# Algorithms for Efficiently and Effectively Using Background Knowledge in `Tertius`

Peter Flach[1] Valentina Maraldi[2] Fabrizio Riguzzi[3]
[1]Department of computer science, University of Bristol,
Woodland Road, Bristol BS8 1UB, United Kingdom,
peter.flach@bristol.ac.uk
[2]DEIS, Università di Bologna,
Viale Risorgimento 2, 40136 Bologna, Italy,
valentina.maraldi@studio.unibo.it
[3]Dipartimento di Ingegneria, Università di Ferrara,
Via Saragat 1, 44100 Ferrara, Italy,
friguzzi@ing.unife.it

## SOMMARIO/*ABSTRACT*

`Tertius` is an Inductive Logic Programming system that performs confirmatory induction, i.e., it looks for the $n$ clauses that have the highest value of a confirmation evaluation function. In this setting, background knowledge is very useful because it can improve the reliability of the evaluation function, assigning minimal confirmation to clauses that are implied by the background knowledge and increasing the confirmation of the remaining clauses. We propose the algorithms *Background1* and *Background2* that look for clauses in the background that imply the clause under evaluation by `Tertius`. Both are based on a simplified implication test that is correct with respect to $\theta$-subsumption but not complete. The implication test is not complete because we want to keep the run time inside acceptable bounds. We compare *Background1* with *Background2* on two datasets. The results show that *Background2* is more efficient than *Background1*. Moreover, we also present the algorithm *Preprocess* that infers new clauses from the background knowledge in order to exploit it as much as possible. The algorithm modifies the consequence finding algorithm proposed by Inoue by reducing its execution time while giving up completeness.


## 1 Introduction

Confirmatory induction is an important task of Knowledge Discovery in Databases. It consists in finding the $n$ rules of the form $H \leftarrow B$ most "confirmed" by the data, i.e., with the highest value of a confirmation measure. [6] proposes three axioms that a confirmation measure should obey. Various confirmation measures have been proposed. The simplest one is weighted relative accuracy. In general, a confirmation measure combines the novelty [3] of a rule, i.e. the unexpectedness of a rule, and its satisfaction [3], i.e. the fraction of expected counter-instances that are not observed.

[3] proposes a new axiom to be added to those of [6] and a new confirmation measure that satisfies all four axioms. It also proposes a learning system, `Tertius` for performing confirmatory induction with the new measure in the domain of Inductive Logic Programming (ILP in the following). In other words, the data can be represented using multiple relations and the rules are first-order logic clauses.

In order to compute its confirmation measure, `Tertius` needs to populate two contingency tables for each clause, one for observed values and one for expected values. When computing these tables, `Tertius` can exploit background knowledge in order to improve the reliability of the evaluation function, assigning minimal confirmation to clauses that are implied by the background knowledge (they have very low novelty) and increasing the confirmation of the remaining clauses.

In this paper we propose two algorithms, *Background1* and *Background2*, that allow the exploitation of the background knowledge by `Tertius`. They are both based on an implication test between two clauses that is simpler than $\theta$-subsumption. The simplification was obtained by giving up the completeness of the test (but not the correctness), thus obtaining a fast approximate implication algorithm that could be easily implemented in C and interfaced with the current system, also written in C.

Moreover, we extended `Tertius` also with the algorithm *Preprocess* that completes the background knowledge with clauses not present in it but implied by it. The added clauses can then be used during learning for improving the reliability of the confirmation measure. *Preprocess* modifies the consequence finding algorithm proposed by [4] in order to reduce the execution time while giving up completeness.

Algorithms *Background1* and *Background2* have been thoroughly tested on a number of datasets in order to check their correctness and speed. The experiments show that *Background2* has a better scalability than *Background1*.

Also *Preprocess* has been tested.

The paper is organized as follows. In section 2 we

present some preliminary notions regarding confirmatory induction, `Tertius` and the use of background knowledge in `Tertius`. In section 3 we illustrate our three algorithms: *Background1*, *Background2* and *Preprocess*. Section 4 describes the results of the various experiments that have been performed, while section 5 presents the conclusions and some directions for future works.

## 2 Tertius

### 2.1 Confirmatory Induction

The confirmatory induction task is a specific problem of Knowledge Discovery in Databases. In the domain of ILP it can be formalized in this way:
Given:

- a set of programs in first-order logic $P$,
- a set of ground facts, or evidence (E),
- a confirmation function $c(E, C)$ that, given a clause $C$ and the evidence $E$, returns a real number.

Find:

- the $n$ most confirmed clauses.

`Tertius` [3] is a top-down rule discovery system that solves the confirmatory induction task in the ILP domain. Specifically, `Tertius` finds rules of the form $\forall(H \leftarrow B)$ from the dataset, where $H$ and $B$ are formulas with some free variables and $\forall(\cdot)$ denotes the universal closure over the free variables. We will omit the quantifier in the following. $H$ is called the *head* of the rule and it is a disjunction of literals, while $B$ is called the *body* of the rule and it is a conjunction of literals. The most important aspect of the confirmatory induction task is the definition of the *confirmation* function that measures how much a rule is confirmed by the evidence. [6] proposes three axioms that a confirmation measure should obey. [3] adds to them a fourth axiom. The confirmation measure of `Tertius` satisfies all four axioms.

In order to express the confirmation axioms, we need two values: the *expected probability* and the *observed probability* of the counter-instances of the rule. We consider a sample of $N$ grounding substitutions $\theta$ of $H \leftarrow B$, and we build a contingency table as Table 1. Specifically, $n_{HB}$ denotes the number of grounding substitutions that satisfy both head and body of the rule; similarly, $n_{\overline{H}B}$ denotes the number of grounding substitutions that satisfy the body and falsify the head, also referred to as *counter-instances* of the rule. In Table 1 we have also the one-way marginals $n_i$, where $n_i$ denotes the number of grounding substitution that satisfy $i$. If we assume the null hypothesis of complete independence of $H$ and $B$, the expected frequency of counter-instances is given by:

$$\mu_{\overline{H}B} = \frac{n_{\overline{H}} n_B}{N}$$

We can build a contingency table where the observed frequencies are substituted by the expected frequencies. We can now define the observed and expected probabilities of counter-instances:

$$p_{\overline{H}B} = \frac{n_{\overline{H}B}}{N}$$

$$\pi_{\overline{H}B} = \frac{\mu_{\overline{H}B}}{N}$$

and the marginal probabilities

$$p_i = \frac{n_i}{N}$$

The three axioms that a confirmation measure should satisfy according to [6] are

A1 the confirmation should be 0 if $p_{\overline{H}B} = \pi_{\overline{H}B}$

A2 the confirmation should monotonically decrease with $p_{\overline{H}B}$, all other parameters remaining the same

A3 the confirmation should monotonically increase with $p_{\overline{H}}$ (or $p_B$), all other parameters remaining the same.

The fourth axiom added by [3] is

A4 two rules $H \leftarrow B$ and $B \leftarrow H$ should get equal confirmation only if they have the same number of counter-instances

In order to define `Tertius`'s confirmation measure (simply confirmation from now on), we also define the value $\Phi^2$ that is a measure of the strength of the dependency between $H$ and $B$

$$\Phi^2 = \frac{(n_{HB} n_{\overline{H}\overline{B}} - n_{H\overline{B}} n_{\overline{H}B})^2}{n_H n_{\overline{H}} n_B n_{\overline{B}}}$$

$\Phi^2$ is obtained from Pearson's statistics $X^2$ by dividing for $N$. $\Phi^2$ ranges from 0 (total independence) to 1 (total dependence). The confirmation is obtained by finding the minimum $\Phi^2_{\overline{H}B}$ of $\Phi^2$ by keeping constant only the observed and expected frequency of counter-instances and the population size. $\Phi^2_{\overline{H}B}$ is given by the following formula:

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

When $p_{\overline{H}B} = 0$ and $\pi_{\overline{H}B} = 0$ we assume $\Phi_{\overline{H}B} = 0$. $\Phi_{\overline{H}B}$, being the square root of $\Phi^2_{\overline{H}B}$, ranges from -1 to +1. $\Phi_{\overline{H}B} = 0$ iff $p_{\overline{H}B} = \pi_{\overline{H}B}$ (A1). $\Phi_{\overline{H}B} = 1$ iff $p_{\overline{H}B} = 0$ and $p_{\overline{H}} = p_B = 0.5$ and $\Phi_{\overline{H}B} = -1$ iff $p_{\overline{H}B} = \sqrt{\pi_{\overline{H}B}}$. In general $\Phi_{\overline{H}B}$ increases with decreasing number of observed counter-instances and increasing number of expected counter-instances (A2-A3). Finally, A4 is satisfied since $\Phi_{\overline{H}B}$ is not symmetric in $H$ and $B$.

Table 1: A contingency table.

| Head | Body $B$ | Body $\overline{B}$ | |
|---|---|---|---|
| $H$ | $n_{HB}$ | $n_{H\overline{B}}$ | $n_H$ |
| $\overline{H}$ | $n_{\overline{H}B}$ | $n_{\overline{H}\overline{B}}$ | $n_{\overline{H}}$ |
| | $n_B$ | $n_{\overline{B}}$ | $N$ |

Table 2: Multi-way Contingency table.

| Head | Body1,Body2 $B_1/n_{B_1}$ $B_2$ | $B_1/n_{B_1}$ $\overline{B_2}$ | $\overline{B_1}/n_{\overline{B_1}}$ $B_2$ | $\overline{B_1}/n_{\overline{B_1}}$ $\overline{B_2}$ | |
|---|---|---|---|---|---|
| $H$ | $n_{HB_1B_2}$ | $n_{HB_1\overline{B_2}}$ | $n_{H\overline{B_1}B_2}$ | $n_{H\overline{B_1B_2}}$ | $n_H$ |
| $\overline{H}$ | $n_{\overline{H}B_1B_2}$ | $n_{\overline{H}B_1\overline{B_2}}$ | $n_{\overline{H}\overline{B_1}B_2}$ | $n_{\overline{H}\overline{B_1B_2}}$ | $n_{\overline{H}}$ |
| | $n_{B_2}$ | $n_{\overline{B_2}}$ | | | $N$ |

Table 3: Multi-way contingency table over the dataset.

| Head | Body $B_1/6$ $B_2$ | $B_1/6$ $\overline{B_2}$ | $\overline{B_1}/14$ $B_2$ | $\overline{B_1}/14$ $\overline{B_2}$ | |
|---|---|---|---|---|---|
| $H$ | 2 | 0 | 4 | 3 | 9 |
| $\overline{H}$ | 4 | 0 | 3 | 4 | 11 |
| | 13 | 7 | | | 20 |

Table 4: Contingency table at step 1.

| Head | Body $B_1/6$ $B_2$ | $B_1/6$ $\overline{B_2}$ | $\overline{B_1}/14$ $B_2$ | $\overline{B_1}/14$ $\overline{B_2}$ | |
|---|---|---|---|---|---|
| $H$ | 1 | 1 | 1 | 1 | 9 |
| $\overline{H}$ | 1 | 1 | 1 | 1 | 11 |
| | 13 | 7 | | | 20 |

## 2.2 Background Knowledge in Tertius

Section 2.1 describes how `Tertius` calculates the confirmation of a rule. In this measure, all the literals of the body of a rule are considered together by building a contingency table with two dimensions. In order to incorporate background knowledge, we consider all the literals on their own and not as a unique element.

Consider for example the rule $H \leftarrow B_1, B_2$. For this rule we have to build the multiway contingency table that is shown in table 2.

As stated in Section 2.1, for the evaluation of confirmation we need two values, the observed frequency and the expected frequency of counter-instances. The observed frequency of counter-instances is obtained from the multiway contingency table: it is represented by the quantity $n_{\overline{H}B_1B_2}$.

The expected frequency of counter-instance is the frequency of counter-instances that we would get in the hypothesis of complete independence among the literals, i.e., in the case that $H$, $B_1$ and $B_2$ are independent of each other.

To compute the expected frequency we use an iterative algorithm: we build a contingency table with the correct dimensions and we initialize each cell with the value 1. Then we take the one-way marginals from the observed contingency table and then iteratively change the cell values in order to fit these marginals. At the end of the fitting phase, in the cell corresponding to $\overline{H}, B_1, B_2$ we have the expected frequency that we were looking for. The following example shows the behavior of the algorithm.

**Example 2.1.** *Suppose we want to evaluate the rule $H \leftarrow B_1 \wedge B_2$. and suppose that Table 3 is the table of the observed frequencies built over the dataset. The first step of the algorithm is to build contingency Table 4 in which all the cells are assigned value 1. The next step of the algorithm is to adjust the value of each cells in order to fit the one-way marginals that are taken from contingency Table 3. The first marginal that we want to fit is the one on $H$: to this purpose, we multiply the first row for $9/4 = 2.25$ and the second row for $11/4 = 2.75$ obtaining contingency Table 5. Then, we consider the literal $B_1$ and, after fitting its one-way marginal we obtain Table 6. The last step is the one that fits the marginal on $B_2$ obtaining Table 7. Now, if we cycle again over the dimensions, we obtain no change in the table: we have reached a fix point. So it is enough to fit the marginals for each dimension once to obtain the expected frequencies. The expected frequency of the counter-instances for the rule that we want to evaluate is now present in the cell corresponding to $\overline{H}, B_1, B_2$.*

We want to exploit the background knowledge in order to improve the accuracy of the confirmation of the rules. For example, if we have the rule $\forall(B_2 \leftarrow B_1)$ in the background knowledge, we know that there will be no substitutions for which $B_1$ is true and $B_2$ false. If we trust the background knowledge more than the evidence, we can set

Table 5: Contingency table after fitting the one-way marginal on $H$.

| Head | Body $B_1/6$ $B_2$ | $B_1/6$ $\overline{B_2}$ | $\overline{B_1}/14$ $B_2$ | $\overline{B_1}/14$ $\overline{B_2}$ | |
|---|---|---|---|---|---|
| $H$ | 2.25 | 2.25 | 2.25 | 2.25 | 9 |
| $\overline{H}$ | 2.75 | 2.75 | 2.75 | 2.75 | 11 |
| | 13 | 7 | | | 20 |

Table 6: Contingency table after fitting the marginal on $B_1$.

| | Body | | | | |
|---|---|---|---|---|---|
| | $B_1/6$ | | $\overline{B_1}/14$ | | |
| Head | $B_2$ | $\overline{B_2}$ | $B_2$ | $\overline{B_2}$ | |
| $H$ | 1.35 | 1.35 | 3.15 | 3.15 | 9 |
| $\overline{H}$ | 1.65 | 1.65 | 3.85 | 3.85 | 11 |
| | 13 | 7 | | | 20 |

Table 7: Contingency table after fitting the marginal on $B_2$.

| | Body | | | | |
|---|---|---|---|---|---|
| | $B_1/6$ | | $\overline{B_1}/14$ | | |
| Head | $B_2$ | $\overline{B_2}$ | $B_2$ | $\overline{B_2}$ | |
| $H$ | 1.755 | 0.945 | 4.095 | 2.205 | 9 |
| $\overline{H}$ | 2.145 | 1.155 | 5.005 | 2.695 | 11 |
| | 13 | 7 | | | 20 |

to 0 the cells of the observed contingency table associated to $B_1\overline{B_2}$. This is equivalent to considering the instances that are not consistent with the background knowledge as noise in the data. Moreover, we can set to 0 also the cells of the expected contingency table associated to $B_1\overline{B_2}$, since we do not expect any substitution to satisfy $B_1$ and $\overline{B_2}$. By setting some of the cells of the expected contingency table to 0, we change the values of the expected probabilities: those of the counter-instances of clauses implied by the background go to 0 while those of the counter-instances of the other clauses built with the same literals increase, in order to fit the marginals. This leads to a confirmation of 0 for the clauses implied by the background knowledge (when $\pi_{\overline{H}B} = 0$ and $p_{\overline{H}B} = 0$ the confirmation is assumed to be 0) and to a higher confirmation for the other clauses built with the same literals. Thus, we increase the confirmation difference between clauses and we gain more precise information.

In order to compute the expected frequency considering the background knowledge, we modify the previous algorithm in two ways: first, we add an initial step before the phase of iterative fitting and, second, we repeat the iterative fitting phase until the absolute differences between the expected and observed marginals fall below a certain threshold. In the initial step we set to 0 all the cells corresponding to counter-instances of clauses implied by the background. For example, if we have the rule $\forall(B_2 \leftarrow B_1)$ in the background and we are considering the rule $\forall(H \leftarrow B_1 \wedge B_2)$, we can set to 0 the two cells $H, B_1, \overline{B_2}$ and $\overline{H}, B_1, \overline{B_2}$ of the initial expected contingency table. In fact, this would be exquivalent to setting to 0 the number of counter-instances of the clauses $B_2 \leftarrow B_1, H$ and $B_2, H \leftarrow B_1$. In the case of example 2.1, the initial contingency table of the expected frequencies is Table 8.

Table 8: Contingency table that incorporate the background knowledge at step 1.

| | Body | | | | |
|---|---|---|---|---|---|
| | $B_1/6$ | | $\overline{B_1}/14$ | | |
| Head | $B_2$ | $\overline{B_2}$ | $B_2$ | $\overline{B_2}$ | |
| $H$ | 1 | 0 | 1 | 1 | 9 |
| $\overline{H}$ | 1 | 0 | 1 | 1 | 11 |
| | 13 | 7 | | | 20 |

Table 9: Contingency table that incorporate the background knowledge after the first iteration.

| | Body | | | | |
|---|---|---|---|---|---|
| | $B_1/6$ | | $\overline{B_1}/14$ | | |
| Head | $B_2$ | $\overline{B_2}$ | $B_2$ | $\overline{B_2}$ | |
| $H$ | 2.7 | 0 | 3.15 | 3.15 | 9 |
| $\overline{H}$ | 3.3 | 0 | 3.85 | 3.85 | 11 |
| | 13 | 7 | | | 20 |

The second modification is necessary because with zeros in the initial table we do not reach a fix point anymore after fitting over all the dimensions. After a first iteration over all the dimensions is performed, the expected marginals would not be equal to the observed marginals and a second iteration would further reduce the differences between them. Therefore, we iterate over all the dimensions until the maximum of the absolute value of the differences between the expected marginals and the observed marginals falls below a threshold defined by the user.

In the case of example 2.1, after the first iteration over all the dimensions we get contingency table 9. After a number of iterations we get contingency table 10. Supposing the threshold is 0.21, this table is the final table because the maximum difference is the one relative to $B_2$ that is 0.2. As you can see, the number of expected counter-instances of the clauses not implied by $B_2 \leftarrow B_1$, such as $H \leftarrow B_1, B_2$, is increased.

[1] proposed an approach for taking into account background knowledge in `Tertius` by interfacing it with Prolog. In particular, in this approach, each cell of the contingency table is tested for implication against the background knowledge by asking the appropriate Prolog query. The problem with this approach is that the test of implication with Prolog can be slow.

# 3 Algorithms

## 3.1 Implication Algorithms

In this section we describe two algorithms that look in the background knowledge for a rule that implies the rule

Table 10: Contingency table that incorporate the background knowledge at the final step.

| | Body | | | | |
|---|---|---|---|---|---|
| | $B_1/6$ | | $\overline{B_1}/14$ | | |
| Head | $B_2$ | $\overline{B_2}$ | $B_2$ | $\overline{B_2}$ | |
| $H$ | 2.7 | 0 | 3.14 | 3.16 | 9 |
| $\overline{H}$ | 3.3 | 0 | 3.84 | 3.86 | 11 |
| | 13 | 7 | | | 20 |

**for** every cell of the contingency table, generate the corresponding clause $D$,
  **for** every rule $C$ of the background knowledge:
    verify if $C$ implies $D$ using the simplified subsumption test,
    **if** so, set to 0 the corresponding cell of the contingency table and **exit** from the inner cycle.

Figure 1: Algorithm *Background1*.

found by `Tertius`.

The first algorithm, *Background1*, consider each cell of the contingency table in turn and generates the clause $D$ for which that cell represent the number of counter-instances. For example, cell $H, \overline{B}_1, B_2$ corresponds to clause $\{\overline{H}, B_1, \overline{B}_2\}$. Then the algorithm looks in the background knowledge for a clause $C$ that implies $D$. To this purpose, it checks whether $D$ is implied by $C$ for each clause $C$ of the background knowledge. In order to test implication, we use a simplified version of $\theta$-subsumption. Note that $\theta$-subsumption is correct but not complete with respect to implication: if a $C$ $\theta$-subsumes $D$ then $C$ implies $D$ but the converse is not true. The test we employ is correct but not complete with respect to $\theta$-subsumption: if it answers yes then $C$ $\theta$-subsumes $D$ (and thus $C$ implies $D$) but if it answers no $C$ may still $\theta$-subsumes (and imply) $D$.

Specifically, algorithm *Background1* is composed by the steps described in Figure 1. The *simplified subsumption test* between two clauses $C$ and $D$ is shown in Figure 2.

Now, we consider the computational complexity of *Background1*. The first step of *Background1* is the generation of the sign combination to be tested, which imposes the repetition of the test $2^m$ times, where $m$ is the number of literals of the rule that we are evaluating. The next step is the *simplified subsumption test*, which has a complexity proportional to $O(q \times m^2)$ where $m$ is the number of literals of the rule and $q$ is the number of rules in the background knowledge. Summarizing, we can say that the complexity of the algorithm *Background1* is $O(2^m \times q \times m^2)$.

The second algorithm, *Background2*, differs from the

**if** for every literal of $C$ there is a literal in $D$ with the same predicate symbol and sign **then**,
  **for all** couples $(L_1, L_2)$ of literals in $C$:
    consider the couple $(M_1, M_2)$ formed by the literals of $D$ that correspond to $(L_1, L_2)$,
    **for all** couples of arguments of $(L_1, L_2)$ that are identical,
      **if** the corresponding arguments of $(M_1, M_2)$ are different **then**,
        **return** failure.
  **return** success.
**else return** failure.

Figure 2: The *simplified subsumption test*.

first because it does not perform a separate search in the background for every cell of the contingency table. Rather, it considers the set of *atoms* $D'$ involved in a contingency table (i.e. it considers the set of literals corresponding to a cell stripped of their signs). Then, for each clause $C$ in the background, it removes the signs of the literals as well, obtaining $C'$, and it tests whether $C'$ implies $D'$ using the *simplified subsumption test*. When it finds a clause $C'$ implying $D'$, it assigns to the atoms of $D'$ the opposite of the signs of the corresponding literals in $C$. The assignment of signs determines which cells have to be set to 0. *Background2* is shown in Figure 3.

From the logic point of view the two algorithms are equivalent, i.e., they set to zero the same number of cells in the contingency table. As a consequence of this they return the same set of rules. The difference between them is in terms of computational complexity.

The algorithm *Background2* has a lower complexity than *Background1*. In particular, there is no exponential term because we perform only one query to the background knowledge. Therefore, the complexity is $O(q \times m^2)$ where $m$ is the number of literals of the rule we are evaluating and $q$ is the number of rules in the background knowledge.

For example, suppose that `Tertius` found the rule $D = \forall(H \leftarrow B_1 \wedge B_2)$. The set of atoms $D'$ is $D' = \{H, B_1, B_2\}$. Suppose also that we have the clause $C = \forall(B_2 \leftarrow B_1)$ in the background knowledge. Clause $C'$ is $\{B_1, B_2\}$. Thus $C'$ implies $D'$ and we can set some of the cells of the contingency table for $D$ to 0. We should set to 0 all the cells corresponding to the sign combination $\{B_1, \overline{B}_2\}$, i.e. $\{H, B_1, \overline{B}_2\}$ and $\{\overline{H}, B_1, \overline{B}_2\}$.

As stated before, the two algorithms use an implication test that is not complete. However, we can say that there is a particular class of problems where this implication test is complete. In our test, for each atom of the rule of the background we look for it in the clause found by `Tertius`. In particular, this search stops at the first failure. This implies that the implication test is complete only if we consider rules where each predicate symbol appears at most once because, in this way, we have only one possibility to try

generate clause $D'$ by removing the signs of literals from clause $D$,
**for each** clause $C$ in the background knowledge:
generate clause $C'$ by removing the signs of literals from $C$,
use the *simplified subsumption test* to see whether $C'$ implies $D'$,
**if** so, set to $0$ the corresponding cells in the contingency table.

Figure 3: Algorithm *Background2*.

**for** $i := 1$ **to** $number\_of\_rules - 1$ **do**
$Cur := C_i$
**for** $j := i + 1$ **to** $number\_of\_rules$ **do**
**if** $Cur$ **and** $C_j$ can be resolved **then**
$Cur := resolve(Cur, C_j)$
**output** $Cur$

Figure 4: Algorithm *Preprocess*.

the match between the atom of the background rule and the atom of the rule found by `Tertius`.

### 3.2 Preprocessing

In this section we describe the algorithm *Preprocess* that, from an initial background knowledge, infers clauses which are theorems of it. The importance of this algorithm is that the more rules in the background knowledge are present the more cells in the contingency table will be set to $0$.

*Preprocess* is inspired by the consequence finding concept that was defined by Lee [5] and recently extended to ILP by Inoue in [4]. In our algorithm we generate new rules by applying the resolution principle [7] to couples of clauses. The algorithm *Preprocess* is shown in Figure 4. Let us now see an example of application of our algorithm: suppose we have the rules $\{a(X) \leftarrow b(X),\ b(X) \leftarrow c(X)\}$ in the background knowledge, our algorithm generates the rule $a(X) \leftarrow c(X)$ that is a theorem of the initial theory.

## 4 Experiments

In this section we describe some experiments that compare the performance of *Background1* and *Background2*.

We have used two standard datasets. The first is the *Eastwest* dataset, where we have 20 instances of trains and we want to predict their direction on the basis of their length, the shape of their cars, etc. The second is the *Mutagenesis* dataset, where we have 188 instances of molecules and we want to predict their mutagenicity on the basis of the atoms and bonds that are part of it.

The rules that compose the background knowledge are obtained by starting `Tertius` with an empty background knowledge and taking the most confirmed rules.

The first result that we show is on the *Eastwest* dataset where we fix the maximum number of variables in the rules to $2$ and of literals in the body to $4$, and we analyze the behavior of the two algorithms by varying the number of rules in the background knowledge from $0$ to $500$. We can see the result of this experiment in Figure 5. The x-axis

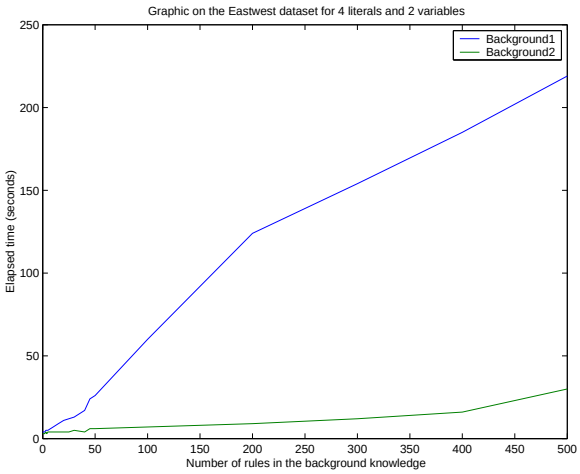


Figure 5: Response times of *Background1* and *Background2* on the *Eastwest* dataset, with 2 maximum variables and 4 maximum literals, as a function of the number of rules in the background.

of the graph in Figure 5 represents the number of rules in the background knowledge, while the y-axis represents the time of response of the algorithms. We can see that the performance of *Background2* are better than the performance of *Background1*.

Another experiment on the *Eastwest* dataset is done by increasing the maximum number of variables to $3$ and of literals to $5$. As in the previous experiment, we analyze the behavior of the two algorithms by varying the number of rules in the background knowledge from $0$ to $15$. Figure 6 shows the results of this experiment. As in the previous experiment, in the x-axis we have the number of rules in the background knowledge and in the y-axis we have the response time of the algorithms. By increasing the search space we can observe a general increase of the response time but the performance of *Background2* is again better than that of *Background1*.

Now, we show the results of an experiment on the *Mutagenesis* dataset using a search space limited to horn-clauses and with a maximum number of variables of $2$ and a maximum number of literals of $4$. As in the previous experiments, we vary the number of rules in the background knowledge from $0$ to $15$. Figure 7 shows the result of this experiment. The search space was limited to horn-clauses in order to keep the response time low. The x-axis and y-

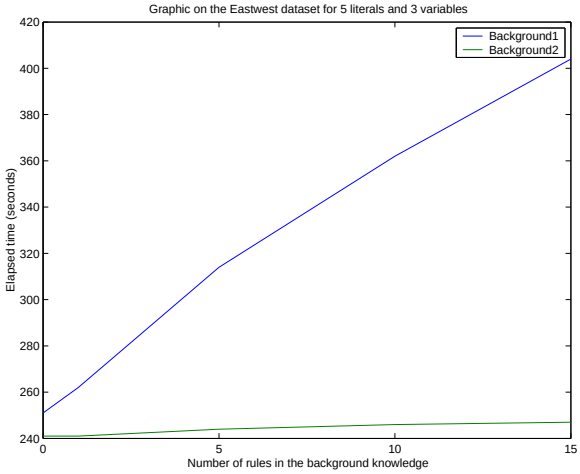


Figure 6: Response times of *Background1* and *Background2* on the *Eastwest* dataset, with 3 maximum variables and 5 maximum literals, as a function of the number of rules in the background.

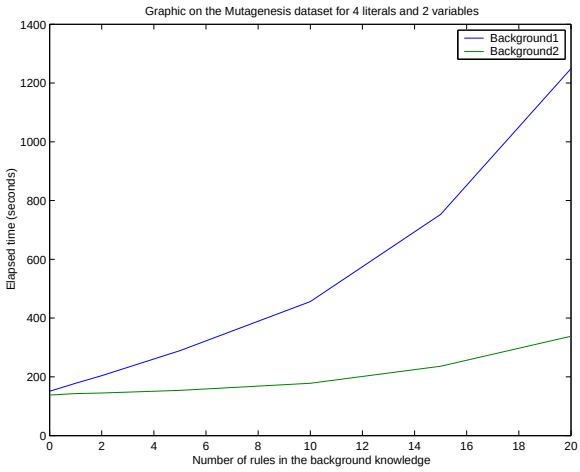


Figure 7: Response times of *Background1* and *Background2* on the *Mutagenesis* dataset, with 2 maximum variables and 4 maximum literals, as a function of the number of rules in the background.

axis have the same meaning as before. In general, due to the dimension of the dataset, the response time is greater that the one on the *Eastwest* dataset, but the relative behavior of the two algorithms remains unchanged.

In the next experiment we analyze the behavior of *Background1* and *Background2* in relation to the search space. This experiment is done on the *Eastwest* dataset and uses, for the two algorithms, a set of 20 rules in the background knowledge. The maximum number of variables is 4 and the maximum number of literals vary from 5 to 8. The result is shown in Figure 8. In the x-axis of this graph we

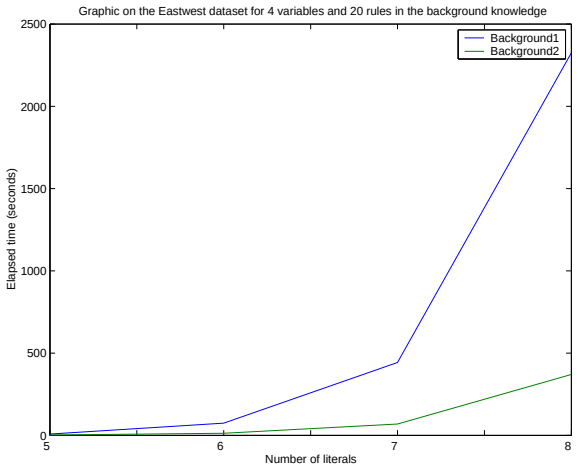


Figure 8: Response times of *Background1* and *Background2* on the *Eastwest* dataset, with 4 maximum variables and 20 rules in the background, as a function of the maximum number of literals.

have the maximum number of literals of the search space while in the y-axis we have the response time of the algorithms. In particular, we can see that, for a low number of literals in the search space, the performance of the two algorithms are comparable, but, when the number of literals increases, *Background1* increases its response time more than *Background2*.

After this set of experiments, we can say that, for a limited number of rules in the background knowledge and for a small search space, the performance of the two algorithms are similar. However, if we increase the number of rules in the background knowledge or the number of literals of the search space, the response time of *Background1* increases more than that of *Background2*.

Now, we want to show the percentage of cells of the contingency tables that are set to zero due to the background knowledge. In particular, the two algorithms set to zero the same number of cells. In the following table we represent an experimental result on the Eastwest dataset with a maximum number of literal of 3 and a maximum number of variables of 2. In particular, Table 11 presents the relation between the percentage of the total number of cells of the contingency tables that are set to zero and the number of rules of the background knowledge. In the following

Table 11: Percentages of cells set to 0 for 3 maximum literals and 2 maximum variables.

| N rules | 2 | 4 | 6 | 8 | 10 |
|---|---|---|---|---|---|
| P cells | .21% | .42% | .63% | .84% | 1.06% |

Table 12: Percentages of cells set to 0 for 4 maximum literals and 3 maximum variables.

| N rules | 1 | 5 | 10 | 15 | 20 |
|---|---|---|---|---|---|
| P cells | .009% | .04% | .28% | .32% | .73% |

table we represent another experimental result on the East-west dataset with a maximum number of literal of $4$ and a maximum number of variables of $3$. As in the previous experiment, Table 12 presents the relation between the percentage of cells set to zero and the number of rules in the background knowledge. Now, we want to show some examples of how *Preprocess* works. In the first experiment we consider this initial background knowledge:

```
1.  a(X,Y):-c(Y),b(X).
2.  b(X):-d(X).
3.  d(X):-h(X).
```

The result of this experiment is the following:

```
4.  a(X,Y):-c(Y),d(X).
5.  a(X,Y):-c(Y),h(X).
6.  b(X):-h(X).
```

Let us follow the behavior of the algorithm. It starts from rule `1` and it tries to resolve it with the following rules. So, rule `4` is obtained by the resolution of rule `1` with rule `2`. Then, rule `4` is resolved with rule `3` obtaining rule `5`. Finally, rule `6` is obtained by resolving rule `2` with rule `3`.

Another experiment is done with a background knowledge where some rules have constants. We consider this background knowledge:

```
1.  a(X,Y):-c(X),b(X).
2.  b(g):-d(g).
3.  d(X):-h(X).
```

The result of the experiment is:

```
4.  a(g,X):-c(g),d(g).
5.  a(g,X):-c(g),h(g).
6.  b(g):-h(g).
```

Rule `4` is obtained by resolving rule `1` with rule `2`. Then, the algorithm obtains rule `5` resolving rule `4` with rule `3`. Finally, rule `6` is the result of the resolution of rule `2` with rule `3`.

In the last experiment we consider a background knowledge where all the rules have all their variables instantiated to constants:

```
1.  a(g,f):-c(f),b(g).
2.  b(d):-d(g).
3.  d(g):-h(g).
```

The only rule that can be obtained is the one from the resolution of rule `2` with rule `3`:

```
b(d):-h(g).
```

## 5 Conclusions and Future Works

We have presented three algorithms that can be used in order to improve the handling of background knowledge by the ILP system `Tertius`. *Background1* and *Background2* exploit background during learning for setting to 0 some of the cells of the contingency tables. They are both based on a simplified subsumption algorithm that is correct but not complete. The experiments show that, as was expected, *Background2* is much more efficient that *Background1*.

The last algorithm performs a preprocessing of the background knowledge by adding to it some of the clauses that are its logical consequences. In this way more cells of the contingency tables can be set to 0 thus improving the reliability of the confirmation function.

An interesting line of future research is to integrate in *Background2* the algorithm proposed in [2] for performing $\theta$-subsumption. Such an algorithm takes as input two clauses $C$ and $D$ and returns the set of all the substitutions $\theta$ such that $C\theta \subseteq D$. This set is useful in our case because, for each substitution, a different set of cells of the contingency table can be set to 0. In fact, different substitutions may map the same literal of $C$ to different literals of $D$, thus leading to different corresponding cells of the table.

## REFERENCES


[1] T. S. Dahl. Background knowledge in the tertius first order knowledge discovery tool. *Technical Report CSTR-99-006, Department of Computer Science, University of Bristol*, March 1999.

[2] Stefano Ferilli, Nicola Di Mauro, Teresa Maria Altomare Basile, and Floriana Esposito. A complete subsumption algorithm. In *AI*IA 2003: Advances in Artificial Intelligence, 8th Congress of the Italian Association for Artificial Intelligence, Pisa, Italy, September 23-26, 2003, Proceedings*, pages 1–13, 2003.

[3] Peter A. Flach and Nicolas Lachiche. Confirmation-guided discovery of first-order rules with tertius. *Machine Learning*, 42:61–95, January 2001.

[4] Katsumi Inoue. Induction as consequence finding. *Machine Learning*, 55:109–135, May 2004.

[5] Char-Tung Lee. *A completeness theorem and a computer program for finding theorems derivable from given axioms*. PhD thesis, 1967.

[6] Gregory Piatetsky-Shapiro. Discovery, analysis, and presentation of strong rules. In *Knowledge Discovery in Databases*, pages 229–248. AAAI/MIT Press, 1991.

[7] J. A. Robinson. A machine-oriented logic based on the resolution principle. *J. ACM*, 12(1):23–41, 1965.