 0.086 0.011 */ mutagenic(A):-atm(A,B),atomel(B,c),
                                    atomty(B,22),atomch(B,-0.114).
p6. /* 0.269 0.011 */ mutagenic(A):-atm(A,B),atomel(B,c),
                                    atomty(B,29),atomch(B,C),C=<0.011.
p7. /* 0.171 0.000 */ mutagenic(A):-atm(A,B),atomel(B,c),atomty(B,195).
p8. /* 0.028 0.021 */ mutagenic(A):-atm(A,B),atm(A,D),atomel(B,o),
                                    atomty(B,40),atomch(B,C),
                                    atomel(D,n),atomty(D,32),atomch(D,C).
p9. /* 0.211 0.000 */ mutagenic(A):-atm(A,B),atm(A,D),atomel(B,c),
                                    atomty(B,10),atomch(B,C),atomel(D,c),
                                    atomty(D,10),atomch(D,C),bond(D,B,1).
p10. /* 0.104 0.000 */ mutagenic(A):-atm(A,B),atm(A,D),atomel(B,o),
                                    atomty(B,40),atomch(B,C),
                                    C=<-0.403,bond(B,D,2).
```

```
p11. /* 0.092 0.000 */ mutagenic(A):-atm(A,B),atomel(B,c),
                                     atomty(B,16),atomch(B,-0.191).
p12. /* 0.097 0.011 */ mutagenic(A):-atm(A,B),atm(A,D),atomel(B,c),
                                     atomty(B,26),atomel(D,c),
                                     atomty(D,22),atomch(D,E),E=<-0.110.
p13. /* 0.063 0.000 */ mutagenic(A):-atm(A,B),atomel(B,c),
                                     atomty(B,29),atomch(B,0.010).
p14. /* 0.079 0.000 */ mutagenic(A):-atm(A,B),atm(A,C),atm(A,D),
                                     atm(A,E),atomel(B,cl),atomty(B,93),
                                     bond(D,E,2),bond(C,E,1).
p15. /* 0.163 0.000 */ mutagenic(A):-atm(A,B),atm(A,C),atm(A,D),atomel(B,o),
                                     atomty(B,40),atomch(B,-0.389),
                                     bond(C,B,2),bond(D,C,1).
```

Rule t1 found by `Tertius` covers rules p3 and p4 found by `Progol`, and rule t2 covers rules p1, p2, p6, and p13. Note that p4 and p6 are refinements of t1 and t2, respectively, that get higher confirmation and less counterinstances (they weren't found by `Tertius` because they are outside the specified language bias).

On the other hand, `Tertius` finds also some considerably simpler rules that are equivalent to `Progol` rules in terms of confirmation and counterinstances. For instance, rule t11 covers rule p7; since both rules have exactly the same $\Phi_{\bar{H}B}$ and $p_{\bar{H}B}$, the condition that `B` is a carbon atom in the `Progol` rule is redundant. Similarly, further down the list (not shown above) `Tertius` finds some rules subsuming other `Progol` rules (p11 and p5) with the same confirmation and counterinstances:

```
t42. /* 0.092226 0.000000 */ mutagenic(A) :- atm(A,B), atomch(B,-0.191).
t51. /* 0.086106 0.010638 */ mutagenic(A) :- atm(A,B), atomch(B,-0.114).
```

Interestingly, the most highly confirmed satisfied rule found by `Tertius` (t3), stating each of the 17 molecules containing an atom of type `28` is mutagenic, is not found by `Progol`, nor is any of its refinements. Similarly, the remaining rules (t4–t10, t12) found by `Tertius` are relatively highly confirmed, some are even satisfied, and none of them subsume any of the rules found by `Progol`.

The reader will have noticed that one rule found by `Progol` gets a negative confirmation. This can be explained by the sequential covering algorithm employed by `Progol`, which repeatedly changes the dataset (and thereby the confirmation values) by throwing out covered positive examples. Specifically, rule p2 gets a negative confirmation on the whole dataset because its expected number of counterinstances (4.7) is lower than the observed number (see the left table).

	Body	$\overline{Body}$	
Head	9	116	125
$\overline{Head}$	5	58	63
	14	174	188

	Body	$\overline{Body}$	
Head	9	0	9
$\overline{Head}$	5	58	63
	14	58	72

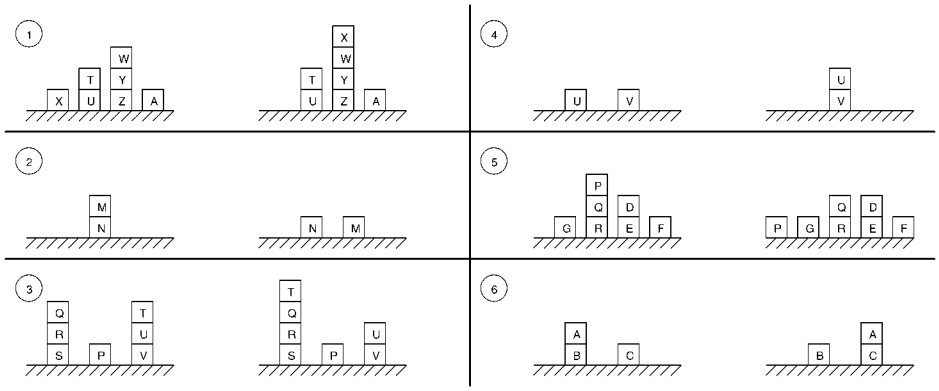

Figure 3. Examples of situations before and after stacking, unstacking, and transfer operations (after Vere, 1978; used with permission).

Without going into a detailed analysis of which positive examples were removed by Progol, we show how this could have led to much higher confirmation of p2. The maximum confirmation one can get by removing positive examples from the left contingency table would be obtained when the 116 instances satisfying $Head$ and $\overline{Body}$ are removed (right table). The expected number of counterinstances $\frac{14 \times 63}{72} = 12.25$ is then greater than the observed number of counterinstances, and the confirmation of the rule becomes 0.42. This is obviously an optimistic estimate, and the actual effect of removing positive examples can lead to a lower degree of confirmation. However, the example demonstrates that sequential covering algorithms can induce locally confirmed but globally unconfirmed rules.

In conclusion, Tertius found simple rules subsuming nine out of the fifteen rules found by Progol using atoms and bonds only. It also found nine highly confirmed rules such as t3, none of which were found by Progol. We should add that Tertius — in its current implementation — cannot learn rules of the complexity of, e.g., p14 and p15, because of the search depth required.

5.2. Problem solving and planning

Tertius can be used in a problem solving and planning context. In this section, we consider a planning domain. Figure 3 shows an experimental framework taken from Vere (1978). It consists of six examples of 'blocks world' pairs representing the situations before and after the application of one of the operators stack, unstack, and transfer. Vere's task was to identify a set of operators, defined by their pre-conditions and post-conditions, such that one of the operator applies to each example. The intended result is:

action	pre-conditions	post-conditions
`stack`	`clear(X), clear(Y), ontable(X)`	`clear(X), on(X,Y)`
`unstack`	`clear(X), on(X,Y)`	`clear(X), ontable(X), clear(Y)`
`transfer`	`clear(X), clear(Y), on(X,Z)`	`clear(X), on(X,Y), clear(Z)`

Vere defines a *production* as a pre/post-condition pair, for instance `clear(X), clear(Y), ontable(X)` → `clear(X), on(X,Y)`. The original learning task was

to learn such productions. However, Tertius is not directly applicable to that task for two reasons. In the first place, productions are not logical rules: A being clear and on the table and T being clear beforehand in example 1 does not imply that A will be stacked on T. Secondly, instead of identifying a set of "rules" completely covering the examples, Tertius learns a set of integrity constraints, each of which applies to all examples. The first learning task considered in this section is therefore to find the set of all satisfied clauses from the complete set of six examples.

In our experiments, each example is described by a set of ground facts representing the properties of each block before and after the operation, plus the identification of the subject of the operation (i.e., the block to be moved) and its destination. We also indicate that the table is a special kind of block.[7] The first example, for instance, is represented by the following facts:

```
clear(table,before).  clear(table,after).  subject(x).
clear(x,before).      clear(x,after).      destination(w).
on(x,table,before).   on(x,w,after).       table(table).
on(u,table,before).   on(u,table,after).
on(t,u,before).       on(t,u,after).
clear(t,before).      clear(t,after).
on(z,table,before).   on(z,table,after).
on(y,z,before).       on(y,z,after).
on(w,y,before).       on(w,y,after).
on(a,table,before).   on(a,table,after).
clear(a,before).      clear(a,after).
clear(w,before).
```

Given that this domain is noise-free, we looked for satisfied rules only. Since each example represents a separate dataset, partitioning was used and the confirmation value was averaged over the partitions. Tertius produced the following rules (since they are all satisfied, we omit the relative frequency of counterinstances):

```
tertius -sat -c 0 -avg 3/2 all
1. /* 0.406156 */ subject(A); on(A,B,after) :- on(A,B,before).
2. /* 0.406156 */ subject(A); on(A,B,before) :- on(A,B,after).
3. /* 0.389417 */ destination(A); on(B,A,before) :- on(B,A,after).
4. /* 0.336546 */ destination(C); clear(C,after) :- clear(C,before).
5. /* 0.329346 */ table(B) :- on(A,B,before), clear(B,before).
6. /* 0.328989 */ table(B) :- on(A,B,after), clear(B,after).
7. /* 0.245619 */ clear(B,before) :- subject(B).
8. /* 0.245619 */ clear(B,before) :- table(B).
9. /* 0.245619 */ clear(B,before) :- destination(B).
10. /* 0.244962 */ clear(B,after) :- subject(B).
11. /* 0.244962 */ clear(B,after) :- table(B).
12. /* 0.217840 */ on(A,B,after) :- subject(A), destination(B).
13. /* 0.203063 */ destination(B) :- subject(A), on(A,B,after).
```

```
14. /* 0.197992 */ table(C) :- destination(C), clear(C,after).
15. /* 0.180560 */ on(A,B,after); clear(A,before) :- on(A,B,before).
16. /* 0.180560 */ on(A,B,before); clear(B,before) :- on(A,B,after).
17. /* 0.180560 */ on(A,B,before); clear(A,before) :- on(A,B,after).
18. /* 0.179924 */ on(A,B,after); clear(B,after) :- on(A,B,before).
19. /* 0.179924 */ on(A,B,after); clear(A,after) :- on(A,B,before).
20. /* 0.179924 */ on(A,B,before); clear(A,after) :- on(A,B,after).
21. /* 0.161257 */ clear(B,before) :- on(A,B,after), clear(B,after).
22. /* 0.161005 */ clear(B,after) :- on(A,B,before), clear(B,before).
23. /* 0.139252 */ on(A,B,before) :- destination(A), on(A,B,after).
24. /* 0.139252 */ on(A,B,after) :- destination(A), on(A,B,before).
25. /* 0.103594 */ on(B,A,after) :- destination(A), on(B,A,before).
26. /* 0.097025 */ table(A) :- destination(A), on(B,A,before).
27. /* 0.091987 */ clear(B,before) :- subject(A), on(A,B,after).
28. /* 0.091679 */ clear(B,after) :- subject(A), on(A,B,before).
29. /* 0.020408 */ clear(A,after) :- destination(A), on(B,A,before).
30. /* 0.000000 */ :- subject(A), on(B,A,before).
31. /* 0.000000 */ :- subject(A), on(B,A,after).
32. /* 0.000000 */ :- on(A,B,before), on(B,A,after).
33. /* 0.000000 */ :- on(A,B,before), table(A).
34. /* 0.000000 */ :- on(B,B,before).
35. /* 0.000000 */ :- on(A,B,after), table(A).
36. /* 0.000000 */ :- on(B,B,after).
37. /* 0.000000 */ :- subject(A), destination(B), on(A,B,before).
38. /* 0.000000 */ :- subject(B), destination(B).
39. /* 0.000000 */ :- subject(B), table(B).
40. /* 0.000000 */ :- subject(A), on(A,B,before), on(A,B,after).
41. /* 0.000000 */ :- destination(A), on(A,B,before), clear(A,after).
42. /* 0.000000 */ :- destination(A), on(A,B,after), clear(A,after).
Number of hypotheses explored: 2869
Average number of ground queries to the database: 471
Time: 7.6s
```

All these rules are integrity constraints on this domain. The four most confirmed ones deal with the frame-problem. For instance, rules 1 and 2 state that a block does not move if it isn't the subject of the action. There are some characteristics of the subject (rules 7, 10, 13, 30, 31, 37, 38, 40) and some of the destination (rules 9, 14, 13, 25, 26, 27, 37, 38). Rules 15–20 and 23–24 are actually variants of rules 1 and 2. For instance, rule 19 can be read as if A is on B beforehand, and A is not clear afterwards, then A is on B afterwards, but we know (rule 10) that if A is not clear afterwards then A isn't the subject, and (rule 1) that if A is on B beforehand, and A isn't the subject, then A is on B afterwards, so rule 19 is a logical consequence of rules 1 and 10. More than half of the rules (rules 5, 6, 8, 11, 14, 21, 22, 25, 26, 29, 33, 35, 39, 41, 42) deal with the special case of the table and could be considered as background knowledge. Rules 34 and 36 are typical background knowledge too.

From the point of view of action learning, some of the rules do give the pre-conditions and post-conditions of the actions performed in the examples. Rules 7 and 9 state that the subject and the destination must be clear before. Rules 10 and 12 state that the subject is clear and on the destination afterwards. Rule 28 models a more subtle post-condition that is used mainly for the unstacking and the transfer operations: if the subject was on B beforehand, then B is clear afterwards (it happens to be true for stacking as well, because in our representation the table is always clear).

In the remainder of this section, we report results of experiments carried out for each operation separately. In a second experiment, Tertius was run on the two stacking examples, searching for satisfied clauses only. It produced 51 rules, many of which are omitted below as they were already found previously. For instance, the second rule learned in this experiment is the same as the first rule learned in the first experiment, the only difference is the confirmation value which was somewhat higher before. The rules shown below are the ones that hold for the stacking examples, but are falsified by the other examples:

```
tertius -sat -c 0 -avg 3/2 stack
s1. /* 0.322138 */ table(A); on(B,A,after) :- on(B,A,before).
s5. /* 0.310329 */ table(A) :- on(B,A,before), clear(A,after).
s10. /* 0.254569 */ clear(B,before) :- clear(B,after).
s11. /* 0.224389 */ on(B,A,before) :- table(A), on(B,A,after).
s12. /* 0.224389 */ on(A,B,before) :- on(A,B,after), clear(B,after).
s21. /* 0.183373 */ on(A,B,before) :- subject(A), table(B).
s23. /* 0.168783 */ table(B) :- subject(A), on(A,B,before).
s24. /* 0.168783 */ subject(A) :- destination(B), on(A,B,after).
s25. /* 0.111757 */ clear(B,after) :- destination(A), on(B,A,after).
s30. /* 0.077907 */ on(A,B,after); clear(B,before) :- on(A,B,before).
s34. /* 0.027631 */ clear(B,before) :- destination(A), on(B,A,after).
s35. /* 0.027631 */ clear(B,before) :- subject(A), on(A,B,before).
s41. /* 0.000000 */ :- destination(A), on(B,A,before).
s45. /* 0.000000 */ :- destination(B), clear(B,after).
s46. /* 0.000000 */ :- destination(B), table(B).
s49. /* 0.000000 */ :- subject(A), table(B), on(A,B,after).
s51. /* 0.000000 */ :- subject(A), on(A,B,after), clear(B,after).
Number of hypotheses explored: 2797
Average number of ground queries to the database: 151
Time: 2.9s
```

The rule s1 is a variant of rule 1 above (that is also found as rule s2), specific to stacking: a block that is not on the table does not move. The rule s5 characterises the table as the place that becomes clear. The rule s10 is a generalisation of the rule 21 we have already seen with the six examples, stating that nothing becomes clear in a stacking operation. The rule s11 states that there is nothing new on the table. The rule s12 is a variant of s11 since only the table can have something on it and be clear at the same time (rule 6 aka s13). The rule s21 is a pre-condition specific to the stacking operation: the subject is on the table before. The rules s23, s24, s25, s34 are variants of known rules: the subject is on the table beforehand, on the

destination afterwards, and the subject is clear beforehand and afterwards. The remaining rules deal with the destination (s41, s45, s46, s51) or with the table (s35, s46, s49).

In a third experiment we ran Tertius on the two unstacking examples, again searching for satisfied clauses only. Below we show only new rules among the 62 found. Moreover, since the first two rules establish an equivalence between table(B) and destination(B), many rules come in two forms, one with table and one with destination; we only show the rules concerning the table.

```
tertius -sat -c 0 -avg 3/2 unstack
u1. /* 0.461982 */ destination(B) :- table(B).
u2. /* 0.461982 */ table(B) :- destination(B).
u3. /* 0.326589 */ table(A) :- on(B,A,after), clear(A,before).
u14. /* 0.241720 */ on(A,B,after) :- on(A,B,before), clear(B,before).
u19. /* 0.217890 */ clear(B,after) :- clear(B,before).
u32. /* 0.066040 */ on(B,A,after); clear(B,after) :- table(A).
u34. /* 0.066040 */ on(B,A,before); clear(B,after) :- table(A).
u39. /* 0.060994 */ on(A,B,before); clear(B,after) :- on(A,B,after).
u40. /* 0.060994 */ table(A); clear(B,after) :- on(B,A,after).
u42. /* 0.060994 */ table(A); clear(B,after) :- on(B,A,before).
u44. /* 0.037557 */ clear(B,before); clear(A,after) :- on(A,B,after).
u45. /* 0.037557 */ clear(B,before); clear(A,after) :- on(A,B,before).
u47. /* 0.022222 */ clear(B,after) :- subject(A), on(A,B,after).
u60. /* 0.000000 */ :- subject(A), on(A,B,before), clear(B,before).
Number of hypotheses explored: 2797
Average number of ground queries to the database: 151
Time: 3.0s
```

The first two rules u1 and u2 ensure the destination is the table. Several rules concern the frame-problem, the subject, or the destination, as in the previous two experiments. However, some rules are worth emphasising. The rules u40 and u42 state that if a block is not on the table, either beforehand or afterwards, then it is clear afterwards. It means that, after unstacking, the stacks cannot be higher than 2 blocks. It is only coincidental in this case and these properties would not hold with more unstacking examples. However, it shows that it is possible to represent concepts such as “not more than two blocks high” in our framework.

Finally, we ran Tertius on the two transfer examples only, and looking for satisfied clauses only. It produced 49 rules, all of which had been found in one of the previous three experiments. This is not surprising since a transfer can be seen as a sequence of unstacking then stacking. We omit the details.

5.3. Functional dependencies

Tertius can be used to learn functional dependencies from a database relation. To illustrate this, Tertius was run on the following data from Flach (1993):

```
train(utrecht,8,8,den-bosch).        train(utrecht,9,8,den-bosch).
train(tilburg,8,10,tilburg).         train(tilburg,9,10,tilburg).
train(maastricht,8,10,weert).        train(maastricht,9,10,weert).
train(utrecht,8,13,eindhoven-bkln).  train(utrecht,9,13,eindhoven-bkln).
train(tilburg,8,17,eindhoven-bkln).  train(tilburg,9,17,eindhoven-bkln).
train(utrecht,8,25,den-bosch).       train(utrecht,9,25,den-bosch).
train(utrecht,8,31,utrecht).
train(utrecht,8,43,eindhoven-bkln).  train(utrecht,9,43,eindhoven-bkln).
train(tilburg,8,47,eindhoven-bkln).  train(tilburg,9,47,eindhoven-bkln).
```

The four arguments of `train` mean, respectively: the train's direction, the departure time in hours and minutes, and the first stop.

The rules found by `Tertius` are the following:

```
tertius 3/6 index
/* 0.143461 0.000000 */ equaldir(G,E) :- train(G,F,C,D), train(E,F,C,D).
/* 0.141036 0.000000 */ equaldir(A,E) :- train(A,B,C,D), train(E,F,C,D).
/* 0.119142 0.000000 */ equalfirst(D,G) :- train(A,E,C,D), train(A,E,C,G).
/* 0.117170 0.000000 */ equalfirst(F,D) :- train(A,B,C,D), train(A,E,C,F).
/* 0.068466 0.006250 */ equalmin(G,F) :- train(A,E,G,D), train(A,E,F,D).
/* 0.066730 0.006250 */ equalmin(C,F) :- train(A,B,C,D), train(A,E,F,D).
/* 0.055864 0.001111 */ equalfirst(F,D) :- train(A,B,C,D), train(E,B,C,F).
/* 0.047506 0.001111 */ equaldir(A,E) :- train(A,B,C,D), train(E,B,C,F).
/* 0.036298 0.002778 */ equaldir(A,E) :- train(A,B,C,D), train(E,B,F,D).
/* 0.026429 0.003333 */ equalfirst(F,D) :- train(A,B,C,D), train(A,B,E,F).
Number of hypotheses explored: 1685
Average number of ground queries to the database: 4293
Time: 1min54s
```

The second rule expresses the functional dependency "minutes and first stop determine direction" and the fourth rule states "direction and minutes determine first stop", which are both valid on the data. These are the same rules as found by `INDEX` (Flach, 1993) and `Claudien` (De Raedt & Dehaspe, 1997). The first and third rule are not found by these systems, because they are refinements of the second and fourth rule, respectively. Here they are kept by `Tertius` because they have a higher degree of confirmation than their ancestor. If this were not the case, `Tertius`' final subsumption test would have removed them as well.

The remaining 6 rules have small but non-zero $n_{\bar{H}B}$, and appear to be 'almost satisfied' functional dependencies. For instance, the first non-satisfied rule says "direction, hour and first stop determines minutes", i.e., trains depart once an hour at most. This dependency has 12 contradicting pairs of tuples, or approximately 4.2% of the total $17 \times 17 = 289$ pairs of tuples. However, the sample consists of all pairs of *possible* tuples that can be constructed from the types, which is 1920 in this case. Here we thus see the need for the sample of grounding substitutions to be constructed in a more sensible way. On the other hand, we have shown that the `Tertius` setting is general enough to allow database dependency inference, and that the confirmation measure can be applied in this context as well.

5.4. Multiple predicate learning

Tertius does not focus on a single predicate unless it is constrained to do so by a language bias. By default, Tertius is able to learn along several dimensions, which is called *multiple predicate learning* when a first-order representation is used. We took the family dataset from De Raedt and Lavrač (1996), and ran Tertius to find all clauses (possibly indefinite) with up to 3 literals and 3 variables. We also restricted output to satisfied clauses only, and set a small positive threshold on the confirmation. The learned clauses were as follows:

```
tertius -sat -c 0.0001 3/3 mpl
/* 0.314447 0.000000 */ father(D,B); mother(D,B) :- parent(D,B).
/* 0.293511 0.000000 */ ancestor(C,B) :- parent(C,B).
/* 0.210408 0.000000 */ male(D); mother(D,B) :- parent(D,B).
/* 0.207553 0.000000 */ father(C,D) :- parent(C,D), male(C).
/* 0.207553 0.000000 */ mother(C,D) :- parent(C,D), female(C).
/* 0.203596 0.000000 */ parent(C,B) :- father(C,B).
/* 0.203596 0.000000 */ parent(C,B) :- mother(C,B).
/* 0.196670 0.000000 */ female(D); father(D,B) :- parent(D,B).
/* 0.191114 0.000000 */ ancestor(C,B) :- father(C,B).
/* 0.191114 0.000000 */ ancestor(C,B) :- mother(C,B).
/* 0.145001 0.000000 */ male(B) :- father(B,C).
/* 0.136544 0.000000 */ female(B) :- mother(B,C).
/* 0.083298 0.000000 */ ancestor(A,B) :- ancestor(A,C), ancestor(C,B).
/* 0.069193 0.000000 */ ancestor(A,B) :- ancestor(C,B), parent(A,C).
/* 0.069193 0.000000 */ ancestor(A,B) :- ancestor(A,C), parent(C,B).
/* 0.059986 0.000000 */ ancestor(A,B) :- ancestor(A,C), mother(C,B).
/* 0.047955 0.000000 */ ancestor(A,B) :- ancestor(C,B), mother(A,C).
/* 0.047955 0.000000 */ ancestor(A,B) :- ancestor(C,B), father(A,C).
/* 0.043326 0.000000 */ ancestor(A,B) :- parent(A,C), mother(C,B).
/* 0.036373 0.000000 */ ancestor(A,B) :- parent(C,B), mother(A,C).
/* 0.036373 0.000000 */ ancestor(A,B) :- parent(C,B), father(A,C).
/* 0.032409 0.000000 */ ancestor(A,B) :- ancestor(A,C), father(C,B).
/* 0.030252 0.000000 */ ancestor(A,B) :- father(A,C), mother(C,B).
/* 0.027945 0.000000 */ ancestor(A,B) :- parent(A,C), father(C,B).
/* 0.024071 0.000000 */ female(C) :- ancestor(A,B), father(B,C).
/* 0.020779 0.000000 */ female(C) :- parent(A,B), father(B,C).
/* 0.019600 0.000000 */ ancestor(A,B) :- father(C,B), mother(A,C).
/* 0.014604 0.000000 */ female(C) :- father(B,C), mother(D,B).
Number of hypotheses explored: 8276
Average number of ground queries to the database: 12105
Time: 28min35.5s
```

Notice the highly confirmed disjunctive clause. Also, notice the clauses for ancestor, which include singly and doubly recursive clauses. While such clauses typically pose problems

to supervised ILP systems, there is no particular difficulty involved in learning recursive clauses in our approach, other than the fact that the search space increases if literals can occur more than once. All the clauses learned by the `MPL` and `Claudien` algorithms in De Raedt and Lavrač (1996) are also learned. In addition, at the bottom `Tertius` lists three satisfied but meaningless clauses that happen to be true in this particular dataset.

`Tertius` took almost half an hour of CPU-time to find these clauses. This is explained by the weak declarative bias imposed by this domain: it is not individual-based, all variables are of the same type, there is no single target predicate, and disjunctive clauses are allowed. Therefore the number of possible hypotheses and the number of possible instantiations are higher than those of previous experiments, and the learning time is proportionally higher. Furthermore, in cases where we ask for all clauses with non-zero confirmation, there will be no best-first pruning as the system returns all solutions and not only the best ones.

6. Related work

In this section we discuss some work in machine learning and knowledge discovery that is closely related to our work. In Section 6.1 we indicate how our confirmation measure differs from other measures found in the data mining literature. In Section 6.2 we compare our `Tertius` system with other published algorithms and systems.

6.1. Related evaluation measures

Pearson's X^2 statistic is intended for hypothesis testing—that is, it is used to asses the confidence with which the null hypothesis of statistical independence between variables A and B can be rejected or accepted. The use of X^2 or its normalised version Φ^2 to assess the *strength* of the dependency between A and B is considered somewhat dubious by many statisticians. On the other hand, since there is no generally agreed upon measure of dependency, Φ^2 or a variant thereof does not seem less acceptable than many of the other measures proposed over the years (Goodman & Kruskal, 1979).

The main problem with using Φ^2 as a measure of dependency is that it is completely undirected: swapping two columns or two rows in the contingency table does not affect the value of Φ^2. This is partly alleviated by e.g., Yule's coefficient $\frac{n_{11}n_{22}-n_{12}n_{21}}{n_{11}n_{22}+n_{12}n_{21}}$, which is -1 in case of a zero on one diagonal, and $+1$ in case of a zero on the other. However, this measure is still partly undirected, since if we swap the two columns **and** the two rows of a two-by-two contingency table, the value of Yule's coefficient remains the same. Thus, it violates principle P4 in Section 3.1.

Much of the work in Section 3 was devoted to devising a measure of dependence fit for assessing directed rules such as implications. To the best of our knowledge, such a directed confirmation measure has not been published in the machine learning and knowledge discovery literature before. This is perhaps not surprising if one considers that in the majority of approaches one searches for rules with a fixed head. In such a case one effectively evaluates the usefulness or interestingness of a body with respect to a given head.

Not only is this true in classification, it also holds for knowledge discovery tasks such as subgroup discovery (Klösgen, 1996). In Klösgen's setting, a concept C indicates an

Table 5. A comparison of evaluation measures for rule discovery.

(1)	$g(p - p_0)$	$\pi_{\bar{H}B} - p_{\bar{H}B}$
(2)	$\sqrt{g}(p - p_0)$	$\frac{\pi_{\bar{H}B} - p_{\bar{H}B}}{\sqrt{p_B}}$
(3)	$\sqrt{\frac{g}{1-g}}(p - p_0)$	$\frac{\pi_{\bar{H}B} - p_{\bar{H}B}}{\sqrt{p_B(1-p_B)}}$
(4)	$\frac{g}{\sqrt{g(1-p_0)-g(1-p_0)}}(p - p_0)$	$\frac{\pi_{\bar{H}B} - p_{\bar{H}B}}{\sqrt{\pi_{\bar{H}B}-\pi_{\bar{H}B}}}$

interesting subgroup with respect to a fixed target group T, if the distribution of T's and non-T's among the individuals satisfying C is significantly different from the distribution over the whole population P. In our more general framework of rule discovery, the target T corresponds to the head H of a clause, and the concept C to its body B. We can compare rules with different heads and bodies, and thus compare the interest of different subgroups associated with different targets.

We compare our confirmation function with the main ones considered by Klösgen (Table 5). The first three evaluation measures are from (Klösgen, 1996),[8] the last one is our confirmation function $\Phi_{\bar{H}B}$. The left column uses Klösgen's notation: $p = p(T \mid C)$, $p_0 = p(T)$, and $g = p(C)$. This can be translated to our notation using $p_0 = p_H$, $g = p_B$, and $pg = p_{HB}$ (right column).

Notice that each of these measures includes the quantity $g(p - p_0) = \pi_{\bar{H}B} - p_{\bar{H}B}$, which is called novelty in this paper. As noted in Section 3.1, this quantity displays unwanted symmetry in that $H \leftarrow B$ and $B \leftarrow H$ always get the same evaluation. The other three measures all break this symmetry in one way or another, by dividing novelty by some denominator D. Notice that (3) introduces another symmetry: if $H \leftarrow B$ gets evaluation c, then $H \leftarrow \neg B$ gets evaluation $-c$. While such a symmetry could make sense in the context of subgroup discovery, where the target T is fixed, it is not obvious why it should be valid in the wider context of rule discovery.[9]

The main difference, then, between our confirmation function (4) and measures (2-3) in Table 5 is that for the latter D is based only on $g = p_B$, while in our case also $p_0 = p_H$ is taken into account. This supports our view that confirmation as defined in this paper is better suited for comparing rules with different heads as well as bodies.

6.2. Related systems

Our approach has initially been motivated by the work of Oates and Cohen on finding structure in data (Oates & Cohen, 1996). Like Tertius, their MSDD system employs an A* algorithm to find the k strongest rules that describe dependencies in the data, where the strength of a rule is defined in terms of X^2. Tertius upgrades their system to using a first-order representation, and also improves their heuristic in order to deal with directed implications.

As a system, Tertius is most closely related to the ILP clausal discovery system Claudien (De Raedt & Dehaspe, 1997). Claudien finds regularities expressed in

first-order logic that hold for the data. Essentially, it performs a top-down refinement search to find all most general non-contradicted formulae. `Tertius` extends `Claudien` by employing a powerful confirmation heuristic, that allows us to rank the induced formulae. Furthermore, by restricting attention to the *k* best confirmations an optimistic estimate of confirmation can be used as a search heuristic. Another difference between `Tertius` and `Claudien` is that the latter employs a grammar-based declarative bias formalism to restrict the search space, while `Tertius` relies on more semantic notions such as individual-based and structural representations.

Like `Claudien`, `Tertius` is a general-purpose system that is not restricted to particular discovery tasks. It is also related to a number of specialised knowledge discovery algorithms, although such algorithms tend to be more efficient because they are tailored towards that specific task. Database dependency discovery is such a restricted form of clausal discovery. The `INDEX` system (Flach, 1993) provided a straightforward top-down refinement approach in a deductive database setting. The `fdep` system (Flach & Savnik, 1999) offers more sophisticated and efficient bottom-up algorithms. `Tertius` can in principle be applied to database dependency discovery tasks, although it will not have the efficiency of the special-purpose `fdep` system. The confirmation heuristic, however, directly carries over to database dependency discovery.

Association rules were originally defined over single relations, but have recently been upgraded to the multi-relational first-order case (Dehaspe & Toivonen, 1999). Association rules have a different semantics from the clauses considered in this paper, as they have a conjunctive head rather than a disjunctive one. Also, association rule algorithms crucially depend on the pruning enabled by the frequency threshold. Although `Tertius` does allow the user to set a frequency threshold, an A-PRIORI-like algorithm has not yet been implemented.

Subgroup discovery (Klösgen, 1996) has also recently been extended to the multi-relational case (Wrobel, 1997). As indicated in the previous section, the goal here is to find subgroups C of a population that display a significantly different distribution with respect to a target property T. `Tertius` is particularly well-suited to this task, employing a similar novelty-based heuristic as discussed above, but allowing a more general hypothesis language. Moreover, `Tertius` allows to discover and compare subgroups related to different targets. Notice that subgroup discovery differs from rule discovery in that $T \leftarrow C$ is, in general, not a satisfied rule. `Tertius`' ability to perform non-categorical confirmatory induction enables it to find such highly confirmed but non-satisfied rules.

Our work is furthermore related to the multiple predicate learning task in ILP (De Raedt & Lavrač, 1996). The approach we take here is similar to the `Claudien` approach: instead of searching for definitions of the predicates to be learned, we search for dependencies between them. This avoids the traditional problems with multiple predicate learning in ILP because there are no proofs involved, and so mutual recursion is not a problem. Finally, there is the task of learning mixed theories of predicate definitions and integrity constraints. Previous work has concentrated on learning them separately by two different learners (Dimopoulos, Dzeroski, & Kakas, 1997). `Tertius` is able to learn such mixed theories in one go.

7. Conclusions

In this paper we described the Tertius approach to first-order unsupervised learning. First-order logic offers the ability to deal with structured, multi-relational knowledge. Tertius is a general-purpose first-order unsupervised rule learner. Compared to other clausal discovery systems such as Claudien, our main contribution is the confirmation measure we propose. This measure uses concepts from categorical data analysis and is thus well-founded in statistics. It also allows the inclusion of first-order background knowledge, and has the potential to provide a real integration between logic and statistics.

Tertius is a fully-fledged and powerful rule discovery system. It takes an approach to declarative bias which is more semantically motivated, distinguishing individual-based and structural domains, among others. The individual-based representation is very intuitive and provides a clear link with propositional attribute-value representations.[10] We have compared our approach with other rule discovery systems on a few benchmark tasks, such as database dependency discovery and multiple predicate learning. We have also shown that Tertius can be applied to supervised learning tasks such as concept learning, where its rich and flexible representation formalism leads to meaningful results. This supports our view that supervised and unsupervised learning represent two distinct but related points on a dimension, rather than completely separate worlds.

Future work includes improvement of Tertius' efficiency and further declarative bias in order to search deeper in the hypothesis space. We are also looking at ways to reduce the overhead of counting in domains with many grounding substitutions. As indicated before, further theoretical analysis of the multiway confirmation measure is needed in order to guarantee the same kind of properties we proved in this paper for the two-way measure. Finally, although induction of novel and interesting rules is often a goal in itself, we will also study post-processing approaches where the learned rules are, for instances, separated into predicate definitions and integrity constraints for use by a reasoner which is able to deal with such mixed theories.

Acknowledgments

Thanks are due to Torbjørn Dahl for implementing the Prolog interface. We would like to thank two anonymous referees for their comments on an earlier draft which helped to improve the quality of the paper, and Doug Fisher for suggesting the blocks world experiments. Part of this work was supported by Esprit Long Term Research Project 20237 *Inductive Logic Programming 2*.

Notes

1. Strictly speaking these are probability *estimates*, as they are taken over the sample and not over the population. For simplicity we will not reflect this in our notation.
2. We do have an extreme form of contraposition: $\forall(H \leftarrow B)$ and $\forall(\neg B \leftarrow \neg H)$ always get the same confirmation.

3. Note that in some ILP systems such as `Progol` (Muggleton, 1995) all the information relative to an example, except its class, is considered background knowledge (e.g., molecule `d101` has a carbon atom `d101_1` with type 22 and charge `-0.121`). In `Tertius`, all information specific to an example is considered foreground knowledge; the background knowledge consists of general properties shared among the examples, such as the rule of symmetry above, or general ground facts stating that, e.g., in a blocks world the constant `t` is a table appearing in all examples (Section 5.2).
4. In a *strongly* typed language, each type has an associated set of operations. See (Flach, Giraud-Carrier & Lloyd, 1998) for a discussion of the utility of strong typing for concept learning.
5. This approach to discretisation was suggested by Stephen Muggleton.
6. The original `Progol` rules used a predicate `atm(Mol,Atom,Element,Type,Charge)`; we adapted them to our individual-based representation, using a structural predicate `atm(Mol,Atom)` and binary properties `atomel(Atom,El)`, `atomty(Atom,Type)` and `atomch(Atom,Charge)`.
7. This information is a typical example of ground background knowledge. In the current implementation of `Tertius`, it has to be either put in a separate Prolog file containing all the background knowledge, or included in each example. We chose the second alternative that allowed us to avoid slower calls to Prolog by storing more facts.
8. The third measure occurs as $\frac{g}{1-g}(p - p_0)^2$ in Klösgen (1996), but this ignores the sign of $(p - p_0)$.
9. Notice that the measure $\frac{\pi_{\bar{H}B} - p_{\bar{H}B}}{\sqrt{\pi_{\bar{H}B}(1-\pi_{\bar{H}B})}}$ would achieve a similar symmetry.
10. This link has already proven useful, as it has given rise to a descendant system called `1BC`, which is a first-order naive Bayesian classifier (Flach & Lachiche, 1999a).

References

Badea, L. & Stanciu, M. (1999). Refinement operators can be (weakly) perfect. In S. Džeroski and P. Flach (Eds.), *Proceedings of the 9th International Workshop on Inductive Logic Programming*, volume 1634 of Lecture Notes in Artificial Intelligence, (pp. 21–32). Springer-Verlag.

Dahl, T. S. (1999). Background knowledge in the Tertius first-order knowledge discovery tool. Technical Report CSTR-99-006, Department of Computer Science, University of Bristol, March 1999.

De Raedt, L. & Lavrač, N. (1996). Multiple predicate learning in two inductive logic programming settings. *Journal on Pure and Applied Logic, 4*(2), 227–254.

De Raedt, L. (1997). Logical settings for concept learning. *Artificial Intelligence 95*(1), 187–201.

De Raedt, L. & Dehaspe, L. (1997). Clausal discovery. *Machine Learning, 26*(2/3), 99–146.

Dehaspe L. & De Raedt, L. (1997). Mining association rules in multiple relations. In S. Džeroski & N. Lavrač (Eds.), In *Proceedings of the 7th International Workshop on Inductive Logic Programming*, volume 1297 of Lecture Notes in Artificial Intelligence (pp. 125–132). Springer-Verlag.

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

Dimopoulos, Y., Dzeroski, S., & Kakas, A. C. (1997). Integrating explanatory and descriptive learning in ILP. In M. E. Pollack (Ed.), *Proceedings of the 15th International Joint Conference on Artificial Intelligence* (pp. 900–906), Morgan Kaufmann.

Flach, P. (1993). Predicate invention in inductive data engineering. In Brazdil, P. (Ed.), *Proceedings of the 6th European Conference on Machine Learning*, volume 667 of Lecture Notes in Artificial Intelligence (pp. 83–94). Springer-Verlag.

Flach, P., Giraud-Carrier, C., & Lloyd, J. (1998). Strongly typed inductive concept learning. In Page, D. (Ed.), *Proceedings of the 8th International Conference on Inductive Logic Programming*, volume 1446 of Lecture Notes in Artificial Intelligence (pp. 185–194). Springer-Verlag.

Flach, P. & Lachiche, N. (1999a). 1BC: A first-order Bayesian classifier. In Džeroski, S. & Flach P. (Eds.), *Proceedings of the 9th International Workshop on Inductive Logic Programming* (pp. 92–103). volume 1634 of Lecture Notes in Artificial Intelligence, Springer-Verlag.

Flach, P. & Savnik, I. (1999). Database dependency discovery: a machine learning approach. *AI Communications*, *12*(3), 139–160.

Flach, P. A. & Lachiche, N. (1999b). The Tertius system. http://www.cs.bris.ac.uk/Research/MachineLearning/Tertius/.

Goodman, L. A. & Kruskal, W. H. (1979). *Measures of association for cross classifications*. Springer-Verlag.

Klösgen, W. (1996). Explora: A multipattern and multistrategy discovery assistant. In Fayyad, U., Piatetsky-Shapiro, G., Smyth, P., & Uthurusamy, R. (Eds.), *Advances in Knowledge Discovery and Data Mining* (pp. 249–271). AAAI Press.

Lavrač, N., Flach, P., & Zupan, B. (1999). Rule evaluation measures: A unifying view. In Džeroski, S. & Flach, P. (Eds.), *Proceedings of the 9th International Workshop on Inductive Logic Programming*, volume 1634 of Lecture Notes in Artificial Intelligence (pp. 174–185). Springer-Verlag.

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

Oates, T. & Cohen, P. (1996). Searching for structure in multiple streams of data. In Saitta, L. (Ed.), *Proceedings of the 13th International Conference on Machine Learning* (pp. 346–354). Morgan Kaufmann.

Piatetsky-Shapiro, G. (1991). Discovery, analysis and presentation of strong rules. In Piatetsky-Shapiro, G. and Frawley, W. (Eds.), *Knowledge Discovery in Databases* (pp. 229–249). AAAI Press.

Plotkin, G. (1970). A note on inductive generalization. In *Machine Intelligence* (vol. 5, pp. 153–163). Edinburgh University Press.

Plotkin, G. (1971). A further note on inductive generalization. In *Machine Intelligence* (vol. 6, pp. 101–124). Edinburgh University Press.

Rymon, R. (1992). Search through systematic set enumeration. In *Proceedings of the 3d International Conference on Knowledge Representation and Reasoning* (pp. 539–550). Morgan Kaufmann.

Srinivasan, A., King, R. D., & Muggleton, S. H. (1996). The role of background knowledge: Using a problem from chemistry to examine the performance of an ILP program. Unpublished manuscript, available from the first author.

Srinivasan, A., Muggleton, S., King, R., & Sternberg, M. (1994). Mutagenesis: ILP experiments in a non-determinate biological domain. In Wrobel, S. (Ed.), *Proceedings of the 4th International Workshop on Inductive Logic Programming* (vol. 237 of GMD-Studien, pp. 217–232). Gesellschaft für Mathematik und Datenverarbeitung MBH.

Vere, S. A. (1978). Inductive learning of relational productions. In Waterman, D. & Hayes-Roth, F. (Eds.), *Pattern-Directed Inference Systems* (pp. 281–295). Academic Press.

Wickens, T. (1989). *Multiway contingency tables analysis for the social sciences*. Lawrence Erlbaum.

Wrobel, S. (1997). An algorithm for multi-relational discovery of subgroups. In Komorowski, J. & Zytkow, J. (Eds.), *Proceedings of the 1st European Symposium on Principles of Data Mining and Knowledge Discovery*. Springer-Verlag.

Received February 15, 1999
Revised February 15, 1999
Accepted July 28, 1999
Final manuscript September 18, 1999