ELSEVIER The Journal of Logic Programming 40 (1999) 215–249

THE JOURNAL OF LOGIC PROGRAMMING

A study of relevance for learning in deductive databases

Nada Lavrač[a,*], Dragan Gamberger[b], Viktor Jovanoski[a]

[a] *Jožef Stefan Institute, Jamova 39, 1000 Ljubljana, Slovenia*
[b] *Rudjer Bošković Institute, Bijenička 54, 10000 Zagreb, Croatia*

Received 16 September 1996; received in revised form 10 April 1998; accepted 12 January 1999

Abstract

This paper is a study of the problem of relevance in inductive concept learning. It gives definitions of irrelevant literals and irrelevant examples and presents efficient algorithms that enable their elimination. The proposed approach is directly applicable in propositional learning and in relation learning tasks that can be solved using a LINUS transformation approach. A simple inductive logic programming (ILP) problem is used to illustrate the approach to irrelevant literal and example elimination. Results of utility studies show the usefulness of literal reduction applied in LINUS and in the search of refinement graphs.

Keywords: Concept learning; Inductive logic programming; Relevance of literals and examples; Cost-sensitive literal reduction

1. Introduction

Inductive concept learning can be viewed as a process of searching a space of concept descriptions (hypotheses) [27] aimed to fit a given dataset. If the learner has no prior knowledge about the learning problem, it learns exclusively from examples. However, difficult learning problems typically require a substantial body of prior knowledge to be considered in the learning process. For instance, in inductive logic programming (ILP) [28,31,19], a relation learning task involves learning of an intensional definition of a target relation (a hypothesis H) from the extensional definition of this relation (training examples E) and definitions of other relations relevant for the task (background knowledge B).

Although the use of a substantial body of background knowledge is usually invaluable for the success of learning, using too much information may sometimes

* Corresponding author. Tel.: +386-61-177-3272; fax: +386-61-125-1038; e-mail: nada.lavrac@ijs.si

0743-1066/99/$ – see front matter
PII: S0743-1066(99)00019-9

have the opposite effect. First of all, substantial amount of background knowledge largely increases the search space of hypotheses, which results in decreased efficiency of learning. Moreover, incorporating inappropriate background knowledge may also cause increased complexity and/or decreased accuracy of induced hypotheses (for experimental evidence, see Ref. [6]). It is therefore important to determine what parts of information contained in the training set and in the background knowledge are relevant for the success of learning; this is the topic of interest of this study.

In inductive concept learning, the hypothesis language and the background knowledge (together with some explicit definitions and/or implicit assumptions) define the basic language elements that constitute the hypothesis space. *Literals* are the basic language elements in a first-order language. Logically speaking, attribute-value pairs, features, selectors, etc., which are the basic ingredients of various propositional representations, can also be considered as literals.

The primary aim of the theory of relevance presented in this paper is to detect which literals are irrelevant for learning and to exclude them, in order to reduce the hypothesis space and facilitate the search for the final solution. The secondary aim is to reduce also the number of training examples. Whereas the elimination of literals may be done more or less regardless of the specific properties of the learning domain and the learning algorithm used, the elimination of training examples needs to be done with caution: the exclusion of examples may change the statistical properties of the training set (the distribution of positive and negative examples) which may be inappropriate for algorithms using statistical measures in learning and noise-handling procedures, such as learning and pruning of decision trees [36].

The paper gives definitions of irrelevant literals and irrelevant examples and presents efficient algorithms that enable their elimination. Besides reducing the hypothesis space, the elimination of irrelevant literals and examples may contribute to the better understanding of the problem domain. For example, this may be important in data analysis where irrelevant literals may indicate that some measurements are not needed, whereas irrelevant examples may indicate the uninteresting cases that do not require further attention when studying the domain.

The developed theory of relevance is applicable in propositional learning. In this paper, the impacts of the theory of relevance are studied in a restricted inductive logic programming (ILP) context, using the LINUS transformation approach which enables learning in deductive databases [18,19].

The outline of the paper is as follows. Section 2 presents the scope of this research and an overview of related work. The transformation approach is sketched in Section 3, which introduces a simple relation learning task, illustrates an example run of LINUS, and gives the complexity of the LINUS transformation approach. It also presents a transformation approach applicable in propositional learning.

A theoretical study of irrelevant literal and example elimination is given in Section 4. It introduces literals and pairs of examples (the so-called p/n pairs), gives definitions of irrelevant literals and irrelevant examples and presents theorems which are the basis for literal and example elimination. Section 5 presents the algorithms for literal and example elimination and illustrates an example run of the algorithms.

Section 6 studies the utility of the irrelevant literal elimination algorithm used as part of the LINUS transformation, and presents the results of experiments in a number of typical relational problem domains. Section 7 is a study of the effectiveness of literal elimination in the search of refinement graphs, a learning technique frequently

used in ILP. The experiments, using a simplified refinement operator, confirm the potential of irrelevant literal elimination for reducing the search space of hypotheses in ILP. The paper concludes with a discussion and directions for further work.

2. Scope and related work

The problem of relevance has been addressed already in early inductive concept learning research [26]. This problem is actually encountered by every inductive learner. Basically, all learners are concerned with the selection of relevant literals. Usually, at each step of learning, the choice of the 'best' or 'most informative' literal needs to be made. This choice is frequently based on the distribution of positive and negative examples covered by the hypothesis before and after literal selection, e.g., Refs. [35,36]. Whereas in most learning systems the selection of significant or informative literals is part of the learning process, the theory of relevance presented in this paper is aimed at pointing out which literals constitute a set of relevant literals and which literals are irrelevant and can be discarded, without even entering the 'best literal' competition. Such a filtering of irrelevant literals can thus be done in preprocessing of the set of training examples. Whereas most other algorithms only consider the 'local training set' (i.e., a subset of examples covered by the currently developed rule, or a subset of examples in the currently developed node of a decision tree) when deciding about the importance/relevance of literals, we are concerned with finding 'globally relevant' literals w.r.t. the entire set of training examples. This is important since the elimination of globally irrelevant literals guarantees that literal elimination will not harm the hypothesis formation process.

The problem of relevance has recently attracted much attention in the context of feature subset selection in propositional learning [2,13,16,24,39]. An extensive discussion of different approaches to feature subset selection can be found in Ref. [15], which distinguishes between filter and wrapper approaches, and introduces the notions of totally irrelevant, weakly relevant and strongly relevant features. In this categorization, our work belongs to filter approaches which eliminate totally irrelevant features in preprocessing. Other filtering approaches include different versions of the RELIEF algorithm [14,17], the FOCUS algorithm [1] and an approach to feature selection proposed in Ref. [33].

The definition of relevance in this work differs from the definitions of relevance in the above approaches. RELIEF [14,17] estimates the relevance of features based on a combination of statistical and topological properties (proportion of positive and negative nearest neighbors to randomly selected instances), and eliminates weakly relevant features. In contrast, our definition of the irrelevance of literals is based on the coverage of pairs of examples, where an example pair consists of one positive and one negative example. The idea of using pairs of examples was developed and used within the ILLM algorithm for learning generalized CNF/DNF descriptions [9], whereas its origins stem from the theory of Boolean functions [3]. This is related to FOCUS [1] which introduces a similar concept of example pairs called a set of conflicts. The main difference to our approach is that the set of conflicts is based on the coverage by features (and not by literals) and that the obtained minimal set of features contains strongly relevant features. A similar approach is iteratively

applied in Ref. [33] with the intention to enable constructive induction based on binary features.

The distinguishing feature of the theory and algorithms presented in this paper is the capability of dealing with costs of literals, which is important for practical domains (e.g., medical diagnostic problems). Cost-sensitive literal reduction was studied in an application of a hybrid genetic algorithm RL-ICET to two "challenges to the international computing community" to discover low size-complexity Prolog programs for classifying trains as Eastbound or Westbound (the so-called East–West challenges). This application showed significant improvement of performance of the hybrid genetic algorithm due to applying irrelevant literal elimination in preprocessing of the dataset. In this application, costs were treated as an estimate of the complexity of literals [21,22].

The approach presented in this paper deals with noiseless domains. However, the presented theory of relevance can be upgraded to dealing with noise by applying an approach to noisy example elimination, presented in Refs. [9–11], provided that a learner based on the minimum description length (MDL) principle is used for learning [5,23]. In comparison with other approaches to relevance, notice that also the adapted RELIEF algorithm presented in Ref. [17] can deal with noisy data, whereas Refs. [1,33] are designed for noiseless domains.

Our approach to cost-sensitive literal elimination can be easily reimplemented in an iterative algorithm, starting with an initial (possibly empty) set of literals and adding relevant literals only [22]. This is particularly useful for tasks in which bias shift is needed. Bias shift can be implemented by shifting to a more expressive hypothesis language in a language hierarchy given by the user. On the other hand, a learner may perform bias shift by constructive induction/predicate invention, where the relevance of newly costructed literals can be tested in data preprocessing.

As noted above, our approch to dealing with relevance is not based on the statistical properties of the training set. On the contrary, it is a deterministic consequence of the known properties and dependencies among literals (and examples). Similar deterministic complexity reduction approaches were studied long ago in the fields of game theory [32,37] and switching circuit design [12]. In game theory, the approach studies the notion of 'dominance': when does a (dominant) strategy dominate another (dominated) strategy. The definition of dominance was and still is essential for some problems in game theory because it is the only way how, by the elimination of dominated startegies, these problems can be solved. Game theory distinguishes the notions of strong dominance, weak dominance and iterative dominance [37]. Our term of relevance corresponds to the definition of strong dominance.

The dominance approach was effectively used in the field of minimal complexity realization for switching functions where the aim is to find a minimal cover from the set of all prime implicants [34]. This field introduced the term 'covering' in the sense that an instance can be (but should not be), covered by some prime implicant. Also the dominance property has been defined both for columns (prime implicants) and rows (instances). It is generally assumed that all prime implicants have the same weight, and the intention is to find the solution with the least number of different prime implicants.

It must be noted that the deterministic theory of dominance (relevance) cannot be applied to inductive learning problems in their original form, except for trivial cases with identical inputs (features or attributes). The necessary prerequisite for the appli-

cability of the deterministic notion of relevance is the transformation of the original problem into the standard covering problem. From the theory of Boolean functions it is known that the transformation can be done by introducing instance (example) pairs where each pair is built from one positive and one negative instance [3]. This approach is much less popular than the well-known Quine–McCluskey prime implicant approach. The reason is that the obtained covering table has the number of rows proportional to the square of the number of instances (examples). But the approach with instance pairs can have significant advantages when the target function is incompletely specified with the unknown output value for many input combinations. This concept is the basis for the suggested transformation of examples into the form of truth-value tuples described by literals and the basis for the theorems presented in this paper.

3. A transformation approach to inductive learning

In order to facilitate the presentation of the proposed theory of relevance, we first need to appropriately incorporate the available background knowledge into the learning task; namely, the set of considered literals directly depends on the background knowledge. This will be done in preprocessing, by applying a transformer which takes as its input the initial dataset, the form of basic language elements (literals) and background knowledge, and gives as its output a transformed set of training examples E, described by literals.

In our work, preprocessing consists of two steps:

1. Transformation of the given dataset into the form of truth-value tuples E described by literals L.
2. Elimination of irrelevant literals and examples, resulting in there duced set of relevant literals RL and relevant examples RE.

Step 2 is the central theme of this paper, whereas step 1 is a necessary prerequisite for step 2. Step 1 is briefly described in this section.

3.1. Prerequisites

Recall the following logic programming terminology [25]. A *term* is a variable or a function symbol followed by a bracketed n-tuple of terms. A *constant* is a function symbol of arity 0. An *atom* (or an atomic formula) is a predicate symbol followed by a bracketed n-tuple of terms. A *literal* is an atom (a positive literal) or a negation of an atom (a negative literal).

For a positive literal l, its negation will be written as $\overline{l}$, or $\neg l$, or *not* l.

Let us illustrate these definitions by examples: $f(g(X), h)$ is a term when f, g and h are function symbols and X is a variable. If $p/2$ is a predicate of arity 2, $p(X, Y)$ is an atom or a positive literal, whereas $\neg p(X, Y)$ is a negative literal. In the intended interpretation (for given values of arguments of the predicate) a literal has a value *true* or *false*.

In a relational data model, a relation is a set of tuples, i.e., a subset of the Cartesian product of one or more *domains* $D_1 \times \cdots \times D_n$. A relation can be viewed as a table, where each row is a tuple, and columns are often given names,

called *attributes*. A *relational database* is a set of relations. A *deductive database*[1] (DDB) is a set of database clauses of the form $A \leftarrow L_1, \ldots, L_m$ where A is an atom of the form $p(X_1, \ldots, X_n)$ with typed argyments X_i, and L_i are literals (atoms or negations of atoms). Atom A is called the *head* and the conjunction of literals L_i the *body* of the clause. A *predicate definition* is a set of clauses with the same predicate in the head. The difference between program clauses (as used in logic programming) and database clauses is that database clauses are typed, i.e., each argument X_i of each predicate p in a DDB is assigned a type T_i. In DDB, a *ground fact* is a unit clause, i.e., a clause with an empty body and no variables in the head of the clause. A ground fact in a DDB is equivalent to a tuple in a relational database.

A relation learning task can be defined as follows.

Given

- a set P of true ground facts of a target relation t (positive examples),
- a set N of false ground facts of the target relation t (negative examples),
- background knowledge B defining relations $q_i,\ i = 1, \ldots, k$ (other than t) which may be used in the definition of t, and
- a hypothesis language $\mathscr{L}$, specifying syntactic restrictions on the definition of t

Find a definition H (hypothesis) of the taget relation $t,\ H \in \mathscr{L}$, such that

- H covers all the positive examples, i.e., $\forall p \in P : B \cup H \models p$ (completeness), and
- H covers none of the negative examples, i.e., $\forall n \in N : B \cup H \not\models n$ (consistency).

This definition of the learning task is appropriate for non-noisy domains. For domains with noise, the completeness and consistency requirements need to be relaxed and replaced by other quality criteria (for different criteria, see Ref. [19]).

In inductive concept learning, we distinguish between the propositional and the first-order learning framework, depending on the choice of the hypothesis language. In the propositional case, learning is usually called *propositional learning* or *attribute-value learning*. Training examples $E = P \cup N$ are expressed as tuples of a relation t and the induced hypothesis is usually represented in rule form (CNF, DNF) or decision tree form. In the first-order case, relation learning is called *inductive logic programming* (ILP) when the selected representation language is the language of logic programs consisting of program clauses. In learning from deductive databases, a sub-language of the language of logic programs is used, e.g., the language of database clauses (DDB – typed program clauses), hierarchical database clauses (DHDB – database clauses restricted to non-recursive predicate definitions and non-recursive types), constrained DHDB clauses (all variables in the body literals appear in the head literal), or Datalog (program clauses with no function symbols of non-zero arity).

In this paper, we assume that B is a deductive database formed of Datalog clauses (which may also be ground facts), $\mathscr{L}$ is the language of function-free constrained DHDB clauses, and training examples are ground facts.

Note that a fact $e \in E$ can be viewed as $t(X_1, \ldots, X_n)\theta$, where a grounding substitution $\theta = \{X_1/v_1, \ldots, X_n/v_n\}$ makes $e = t(X_1, \ldots, X_n)\theta$ *true* for $p \in P$ and *false* for

[1] We use the term *deductive database* (DDB) instead of the term *normal database* for the database that consists of database clauses.

$n \in N$. Moreover, we will assume that a deductive database is implemented in Prolog and that coverage is tested as a Prolog query: having asserted B and H in a Prolog database, the answer *true* to a query $? - q$ means that q is covered by H.

3.2. Transformation in learning from deductive databases

In the transformation approach to inductive logic programming, as implemented in the LINUS and DINUS systems [18,19], a set of all literals to be considered by a propositional learner is determined in preprocessing. The set of literals constructed by LINUS will be denoted by L_p, where the subscript p denotes positive literals. In this paper, these positive literals will also be called *features*.

3.2.1. An example run of LINUS

To illustrate the LINUS transformation approach, consider a first-order relation learning task typical for inductive logic programming. The example is taken from Ref. [19]. It is suited for the LINUS approach whose hypothesis language is limited to constrained DHDB clauses [18,19].

Suppose that the task is to define the target relation $daughter(X, Y)$, which states that person X is a daughter of person Y, in terms of the background knowledge relations *female*, *male* and *parent*. These relations are given in Table 1, where all variables are of type *person*. The type *person* is defined as a set of values: $person = \{ann, eve, pat, sue, tom\}$. There are two positive examples $P = \{daughter(sue, eve), daughter(ann, pat)\}$, assigned $\oplus$, and two negative examples $N = \{daughter(tom, ann), daughter(eve, ann)\}$, assigned $\ominus$, of the target relation *daughter* whose intensional definition is to be learned from the given extensional definition (training examples $P \cup N$) and background knowledge. Notice that LINUS is not limited to extensional background knowledge B. It can use intensional definitions of background predicates as well.

The LINUS transformation procedure [18,19] has as its input the training examples and background knowledge, and as its output a truth-value table of transformed examples.

In the transformation of an ILP problem into the tabular form with elements *true* and *false*, all the possible applications of the background predicates on the arguments of the target relation are determined, taking into account argument types. Each such application introduces a new literal which will, in the learning phase, be considered as an attribute used in learning the target relation. Since the hypothesis language is restricted to non-recursive constrained clauses, the following positive literals L_p are generated by the LINUS transformer: $female(X)$, $female(Y)$, $male(X)$, $male(Y)$, $parent(X, X)$ $parent(X, Y)$, $parent(Y, X)$, and $parent(Y, Y)$. In general, LINUS would generate also the literal $X = Y$, which stands for $equal(X, Y)$. Since

Table 1
A simple ILP problem: learning the *daughter* relationship

Target relation		*Background knowledge*		
$daughter(sue, eve)$	$\oplus$	$parent(eve, sue)$	$female(ann)$	$male(pat)$
$daughter(ann, pat)$	$\oplus$	$parent(ann, tom)$	$female(sue)$	$male(tom)$
$daughter(tom, ann)$	$\ominus$	$parent(pat, ann)$	$female(eve)$	
$daughter(eve, ann)$	$\ominus$	$parent(tom, sue)$		

in most of the domains of experiments in Sections 6 and 7 such a literal makes no sense, it is excluded from the consideration of this paper (only in the chess endgame this predicate is useful, where *equal*/2 is explicitly added to the background knowledge).

Notice that in our example all variables are of the same type *person*. In constrained clauses, for the target relation *daughter*(X, Y) that should appear in the head of a clause, the literals that may appear in the body of a clause may only use X and Y in the arguments of background knowledge predicates, whereas literals introducing new variables such as *parent*(X, Z) are not considered.

The corresponding propositional learning problem is given in Table 2, where C stands for *Class*, d stands for *daughter*, f for *female*, m for *male* and p for *parent*, and *Literals* denote the literals that can be considered as attributes constructed for a propositional learning task.

Observe in Table 2 that the LINUS transformation results in some senseless literals such as $p(X, X)$ which are *false* for all training examples (nobody is his own parent). In LINUS, this could be avoided by adding semantic information on individual predicates which would prevent the generation of such literals. Alternatively, such senseless literals can be eliminated by an algorithm for irrelevant literal elimination, outlined in Section 5.

3.2.2. The complexity of the transformation approach

A complete complexity analysis for learning constrained DHDB clauses with LINUS can be found in Ref. [19]. The relevant part of the complexity analysis is outlined below for the sake of completeness.

The analysis in this paper assumes the function-free restriction and does not consider literals arising from the applications of the built-in predicate *equality*/2. Under these restrictions, the number of positive literals $|L_p|$ is equal only to the number of literals resulting from applications of the b background predicates q_i on the arguments of the target relation:

$$|L_p| = \sum_{i=1}^{b} k_{New,q_i} \tag{1}$$

Suppose that u is the number of distinct types of arguments of the target predicate t, u_i is the number of distinct types of arguments of the background predicate q_i, $n_{i,s}$ is the number of arguments of q_i that are of type T_s and k_{ArgT_s} is the number of arguments of target predicate t that are of type T_s. Then k_{New,q_i} is computed by the following formula:

Table 2
Propositional form of the *daughter* relationship problem

C	*Examples* $d(X, Y)$	*Literals*							
		$f(X)$	$f(Y)$	$m(X)$	$m(Y)$	$p(X,X)$	$p(X,Y)$	$p(Y,X)$	$p(Y,Y)$
$\oplus$	$d(sue, eve)$	*true*	*true*	*false*	*false*	*false*	*false*	*true*	*false*
$\oplus$	$d(ann, pat)$	*true*	*false*	*false*	*true*	*false*	*false*	*true*	*false*
$\ominus$	$d(tom, ann)$	*false*	*true*	*true*	*false*	*false*	*false*	*true*	*false*
$\ominus$	$d(eve, ann)$	*true*	*true*	*false*	*false*	*false*	*false*	*false*	*false*

$$k_{New,q_i} = \prod_{s=1}^{u_i} (k_{ArgT_s})^{n_{i,s}} \quad (2)$$

The $n_{i,s}$ places for arguments of type T_s can be, namely, filled in $(k_{ArgT_s})^{n_{i,s}}$ ways independently from choosing the arguments of q_i which are of different types.

In the *daughter* learning example, $q_1 = female$, $q_2 = male$ and $q_3 = parent$. As all arguments are of the same type $T_1(person), u_1 = u_2 = u_3 = 1$ and $k_{ArgT_1} = 2$. Since there is only one type, n_i can be used instead of $n_{i,s}$. In this notation, $n_1 = n_2 = 1$ and $n_3 = 2$. Thus, according to Eq. (2), $k_{New,q_1} = k_{New,q_2} = (k_{ArgT_1})^{n_1} = (k_{ArgT_1})^{n_2} = 2^1 = 2$. This means that there are two applications of each of the predicates *female* and *male*, namely $f(X)$, $f(Y)$ and $m(X)$, $m(Y)$, respectively. Similarly, $k_{New,q_3} = (k_{ArgT_1})^{n_3} = 2^2 = 4$, the four applications of *parent*/2 being $p(X,X)$, $p(X,Y)$, $p(Y,X)$ and $p(Y,Y)$. Finally, $|L_p| = k_{New,q_1} + k_{New,q_2} + k_{New,q_3} = 2+2+4 = 8$.

It is obvious that the number of generated literals may increase the dimensionality of the problem to an extent that may prevent the practical application of the transformation approach for domains with large numbers of predicates in the background knowledge, especially if these have many arguments (since the number of generated literals grows exponentially with the number of arguments – see Eq. (2)). Thus, when taking into account all the literals, learning may become unfeasible, in particular in the DINUS framework where the language bias is weakened and hypotheses consist of determinate non-constrained clauses which may introduce new variables in the body [9].

3.3. Transformation in propositional learning

The transformation procedure is rather straightforward for propositional learning. In propositional learning, the basic language elements are literals of the form *Attribute* = *Value* and $\neg$(*Attribute* = *Value*) (i.e., *Attribute* $\neq$ *Value*) for discrete attributes. Training examples are bitstrings (tuples) of truth-values of these literals.

To illustrate this representation, consider a problem with two attributes, A and B, and a training set of three examples, two positive examples (a_2, b_1) and (a_3, b_2) and a negative example (a_1, b_2). Then the following literals are created: $A \neq a_1$, $A = a_2$, $A = a_3$, $B = b_1$, $B = b_2$, $B \neq b_2$, and the three truth-value tuples of literals are constructed: $p_1 = [true, true, false, true, false, true]$ and $p_2 = [true, false, true, false, true, false]$ corresponding to the two positive examples, and $n_3 = [false, false, false, false, true, false]$ corresponding to the negative example.[2] Literals $A = a_1$, $A \neq a_2$, ... are not even considered since they are either *false* for all positive examples ($A = a_1$) or *true* for all negative examples ($A \neq a_2$); as such they are useless for constructing a concept description.

[2] Note that the examples in the initial dataset are described in the data description language, whereas the tuples can be interpreted as the training examples, described by the primitives of the hypothesis language. For instance, given the following primitives of the hypothesis language $A \neq a_1$, $A = a_2$, $A = a_3$, $B = b_1$, $B = b_2$, and $B \neq b_2$, the tuple $p_2 = [true, false, true, false, true, false]$ corresponding to the initial training example (a_3, b_2) is actually quivalent to the conjunctive description $A \neq a_1 \wedge A = a_3 \wedge B = b_2$ in the hypothesis language.

Different types of literals are constructed for continuous and integer-valued attributes [21]. To formalise literal construction, let values v_{ix} ($x = 1 \ldots k_{ip}$) denote the k_{ip} different values of attribute A_i that appear in the positive examples and w_{iy} ($y = 1 \ldots k_{in}$) the k_{in} different values of A_i appearing in the negative examples. The transformation results in a set of literals L:

- For discrete attributes A_i, literals of the form $A_i = v_{ix}$ and $A_i \neq w_{iy}$ are generated.
- For continuous attributes A_i, literals of the form $A_i \leqslant (v_{ix} + w_{iy})/2$ are created for all neighboring value pairs (v_{ix}, w_{iy}), and literals literals $A_i > (v_{ix} + w_{iy})/2$ for all neighboring pairs (w_{iy}, v_{ix}). The motivation is similar to one suggested in Ref. [7].
- For integer valued attributes A_i, literals are generated as if A_i were both discrete and continuous, resulting in literals of four different forms: $A_i \leqslant (v_{ix} + w_{iy})/2$, $A_i > (v_{ix} + w_{iy})/2$, $A_i = v_{ix}$, and $A_i \neq w_{iy}$.

4. Theory of relevance

The main aim of the theory of relevance is to reduce the hypothesis space by the elimination of irrelevant literals. Its secondary aim is the reduction of the space of examples by the elimination of irrelevant examples.

4.1. Literals and p/n pairs of examples

In previous sections we have introduced literals as the basic language elements constituting the hypothesis space.

Consider a two-class learning problem where training set E consists of positive and negative examples of a concept ($E = P \cup N$) and examples $e \in E$ are tuples of truth-values of literals L. Training set E is represented as a table where rows correspond to training examples and columns correspond to literals. An element in the table has the value *true* when the example satisfies the condition (literal) in the column of the table, otherwise its value is *false*.

Definition 1. Let $E = P \cup N$, where P are positive and N are negative examples. A p/n pair is a pair of training examples where $p \in P$ and $n \in N$.

Definition 2. Let L denote a set of literals. A literal $l \in L$ covers a p_i/n_j pair if the literal has value *true* for p_i and value *false* for n_j.

Notice that in the standard machine learning terminology we may reformulate the definition of coverage of p/n pairs as follows: literal l covers a p/n pair if l covers (has value *true* for) the positive example p and does not cover (has value *false* for) the negative example n.

Example 1. Recall the learning problem from Section 3.2, where the definition of the target relation *daughter* is to be induced from four training examples and the given background knowledge consisting of definitions of background knowledge predicates $f/1$ (f stands for *female*), $m/1$ (*male*) and $p/2$ (*parent*) (see Table 1). Recall the two positive and two negative examples (symbol d stands for the predicate symbol *daughter*):

$$P = \{p_1, p_2\} = \{d(sue, eve), d(ann, pat)\}$$

$$N = \{n_1, n_2\} = \{d(tom, ann), d(eve, ann)\}$$

Recall that the extensional definition of the background knowledge predicate *female* consists of three ground facts:

$$\{f(ann), f(sue), f(eve)\}$$

Consider now the target relation $d(X, Y)$ and the literal $l = f(X)$. Literal $f(X)$ covers p_1/n_1 since for $p_1 = d(sue, eve)$ the value of $f(X)$ is *true* (since $f(sue)$ can be found in the extensional definition of the predicate $f/1$), and $f(X)$ is *false* for $n_1 = d(tom, ann)$ (since *tom* is not a female, i.e., $f(tom)$ is not in the extensional definition of $f/1$). On the other hand, $f(X)$ does not cover p_1/n_2 since $f(X)$ is *true* for p_1 (*sue* is a female), but $f(X)$ is also *true* for n_2 (*eve* is a female). Furthermore, $f(X)$ covers p_2/n_1 and does not cover p_2/n_2.

The notion of p/n pairs can be used to prove important properties of literals for building complete and consistent concept descriptions [9]. The following theorem assumes that the hypothesis language $\mathscr{L}$ is rich enough to allow for a complete and consistent hypothesis H to be induced from the set of training examples E.

Theorem 1. *Assume a training set E and a set of literals L such that a complete and consistent hypothesis H can be found. Let $L' \subseteq L$. A complete and consistent hypothesis H can be found using only literals from the set L' if and only if for each possible p/n pair from the training set E there exists at least one literal $l \in L'$ that covers the p/n pair.*

Proof. *Necessity* (*only if*): Suppose that the negation of the conclusion holds, i.e., that a p/n pair exists that is not covered by any literal $l \in L'$. Then no rule built of literals from L' will be able to distinguish between these two examples. Consequently, a description which is both complete and consistent cannot be found.

Sufficiency (*if*): Take a positive example p_i. Select from L' the subset of all literals L_i that are true for p_i. A constructive proof of sufficiency can now be presented, based on k runs of a covering algorithm, where k is the cardinality of the set of positive examples ($k = |P|$). In the ith run, the algorithm learns a conjunctive description h_i, $h_i = l_{i,1} \wedge \cdots \wedge l_{i,m}$, from all $l_{i,1}, \ldots l_{i,m} \in L_i$ that are true for p_i. Each h_i will thus be *true* for p_i (h_i covers p_i), and *false* for all $n \in N$. After having formed all the k descriptions h_i, a resulting complete and consistent hypothesis can be constructed: $H = h_1 \vee \cdots \vee h_k$. □

The importance of Theorem 1 for the theory of relevance is manifold. First, it points out that when deciding about the relevance of literals it will be significant to detect which p/n pairs are covered by the literal. Second, the theorem enables us to directly detect useless literals as those that do not cover any p/n pair. In addition, an important property of pairs of literals can now be defined: the property of the so-called coverage of literals.

Definition 3. Let $l \in L$. Let $E(l)$ denote the set of all p/n pairs covered by literal l.

Definition 4. Literal l covers literal l' if $E(l') \subseteq E(l)$.

The example below shows some important properties of sets of literals and examples. It also intuitively introduces the notion of relevance of a literal.

Example 2. Suppose that we have a domain with two positive examples, $P = \{p_1, p_2\}$, two negative examples $N = \{n_1, n_2\}$, and six literals: three positive literals $L_p = \{l_1, l_2, l_3\}$ and three negative literals $L_n = \{\overline{l_1}, \overline{l_2}, \overline{l_3}\}$, as shown in Table 3.

With this example we wish to illustrate that dealing with positive literals L_p only is insufficient and that both positive and negative literals should be considered.

Take for example literal $\overline{l_1}$ which covers two p/n pairs (p_1/n_1 and p_2/n_1) and the literal l_1 which does not cover any p/n pair at all ($E(l_1) = \emptyset$). Because of that, l_1 can be immediately detected as irrelevant for learning a concept description H.

Take as another example literal l_2. It can be seen that it covers only the p/n pair built of p_1 and n_2 while its logical complement $\overline{l_2}$ covers only the pair built of p_2 and n_1. Although $\overline{l_2}$ is a logical complement of l_2, the sets of p/n pairs covered by l_2 and $\overline{l_2}$ are different, therefore both the literal and its negation need to be considered. Consequently, in hypothesis construction and in literal elimination we should consider a set of literals L consisting of positive and negative literals: $L = L_p \cup L_n$.

4.2. Costs of literals

Assume that costs are assigned to literals. Let $c(l)$ denote the cost of literal $l \in L$. If costs are not assigned, all costs are assumed to be equal to 1, i.e., $\forall l \in L$: $c(l) = 1$.

For the theory of relevance it is actually not important how costs are determined. Cost of a literal can be the encoding length of a literal or any other cost function, implicitly or explicitly using Occam's razor, which will affect the potential inclusion of a literal into a concept description. For example, in the East–West challenge, where cost-sensitive literal reduction was studied in an application of a hybrid genetic algorithm RL–ICET to discover low size-complexity Prolog programs for classifying trains as Eastbound or Westbound [21,22], costs were used as a measure of complexity – the more complex is a literal, the higher is its cost. A possible complexity measure for literals can be found in Ref. [5].

Table 3
Coverage of literals and importance of positive and negative literals

Examples		*Literals*					
		l_1	$\overline{l_1}$	l_2	$\overline{l_2}$	l_3	$\overline{l_3}$
p_1	$\oplus$	*false*	*true*	*true*	*false*	*false*	*true*
p_2	$\oplus$	*false*	*true*	*false*	*true*	*true*	*false*
n_1	$\ominus$	*true*	*false*	*true*	*false*	*true*	*false*
n_2	$\ominus$	*false*	*true*	*false*	*true*	*false*	*true*

4.3. Relevance of literals

Example 2 already identified useless literals as those for which $E(l) = \emptyset$. Such literals can be immediately eliminated from the set of literals L, regardless of their cost and regardless of the properties of other literals.

In order to identify other literals that can be eliminated from the set of literals needed for hypothesis construction, we need to make the assumption that hypothesis construction will be performed by a learner that prefers hypotheses of lower cost (complexity). For instance, learners of this type are learners based on MDL which aim at minimizing the Kolmogorov complexity of theories (see for example Ref. [23]).

Definition 5. Literal l' is irrelevant if there exists another literal $l \in L$ such that l covers l' ($E(l') \subseteq E(l)$) and the cost of l is lower than or equal to the cost of l' ($c(l) \leqslant c(l')$).

Definition 6. Let $H(E, L)$ denote a hypothesis H built from example set E and literal set L.

Lemma 1. *Let $l, l' \in L$. Let $P(l)$ denote the subset of positive examples for which l has value true and $N(l)$ denote the subset of negative examples for which l has value false. $E(l') \subseteq E(l)$ implies $P(l') \subseteq P(l)$ and $N(l') \subseteq N(l)$. The reverse is also true: from $P(l') \subseteq P(l)$ and $N(l') \subseteq N(l)$ it follows that $E(l') \subseteq E(l)$.*

Proof. The proof of this lemma can trivially be verified by contradiction. □

Theorem 2. *If a literal $l' \in L$ is irrelevant then for every complete and consistent hypothesis $H = H(E, L)$ whose description includes literal l', there exists a complete and consistent hypothesis $H' = H(E, L')$, built from the literal set L' that excludes l' ($L' = L \setminus \{l'\}$) which has a lower cost than H ($c(H') \leqslant c(H)$).*

Proof. Suppose that $E(l')$ is not an empty set. If l' is irrelevant, there exists l such that $E(l') \subseteq E(l)$, and therefore $P(l') \subseteq P(l)$ and $N(l') \subseteq N(l)$. This implies that each hypothesis, which is built using literal l' and is complete and consistent for all the training examples, will remain complete and consistent if we substitute every occurrence of literal l' by literal l. Since the cost of l is less than or equal to $c(l')$, the cost of the concept description after the substitution of l' by l will not be greater than before. This proves that for each hypothesis eliminated from the hypothesis space by the elimination of l' there remains at least one other hypothesis that is at least as good as the eliminated one. □

This theorem is the basis of an algorithm for the elimination of irrelevant literals from the initial set of literals L, resulting in a reduced set of relevant literals *RL*. It should again be stressed that the (ir)relevance of literals is defined for the given set of training examples E and for the given set of literals L. The implementation of a cost-sensitive literal elimination algorithm (called REDUCE) is given in Section 5.

4.4. Relevance of features

In this work, the term *feature* is used to denote a positive literal l. Let L_p denote a set of positive literals (features).

In a hypothesis language, the existence of a feature l implies the existence of two complementary literals: a positive literal l and a negative literal $\neg l$. Let L denote the set of positive and negative literals, $L = L_p \cup L_n$.

Since each feature implies the existence of two literals, the necessary and sufficient condition that a feature can be eliminated as irrelevant is that both of its literals are irrelevant.

It must be noted that direct detection of irrelevant features (without conversion to and from the literal form) is not possible except in the trivial case where two (or more) features have identical columns in the table E. Only in this case a feature f exists whose literals f and $\neg f$ cover both literals g and $\neg g$ of some other feature. In a general case if a literal of feature f covers some literal of feature g then the other literal of feature g is not covered by the other literal of feature f. But it can happen that this other literal of feature g is covered by a literal of some other feature h. This means that although there is no such feature f that covers both literals of feature g, feature g can still turn out to be irrelevant.

This needs to be considered in literal elimination. One has to eliminate irrelevant literals from the literal set $L = L_p \cup L_n$ in order to get a set of relevant literals. On the other hand, if we wish to construct a set of relevant features, this set should include all the features which have at least one of their literals in the relevant literal set. For the formalization of these notions and the empirical verification of claims see Section 6.

4.5. Relevance of examples

Besides the theory of relevance for literals we can similarly define the theory of relevance for training examples. Its aim is to eliminate irrelevant examples from the training set E in such a way that any hypothesis built of literals from L that is correct (complete and consistent) for all the examples from the reduced set of examples RE will also be correct for all the examples of the initial training set E. By reducing the number of examples in the learning space, the search for the final concept description by an inductive learning algorithm can be made more efficient, without fearing that the result of learning will be incorrect.

At this point we have to make sure that the reader is aware of the preconditions we have made for the validity of our claims. The elimination of irrelevant examples may be applied only in exact domains or when a learner using some minimization/compression measure is used for learning from the reduced example set. Caution is needed because example elimination can change the distribution of positive and negative examples, this distribution being crucial for learners using statistical measures. Notice that, on the other hand, literal elimination can be applied in general without restrictions.

Definition 7. If an example is positive, $p \in P$, then $L(p)$ represents a subset of literals in L that have value *true* for example p. If n is a negative example, $n \in N$, then $L(n)$ represents a subset of literals in L that have value *false* for the example n.

Definition 8. Example e covers e' if $L(e) \subseteq L(e')$ and if examples e and e' are either both positive or both negative.

Definition 9. An example e' is irrelevant if there exists at least one example $e \in E$ such that e covers e'.

Theorem 3. *If a positive example $e' \in E$ is irrelevant then for every complete and consistent hypothesis $H' = H(E', L)$ built from example set E' that excludes e ($E' = E \setminus \{e'\}$) it holds that H' covers e'.*

A dual theorem holds for negative examples: If a negative example $e' \in E$ is irrelevant then for every complete and consistent hypothesis $H' = H(E', L)$ built from example set $E' = E \setminus \{e'\}$ it holds that H' does not cover e'.

Proof. Suppose that e' is a positive irrelevant example eliminated from E because it was covered by e^*. If the example e^* is also eliminated from E because it was covered by another example e then it is true that e covers e' as well (the property of transitivity holds). It means that after all the eliminations of irrelevant examples from E, at least one example that covers the eliminated example e' must remain in the reduced training set RE. Let us use the label e to denote the remaining example. If any literal l is *true* for example e then l is also true for e' (see Definitions 8–10). This means that if we form any conjunctive and/or disjunctive description H' as a combination of literals that are *true* for e then H' must be true for e' as well. This proves that e' is covered by H'. An analogous proof for e' being a negative example can be made. □

Example 3. Reconsider Table 3 from Example 2 where p_1, p_2 are positive examples and n_1, n_2 are negative examples. No example is covered by another example.

On the other hand, consider the situation presented in Table 4. Example p_1 covers p_2, therefore p_2 can be eliminated from the training set. Notice that this situation actually occurs in the problem of learning family relations (see Table 8 of Section 5.3).

The presented theory of relevance is the basis of algorithms for irrelevant literal and example elimination, introduced in the following Section.

5. Literal and example elimination algorithms

Recall Lemma 1 from Section 4.3 stating that for $l, l' \in L$, $E(l') \subseteq E(l)$ is equivalent to $P(l') \subseteq P(l)$ and $N(l') \subseteq N(l)$. This property enables us to execute the search

Table 4
An example showing the coverage of examples

Examples		*Literals*		
		l_1	l_2	l_3
p_1	$\oplus$	*true*	*false*	*true*
p_2	$\oplus$	*true*	*true*	*true*
n_1	$\ominus$	*false*	*false*	*true*
n_2	$\ominus$	*true*	*false*	*false*

for irrelevant literals over two separate example sets (P and N) instead of over the much larger set of p/n pairs.

5.1. Covering tables

The tranformation of examples into truth-value tuples is sufficient for our theoretical investigation of the problem of relevance. However, for efficiency reasons of a practical implementation of literal and example elimination algorithms presented in Section 5.2, a different transformation step is recommended: tuples of truth values are transformed into bitstrings consisting of values 1 and 0. This results in two separate matrices for positive and negative examples (the so-called P and N *tables*). Section 5.2 is based on this representation which allows for efficient bitstring manipulation.

In brief, the transformation goes as follows. For a positive example, a bitstring in a P table has value 1 if a literal has value *true*, and 0 if a literal has value *false*. For a negative example, the bit assignment is just the opposite: 1 is assigned to value *false* and 0 to value *true*.

Intuitively, in a binary decision problem (two classes: $\oplus$ and $\ominus$) literals that have value 1 (*true*) for a positive example and value 0 (*false*) for a negative example have the greatest discriminating power for distinguishing between the two decision classes. Thus, stating it informally, literals with a larger number of 1 elements in the corresponding column of the table are better candidates for inclusion in a hypothesis and should not be eliminated. On the other hand, literals with a smaller discriminating power, having many 0 elements are less promising for inclusion in a hypothesis.

Similar intuitions are the basis for example elimination. In contrast with the above situation, where a literal is more relevant if it has more 1 elements in the corresponding column, an example is more relevant if it has less 1 elements in the corresponding row. To illustrate this, suppose that in P table we have a row 01001 for a positive example whose truth-value tuple is (*false*, *true*, *false*, *false*, *true*) for literals l_1, l_2, l_3, l_4, l_5. This example is covered by a conjunctive description $l_2 \wedge l_5$. Another positive example 01101 which has value 1 in the same columns as example 01001 is covered by the same conjunctive description, since $l_2 \wedge l_3 \wedge l_5$ is more specific than $l_2 \wedge l_5$, and $l_2 \wedge l_5$ covers $l_2 \wedge l_3 \wedge l_5$. Thus, intuitively speaking, the second positive example is irrelevant and we do not need to consider it for learning since only considering example 01001 guarantees the coverage of 01101 as well.

5.2. Elimination algorithms

Algorithms 1 and 2 implement the relevance theory adapted to the above definitions of P and N table. Initial versions of these algorithms, disregarding costs, were developed within the ILLM learner [9]. Algorithm 1 (called REDUCE) can be easily transformed into an iterative algorithm that can be used during the process of generation of literals [22]. In this way the irrelevant literals with respect to the already generated literals can be eliminated without even entering the P and N tables. This approach can significantly reduce the space required for storing P and N tables.

Algorithm 1. *Cost-sensitive literal elimination* (*REDUCE*)
Procedure *LiteralElimination*$(P, N, CL, L, RP, RN, RL)$
Given: CL – costs of literals in L
Input: P, N – tables of positive and negative examples, L – set of literals
Output: RP, RN – reduced tables of positive and negative examples,
RL – reduced set of literals
$RP \leftarrow P$, $RN \leftarrow N$, $RL \leftarrow L$
for $\forall\ l_i \in RL$, $i \in [1, |L|]$ **do**
if l_i has value 0 (*false*) for all rows of RP **then**
eliminate l_i from RL
eliminate column l_i from RP and RN tables
if l_i has value 0 (*true*) for all rows of RN **then**
eliminate l_i from RL
eliminate column l_i from RP and RN tables
if l_i is covered by any $l_j \in RL$ for which $c(l_j) \leqslant c(l_i)$ **then**
eliminate l_i from RL
eliminate column l_i from RP and RN tables
endfor

Algorithm 2. *Example elimination*
Procedure *ExampleElimination*(P, N, E, RP, RN, RE)
Input: P, N – tables of positive and negative examples, L – set of literals
Output: RP, RN – reduced tables of positive and negative examples,
RE – reduced set of examples
$RP \leftarrow P$, $RN \leftarrow N$, $RL \leftarrow L$
for $\forall\ e_i \in RE$, $i \in [1, |E|]$ **do**
if positive e_i is covered by any $e_j \in RP$ **then**
eliminate e_i from RE
eliminate row e_i from RP table
if negative e_i is covered by any $e_j \in RN$ **then**
eliminate e_i from RE
eliminate row e_i from RN table
endfor

The complexity of Algorithm 1 is $\mathcal{O}(|L|^2 \times |E|)$ while the complexity of Algorithm 2 is $\mathcal{O}(|L| \times |E|^2)$ where $|L|$ is the number of literals and $|E|$ is the number of examples.

Algorithm 3. *Iterative elimination of literals and examples*
Procedure *IterativeElimination*(E, CL, L, RE, RL)
Given: CL (costs of literals in L)
Input: E (initial example set), L (initial literal set)
Output: RE – reduced table of positive and negative examples,
RL – reduced set of literals
build P and N tables from E
reset loop counter $i \leftarrow 0$
repeat
call *LiteralElimination*$(P, N, CL, L, RP, RN, RL)$
if $i > 0$ and $RL = L$ **then** exit repeat loop

$L \leftarrow RL$, $P \leftarrow RP$, $N \leftarrow RN$
call $ExampleElimination(P, N, E, RP, RN, RE)$
if $RE = E$ **then** exit repeat loop
$E \leftarrow RE$, $P \leftarrow RP$, $N \leftarrow RN$
$i \leftarrow i + 1$
endrepeat

Algorithm 3 combines Algorithms 1 and 2 into an iterative loop in which both irrelevant literals and examples are eliminated. It must be noted that the elimination process must be iterative because example eliminations can enable that some literals become irrelevant although they have not been irrelevant before. The same is true for literals. After literal eliminations some examples can become irrelevant although they have not been irrelevant before. Note that the final result of irrelevant literal and example elimination is unique regardless of the order of eliminations. It is not important whether Algorithm 3 starts with literal or example eliminations. The order of eliminations does not matter because every literal that is irrelevant remains irrelevant also after some example eliminations. The same is true for examples.

5.3. An example run of elimination algorithms

To illustrate an application of the presented theory of relevance and a run of the elimination algorithms consider again the example of learning an intensional definition of the relation *daughter*, introduced in Section 3.2, where the LINUS transformation results in Table 2.

As already discussed in Example 2 of Section 4.1 and more formally in Section 4.4, for our study of relevance the negative counterparts of the generated literals L_p need to be considered as well, which gives rise to nine more literals $L_n = \{\overline{f(X)}, \overline{f(Y)}, \ldots\}$ in the transformed table of examples E for the daughter learning problem. A part of the truth-value table for only 6 out of 18 literals constituting $L = L_p \cup L_n$ is presented in Table 5.

In Table 6, Table 5 is transcribed into the form of P and N tables. This relation learning problem will be used to illustrate the process of literal and example elimination. In the example it is assumed that all literals have equal costs.

From Table 6 it can be noticed that literals $\overline{f(X)}$, $\overline{m(Y)}$, and $\overline{p(Y,X)}$ are useless, either having 0 values in all P rows or 0 values in all N rows. In the starting P and N tables there are in total 13 such literals. After their elimination, five literals remain. These literals are presented in Table 7.

It can now be noticed that columns $f(X)$ and $\overline{m(X)}$ are identical as well as columns $m(Y)$ and $\overline{f(Y)}$. As identical columns cover each other, the negative literals

Table 5
Six out of 18 literals in the transformed example set for learning the *daughter* relationship

C	*Examples* $d(X,Y)$	*Literals*					
		$f(X)$	$\overline{f(X)}$	$m(Y)$	$\overline{m(Y)}$	$p(Y,X)$	$\overline{p(Y,X)}$
$\oplus$	p_1	*true*	*false*	*false*	*true*	*true*	*false*
$\oplus$	p_2	*true*	*false*	*true*	*false*	*true*	*false*
$\ominus$	n_1	*false*	*true*	*false*	*true*	*true*	*false*
$\ominus$	n_2	*true*	*false*	*false*	*true*	*false*	*true*

Table 6
A part of P and N tables for the *daughter* relationship problem

Examples $d(X,Y)$	*Literals*					
	$f(X)$	$\overline{f(X)}$	$m(Y)$	$\overline{m(Y)}$	$p(Y,X)$	$\overline{p(Y,X)}$
P						
p_1	1	0	0	1	1	0
p_2	1	0	1	0	1	0
N						
n_1	1	0	1	0	0	1
n_2	0	1	1	0	1	0

Table 7
P and N tables after the elimination of useless literals

Examples $d(X,Y)$	*Literals*				
	$f(X)$	$m(Y)$	$p(Y,X)$	$\overline{f(Y)}$	$\overline{m(X)}$
P					
p_1	1	0	1	0	1
p_2	1	1	1	1	1
N					
n_1	1	1	0	1	1
n_2	0	1	1	1	0

can be eliminated from the P and N tables of Table 7. In this way, Table 8 with only 3 columns is obtained.

It is interesting to notice that only after these column eliminations have been performed, there is finally a possibility also for a row elimination. Example p_2 can be eliminated since it is covered by example p_1. The result is given in Table 9.

Now there is a possibility for an additional column elimination because the literal $m(Y)$ has value 0 in all rows of the P table. The final P and N tables with only two columns and three rows are presented in Table 10.

In the selected example of relation learning, the two literals that remain after all eliminations represent the minimal subset of literals that needs to be used in hypothesis construction. Actually, by now running a propositional learner within LINUS, e.g., CN2 [4], the following if–then rule can be trivially induced:

$$daughter(X,Y) = true \textbf{ if } [female(X) = true] \wedge [parent(Y,X) = true]$$

Table 8
P and N tables after column eliminations

Examples	*Literals*		
	$f(X)$	$m(Y)$	$p(Y,X)$
P			
p_1	1	0	1
p_2	1	1	1
N			
n_1	1	1	0
n_2	0	1	1

Table 9
P and *N* tables after row elimination

Examples	*Literals*		
	$f(X)$	$m(Y)$	$p(Y,X)$
P			
p_1	1	0	1
N			
n_1	1	1	0
n_2	0	1	1

Table 10
Reduced *P* and *N* tables

Examples	*Literals*	
	$f(X)$	$p(Y,X)$
P		
p_1	1	1
N		
n_1	1	0
n_2	0	1

If transformed to clausal form as done in LINUS, the resulting hypothesis is as follows:

$$daughter(X,Y) \leftarrow female(X), parent(Y,X)$$

Notice that in our example the task of an inductive learning algorithm has been significantly simplified due to irrelevant literal and example elimination.

6. Utility study of literal reduction for LINUS

The experiments described in this section study the usefulness of the irrelevant literal elimination algorithm REDUCE used as part of the LINUS transformation algorithm.

6.1. Experimental settings and domains

Three settings are used, in which the input to the irrelevant literal elimination algorithm is a set of examples E described as truth-value tuples of the following sets of literals:

Setting 1: Use LINUS to generate positive literals L_p called *features*. Select from L_p the set of *relevant features* RL_p.

Setting 2: Take L_p from Setting 1. Generate negative literals, denoted by $L_n = \{\neg l \mid l \in L_p\}$. Select from literals $L = L_p \cup L_n$ the set of *relevant literals* $RL = RL_p \cup RL_n$.

Setting 3: Take $L = L_p \cup L_n$ from Setting 2. Select from L the set of *relevant features RF*. To do so, first select relevant literals RL as in Setting 2, and then construct the reduced set of features including all the features which have at least one of their literals in the reduced literal set: $RF = RL_p \cup RL_{np}$, where $RL_{np} = \{l \mid \neg l \in RL_n\}$.

The experimental setting was designed with the goal to verify that Setting 3 is the best setting when using LINUS for solving ILP problems, assuming that the propositional learner used in LINUS is able to generate hypotheses with negated features; if this is not the case, Setting 2 is the best setting for LINUS. It is also shown that Setting 1, in which negative literals are not generated and in which features are eliminated as literals, may lead to inappropriate reductions.

The performance of the literal reduction algorithm REDUCE is evaluated on learning tasks taken from Ref. [19] (the original references for the individual tasks can be found in this source). The first three domains involve a very small number of examples: family relationships, arches and the Eleusis card game consisting of three different training sets, whereas the fourth domain, the King–Rook–King chess endgame, involves five training sets of 100 examples each as well as one training set of 5000 examples.

Each domain is described by the target predicate, predicates in the background knowledge, and the list of positive literals (features) constructed by LINUS (this set is denoted by L_p).

6.2. Learning family relationships

In the family relationships learning task, two stylized families of 12 members each are given, as shown in Fig. 1, taken from Ref. [35] (first described by Hinton in 1986).

The task is to learn the definition of the target predicate $mother(A, B)$ from examples of this relation and the background relations $father(X, Y)$, $wife(X, Y)$, $son(X, Y)$ and $daughter(X, Y)$. Negative examples are generated under the closed-world assumption. There are 10 positive and 50 negative examples.

LINUS constructs eight features $father(A, B)$, $father(B, A)$, $daughter(A, B)$, $daughter(B, A)$, $son(A, B)$, $son(B, A)$, $wife(A, B)$, $wife(B, A)$. In addition, LINUS creates 18 senseless features, $father(A, A)$, $father(B, B)$, $daughter(A, A)$, $daughter(B, B)$, $son(A, A)$, $son(B, B)$, $wife(A, A)$, $wife(B, B)$, which are irrelevant by definition since they are *false* for all the training examples.

From 10 positive and 50 negative examples, LINUS induces the following hypothesis [19].

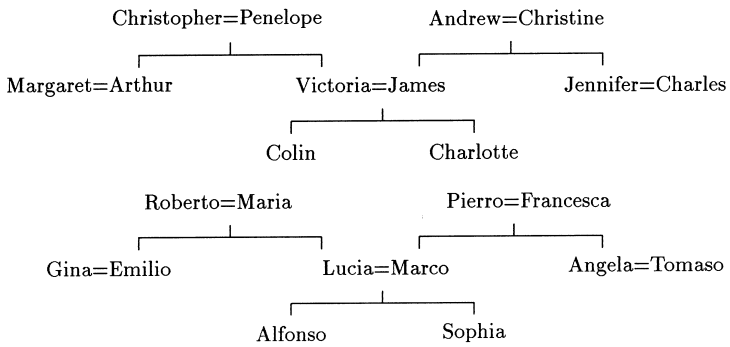

Fig. 1. Two family trees, where = means *married to*.

$$
\begin{aligned}
mother(A,B) \quad &\leftarrow \quad daughter(B,A)\\
&\qquad\quad\ \ not\ father(A,B)\\
mother(A,B) \quad &\leftarrow \quad not\ daughter(B,A)\\
&\qquad\quad\ \ son(B,A)\\
&\qquad\quad\ \ not\ father(A,B)
\end{aligned}
$$

Setting 1: L_p contains 16 features. Only two features are relevant, $RL_p = \{daughter(B,A), son(B,A)\}$, all others are eliminated. These two features do not suffice for inducing the target hypothesis. Since the negative literals are not available, the algorithm eliminates the feature $father(A,B)$ which is needed to induce the above hypothesis.

Setting 2: L contains 32 literals. The following eight literals are relevant: $RL = \{daughter(B,A), son(B,A), not\ father(A,B), not\ father(B,A), not\ son(A,B), not\ wife(A,B), not\ wife(B,A), not\ daughter(A,B)\}$. The other eight literals are eliminated as irrelevant. All the literals needed for hypothesis construction remain in the relevant literal set, including $not\ father(A,B)$.

Setting 3: There are eight relevant features: $RF = \{daughter(B,A),\ son(B,A), father(A,B), father(B,A), daughter(A,B), son(A,B), wife(A,B), wife(B,A)\}$. Again, all the features needed for hypothesis construction remain in the relevant feature set, including $not\ father(A,B)$.

In Settings 2 and 3, eight literals out of 32 are relevant. Irrelevant literal elimination is not particularly effective due to a large amount of negative examples which prevent the elimination of negative literals.

6.3. Learning the concept of an arch

In this example, taken from Ref. [35] (first described by Winston in 1975), two given objects are arches (positive examples: the first and fourth object) and others are not (negative examples). The original problem consisted of two positive and two negative examples (the first four objects in Fig. 2).

For inducing the target relation $arch(A,B,C)$, stating that A, B and C form an arch with columns A and B and lintel C, the following background relations were used: $supports(X,Y)$, $left\_of(X,Y)$, $touches(X,Y)$, $brick(X)$, $wedge(X)$ and $parallelepiped(X)$.

LINUS constructs the following 27 features: $supports(A,B)$, $supports(A,C)$, $supports(B,A)$, $supports(B,C)$, $supports(C,A)$, $supports(C,B)$, $leftof(A,B)$, $leftof(A,C)$, $leftof(B,A)$, $leftof(B,C)$, $leftof(C,A)$, $leftof(C,B)$, $brick(A)$, $brick(B)$, $brick(C)$, $touches(A,B)$, $touches(A,C)$, $touches(B,A)$, $touches(B,C)$, $touches(C,A)$, $touches(C,B)$, $wedge(A)$, $wedge(B)$, $wedge(C)$, $parallelepiped(A)$, $parallelepiped(B)$, $parallelepiped(C)$. In addition, nine senseless features are constructed: $supports(A,A)$, $supports(B,B)$, ..., $touches(C,C)$.

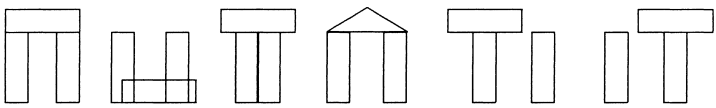

Fig. 2. Arches and near misses.

6.3.1. First example set

From two positive and two negative examples, LINUS induces the following hypothesis [19].

$$arch(A,B,C) \leftarrow supports(A,C) \\ \quad\quad\quad\quad\quad\quad not\ touches(A,B)$$

Setting 1: Only two features are relevant: $supports(A,C)$ and $wedge(C)$. The feature $touches(A,B)$ is eliminated despite the fact that it is needed for hypothesis formation. Similar as in the family example, the feature that is (but should not have been) eliminated appears in the hypothesis as a negative literal.

Setting 2: $RL = \{supports(A,C), wedge(C), not\ touches(A,B)\}$. All the needed literals remain available after the elimination.

Setting 3: There are again three relevant features: $supports(A,C)$, $wedge(C)$ and $touches(A,B)$. All the needed features remain available.

6.3.2. Second example set

From the entire set of examples shown in Fig. 2, consisting of two positive and four negative examples, LINUS induces the following hypothesis [19].

$$arch(A,B,C) \leftarrow supports(A,C) \\ \quad\quad\quad\quad\quad\quad supports(B,C) \\ \quad\quad\quad\quad\quad\quad not\ touches(A,B)$$

Setting 1: Three features are relevant: $supports(A,C)$, $supports(B,C)$ and $wedge(C)$. The feature $touches(A,B)$ is eliminated despite the fact that it is needed for hypothesis formation.

Setting 2: $RL = \{supports(A,C), supports(B,C), wedge(C), not\ touches(A,B)\}$. All the needed literals remain available after the elimination.

Setting 3: There are four relevant features: $supports(A,C)$, $supports(B,C)$, $wedge(C)$ and $touches(A,B)$. Again, all the needed features remain available.

6.4. Learning rules that govern card sequences

The Eleusis learning problem, taken from Ref. [35], was first described by Dietterich and Michalski in 1986. In the Eleusis card game, the dealer invents a secret rule specifying conditions under which a card can be added to a sequence of cards. The players attempt to add a card to the current sequence. If a card is a legal successor, it is placed to the right of the last card, otherwise it is placed under the last card. The horizontal *main line* represents the sequence as developed so far, while the vertical *side lines* show incorrect plays. Three layouts, reproduced from Ref. [35], are given in Fig. 3.

Each card other than the first in the sequence provides an example for learning the target relation *can_follow*. The example is a positive example if the card appears in the main line, and it is a negative example if it is in a side line.

In all the layouts, the target predicate is $can\_follow(R,S,PR,PS,CS,CL)$, where arguments denote: a card of R – rank and S – suit, that follows a card of PR – previous rank and PS – previous suit, CS – number of consecutive cards in same suit, CL – number of consecutive cards in same color.

main line	A♡	7♣	6♣	9♠	10♡	7♡	10♢	J♣	A♢	4♡	8♢	7♣	9♠
side lines			K♢			5♠		Q♢		3♠			9♡
			J♡										6♡
main (ctd)	10♣	K♠	2♣	10♠	J♠								
side (ctd)		Q♡											
		A♢											
main line	J♣	4♣	Q♡	3♠	Q♢	9♡	Q♣	7♡	Q♢	9♢	Q♣	3♡	K♡
side lines	K♣	5♠				4♠		10♢					
		7♠											
main (ctd)	4♣	K♢	6♣	J♢	8♢	J♡	7♣	J♢	7♡	J♡	6♡	K♢	
mainline	4♡	5♢	8♣	J♠	2♣	5♠	A♣	5♠	10♡				
side line	7♣	6♠	K♣	A♡		6♣		A♠					
	J♡	7♡	3♡	K♢									
	4♣	2♣		Q♠									
	10♠	7♠											
	8♡	6♢											
	A♢	6♡											
	2♢	4♣											

Fig. 3. Three layouts of the Eleusis card game.

Background relations that can be used in induced clauses are the following: *precedes_rank*(X, Y), *precedes_suit*(X, Y), *lower_rank*(X, Y), *same_color*(X, Y), *face*(X), *odd_rank*(X), and *odd_num*(X).

LINUS generates 14 features: *precedes_rank*(R, PR), *precedes_rank*(PR, R), *precedes_suit*(S, PS), *precedes_suit*(PS, S), *face*(R), *face*(PR), *lower_rank*(R, PR), *lower_rank*(PR, R), *same_color*(S, PS), *same_color*(PS, S), *odd_rank*(R), *odd_rank*(PR), *odd_num*(CS), *odd_num*(CL), as well as eight senseless features.

6.4.1. First layout

In the first layout, the intended dealer's rule is: 'Completed color sequences must be of odd length and a male card may not appear next to a female card'. LINUS cannot discover the intended rule, because no information on the gender of cards is encoded in the background relations. From 17 positive and 9 negative examples, LINUS induces the following clauses [17].

$$\begin{aligned} &can\_follow(R, S, PR, PS, CS, CL) \leftarrow same\_color(S, PS)\\ &can\_follow(R, S, PR, PS, CS, CL) \leftarrow odd\_num(CL), odd\_rank(R)\\ &can\_follow(R, S, PR, PS, CS, CL) \leftarrow not\ face(R), lower\_rank(PR, R) \end{aligned}$$

Setting 1: All the features are kept as relevant except for *same_color*(PS, S), which is a symmetric variant of *same_color*(S, PS).

Setting 2: Out of 44 literals, 21 literals are relevant. In addition to the initial 13 literals as in Setting 1, *RL* includes literals *not precedes_rank*(R, PR), *not precedes_rank*(PR, R), *not precedes_suit*(S, PS), *not precedes_suit*(PS, S), *not face*(R), *not face*(PR), *not odd*(R), *not odd*(PR).

Setting 3: Relevant literals are identical as in Setting 1.

In all the settings, all literals needed for hypothesis formation are available as relevant.

6.4.2. Second layout

In the second layout, given 24 positive and 5 negative examples, LINUS correctly induces the intended rule: 'Play alternate face and non-face cards' [19].

$$can\_follow(R,S,PR,PS,CS,CL) \leftarrow face(R), not\ face(PR)$$
$$can\_follow(R,S,PR,PS,CS,CL) \leftarrow face(PR), not\ face(R)$$

Setting 1: Nine features are relevant, including all features needed for hypothesis formation: $lower\_rank(R,PR)$, $precedes\_suit(S,PS)$, $precedes\_suit(PS,S)$, $face(R)$, $face(PR)$, $same\_color(S,PS)$, $odd(R)$, $odd(PR)$, $odd(CL)$.

Setting 2: Out of 44 literals, 18 literals are relevant: in addition to the literals of Setting 1, the following literals are relevant: $not\ precedes\_rank(PR,R)$, $not\ precedes\_suit(S,PS)$, $not\ precedes\_suit(PS,S)$, $not\ face(PR)$, $not\ odd(R)$, $not\ odd(PR)$, $not\ odd(CS)$, $not\ odd(CL)$, $not\ same\_color(S,PS)$.

All the literals, which are used in the hypothesis are listed here, although not all are kept in the appropriate form. For example, $not\ face(R)$ is eliminated because it is covered by the literal $lower\_rank(R,PR)$. This is acceptable: the induced hypothesis would look different but would cover the same positive examples (if the previous card is a face card, the next one must not be a face card, therefore it has a lower rank). This situation occurs due to a small number of negative examples.

Setting 3: Eleven features are relevant: nine features of Setting 1, as well as $precedes\_rank(PR,R)$ and $odd(CS)$. All features needed for hypothesis formation are available as relevant.

6.4.3. Third layout

In the third layout, the intended rule is: 'Play a higher card in the suit preceding that of the last card; or, play a lower card in the suit following that of the last card'. LINUS discovers an approximation of the rule: 'Play a higher or equal card in the suit preceding that of the last card; or, play a lower card in the suit following that of the last card'. From 8 positive and 21 negative examples, LINUS generates the following hypothesis [19]:

$$can\_follow(R,S,PR,PS,CS,CL) \leftarrow lower\_rank(R,PR), precedes\_suit(PS,S)$$
$$can\_follow(R,S,PR,PS,CS,CL) \leftarrow lower\_rank(PR,R), precedes\_suit(S,PS)$$

Setting 1: Three features that are are eliminated are $precedes\_rank(R,PR)$, $same\_color(PS,S)$, and $odd(CS)$, the others are relevant.

Setting 2: There are 17 relevant literals: 11 features from Setting 1, and in addition: $not\ precedes\_rank(R,PR)$, $not\ precedes\_rank(PR,R)$, $not\ face(R)$, $not\ face(PR)$, $not\ same\_color(S,PS)$, $not\ odd(R)$.

Setting 3: There are 12 relevant features: 11 features from Setting 1 and $precedes\_rank(R,PR)$.

In all the settings, all literals needed for hypothesis formation are available as relevant.

6.5. Learning illegal chess endgame positions

In the chess endgame domain White King and Rook vs. Black King, taken from [35] (first described in Ref. [29]), the target relation $illegal(A,B,C,D,E,F)$ states

whether a position where the White King is at file and rank (A, B), the White Rook at (C, D) and the Black King at (E, F) is an illegal White-to-move position. For example, $illegal(g, 6, c, 7, c, 8)$ is a positive example, i.e., an illegal position.

Given the background knowledge predicates, LINUS creates the following 36 features.

Twelve for equal/2: $equal(A, C)$, $equal(A, E)$, $equal(C, A)$, $equal(C, E)$, $equal(E, A)$, $equal(E, C)$, $equal(B, D)$, $equal(B, F)$, $equal(D, B)$, $equal(D, F)$, $equal(F, B)$, $equal(F, D)$.
Six for adj_file/2: $adj\_file(A, C)$, $adj\_file(A, E)$, $adj\_file(C, A)$, $adj\_file(C, E)$, $adj\_file(E, A)$, $adj\_file(C, E)$.
Six for adj_rank/2: $adj\_rank(B, D)$, $adj\_rank(B, F)$, $adj\_rank(D, B)$, $adj\_rank(D, F)$, $adj\_rank(F, B)$, $adj\_rank(F, D)$.
Six for less_file/2: $less\_file(A, C)$, $less\_file(A, E)$, $less\_file(C, A)$, $less\_file(C, E)$, $less\_file(E, A)$, $less\_file(E, C)$.
Six for less_rank/2: $less\_rank(B, D)$, $less\_rank(B, F)$, $less\_rank(D, B)$, $less\_rank(D, F)$, $less\_rank(F, B)$, $less\_rank(F, D)$.

In addition, LINUS creates 18 senseless features such as $equal(A, A)$, $equal(B, B)$, ..., $less\_rank(F, F)$, which are irrelevant by definition since they are *false* for all the training examples.

By considering both positive and negative literals, this leads to examples described by 108 literals.

In this experiment, the definition of the target relation $illegal(A, B, C, D, E, F)$ is induced from five sets of 100 examples each. The numbers of positive examples are 49, 33, 32, 39, 37, respectively. The hypothesis is the same for all five sets of examples:

$$
\begin{aligned}
illegal(A,B,C,D,E,F) &\leftarrow equal(C,E)\\
illegal(A,B,C,D,E,F) &\leftarrow equal(D,F)\\
illegal(A,B,C,D,E,F) &\leftarrow adj\_file(A,E), equal(B,F)\\
illegal(A,B,C,D,E,F) &\leftarrow adj\_file(A,E), adj\_rank(B,F)\\
illegal(A,B,C,D,E,F) &\leftarrow equal(A,E), adj\_rank(B,F)\\
illegal(A,B,C,D,E,F) &\leftarrow equal(A,E), equal(B,F)
\end{aligned}
$$

These clauses may be paraphrased as: 'A position is illegal if the Black King is on the same rank or file as (i.e., is attacked by) the Rook, or the White King and the Black King are next to each other, or the White King and the Black King are on the same square'. Although these clauses are neither consistent nor complete, they correctly classify 98.5% of the unseen cases.

The overall results of literal elimination are shown in Table 11. In all the five domains, Setting 3 results in 24 features; all the symmetric features of the initial set of 36 features are eliminated, for example: $equal(A, C)$ contains the same information as $equal(C, A)$, hence one of them is irrelevant. In two domains, in Setting 1 one additional feature is eliminated: either $adj\_file(C, E)$ or $adj\_rank(B, D)$. Setting 2 keeps 41–45 literals, depending on the domain. The individual results for the five training sets are given in Appendix. In all the settings, all the needed literals remain available in the relevant literal set.

Table 11
Results of the utility study for LINUS

Domain	*Ex*	*Pos*	*Setting* 1	*Setting* 2	*Setting* 3
Family	60	10	$8+8 \rightarrow 2$	$16+16 \rightarrow 8$	$16+16 \rightarrow 8$
Arch1	4	2	$27+9 \rightarrow 2$	$54+18 \rightarrow 3$	$54+18 \rightarrow 3$
Arch2	6	2	$27+9 \rightarrow 3$	$54+18 \rightarrow 4$	$54+18 \rightarrow 4$
Eleusis1	26	17	$14+8 \rightarrow 13$	$28+16 \rightarrow 21$	$28+16 \rightarrow 13$
Eleusis2	29	24	$14+8 \rightarrow 9$	$28+16 \rightarrow 18$	$28+16 \rightarrow 11$
Eleusis3	29	8	$14+8 \rightarrow 11$	$28+16 \rightarrow 17$	$28+16 \rightarrow 12$
Krk100-1	100	49	$36+18 \rightarrow 24$	$72+36 \rightarrow 42$	$72+36 \rightarrow 24$
Krk100-2	100	33	$36+18 \rightarrow 23$	$72+36 \rightarrow 41$	$72+36 \rightarrow 24$
Krk100-3	100	32	$36+18 \rightarrow 24$	$72+36 \rightarrow 42$	$72+36 \rightarrow 24$
Krk100-4	100	39	$36+18 \rightarrow 24$	$72+36 \rightarrow 45$	$72+36 \rightarrow 24$
Krk100-5	100	37	$36+18 \rightarrow 23$	$72+36 \rightarrow 41$	$72+36 \rightarrow 24$

6.6. Summary and further work

Table 11 summarizes the experimental results. The meaning of the abbreviations are as follows: *Ex* – number of examples, *Pos* – number of positive examples, *Setting* 1–3 – reductions of literals in different settings.

The effectiveness of the literal elimination algorithm REDUCE depends on several parameters. The most important is the shape of table E of examples described by literals L. If this table is high (large number of examples in E), the chance for literal coverage is low, therefore not many literals will be eliminated as irrelevant. On the other hand, if the table is wide, there are more literals and the chance of coverage is higher. The algorithm is more effective when there are many literals and a small number of examples (e.g., the arch domain). The minimal result achieved, regardless of the shape of the table, is that REDUCE will always discover and eliminate redundant literals which are senseless or symmetric.

In addition to experiments concerning irrelevant literal elimination, and experiment involving also irrelevant example elimination was performed on an enlarged training set of 5000 examples of King–Rook–King chess endgame positions, described by 108 literals. In this experiment, the application of example and literal elimination algorithms reduces the number of examples from 5000 to 1216, and the number of literals from 108 to 48. The reduction in the number of examples is significant, whereas the drop from 108 to 48 literals is only due to the elimination of senseless literals, and symmetric literals for predicates $equal/2$, $adj\_file/2$ and $adj\_rank/2$. For example, literal $equal(X,Y)$ is symmetric which means that $equal(X,Y) = equal(Y,X)$, therefore only one of these two literals should remain in the relevant literal set. Notice that this reduction could have been achieved in LINUS itself by declaring the predicate $equal/2$ as symmetric.

In further work, problems with higher dimensionality will be studied, for which literal generation and selection should be interleaved. The algorithm REDUCE can easily be reimplemented as part of the LINUS and DINUS algorithms to enable incremental literal generation and selection: every time a new literal is constructed, an attempt should be made to eliminate this literal or one of the previously introduced literals. This would significantly reduce the complexity of the LINUS and DINUS transformation approaches to ILP.

The switch from the LINUS to the DINUS learning framework can be seen as bias shift (notice that DINUS is not restricted to constrained clauses but allows new variables in clause body to be introduced by determinate literals). Using the incremental literal generation and selection procedure, one could start by generating literals which do not introduce new variables first, and eliminating the ones that are irrelevant. If there is no solution to the learning problem in this language bias (i.e., if the set of literals generated is not discriminating), one might switch to a weaker language bias within DINUS (with a greater depth of new variables) and continue the incremental generation and selection of literals until a discriminating literal set is found.

7. Reducing the search of refinement graphs

This section studies the utility of literal elimination in the search of refinement graphs.

7.1. Refinement graphs

Recall that learning can be viewed as search, either heuristic or exhaustive [27]. The states in the search space are descriptions in the hypothesis space defined by the language of hypotheses $\mathscr{L}$, and the goal is to find one or more states satisfying the quality criterion (e.g., completeness and consistency). A learner can be described in terms of the structure of its search space, its search heuristics and search strategy.

In this study, the structure of the hypothesis space is defined by a specialization operator which, for a selected language bias $\mathscr{L}$ and given background knowledge $\mathscr{B}$, defines the so-called *refinement graph*.

Let $\mathscr{L}$ be a set of program clauses. We assume the following definitions, adapted from Ref. [38].

Definition 10. Clause c is at least as general as clause c' ($c \leqslant c'$) if c θ-subsumes c', i.e., if there exists a substitution θ, such that $c\theta \subseteq c'$. Clause c is more general than c' ($c < c'$) if $c \leqslant c'$ holds and $c' \leqslant c$ does not. In this case, we say that c' is a specialization of c.

Definition 11. A refinement operator ρ is a mapping from $\mathscr{L}$ to finite subsets of $\mathscr{L}$, $\rho : \mathscr{L} \rightarrow \mathscr{P}(\mathscr{L})$, such that all clauses in $\rho(c)$ are specializations of the clause c.

Definition 12. Let ρ be a refinement operator and c and c' two clauses in $\mathscr{L}$. Clause c' is a refinement of clause c if c' is a specialization of c, i.e., if $c' \in \rho(c)$.

Definition 13. A refinement graph is a directed, acyclic graph in which nodes are program clauses and *arcs* correspond to refinement operations, i.e., there is an arc from c to c', if $c, c' \in \mathscr{L}$, and $c' \in \rho(c)$.

7.2. A simplified refinement operator for DHDB clauses

In this study we consider a simplified refinement operator, defined as follows. Clause c' is a refinement of clause c, i.e., $c' \in \rho(c)$, if:

1. Clause $c = \square$ and $c' = t(X_1, X_2, \ldots, X_n) \leftarrow$, where $\square$ stands for the empty clause and $X_1, X_2, \ldots, X_n$ are distinct variables of types $T_1, T_2, \ldots, T_n$ as specified for the arguments of target predicate t.
2. Clause c is a DHDB clause and c' is obtained from c by adding to the body of c literals $q_i(Y_1, Y_2, \ldots, Y_{n_i})$ or *not* $q_i(Y_1, Y_2, \ldots, Y_{n_i})$, where $Y_1, Y_2, \ldots, Y_{n_i}$ appear in c and are of the corresponding types specified for the arguments of predicate q_i.

Notice that the operator is not a general refinement operator for constrained DHDB clauses since it does not introduce literals (and negations of literals) of the form $X_j = X_s$, $X_j = v$ and does not deal with function symbols [19].

In addition to the above refinement operator ρ, consider a modified refinement operator ρ_R which is identical to ρ except that it does not generate refinements c', $c' \in \rho(c)$, which would have been obtained by adding to the body of clause c a literal $q_i(Y_1, Y_2, \ldots, Y_{n_i})$ or *not* $q_i(Y_1, Y_2, \ldots, Y_{n_i})$ which is irrelevant w.r.t. the set of literals occuring in the body of c.

7.3. Experimental setting and results of experiments

In the experiments, a program for searching refinement graphs was adapted from Ref. [8]. The modified program allows for empirical (batch) learning by performing breadth-first exhaustive search. Search of the refinement graph stops (clause c is not refined) if:

1. clause c covers no positive example (useless clause),
2. clause c covers no negative example (consistent clause).

A clause covering no positive example is useless and is discarded. A clause covering no negative example is a consistent clause. The algorithm outputs all consistent clauses as possible hypothesis clauses, and eliminates the covered positive examples from the training set.

To test whether the search of refinement graphs can be reduced, the number of nodes generated by the refinement operator ρ are compared with the number of nodes generated by the refinement operator ρ_R.

Experiments are performed on the problem domains described in Section 6. Table 12 summarizes the experimental results. The meaning of the abbreviations are as follows: *Depth* – depth of the refinement graph (max. number of body literals), *All* – number of all nodes, *Rest* – number of nodes to be refined at the next depth of the refinement graph ($Rest = All - Useless - Consistent$), All_R– number of all nodes, $Elim_R$– number of eliminated nodes due to irrelevant literals, $Rest_R$– number of nodes to be refined at the next depth of the refinement graph ($Rest_R = All_R - Useless_R - Elim_R - Consistent_R$). Columns with subscript R denote the results using the refinement operator ρ_R, and the others are the results using the operator ρ.

Notice that useless clauses are due to irrelevant literals that are *false* for all positive examples. This type of irrelevance is eliminated both in the ρ and the ρ_R refinement graph search. On the other hand, eliminated $Elim_R$ nodes which are due to irrelevant literals are only eliminated in the refinement graph constructed by ρ_R.

Notice that the first level of the refinement ($Depth = 1$) results in a list of literals that is identical to the list of literals generated in Setting 2 of the LINUS transformation (see Section 6).

Table 12
Results of refinement graph reduction experiments

Domain	*Depth*	*All*	*Rest*	All_R	$Elim_R$	$Rest_R$
Family	1	32	18	32	10	8
	2	576	312	256	125	13
Total		608	330	288	135	21
Arch1	1	72	36	72	35	2
	2	2592	1292	144	71	0
Total		2664	1328	216	106	2
Arch2	1	72	38	72	34	4
	2	2736	1432	288	146	3
	3	103104	46638	216	107	0
Total		105912	48108	576	287	7
Eleusis1	1	44	34	44	15	20
	2	1496	896	880	557	1
Total		1540	930	924	572	21
Eleusis2	1	44	33	44	16	17
	2	1452	929	748	494	28
Total		1496	962	792	510	45
Eleusis3	1	44	34	44	17	17
	2	1496	992	748	470	20
Total		1540	1026	792	487	37
Krk1	1	108	86	108	48	40
	2	9288	6846	4320	2840	261
Total		9396	6932	4428	2888	301

As shown in Table 12, irrelevant literal elimination is effective in all the domains, since it significantly reduces the size of refinement graphs. To see the effectiveness of reduction, see the column $Elim_R$, and compare *All* vs. All_R, and *Rest* vs. $Rest_R$ in rows Total, as well as in rows above.

The most interesting domain is Arch2, i.e., the second example set of the arches learning problem, since the induced hypothesis contains a clause with three literals in the body. This involves the search of the refinement graph at depth 3, where the stopping of further specializations at depths 1 and 2 has the largest effect: instead of checking 103104 nodes (*All*) at depth 3, the search in the reduced graph involves only 216 nodes (All_R). Notice that the number of *All* nodes at a certain depth, say depth 3, depends on the *Rest* of nodes at the previous depth and the branching factor, where the branching factor is equal to *All* nodes at depth 1. For instance, in the Arch2 problem, $103104 = 1432 \times 72$, and for the reduced case $216 = 3 \times 72$. This confirms the intuition that with larger depth, irrelevant literal elimination will be more effective.

7.4. Summary and further work

Results of the experiments in the simplified refinement graph search setting used in this work show that the number of nodes in a refinement graph can be substantially reduced due to irrelevant literal elimination.

The experiments test the effectiveness of irrelevant literal elimination in a procedure for exhaustive searching of a refinement graph. We are aware of the limitations

of the simplified refinement graph search setting, which is in our experiments limited to the language of function-free constrained DHDB clauses. In further work the limitations of the language will be relaxed, and the irrelevant literal elimination procedure adapted and incorporated either in an existing, or newly developed, refinement graph search procedure, preferably using an optimal refinement operator (with no node duplications). In addition, bias shift will be studied by first doing the refinements within a limited language bias and then relaxing the bias if a consistent hypothesis cannot be found.

Complete search procedures have recently again gained popularity, for instance in Progol [30] which has been used in many successful data mining applications. Due to complete search, limited by bottom clauses generated for randomly selected positive examples, Progol is slow and can induce only short clauses containing few body literals. It is planned to study the possible improvements of the Progol refinement operator by disregarding refinements of nodes containing irrelevant literals.

In further work our plan is to investigate also irrelevant literal elimination in the heuristic search of refinement graphs, using the heuristics developed in the literal minimization algorithm [20,21], as well as the noise-handling based on the elimination of potentially noisy examples [11,10].

8. Conclusion

This work is a study of the problem of relevance for inductive learners, applicable both in propositional learning and in the LINUS transformation approach to inductive logic programming. The only condition for irrelevant literal elimination is that the learning algorithm uses some sort of minimization in the search of the hypothesis space. This is not a real limiting condition for existing learning systems. In contrast to this, the elimination of irrelevant training examples changes the statistical properties of the problem domain and must not be used when learners using statistical measures are used for hypothesis construction.

The problem of relevance is encountered by every inductive learner. Basically all learners are concerned with the selection of 'best' literals among the relevant literals. The presented theory of relevance is aimed at pointing out which literals constitute a set of relevant literals and which literals are irrelevant and can be disregarded, without even entering the learning process. We are thus concerned with finding globally relevant/irrelevant literals w.r.t. the entire set of training examples. This is important since the elimination of globally irrelevant literals guarantees that literal elimination will not harm the hypothesis formation process, i.e., that during the reduction of the hypothesis space the optimal problem solution will not be eliminated.

Although most of this study is devoted to the problem of relevance in preprocessing of the set of training examples, the results of this study are applicable also in the learning process itself, i.e., in the search of refinement graphs for ILP [38,35,30].

The presented work is also a step towards the detection of interesting chunks of knowledge. In our case these chunks (cliches) are actually the reduced training examples themselves, if described in the hypothesis language. Detection of these rudimentary cliches may be seen as a step towards easier predicate invention. Namely, the result of the reduction (described in this work) and minimization (described in previous work by the authors [9,20]) is a minimal set of literals needed for forming a

concept description. By eliminating the entire set of literals first to the reduced set, and further to the minimal set of literals, the complexity of the predicate invention task may considerably be reduced. This is one of the topics of further research.

This study is also a step towards a better understanding of the notion of relevance for inductive concept learning. We are aware of some assumptions and simplifications which need to be elaborated in further work since they may hinder the application of the proposed approach in real-life applications. For example, we do not consider missing values of training examples. In this work we also disregard the problem of noise and assume that the goal of a learner is to find a consistent and complete DNF description. The problem of noise is successfully solved in related work of the authors [11,10], whereas the limitation to DNF learning is solved in the ILLM algorithm which learns combined CNF/DNF descriptions [9].

Some of the important practical aspects such as the costs of literals are taken into account in this work, while disregarded by other authors concerned with feature selection [2,13,16,24,39]. A case study of cost-sensitive feature elimination in data preprocessing for a hybrid genetic decision tree induction algorithm RL-ICET on two East–West Challenge problems [21,22] shows that cost-sensitive elimination of irrelevant features can substantially improve the efficiency of learning and can reduce the costs of induced hypotheses.

Acknowledgements

We are grateful to Peter Flach, Claude Sammut and Dunja Mladenić for their comments on an earlier draft of this paper, as well as to anonymous reviewers for many useful remarks. We wish to thank Peter Turney and Sašo Džeroski for their collaboration in other experiments, and Donald Michie for his vivid interest in this work and stimulating discussions that motivated further research on this topic. This work has been financially supported by the Slovenian Ministry of Science and Technology, the Croatian Ministry of Science, and the ESPRIT project 20237 Inductive Logic Programming 2.

Appendix: Detailed results of literal elimination in five King–Rook–King chess endgame problems

This section presents results of irrelevant literal elimination for the five training sets of the chess endgame domain White King and Rook vs. Black King, presented in Section 6.5.

First example set

There are 49 positive examples in this set.

Setting 1: The following features are eliminated: $equal(C,A)$, $equal(E,A)$, $equal(E,C)$, $equal(D,B)$, $equal(F,B)$, $equal(F,D)$, $adj\_file(C,A)$, $adj\_file(E,A)$, $adj\_file(C,E)$, $adj\_rank(D,B)$, $adj\_rank(F,B)$, $adj\_rank(F,D)$. The other features are relevant. As we can see, only symmetric features are eliminated.

Setting 2: *RL* contains 42 literals: $equal(A,C)$, $equal(A,E)$, $equal(C,E)$, $equal(B,D)$, $equal(B,F)$, $equal(D,F)$, $adj\_file(A,C)$, $adj\_file(A,E)$, $adj\_file(C,E)$, $adj\_rank(B,D)$, $adj\_rank(B,F)$, $adj\_rank(D,F)$, $less\_file(A,C)$, $less\_file(A,E)$, $less\_file(C,A)$, $less\_file(E,A)$, $less\_rank(B,D)$, $less\_rank(B,F)$, $less\_rank(D,B)$, $less\_rank(F,B)$, $not\ equal(A,C)$, $not\ equal(A,E)$, $not\ equal(B,D)$, $not\ equal(B,F)$, $not\ adj\_file(A,C)$, $not\ adj\_file(A,E)$, $not\ adj\_file(C,E)$, $not\ adj\_rank(B,D)$, $not\ adj\_rank\ (B,F)$, $not\ adj\_rank(D,F)$, $not\ less\_file(A,C)$, $not\ less\_file(A,E)$, $not\ less\_file(C,A)$, $not\ less\_file(C,E)$, $not\ less\_file(E,A)$, $not\ less\_file(E,C)$, $not\ less\_rank(B,D)$, $not\ less\ \_rank(B,F)$, $not\ less\_rank(D,B)$, $not\ less\_rank(D,F)$, $not\ less\_rank(F,B)$, $not\ less\_rank\ (F,D)$.

Setting 3: The result in Setting 3 is identical to the one in Setting 1.

Second example set

There are 33 positive examples in this set.

Setting 1: RL_p contains 23 features. The result is the same as in the first example set except that feature $adj\_rank(B,D)$ is eliminated as well.

Setting 2: Results in Setting 2 are the same as in the first example set. The only difference is that literal $adj\_rank(B,D)$ is also eliminated.

Setting 3: The result in Setting 3 is identical to the one in the first example set.

Third example set

There are 32 positive examples in this set. Results in all settings are identical to the results in the first example set.

Fourth example set

There are 39 positive examples in this set.

Setting 1: The result in Setting 1 is the same as in the first example set.

Setting 2: Results in Setting 2 are similar as in the first example set. In addition to the 42 literals from the first example set REDUCE keeps literals $less\_rank(D,F)$, $less\_rank(F,D)$ and $notequal(D,F)$.

Setting 3: Results in Setting 3 are identical to those in the first example set.

Fifth example set

There are 37 positive examples in this set.

Setting 1: The result in Setting 1 is similar to that in the second example set. Instead of feature $adj\_rank(B,D)$ REDUCE eliminates feature $adj\_file(C,E)$.

Setting 2: The result in Setting 2 is similar as in the first example set. Instead of literal $adj\_rank(B,D)$, literal $adj\_file(C,E)$ is eliminated.

Setting 3: The result in Setting 3 is identical to the one in the first example set.

References

[1] H. Almuallim, T.G. Dietterich, Learning with many irrelevant features, Proceedings of the Ninth National Conference on Artificial Intelligence, MIT Press, Cambridge, 1991, pp. 547–552.

[2] R. Caruana, D. Freitag, Greedy attribute selection, Proceedings of the 11th International Conference on Machine Learning, Morgan Kaufmann, Los Altos, 1994, pp. 28–36.

[3] J. Chinal, Design Methods for Digital Systems, Akademie-Verlag, Berlin, 1973.

[4] P. Clark, R. Boswell, The CN2 induction algorithm, Machine Learning 3 (4) (1989) 261–283.

[5] D. Conklin, I.H. Witten, Complexity-based induction, Machine Learning 16 (1994) 203–225.

[6] S. Džeroski, Handling imperfect data in inductive logic programming, in: Proceedings of the Fourth Scandinavian Conference on Artificial Intelligence, IOS Press, 1993, pp. 111–125.

[7] U.M. Fayyad, K.B. Irani, On the handling of continuous-valued attributes in decision tree generation, Machine Learning 8 (1992) 87–102.

[8] P. Flach, Simply Logical: Intelligent Reasoning by Example, Wiley, New York, 1994.

[9] D. Gamberger, A minimization approach to propositional inductive learning, Proceedings of the Eighth European Conference on Machine Learning (ECML-95), Springer, Berlin, 1995, pp. 151–160.

[10] D. Gamberger, N. Lavrač, Noise detection and elimination applied to noise handling a KRK chess endgame, Proceedings of the Sixth International Workshop on Inductive Logic Programming, Lecture Notes in Artificial Intelligence, vol. 1314, Springer, Berlin, 1997, pp. 72–88.

[11] D. Gamberger, N. Lavrač, S. Džeroski, Noise elimination in inductive concept learning: A case study in medical diagnosis, Proceedings of the Seventh International Workshop on Algorithmic Learning Theory, Lecture Notes in Artificial Intelligence, vol. 1160, Springer, Berlin, 1996, pp. 199–212.

[12] D. Givone, Introduction to Switching Circuit Theory, McGraw-Hill, New York, 1970.

[13] G.H. John, R. Kohavi, K. Pfleger, Irrelevant features and the subset selection problem, Proceedings of the 11th International Conference on Machine Learning, Morgan Kaufmann, Los Altos, 1994, pp. 190–198.

[14] K. Kira, L.A. Rendell, A practical approach to feature selection, Proceedings of the Ninth International Conference on Machine Learning, Morgan Kaufmann, Los Altos, 1992, pp. 249–256.

[15] R. Kohavi, G.H. John, Wrappers for feature subset selection, Artificial Intelligence, Special Issue on Relevance, 1997.

[16] D. Koller, M. Sahami, Toward optimal feature selection, Proceedings of the 13th International Conference on Machine Learning, Morgan Kaufmann, Los Altos, 1996, pp. 284–292.

[17] I. Kononenko, Estimating atributes: Analysis and extensions of Relief, in: Proceedings of the Seventh European Conference on Machine Learning, Lecture Notes in Artificial Intelligence, vol. 784, Springer, Berlin, 1994, pp. 171–182.

[18] N. Lavrač, S. Džeroski, M. Grobelnik, Learning non-recursive definitions of relations with LINUS, Proceedings of the Fifth European Working Session on Learning (EWSL-91), Springer, Berlin, 1991, pp. 265–281.

[19] N. Lavrač, S. Džeroski, Inductive Logic Programming: Techniques and Applications, Ellis Horwood, Chichester, 1994.

[20] N. Lavrač, D. Gamberger, S. Džeroski, An approach to dimensionality reduction in learning from deductive databases, in: Proceedings of the Fifth International Workshop on Inductive Logic Programming (ILP-95), Technical report of the Katholieke Universiteit Leuven, 1995, pp. 337–354.

[21] N. Lavrač, D. Gamberger, P. Turney, Cost-sensitive feature reduction applied to a hybrid genetic algorithm, Proceedings of Seventh International Workshop on Algorithmic Learning Theory, Lecture Notes in Artificial Intelligence, vol. 1160, Springer, Berlin, 1996, pp. 127–134.

[22] N. Lavrač, D. Gamberger, P. Turney. A relevancy filter for constructive induction, IEEE Expert, Special Issue on Feature Transformation and Subset Selection, 1998.

[23] M. Li, P. Vitanyi, An Introduction to Kolmogorov Complexity and its Applications, Springer, Berlin, 1993.

[24] H. Liu, R. Setiono, A probabilistic approach to feature selection – A filter solution, Proceedings of the 13th International Conference on Machine Learning, Morgan Kaufmann, Los Altos, 1996, pp. 319–327.

[25] J. Lloyd, Foundations of Logic Programming, 2nd ed., Springer, Berlin, 1987.

[26] R.S. Michalski, A theory and methodology of inductive learning, in: R. Michalski, J. Carbonell, T. Mitchell (Eds.), Machine Learning: An Artificial Intelligence Approach, Tioga, Palo Alto, 1983, pp. 83–134.

[27] T. Mitchell, Generalization as search, Artificial Intelligence 18 (2) (1982) 203–226.
[28] S. Muggleton, Inductive logic programming, New Generation Computing 8 (4) (1991) 295–318.
[29] S. Muggleton, M. Bain, J. Hayes-Michie, D. Michie, An experimental comparison of human and machine learning formalism, Proceedings of the Sixth International Workshop on Machine Learning, Morgan Kaufmann, Los Altos, 1989, pp. 113–118.
[30] S. Muggleton, Inverse entailment and Progol: New generation computing, Special issue on inductive logic programming 13 (3-4) (1995) 245–286.
[31] S. Muggleton, L. De Raedt, Inductive logic programming theory and methods, Journal of Logic Programming 19/20 (1994) 629–679.
[32] R.B. Myerson, Game Theory: Analysis of Conflict, Harvard University Press, Cambridge, 1991.
[33] A.L. Oliveira, A. Sangiovanni-Vincentelli, Constructive induction using a non-greedy strategy for feature selection, Proceedings of the Ninth International Conference on Machine Learning, Morgan Kaufmann, Los Altos, 1992, 354–360.
[34] I.B. Pyne, E.J. McCluskey, The reduction of redundancy in solving prime implicant tables, IRE Trans. on Electronic Computers 11 (1962) 473–482.
[35] J.R. Quinlan, Learning logical definitions from relations, Machine Learning 5 (3) (1990) 239–266.
[36] J.R. Quinlan, C4.5: Programs for Machine Learning, Morgan Kaufmann, Los Altos, 1993.
[37] E. Rasmusen, Games and Information: An Introduction to Game Theory, Basil Blackwell, Oxford, 1989.
[38] E. Shapiro, Algorithmic Program Debugging, MIT Press, Cambridge, 1983.
[39] D. Skalak, Prototype and feature selection by sampling and random mutation hill climbing algorithms, Proceedings of the 11th International Conference on Machine Learning, Morgan Kaufmann, Los Altos, 1994, pp. 293–301.