Data Min Knowl Disc
DOI 10.1007/s10618-010-0174-x

Optimal constraint-based decision tree induction from itemset lattices

Siegfried Nijssen · Elisa Fromont

Received: 15 April 2009 / Accepted: 6 March 2010

Abstract In this article we show that there is a strong connection between decision tree learning and local pattern mining. This connection allows us to solve the computationally hard problem of finding *optimal* decision trees in a wide range of applications by post-processing a set of patterns: we use local patterns to construct a global model. We exploit the connection between constraints in pattern mining and constraints in decision tree induction to develop a framework for categorizing decision tree mining constraints. This framework allows us to determine which model constraints can be pushed deeply into the pattern mining process, and allows us to improve the state-of-the-art of optimal decision tree induction.

Keywords Decision tree learning · Formal concepts · Frequent itemset mining · Constraint based mining

1 Introduction

Decision trees are among the most popular predictive models and have been studied from many perspectives. However, no general framework exists to constrain the

Responsible editor: Johannes Fürnkranz and Arno Knobbe.

S. Nijssen (✉)
Department of Computer Science, Katholieke Universiteit Leuven, Celestijnenlaan 200A, 3001 Leuven, Belgium
e-mail: Siegfried.Nijssen@cs.kuleuven.be

E. Fromont
Laboratoire Hubert Curien, CNRS UMR 5516, Université de Saint-Étienne Jean Monnet, Université de Lyon, 42023 Saint-Étienne, France
e-mail: Elisa.Fromont@univ-st-etienne.fr

Published online: 06 April 2010

induction of decision trees and guarantee an exact result with respect to the given constraints. On the other hand, the topic of exhaustively (i.e. exactly) determining all patterns satisfying certain constraints has been studied extensively in the area of local pattern mining (Agrawal et al. 1996; Zaki et al. 1997a; Han et al. 2000). A natural question is hence if we can exploit the experience *in local pattern mining* for the discovery of decision trees under constraints. This question will be addressed in this article.

Our main starting point is that many decision tree learning problems can be formulated as *queries* of the following canonical form:

$$\operatorname*{argmin}_{T} f(T) \text{ subject to } \varphi(T), \qquad \textit{(Canonical Decision Tree Learning Query)}$$

i.e, we are interested in finding the best tree(s) according to a function $f(T)$, among all trees which fulfill the constraints specified in the formula $\varphi(T)$.

For instance, the following questions could be of interest for a decision tree user:

- Which tree has the smallest error? In this case $f(T)$ is an error function that we wish to minimize.
- Which is the smallest tree with sufficiently high accuracy? In this case the (ranking) function $f(T)$ should prefer smaller trees among sets of sufficiently accurate trees. Alternatively, we can reformulate the problem in a Bayesian setting (Buntine 1992; Chipman et al. 1998).
- Which tree is least sensitive to noise in the class labels? This could require that every leaf of a decision tree has at least a significant majority class. The latter can be seen as a constraint $\varphi(T)$ on the trees of interest.
- Which tree preserves privacy best by being well-balanced? This would impose a constraint $\varphi(T)$ on the trees of interest (Friedman et al. 2006; Machanavajjhala et al. 2007).
- Which tree incurs the smallest amount of classification costs? For example, it can be desirable that the expected costs for classifying examples do not exceed a certain predefined threshold value (Turney 1995).
- Which tree is most *justifiable* from an expert's perspective, by satisfying predefined constraints on the predictions that can be made by the tree? For instance, one could wish to enforce that certain examples are never misclassified, or certain tests are always executed in a given order.

Observe that some of these problem settings are conventional, in the sense that they are formalizations of the problem of finding models of good predictive accuracy. Other problems are less conventional, the main focus being on the syntax of the predictive model.

Many algorithms have been proposed to address these learning problems. Most common are the algorithms that rely on the principle of top-down induction through heuristics (for example C4.5 (Quinlan 1993) and CART (Breiman et al. 1984)). These algorithms do not explicitly minimize a global optimization criterion, but rely on the development of a good heuristic to obtain reasonable solutions. In practice, for each new problem setting that was studied, a new heuristic was proposed in the literature.

To the best of our knowledge, no framework has been proposed which encapsulates the large number of decision tree learning problems listed above and no general algorithm is known for answering the canonical decision tree query exactly. The question that we study in this paper is how pattern mining techniques can contribute to the development of a framework and an algorithm for this task. As such, our work takes the LeGo approach (Knobbe et al. 2008; Bringmann et al. 2009), in that it studies how local pattern mining techniques can be used to build global models.

The benefit of an exact algorithm is that we do not need to develop new heuristics to deal with many types of learning problems and constraints. We are sure that its result is the best that one can hope to achieve according to the predefined optimization criterion and constraints; no fine-tuning of heuristics is necessary. Hence, the results of an exact algorithm can also be used to determine how well an existing heuristic decision tree learner approximates a global optimization criterion.

The development of an *exact* algorithm for learning decision trees has seldom been considered because many decision tree learning problems are known to be NP-complete (Hyafil and Rivest 1976). Therefore an efficient algorithm for the general case most likely does not exist. This theoretical result however does not imply that the problem is intractable in all cases. Many frequent itemset mining algorithms have been applied successfully despite the exponential nature of the itemset mining problem. This is an indication that, on some datasets, exact decision tree induction may still be feasible if we can do this by using itemset mining results. We will provide evidence that for a reasonable number of datasets, exact decision tree induction is indeed practically feasible by taking this approach. An important technical contribution is that we show that decision trees can also be learned from the condensed itemset representation of closed itemsets (Pasquier et al. 1999). This observation allows us to obtain better practical performance.

The article is organized as follows. In Sect. 2, we introduce basic notions on decision trees and itemsets and we focus on their relationships. In Sect. 3, we discuss related work on both exact decision tree learning and itemset mining. In Sect. 4, we propose a framework for constraining decision trees and show how the framework can be used in practice. In Sect. 5 we introduce the DL8 algorithm which uses local patterns to construct our global model. Section 5 also gives some optimizations for DL8. In Sect. 6, we evaluate the performance and the effect of different constraints handled in our algorithm. We conclude in Sect. 7.

2 Itemsets, decision trees, and their relationships

Let us first introduce some terminology concerning *frequent itemsets* and *decision trees* before studying the relationships between these domains.

2.1 Itemsets

Let $\mathcal{I} = \{i_1, i_2, \ldots, i_m\}$ be a set of items and let $D = \{t_1, t_2, \ldots, t_n\}$ be a bag of transactions, where each transaction t_k is an itemset such that $t_k \subseteq \mathcal{I}$. A transaction t_k contains a set of items $I \subseteq \mathcal{I}$ iff $I \subseteq t_k$. The transaction identifier set (TID-set)

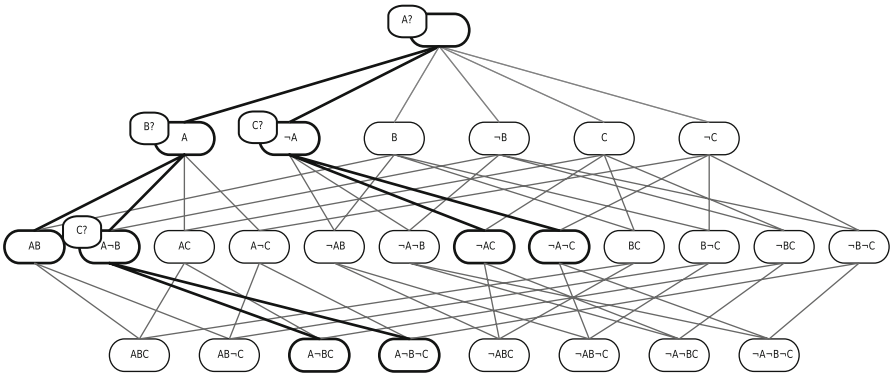


Fig. 1 The Hasse diagram of a part of an itemset lattice for items $\{A, \neg A, B, \neg B, C, \neg C\}$; binary decision tree A(B(1,C(l,l)),C(l,l)) is marked in this diagram

$tid(I) \subseteq \{1, 2, \ldots n\}$ of an itemset $I \subseteq \mathcal{I}$ is the set of identifiers of all transactions that contain itemset I. The frequency of an itemset $I \subseteq \mathcal{I}$ is defined to be the number of transactions that contain the itemset, i.e., $freq(I) = |tid(I)|$; the support of an itemset is $support(I) = freq(I)/|D|$. An itemset I is said to be frequent if its support is higher than a given threshold *minsup*; this is written as $support(I) \geq minsup$ (or, equivalently, $freq(I) \geq minfreq$).

A useful property of itemsets is that they constitute a lattice.

Definition 1 A complete *lattice* is a partially ordered set in which any two elements have a unique least upper bound and a unique greatest lower bound.

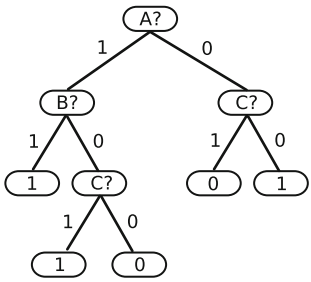
In this case the partial order is defined by the subset relationship $\subseteq$ on the elements in the set $2^{\mathcal{I}}$. The least upper bound of two sets is computed by the intersection ($\cap$) operator, the greatest lower bound by the union ($\cup$) operator. The lower bound $\bot$ of this lattice is $\emptyset$; the higher bound $\top$ is the set $\mathcal{I}$.

Part of a lattice is depicted in Fig. 1 in a *Hasse diagram*, where we assume $\mathcal{I} = \{A, \neg A, B, \neg B, C, \neg C\}$; we only depict itemsets in which an item i and its negation $\neg i$ do not occur together. Edges denote a subset relation between sets; sets are depicted as nodes. On top of the lattice is the lower bound which corresponds to the empty set $\emptyset$ (level 0); the higher bound $\{A, \neg A, B, \neg B, C, \neg C\}$ is not depicted as it includes items as well as their negations. There is an edge between a node in a given level and a node in the next level if the set of the former is strictly included in the set of the latter and if the size of the two sets only differs by one item.

2.2 Decision trees

An example of a decision tree is given in Fig. 2. A decision tree aims at classifying a set of examples by sorting them down the tree. The leaves of the tree provide the classifications of examples (Quinlan 1993). Each node of the tree specifies a test on one attribute of an example and each branch of a node corresponds to one of the possible outcomes of the test. We assume that all tests are Boolean; non-binary attributes

Fig. 2 An example tree



are transformed into Boolean attributes by mapping each possible value to a separate attribute. Numerical attributes are discretized and binarized beforehand (they will then be called *features*). The input of a decision tree learner is hence a binary matrix B, where B_{ij} contains the value of feature j of example i.

A common way to represent a decision tree is as a set of rules (Quinlan 1993). Each leaf of the tree corresponds to a rule. Our example tree can be represented in the following way:

if $A = 1$ and $B = 1$ then predict 1
if $A = 1$ and $B = 0$ and $C = 1$ then predict 1
if $A = 1$ and $B = 0$ and $C = 0$ then predict 0
if $A = 0$ and $C = 1$ then predict 0
if $A = 0$ and $C = 0$ then predict 1

Hence we can see decision tree learning as finding a set of rules with certain properties that allow the set to be represented as a tree.

2.3 The link between decision trees and itemsets

A main observation in the LeGo framework (Knobbe et al. 2008) is that there is a link between rules in predictive models and patterns in pattern mining. Assume that we are given an attribute-value table B in which all features are binary. We can transform table B into a transactional form D such that $t_j = \{i \mid B_{ij} = 1\} \cup \{\neg i \mid B_{ij} = 0\}$. Thus, every feature value is mapped to a positive i or a negative item $\neg i$. The head of a rule, for instance,

$A = 1$ and $B = 0$ and $C = 1$

can now be transformed into an itemset $\{A, \neg B, C\}$. Transactions in which the head of the rule is true correspond to transactions in which the itemset is contained. Hence the decision tree of Fig. 2 can equivalently be represented by a set of class *association rules*:

$\{A, B\} \rightarrow 1$
$\{A, \neg B, C\} \rightarrow 1$
$\{A, \neg B, \neg C\} \rightarrow 0$
$\{\neg A, C\} \rightarrow 0$
$\{\neg A, \neg C\} \rightarrow 1$

A class association rule $I \rightarrow c$ (Liu et al. 1998) consists of an itemset I and a class value c.

The problem of learning a decision tree is now a problem of finding a set of class association rules. As we are usually interested in finding accurate trees, we can reduce this further to a problem of finding itemsets, that is, class association rules without heads: assume we compute the frequency $freq_c(I)$ of an itemset I for each class c separately, we can associate to each itemset the class label for which its frequency is highest,

$$c(I) = \operatorname*{argmax}_{c' \in C} freq_{c'}(I),$$

as this will minimize the prediction error for the examples in the leaf. Given a decision tree T, we denote the set of itemsets corresponding to leaves by $leaves(T)$; in our example,

$$leaves(T) = \{\{A, B\}, \{A, \neg B, C\}, \{A, \neg B, \neg C\}, \{\neg A, C\}, \{\neg A, \neg C\}\};$$

itemsets corresponding to internal nodes are denoted by $internal(T)$, in our example,

$$internal(T) = \{\emptyset, \{A\}, \{A, \neg B\}\};$$

Finally, all itemsets that correspond to paths in the tree are denoted with $paths(T) = internal(T) \cup leaves(T)$.

The problem of finding a decision tree can now alternatively also be formulated as follows. We are interested in finding a set of itemsets $\mathcal{P} \subseteq 2^{\mathcal{I}}$ such that

$$\exists T : paths(T) = \mathcal{P} \text{ and } T = \operatorname*{argmin}_T f(T) \text{ subject to } \varphi(T).$$

Note that we can easily characterize which sets of itemsets represent decision trees.

Lemma 1 *Given a set of itemsets $\mathcal{P} \subseteq 2^{\mathcal{I}}$, then $\exists T : paths(T) = \mathcal{P}$ if and only if for every itemset $I \in \mathcal{P}$ either:*

(1) *there is no $I' \in \mathcal{P}$ such that $I \subset I'$ (in this case $I \in leaves(T)$);*
(2) *there is exactly one item $i \in \mathcal{I}$ such that $I \cup \{i\}, I \cup \{\neg i\} \in \mathcal{P}$ (in this case $I \in internal(T)$).*

Proof "$\Rightarrow$" is straightforward. For "$\Leftarrow$" we can observe the following. Every itemset I in $\mathcal{P}$ can be converted into a node in a decision tree, as follows. Itemsets fulfilling condition (1) we turn into leaves. Itemsets I fulfilling condition (2) are converted into internal nodes which are connected to the nodes representing itemsets $I \cup \{i\}$ and $I \cup \{\neg i\}$. □

Hence, the problems of finding decision trees T and sets of itemsets $\mathcal{P}$ fulfilling the conditions of Lemma 1 are equivalent. Indeed, the reader can check in our example that a set $\mathcal{P}$ fulfilling these conditions corresponds to a decision tree with $paths(T) = \mathcal{P}$.

An important observation that we will exploit is that a lattice of itemsets can be thought of as a compact representation of a set of decision trees. This is illustrated in Fig. 1, where we have highlighted the decision tree of Fig. 2; in principle any decision tree over binary features $\{A, B, C\}$ consists of a similar set of paths in this lattice. Note that we assume that trees never have an item and its negation in one path and that we hence do not need to consider the part of the lattice containing such itemsets.

The most basic problem one could be interested in is that of finding an accurate decision tree. The *accuracy* of a decision tree is derived from the number of misclassified examples in the leaves:

$$accuracy(T) = \frac{|D| - error(T)}{|D|} \qquad \text{where} \qquad error(T) = \sum_{I \in leaves(T)} error(I)$$

and $error(I)$ is the number of examples ending up in leaf I not labeled with the majority class of the examples in I:

$$error(I) = freq(I) - freq_{c(I)}(I)$$

For the *size* of a tree we take the size of the set $paths(T)$.

An example of a decision tree learning problem is to find the tree

$$\operatorname*{argmin}_{T}(error(T), size(T)),$$

that minimizes error in the first place and cuts ties between trees of equal error using the size function. The exploration of other learning problems will be deferred to a later section.

3 Related work

The results in this article are built on the foundations of two different research areas: decision tree induction and pattern mining. This section provides an overview of the relevant results which have been obtained in these areas.

3.1 Exact decision tree induction

The search for exact decision trees with respect to a given optimization criterion dates back to the 1970s, when several algorithms for building such trees were proposed. Their applicability was however limited and the development of heuristic tree learners, such as CART (Breiman et al. 1984) and C4.5 (Quinlan 1993) became most popular. Only recently new attempts have been made to develop more complete tree learners. We will first discuss the early results using modern terminology for clarification.

Garey (1972) proposed an algorithm for constructing an optimal *binary identification procedure*. In this setting, a binary database is given in which every example has a different class label. Furthermore, every example has a weight and every attribute

has a cost. The aim is to build a decision tree in which there is exactly one leaf for every example; the expected cost for classifying examples should be minimal.

Meisel and Michalopoulos (1973) studied a setting which is more common today, in which multiple examples can have the same class label, and numerical attributes are allowed. To tackle the problem of discretization, an overfitting decision tree is greedily constructed first. Its tests are collected. Using these tests, the task is then to find a 100% accurate decision tree with lowest expected cost, where every test has unit cost and examples are distributed according to a previously determined distribution. In 1977, it was shown by Payne and Meisel (1977) that Meisel's algorithm can also be applied for finding optimal decision trees under many other types of optimization criteria, for instance, for finding trees of minimal height or minimal numbers of nodes.

A parallel line of research was explored by Schumacher and Sevcik (1976) and Lew (1978), who studied the problem of converting decision tables into decision trees. A decision table is a table which contains (1) a row for every possible example in the feature space (including a class attribute) and (2) a probability for every example. The aim is to compress the decision table into a compact representation that allows to retrieve the class of an example as quickly as possible. An extension was studied by Lew (1978), in which it is possible to specify the input decision table in a condensed form, for instance, by using wild cards as attributes.

All these problems were solved by dynamic programming algorithms which bottom-up consider all subsets of attribute-values of the examples. These algorithms are very similar to the algorithm we will propose (see Sect. 5). The main difference is that we build a lattice of tests under different types of constraints, point out the connection to itemset mining, and employ modern techniques such as closed itemset mining.

More recently, pruning strategies of decision trees have been studied by Garofalakis et al. (2003). Garofalakis et al.'s algorithm can be seen as an application of the bottom-up algorithm on a greedily constructed tree instead of a lattice of itemsets.

Related is also the work of Moore and Lee on the *ADtree* data structure (Moore and Lee 1998). Both ADtrees and itemset lattices can be used for speeding up the lookup of itemset frequencies during the construction of decision trees, where ADtrees have the benefit that they are computed without frequency constraint. However, this is achieved by not storing specializations of itemsets that are already relatively infrequent; for these itemsets subsets of the data are stored instead. In our bottom-up procedure it is necessary that all itemsets that fulfill the given constraints are stored with associated information. This is not straightforwardly achieved in ADtrees.

Recently the problem of learning optimal decision trees has gained interest again. Esmeir and Markovitch (2007a,b) proposed an any-time algorithm, which essentially performs a brute-force enumeration of trees as long as the algorithm is not interrupted by the user. The algorithm attempts to enumerate more promising regions earlier, and prunes unpromising regions of the search space. However, the algorithm does not exploit dynamic programming strategies. It was applied both on conventional learning problems as problems with cost constraints.

Blanchard et al. (2007) proposed an algorithm for mining optimal dyadic decision trees. This algorithm operates on numerical data. It makes specific choices with respect to the discretization of the data and the optimization criterion used. The optimization criterion includes a regularization parameter which weighs decision tree size

and decision tree accuracy. It is shown that the generalization error of an optimal tree is bounded by this parameter, while also in practice the resulting trees are sometimes better than trees found by traditional tree learners. A dynamic programming algorithm is used to induce the tree. In Sect. 4.2.2 we show that this algorithm can be seen as one instance of our algorithm, but that the specific choice of constraints makes it impossible to apply certain optimizations which are available within our framework, the most important one being that closed itemsets cannot be used when inducing dyadic trees.

Other approaches that aim at more completely traversing the space of decision trees are those that apply variations of genetic algorithms (Turney 1995), and Markov chain Monte Carlo sampling approaches (Chipman et al. 1998).

In Murphy and Pazzani (1997) an exhaustive enumeration of decision trees on small datasets was performed in order to determine the validity of the principles of Occam's razor and oversearch for decision tree learning. In these experiments it was found that slightly larger trees can be found using complete search methods and that these trees can sometimes perform better than smaller trees found using heuristic trees. Our algorithm allows to perform similar experiments on a larger scale.

3.2 Pattern mining

The best-known pattern mining algorithm is the APRIORI algorithm (Mannila et al. 1994; Agrawal and Srikant 1994; Agrawal et al. 1996), whose aim is to find *frequent itemsets*. Subsequently, a wide variety of algorithms has been proposed to find this set more efficiently in dense datasets, i.e., binary datasets in which the number of ones is large. Among these algorithms are ECLAT (Zaki et al. 1997b), FPGROWTH (Han et al. 2000) and LCM (Uno et al. 2004).[1]

In many datasets it was found that the number of frequent itemsets is impractically large, and methods were investigate to find *condensed representations*. Condensed representations consist of smaller sets of itemsets sufficient to reproduce the full set of itemsets, or allow to approximate the full set. Well-known exact condensed representations are the *closed itemsets* (Pasquier et al. 1999) and *free itemsets* (Boulicaut et al. 2000). Among the approximative representations are the δ-free itemsets (Boulicaut et al. 2003).

In this article, we investigate the use of both frequent itemsets and condensed representations during the construction of decision trees. The use of condensed representations could allow us to search for trees more efficiently. Among others, we show that in many cases we can limit ourselves to closed itemsets; in other cases, it can be shown that the itemsets that are needed during tree construction are also $\delta-$free.

The frequent itemset mining problem was extended towards other types of constraints than support; several categories of constraints were identified, among which *monotonic, anti-monotonic*, and *convertible* constraints, and algorithms were introduced to mine itemsets under these constraints (Pei et al. 2001; Bucila et al. 2003; Bonchi and Lucchese 2007). We can show that these categories can also be applied in

[1] A repository of implementations is available here: http://fimi.cs.helsinki.fi/.

decision tree induction, and that algorithms for mining patterns under these constraints can be used in corresponding settings in decision tree induction.

The use of itemsets in predictive models is a topic that has been studied extensively. Well-known algorithms include CBA (Liu et al. 1998), CMAR (Li et al. 2001) and CAEP (Dong et al. 1999); more general overviews can be found in (Knobbe et al. 2008; Bringmann et al. 2009). In these algorithms an initial set of patterns is mined first. Subsequently, a set of patterns is selected with corresponding weights. Classification is based on a vote of the selected patterns. The approach that we will propose is similar in the sense that our algorithm can also be applied to select a subset of patterns from an initial set of patterns, which are subsequently interpreted as rules for classification. The main difference is that the model that we induce takes the particular shape of a tree, is optimal under well-defined conditions, and that we can push model constraints in the pattern mining process or even combine these two.

An alternative approach to exploit patterns in classification is to construct features from patterns. The selection of patterns can in this case be seen as feature selection, where desirable properties are that a set of patterns is chosen that is diverse, covers the data sufficiently, and correlates with the class attribute. Approaches of this kind are discussed in (Yan et al. 2005; Knobbe and Ho 2006; De Raedt and Zimmermann 2007). Even though our approach also selects a subset of patterns satisfying similar constraints, we will however use the patterns directly in classification models.

4 Constraints on decision trees

As stated in the introduction, we are interested in expressing decision tree learning problems as queries of the form

$$\operatorname*{argmin}_{T} f(T) \text{ subject to } \varphi(T),$$

which corresponds to finding the best tree(s) according to the function $f(T)$ among all trees which fulfill the constraints specified in the formula $\varphi(T)$.

In this section we specialize this formula. We argue that in most applications the constraints in $\varphi(T)$ and the criteria in $f(T)$ have properties that can be exploited. The main contributions in this section are:

1. We propose a categorization of constraints and criteria that can be used in the above formula. The aim of this categorization is to introduce the types of constraints that will be exploited in the pattern-based algorithm introduced in the next section. In this discussion, we will discuss desirable properties of *optimization criteria* $f(T)$, as well as properties of constraints $\varphi(T)$.
2. We show that constraints and criteria with useful properties are commonly used in a wide range of applications and hence our algorithm can be used in many applications. The applications were chosen such that it is possible to compare our algorithm to already existing (and most of the time, heuristic) algorithms in the literature.

In the following, the functions $I_1 = child_1(I, T)$ and $I_2 = child_2(I, T)$ return the itemsets representing respectively the left-hand and right-hand child node of the internal node I in binary tree T.

4.1 Properties of constraints and criteria

4.1.1 Optimisation criteria

When an optimization criterion $f(T)$ is specified, this criterion may have properties that we will refer to as *additivity* and *structure independence*.

Additivity An *additive* optimization criterion is a function $f(T)$ over a tree T which can be rewritten as follows:

$$f(T) = \sum_{I \in leaves(T)} f_{leaf}(I) + \sum_{I \in internal(T)} f_{internal}(I, child_1(I, T), child_2(I, T)),$$

where function $f_{leaf}(I) \geq 0$ is a *leaf criterion* and function $f_{internal}(I, I_1, I_2) \geq 0$ is an *internal criterion*. An example of an additive optimization criterion is *size*, in which $f_{leaf}(I) = 1$ and $f_{internal}(I, I_1, I_2) = 1$.

Structure independence An additive optimization criterion $f(T)$ is *structure independent* if we can rewrite the leaf criteria and internal criteria as follows: $f_{internal}(I, I_1, I_2) = f'_{internal}(tid(I), tid(I_1), tid(I_2))$ and $f_{leaf}(I) = f'_{leaf}(tid(I))$, for functions $f'_{internal}$ and $f_{leaf}{}'$ over sets of transactions. Hence, the evaluation depends only on the transactions covered by the nodes, not on the structure of the tree. Please note that *size* is also a structure independent criterion according to our definition; the reasoning is that the size of a tree is only determined by the number of partitions induced by the tree in the set of transactions; otherwise the structure of the tree is unimportant.

In the next section we will show that many common optimization criteria are additive and that a restriction to such criteria is not very restrictive.

4.1.2 Path constraints

For constraints we can formulate similar properties as for criteria. In most cases, the constraint $\varphi(T)$ is a conjunction of a number of independent constraints, which can have the following properties.

Conjunctivity over Paths A *conjunctive path* constraint is a formula over a tree which can be written as:

$$\varphi_{conjunctive}(T) = \bigwedge_{I \in leaves(T)} \varphi_{leaf}(I) \wedge \bigwedge_{I \in internal(T)} \varphi_{internal}(I, child_1(I, T), child_2(I, T)),$$

where formula $\varphi_{leaf}(I)$ is a *leaf constraint* and formula $\varphi_{internal}(I, I_1, I_2)$ is an *internal constraint*.

An example of an internal constraint is that internal nodes should not have *pure* class distributions:

$$\varphi_{internal}(I, I_1, I_2) = (|tid(I)| \neq \max_{c \in C} |tid_c(I)|). \tag{1}$$

This internal constraint is special in the sense that it only takes I, not its children, into account. An example of a leaf constraint is that the number of examples not belonging to a majority class is small:

$$\varphi_{leaf}(I) = (|tid(I)| - \max_{c \in C} |tid_c(I)|) \leq maxfreq). \tag{2}$$

An example of an internal constraint in which the left-hand and right-hand child are used, is:

$$\varphi_{internal}(I, I_1, I_2) = (||tid(I_1)| - |tid(I_2)|| \geq mindif),$$

which states that an internal node splits examples in balanced proportions.

Structure independence A *structure independent* constraint $\varphi_{structure\_ind}(T)$ is a conjunctive path constraint in which $\varphi_{internal}(I, I_1, I_2) = \varphi'_{internal}(tid(I), tid(I_1), tid(I_2))$ and $\varphi_{leaf}(I) = \varphi'_{leaf}(tid(I))$, for formulas $\varphi'_{internal}$ and $\varphi_{leaf}{}'$ over sets of transactions.

An example of a structure independent path constraint is *minimum support*, in which

$$\varphi_{leaf}(I) = \varphi_{internal}(I, I_1, I_2) = (|tid(I)| \geq minfreq);$$

it is easy to see that this constraint is computed from $tid(I)$ only.

Anti-monotonicity An *anti-monotonic* constraint is a formula $\varphi_{antim}(I)$ over paths which ignores the left-hand and right-hand children of internal nodes and satisfies:

$$\forall I \subseteq I' : \varphi_{antim}(I) \rightarrow \varphi_{antim}(I').$$

Minimum support is an anti-monotonic constraint. The constraint in Eq. 2 is an example of a constraint which is not anti-monotonic. If an anti-monotonic constraint is used as leaf constraint, the internal nodes will also satisfy the constraint. Internal node constraints can also be anti-monotonic if they only have one itemset as parameter; for instance, the impurity constraint (see Eq. 1) is anti-monotonic; however, note that this constraint will usually not be used as a leaf constraint. Hence, we can distinguish internal and leaf anti-monotonic constraints; the one type will be denoted with $\varphi_{internal,antim}$, the other with $\varphi_{leaf,antim}$.

Constraints of these types can freely be combined. For instance, if we are searching for trees in which leaves are frequent, internal nodes are not pure and leaves have strong majority classes, we have a problem in which:

$$\varphi(T) = \bigwedge_{I \in internal(T)} (|tid(I)| \neq \max_{c \in C} |tid_c(I)|)$$
$$\bigwedge_{I \in leaves(T)} (((|tid(I)| - \max_{c \in C} |tid_c(I)|) \leq maxfreq) \wedge (|tid(I)| \geq minfreq)).$$

We can categorize these constraints as follows according to their properties:

$$\varphi_{internal}(I, I_1, I_2) = (|tid(I)| \neq \max_{c \in C} |tid_c(I)|),$$
$$\varphi_{leaf}(I) = ((|tid(I)| - \max_{c \in C} |tid_c(I)|) \leq maxfreq) \wedge (|tid(I)| \geq minfreq),$$
$$\varphi_{leaf,antim}(I) = (|tid(I)| \geq minfreq).$$

Note that some constraints (for example $|tid(I)| \geq minfreq$) may belong to multiple categories.

4.1.3 Optimization constraints

If a constraint can be written as

$$\varphi(T) = (g(T) \leq \theta),$$

where $g(T)$ is an integer optimization criterion and θ is a threshold value, the constraint is called an optimization constraint. Properties of optimization criteria, such as additivity and structure independence, extend to optimization constraints. In particular, if $g(T)$ returns a vector of values, θ can also be a vector of thresholds, each of which should be satisfied.

4.2 Showcases

In this section we list several existing decision tree learning problems. Our aim is to explain how such problems can be solved exactly and generally in our framework. We distinguish two classes of mining problems.

1. Traditional learning settings in which the focus is primarily on finding highly predictive trees; these settings differ in the criteria used to measure the predictive value of a tree and achieve generalization. We take as examples the tree-pruning algorithms, the Bayesian learning setting, and dyadic decision tree construction.
2. A learning setting in which the focus is less on accuracy, and the primary focus is on finding trees that also satisfy other desirable criteria. We take as examples cost-based tree learning and privacy-preserving tree induction.

By showing such a diverse list and by combining different settings, we hope to convince the reader that our categorization of constraints is very general and that the scope of our exact algorithm is not limited to the applications presented here.

4.2.1 Error-based pruning

In general we are interested in predictors that perform well on unseen data. The idea behind error-based pruning, which was developed as a pruning measure in the C4.5 algorithm (Quinlan 1993), is to estimate the true error rate of a leaf given the empirical error. An internal node is then turned into a leaf if this reduces the error estimate for the node.

The true error is estimated by assuming that the class labels of the examples are the result of sampling with the unknown true error rate. Consequently, the observed errors are binomially distributed, and a worst-case estimate on the true error can be computed from the observed error. It can be shown that the number of errors estimated by this procedure in a leaf is at least 0.5 higher than the empirical error count. Hence a tree with many leaves is penalized when compared to a tree with few leaves; implicitly, one can think of the error-based pruning criterion as the sum of error and a penalty term for the size of the tree, where the penalty term per leaf is dependent on the class distribution in the leaf.

In our framework, we can see the pruning measure as an optimization criterion. Let us denote the estimated error of a leaf I with $ee(I)$, then we are minimizing

$$f_p(T) = \sum_{I \in leaves(T)} ee(I).$$

Consequently we can categorize C4.5 error-based pruning as applying an *additive, structure independent optimization criterion*.

In C4.5, pruning is often only applied to an existing tree. Usually, this is done in a heuristic fashion: for instance, for each internal node in the tree it is tested in some order whether it is beneficial to replace the node by a leaf; additionally, *lifting* a subtree is sometimes also considered, in which case an internal node may be replaced by the subtree of one of its children if this improves the score. It is not always clear in which order these operations need to be performed. Within our framework, we can also formulate this problem of pruning an existing tree, which allows us to solve it in a non-heuristic way. Given a predefined tree T, we are only interested in finding a tree T' which minimizes $f_p(T')$ under the constraint that every leaf I' in T' is a subset of a path I in T. This constraint is *path conjunctive, structure dependent and anti-monotonic*. The benefit of starting from an existing tree is that the search-space of decision trees is significantly restricted in this way.

4.2.2 Optimal dyadic decision trees

Similarly to what we aim for in this article, Blanchard et al. (2007) studied how to learn *optimal dyadic* decision trees. Dyadic trees are trees on numerical data in which the discretization is limited to equi-width binning with a number of bins that always is a power of 2. Tests are hence always of the kind $a \geq i(r-l)/2^{\ell} + l$, where $[l, r]$ is the range of the attribute a and $0 \leq i \leq 2^{\ell}$ and $\ell \geq 0$ are integers; parameter ℓ is determined during the learning procedure. To limit the complexity of this learning

problem, in (Blanchard et al. 2007) the problem of learning optimal dyadic decision trees was constrained as follows:

- The optimization criterion is a function

$$error(T) + \lambda \cdot size(T)$$

  where λ is a regularization parameter that is determined on validation data. Note that this function is the sum of two additive optimization functions, and is hence also *additive*; compared to error-based pruning large trees are punished directly in this case.
- The first constraint imposes a threshold on the number of tests for each numerical attribute. This is an *anti-monotonic, structure dependent, conjunctive* path constraint:

$$\varphi_{leaf}(I) = \varphi_{internal}(I, I_1, I_2) = \bigwedge_a |I \cap I(a)| \leq k,$$

  where $I(a)$ is the set of items corresponding to tests on attribute a;
- The second constraint states that every path I which contains a split $a \geq v$ on attribute a must also contain a split for value $2\lfloor (v-l)/2 \rfloor + l$ and $2\lceil (v-l)/2 \rceil + l$, except for values l or r of that attribute. We can model this as an *internal node* constraint $\varphi_{internal}(I, I \cup \{i\}, I \cup \{\neg i\})$ which is true if i is a test that is allowed in itemset I.
- It does not make sense to continue splitting for leaves that do not contain examples. Therefore, internal nodes are required to have non-zero support. However, leaves with zero support are allowed, as they can be necessary to allow for more fine-grained splits later on.

4.2.3 Bayesian probability estimation trees

By Buntine (1992), Chipman et al. (1998), and Angelopoulos and Cussens (2005) a Bayesian approach was proposed for weighing decision tree size and decision tree accuracy, hence providing an alternative strategy for finding trees which are both accurate and small. It is assumed that a prior is given on the structure of the *probability estimation trees*:

$$p(T|D) = \prod_{I \in paths(T)} P_{node}(I, T, D)$$

where

$$P_{node}(I, T, D) = \begin{cases} 1, & \text{if } children(I) = 0; \\ 1 - \alpha(1 + |I|)^{-\beta}, & \text{if } I \text{ is a leaf in } T \text{ and } children(I) \neq 0; \\ \frac{\alpha(1+|I|)^{-\beta}}{children(I)}, & \text{otherwise;} \end{cases}$$

Here $children(I)$ is the number of tests that can still be performed to split the examples in $tid(I)$; D is the training data, excluding the class labels; α and β are parameters. In (Buntine 1992; Chipman et al. 1998; Angelopoulos and Cussens 2005) the prior only allows for paths that contain at least a minimum number of examples, which corresponds to the *anti-monotonic, structure independent* minimum support constraint.

The distinguishing feature of density estimation trees is that they have class distributions in the leaves instead of single class labels.

Given the Bayesian setting, in (Buntine 1992; Chipman et al. 1998; Angelopoulos and Cussens 2005) a prior distribution over the parameters was defined, as well as the probability of the training data given a tree structure and its parameters. After rewriting, we can formulate this optimization criterion as follows:

$$f_b(T) = -\log(p(\mathbf{c}|T, D)) - \log(p(T|D)).$$

where

$$p(\mathbf{c}|T, D) = \prod_{I \in leaves(T)} \left(\frac{\Gamma(\sum_c \alpha_c)}{\prod_c \Gamma(\alpha_c)} \right) \left(\frac{\prod_c \Gamma(freq_c(I) + \alpha_c)}{\Gamma(freq(I) + \sum_c \alpha_c)} \right);$$

here Γ is the standard gamma function that extends the factorial to real numbers; vector $\boldsymbol{\alpha}_c$ is a parameter of the optimization criterion. Vector $\mathbf{c}$ represents the class labels of the training examples. The overall optimization criterion is *additive and structure dependent*.

4.2.4 Cost-sensitive decision trees

A benefit of decision trees is that they are easily interpretable models that can be used as questionnaires. For instance, in the medical domain, a decision tree can be interpreted by a doctor as a sequence of tests to diagnose a patient; an insurance company can interpret it as a sequence of questions to determine if a person is a desirable customer. In such cases, the application of a tree on an example incurs a certain cost: every question might require a certain amount of money or time to be answered. Furthermore, if a person is classified incorrectly, this might induce additional costs, in terms of expected missed revenue, or higher treatment costs. To induce trees under such cost constraints, algorithms for decision tree induction under cost constraints have been proposed (Turney 1995; Esmeir and Markovitch 2007a).

Formally, these algorithms assume that the following information is given:

- a $c \times c$ misclassification cost matrix Q where $Q_{i,j}$ is the cost of predicting that an example belongs in class i, when it actually belongs in class j;
- for every attribute i, a cost tq_i for a test on this attribute;
- for every attribute i, a group g_i that it is contained in;
- for every group g, an additional cost tq_g for the first test on an attribute in this group.

The motivation for having both a cost per group and per attribute is that it is often cheaper to ask related questions or perform related medical tests. Therefore later tests in a group of related tests are usually cheaper.

For a path $I \in leaves(T)$ we can formally define the cost of classifying an example in $tid(I)$ as follows:

$$tq(I) = \sum_{i \in I} tq_i + \sum_{g \in \{g_i | i \in I\}} tq_g.$$

The expected costs for performing the tests in a tree are therefore:

$$f_{tq}(T) = \sum_{I \in leaves(T)} \frac{freq(I)}{|D|} tq(I).$$

The expected misclassification costs are:

$$f_{mq}(T) = \frac{1}{|D|} \sum_{I \in leaves(T)} \sum_{c \in C} Q_{c,c(I)} freq_c(I).$$

Combining these costs, the following criterion was proposed (Turney 1995; Esmeir and Markovitch 2007a), which is *additive and structure dependent*:

$$f_q(T) = f_{tq}(T) + f_{mq}(T).$$

A possibility which has not been studied in the literature, is the use of costs in path constraints. This may also be useful in practice. For instance, assume that the cost of a test is expressed in terms of the time that is needed to perform the test, while the misclassification cost is in terms of dollars. Combining these costs in a single measure would require time to be expressed in monetary terms, which may be undesirable and unpractical. An alternative could be to explicitly search for a tree that minimizes $f_{mq}(T)$, under the constraint that $tq(I) \leq maxtime$, for every itemset $I \in leaves(T)$. This *conjunctive, anti-monotonic* constraint would allow us to find inexpensive trees that have bounds on prediction times. One could evaluate such a query for multiple values of *maxtime* to come to a well-motivated trade-off between classification time and misclassification costs.

4.2.5 Privacy preservation

The main motivation behind privacy preserving decision tree learning, such as first performed by Friedman et al. (2006), is as follows. Assume we have a credit card company with a database D in which good and bad clients are distinguished from each other. The company learns a decision tree on this data, and uses this tree to accept or reject customers. Then regulations may require that the company publishes this tree. How can the company avoid that the tree provides information about individual customers in its database?

By Friedman et al. (2006) the following *attack* was studied. Assume that an attacker has public information, such as the address and telephone number of a customer, and wishes to know if this customer is a good customer. How can the company ensure its customers that the prediction that an attacker obtains from the tree using public information, is never based on less than k individuals, and hence, can never be traced to one individual customer? This problem can be formalized as follows: we wish to find a tree in which the following constraint is satisfied for every example t in the data:

$$\sum_{\substack{I' \in leaves(T) \\ (I' \cap \mathcal{I}_{public}) \subseteq (t \cap \mathcal{I}_{public})}} freq(I') \geq k :$$

here, $\mathcal{I}_{public}$ represents attributes assumed to be publicly available; attributes in $\mathcal{I}_{private}$ are not publicly available. The constraint expresses that a particular example t may not end up in a set of leaves containing in total less than k examples, if we assume that for private attributes we try both branches when we pass the example down the tree. This constraint is an *additive and structure dependent* optimization constraint, and needs to be applied for every customer. We will see that such a large number of constraints is hard to deal with in our framework. However, if we do not wish to make assumptions about which attributes are private, and all attributes except the target are assumed public, the constraint reduces to a minimum support constraint.

It is known in the database community that the protection provided by this $k-$*anonymity* is limited (Sweeney 2002; Samarati 2001; Machanavajjhala et al. 2007), in particular when regulations require the class distribution of the decision tree also to be provided. As solution to address this problem the ℓ-diversity principle was proposed. Using our framework, we can extend ℓ-diversity to decision tree learning:

- under k-anonymity the prediction performed by a leaf may be 100% accurate if there is no diversity in the class labels. To ensure customers that a tree never performs a 100% accurate prediction—and hence, every customer could be one of the exceptions to the prediction of the tree—we could require that the class distribution in every leaf has sufficiently high entropy, i.e.,

  $$\forall I \in leaves(T) : H_l(I) = H\left(\frac{freq_1(I)}{freq(I)}, \frac{freq_2(I)}{freq(I)}, \ldots, \frac{freq_c(I)}{freq(I)}\right) \geq \ell;$$

  here, $H(p_1, \ldots, p_n)$ computes the entropy of a distribution,

  $$H(p_1, p_2, \ldots, p_n) = -\sum_{i=1}^{n} p_i \log p_i.$$

  This constraint is a *conjunctive, non-anti-monotonic, structure independent*.
- in addition to a high diversity on class labels, we can also require a high diversity for other attributes by imposing as constraint:

  $$\forall I \in internal(T) : H_{in}(I) = H\left(\frac{freq(child_1(I,T))}{freq(I)}, \frac{freq(child_2(I,T))}{freq(I)}\right) \geq \ell.$$

This constraint avoids the following type of leak: in many cases the most accurate decision trees are obtained when the supports in the leaves reach the frequency threshold k. If an attacker knows this k, the attacker could derive how many examples are approximately in which part of the tree, and could gather an impression of the values of attributes used in internal nodes. By requiring tests to be performed on balanced attributes, some information is leaked; however, this information is often least useful.

Observe that these constraints can be hard to optimize in combination with accuracy. Indeed, ℓ-diversity and accuracy are opposing requirements; traditional entropy-based decision tree learners discourage high entropy in the leaves.

5 Building optimal decision trees from lattices

In this section we develop an algorithm for finding decision trees. Given our categorization of the previous section, these are the requirements for this algorithm:

1. The optimization criterion must be additive.
2. The constraints must be either conjunctive over paths or based on an additive optimization criterion.
3. There should be at least one anti-monotonic path constraint.

As seen in the previous section, we decompose a query in the following components, some of which may be empty:

- the anti-monotonic leaf constraint $\varphi_{leaf,antim}(I)$;
- the leaf constraint $\varphi_{leaf}(I)$, which includes the anti-monotonic leaf constraint;
- the internal constraint $\varphi_{internal}(I, I_1, I_2)$;
- the leaf optimization criterion $f_{leaf}(I)$;
- the internal optimization criterion $f_{internal}(I, I_1, I_2)$;
- the leaf optimization constraint $g_{leaf}(I)$;
- the internal optimization constraint $g_{internal}(I, I_1, I_2)$;
- the optimization constraint threshold(s) θ.

The algorithm, which we called DL8 (Decision Trees from Lattices), is based on the link between itemset mining and decision tree learning. In this section, we first discuss how to compute trees from itemset lattices. Next, we discuss how to compute these lattices, where we consider two options:

1. Building the trees from pre-computed itemsets (the lattice is computed *before* building the decision trees).
2. Integrating itemset mining into the decision tree construction (the lattice is computed *while* building the decision trees).

Finally, we study how queries can be rewritten to improve the efficiency of their evaluation in our algorithm.

5.1 Building decision trees from lattices

The algorithm for constructing decision trees from lattices is given in Algorithm 1. Its main component is the DL8- RECURSIVE procedure, which is called for an itemset and

Algorithm 1 DL8($\varphi_{leaf,antim}, \varphi_{leaf}, \varphi_{internal}, f_{leaf}, f_{internal}, g_{leaf}, g_{internal}, \theta$)

```
1: 𝒯 ←DL8- RECURSIVE(∅)
2: Compute argmin_{T∈𝒯} T.f
3:
4: procedure DL8- RECURSIVE((I))
5:     if DL8- RECURSIVE(I) was computed before then
6:         return stored result
7:     end if
8:     initialize 𝒯 to be an empty associative array with domain {0, ..., θ}
9:     if φ_leaf(I) then
10:        T.tree ← leaf(c(I))
11:        T.f ← f_leaf(I)
12:        T.g ← g_leaf(I)
13:        if T.g ≤ θ then
14:            𝒯[T.g] = T
15:        end if
16:    end if
17:    for all i ∈ ℐ do
18:        if φ_internal(I, I ∪ {i}, I ∪ {¬i}) and φ_leaf,antim(I ∪ {i}) and φ_leaf,antim(I ∪ {¬i}) then
19:            𝒯_1 ← DL8- RECURSIVE(I ∪ {i})
20:            𝒯_2 ← DL8- RECURSIVE(I ∪ {¬i})
21:            for all T_1 ∈ 𝒯_1, T_2 ∈ 𝒯_2 do
22:                T.tree ← node(i, T_1.tree, T_2.tree)
23:                T.f ← f_internal(I, I ∪ {i}, I ∪ {¬i}) + T_1.f + T_2.f
24:                T.g ← g_internal(I, I ∪ {i}, I ∪ {¬i}) + T_1.g + T_2.g
25:                if T.g ≤ θ and (𝒯[T.g] is empty or 𝒯[T.g].f > T.f) then
26:                    𝒯[T.g] = T
27:                end if
28:            end for
29:        end if
30:    end for
31:    store 𝒯 as the result for I and return 𝒯
32: end procedure
```

computes decision trees for that itemset. The main reasons why DL8 is more efficient than naïve enumeration algorithms are:

- We optimize the left-hand and right-hand branch of a node in a tree independently from each other, hence avoiding that we enumerate all possible combinations of sub-trees for the left-hand and right-hand branch of a test.
- When we compute a tree for an itemset, we store the result, and reuse it later on, hence avoiding that we compute the same result for other possible orders in which the same tests can occur in a path.
- We do not recurse the search when the anti-monotone constraints are not satisfied.

The correctness of this approach follows from the following facts.

- We consider queries which are additive and conjunctive, and hence, we can evaluate optimization criteria and constraints for the left-hand and right-hand branch of a node in a tree independently from each other.
- All constraints and optimization criteria are computed for itemsets, independent of the order of the items in these sets.

– If an anti-monotonic constraint is not satisfied for a path, any tree which contains this path cannot be a solution to the query either.

If κ is the number of edges in a lattice, the complexity of the algorithm is $\Theta(\kappa)$, as we consider every edge in this lattice exactly once.

In our algorithm, we use several data structures. The main data structure is the one in which the lattice is stored. For every itemset I we have an associative data structure $\mathcal{T}$ which allows us to associate a tree and its attributes to a vector of integers. In case no optimization constraints are specified, this structure $\mathcal{T}$ contains at most one tree. Note that at the implementation level we do not need to store associated trees in their entirety: it is sufficient to only store the roots of these trees, as the subtrees can be recovered from the lattice recursively, by searching for the trees associated to the left-hand and right-hand branch of an internal node.

In more detail, our algorithm works as follows:

Line 8:	For each possible value that the optimization constraint can take we will store one associated tree. Initially, this data structure is empty. Note that we require an optimization criterion that is used as optimization constraint to have an integer codomain.
Line 9–14:	In case the itemset corresponds to a possible leaf, we initialize this leaf and its statistics $T.f$ and $T.g$.
Line 17–31:	We iterate over all possible tests to split the examples further.
Line 19:	For a possible test, we determine whether or not we create a tree in which this test is a valid internal node; furthermore we determine if we create two paths that can be part of a tree in which the anti-monotonic constraints are satisfied.
Line 20–21:	If we can satisfy the constraints, we determine the best trees for the left-hand and right-hand branches, independently from each other; both calls return sets of trees, each tree associated to a vector of integers, each integer representing a possible value of one of the optimization constraints for the tree (if we do not have an optimization constraint, each set contains at most one tree).
Line 22–29:	We consider all combinations of left-hand and right-hand trees.
Line 27:	The optimization constraint of the generated tree is evaluated; if the best known tree for this constraint value is improved, we store the new tree. We only need to store intermediate trees for which the optimization constraint is not higher than the threshold value, as the additivity means that other sub-trees cannot be part of the final tree.

5.2 Computing lattices beforehand

While DL8 is executing, it needs to evaluate constraints based on the data. In this section we study the following question: assuming that we would like to use an itemset mining algorithm beforehand to find the itemsets and their properties in the data, which constraints should we use in this itemset miner? In other words, how do we push the decision tree mining constraints in the itemset mining process?

First, let us consider why we may be interested in separating the execution of DL8 from the itemset mining process. We believe there could be two reasons for this.

1. There are many optimized itemset mining algorithms; by using these, we exploit these optimizations, and reduce implementation efforts.
2. We might consider decision tree construction as one part of an interactive data analysis process, in which it could be of interest to know in which cases we can reuse a set of itemsets to build multiple decision trees.

The main class of constraints used by itemset miners is the class of *anti-monotonic* constraints. We can see that if we find all itemsets satisfying $\varphi_{leaf,antim}(I)$, we find sufficiently many itemsets to build decision trees for the case that $\varphi_{leaf,antim}(I)$ is the leaf constraint. A more interesting question is the reverse question: are all these itemsets needed? The following example illustrates that this is not the case. Assume that $\{A\}$ is a frequent itemset, but $\{\neg A\}$ is not; then no tree will contain a test on feature A, as one of the branches resulting from this test will lead to an infrequent leaf. Consequently, itemset $\{A\}$, even though frequent, is redundant. The following explains how we can characterize the itemsets *relevant* to decision trees induction.

If we consider the DL8 algorithm, an itemset $I = \{i_1, \ldots, i_n\}$ is needed only if there is an order $[i_{k_1}, i_{k_2}, \ldots, i_{k_n}]$ of the items in I (which corresponds to an order of recursive calls of DL8- RECURSIVE) such that for none of the proper prefixes $I' = [i_{k_1}, i_{k_2}, \ldots, i_{k_m}]$ $(m < n)$ of this order:

1. the $\varphi_{internal}(I', I' \cup \{i_{k_{m+1}}\}, I' \cup \{\neg i_{k_{m+1}}\})$ predicate is false;
2. the conjunction $\varphi_{leaf,antim}(I' \cup \{i_{k_{m+1}}\}) \wedge \varphi_{leaf,antim}(I' \cup \{\neg i_{k_{m+1}}\})$ is false.

Definition 2 Let $\varphi_{leaf,antim}$ be an anti-monotonic constraint and $\varphi_{internal}$ be an internal constraint. Then the *relevance* of an itemset I, denoted by $rel(I)$, is defined by

$$rel(I) = \begin{cases} \varphi_{internal}(I), \text{ if } I = \emptyset & \text{(Case 1)} \\ \\ true, \quad \text{if } \exists i \in I \text{ such that} \\ \quad rel(I - i) \wedge \varphi_{internal}(I - i, I, I - i \cup \{\neg i\}) \wedge \\ \quad \varphi_{leaf,antim}(I) \wedge \varphi_{leaf,antim}(I - i \cup \neg i) & \text{(Case 2)} \\ \\ false, \quad \text{otherwise} & \text{(Case 3)} \end{cases}$$

Theorem 1 *Let $\mathcal{L}_1$ be the set of itemsets stored by* DL8*, and let $\mathcal{L}_2$ be the set of itemsets $\{I \subseteq \mathcal{I} | rel(I) = true\}$. Then $\mathcal{L}_1 = \mathcal{L}_2$.*

Proof We consider both directions.
"$\Rightarrow$": if an itemset is stored by DL8, there must be an order of the items in which each prefix satisfies the constraints. Then we can repeatedly pick the last item in this order to find the items that satisfy the constraints in case 2 of the definition of $rel(I)$.
"$\Leftarrow$": if an itemset is relevant, we can construct an order in which the items can be added in the recursion without violating the constraints, as follows. For a relevant itemset there must be an item $i \in I$ such that case 2 holds. Let this be the last item in

the order; then recursively consider the itemset $I - i$. As this itemset is also relevant, we can again obtain an item $i' \in I - i$, and put this on the second last position in the order, and so on. □

If we assume that the internal constraint is also anti-monotonic, relevance can also be used in itemset miners that exploit anti-monotonic constraints.

Theorem 2 *If both the internal constraint and the leaf constraint are anti-monotonic, itemset relevance is an anti-monotonic property.*

Proof By induction. The base case is trivial: if the $\emptyset$ itemset is not relevant then none of its supersets is relevant. Assume that for all itemsets X', X up to size $|X| = n$ we have shown that if $X' \subset X$: $\neg rel(X') \Rightarrow \neg rel(X)$. Assume that $Y = X \cup i$ and that X is not relevant. To prove that Y is not relevant, we need to consider every $j \in Y$, and consider whether case 2 of the definition is true for this j:

- If $i = j$. certainly $Y - i = X$ is not relevant;
- If $i \neq j$. We know that $j \in X$, and given that X is not relevant, either:
  - $rel(X - j) = false$: in this case $rel(Y - j) = rel(X - j \cup i) = false$ (inductive assumption);
  - $\varphi_{leaf,antim}(X) = false$: in this case $\varphi_{leaf,antim}(Y) = false$ (anti-monotonicity of $\varphi_{leaf,antim}$);
  - $\varphi_{leaf,antim}(X - j \cup \neg j) = false$: in this case $\varphi_{leaf,antim}(Y - j \cup \neg j) = \varphi_{leaf,antim}(X - j \cup \neg j \cup i) = false$ (anti-monotonicity of $\varphi_{leaf,antim}$);
  - $\varphi_{internal,antim}(X - j) = false$: in this case $\varphi_{internal,antim}(Y - j) = \varphi_{internal,antim}(X - j \cup i) = false$ (anti-monotonicity of $\varphi_{internal,antim}$).

Consequently, $rel(Y)$ can only be *false*. □

It is relatively easy to integrate the computation of relevance in both breadth-first and depth-first frequent itemset mining algorithms, as long as the order of itemset generation is such that all subsets of an itemset I are enumerated before I is enumerated itself.

We implemented two versions of DL8 in which the relevance constraints are pushed in the frequent itemset mining process: DL8-APRIORI, which is based on APRIORI (Agrawal et al. 1996), and DL8-ECLAT, which is based on ECLAT (Zaki et al. 1997a).

5.3 Computing lattices on the fly

The second option is to access the data while building decision trees. One reason for doing this could be to avoid possible overhead caused by traversing the lattice multiple times. Another, more important, reason involves the possibility to use closed itemsets effectively.

The main observation that we exploit to this purpose is that if we are dealing with a *structure independent* query we can restrict our attention to an even smaller set of itemsets than the relevant itemsets.

The main reason for this is that if two itemsets I and I' cover the same set of examples (i.e., $tid(I) = tid(I')$), and the query is structure independent, the tree(s)

we find for both itemsets must be the same. To reduce the number of itemsets that we have to store, we should avoid storing such duplicate sets of results.

To ensure that results are re-used between itemsets covering exactly the same examples, we propose to compute for every itemset its *closure*. The closure of an itemset I is the largest itemset that all transactions in $tid(I)$ have in common. More formally, let *items* be the function which computes

$$items(tids) = \cap_{k \in tids} t_k$$

for a TID-set *tids*, then the *closure* of itemset I is the itemset $items(tid(I))$. An itemset I is called *closed* iff $I = items(tid(I))$ (Pasquier et al. 1999). If $tid(I_1) = tid(I_2)$ it is easy to see that also $items(tid(I_1)) = items(tid(I_2))$.

We can use this observation by modifying DL8: instead of associating decision trees to itemsets, we associate decision trees to closed itemsets. We change line 5 such that it checks if a decision tree has already been computed for the closure of I; in line 31, we associate computed decision tree(s) to the closure of I instead of to I itself. We refer to this algorithm as DL8- CLOSED.

In practice this means that we build a data structure of closed itemsets instead of ordinary itemsets. Lattices of closed itemsets are also known as *concept lattices*; closed itemsets are also known as *formal concepts*, and have been studied extensively in the literature (Ganter and Wille 1999). In principle, one could also develop a stepwise approach in which one first computes closed itemsets and subsequently mines decision trees. However, in our algorithm we do not only need the closed itemsets; we also need the relationships between them, i.e., if we add an item to an itemset we need to know what the closure of the resulting itemset is. In other words, we do not only need to know the formal concepts, we also need to know the edges in the Hasse diagram of these itemsets. Storing these edges would not only increase the memory requirements of our algorithm, determining them in a post processing step is also not straightforward: a naïve algorithm for computing this diagram would take quadratic time, while also less naïve recent algorithms (such as (Baixeries et al. 2009)) require significant computation times. An approach in which itemsets are mined and decision trees are built at the same time hence seems more promising. The remainder of this section is devoted to an outline of the choices that we made in the integrated approach that we used in our experiments. This approach builds on choices that are commonly made in closed itemset mining algorithms.

The main idea is that during the search, we keep track of those items and transactions that are 'active'. As parameters to DL8- RECURSIVE we add the following:

- the item i that was last added to I;
- a set of *active items*, which includes item i, and represents all tests that can still be added to the itemset $I - i$;
- a set of *active transaction identifiers* representing $tid(I - i)$;
- the set of all items $\mathcal{C}$ that are in the closure of $I - i$, but are not part of the set of active items.

In the first call to DL8- RECURSIVE, all items and transactions are active. At the start of each recursive call (before line 5 of DL8- RECURSIVE is executed) we scan each

active transaction, and test if it contains the last added item i; for each active transaction that contains item i, we determine which other active items it contains. We use this scan to compute the frequency of the active items, and build the new set of active transaction identifiers $tid(I)$. Those active items of which the frequency equals that of I, are added to the closure $\mathcal{C}$. If it turns out we have encountered this closure before, we return the corresponding previously computed result. Otherwise, we now build a new set of active items. For every item we determine if $\varphi_{leaf,antim}(I \cup \{i\})$, $\varphi_{leaf,antim}(I \cup \{\neg i\})$ and the internal constraint $\varphi_{internal}(I, I \cup \{i\}, I \cup \{\neg i\})$ are true; if so, we add the item to the new set of active items. In line 17 we traverse the set of active items. In line 19 and 20 the updated sets of active transactions and active items are passed to the recursive calls. By computing the closure of every itemset, we traverse the Hasse diagram of closed itemsets.

Our approach for maintaining sets of active transactions is akin to the idea of maintaining projected databases that is implemented in ECLAT (Zaki et al. 1997a) and FP- GROWTH (Han et al. 2000). In contrast to these algorithms, we know in our case that we have to maintain projections that contain both an item i and its negation $\neg i$. As we know that $|tid(I)| = |tid(I \cup i)| + |tid(I \cup \neg i)|$, it is less beneficial to maintain TID-sets as in ECLAT, and we prefer a solution in which we call DL8- RECURSIVE with the set of active transactions $tid(I - i)$ instead of $tid(I)$. We project a transaction set by reordering the transactions in an array. Consequently, the memory use of our algorithm is determined by the amount of memory that is needed to store the database and the closed itemsets with associated information. Per closed itemset we only store the associative array $\mathcal{T}$ for later retrieval; to reduce memory demands, we do not store support values, edges of the Hasse diagram, or TID sets. A tree in the associative array is only represented by its root node, as any subtrees can be recovered recursively from information associated to other itemsets. The information that we store for every itemset is hence only determined by the optimization criteria that are used; if we assume the query given, the information stored per itemset is constant. Consequently, the memory use is $\theta(|D| + |\mathcal{S}|)$, where $|\mathcal{S}|$ is the size of a data structure storing all closed itemsets.

Even though we hence attempt to limit the memory required by our algorithm, it should be repeated that the number of closed itemsets can be exponential in the size of the database; in practice the complexity remains high.

5.4 Efficiency improvements

A common trick to improve the efficiency of constraint-based search is to rewrite constraints or to add redundant constraints. For instance, both in research devoted to constraint programming and to database systems it is common to rewrite queries to a form which can be more efficiently evaluated. While some systems—database systems for instance—attempt to optimize certain queries, in many cases such optimization is still a manual effort that is undertaken by a programmer. A similar observation can also be made for decision tree learning queries. The aim of this section is to suggest the use of query rewriting to improve the overall efficiency of our algorithm. We illustrate this

for two tasks that can be performed by our algorithm. We leave the topic of general query optimization as future work.

Finding the smallest most accurate tree Let us consider the following query:

$$\operatorname*{argmin}_{T \in \mathcal{T}}(error(T), size(T)) \text{ subject to } \varphi(T) = \forall I \in paths(T) : freq(I) \geq minfreq$$

We can optimize this query further by the following simple observation: if a tree contains an internal node which is pure, we can remove the tree below this internal node to obtain a tree that is equally accurate but smaller. Hence, in the smallest most accurate tree we will never have pure internal nodes. We can pose this as an additional, redundant internal constraint, and push this in the mining process. In the following theorem we show that this constraint on the global model can even be relaxed further to prune additional itemsets during the local pattern mining step.

Definition 3 For a given itemset I, let us sort the frequencies in the classes in descending order, $freq_1 \geq \ldots \geq freq_n$ (hence $freq_1$ is the frequency of the majority class). Let *minfreq* be the minimum frequency used to build the lattice. An itemset I is *almost-pure* if $(minfreq - \sum_{i=2}^{n} freq_i(I)) > freq_2(I)$.

Theorem 3 *If $freq(I) \geq minfreq$, $almost-pure(I) = true$ and class k is the majority class in I, then for all $I' \supset I$ such that $freq(I') \geq minfreq$, class k is the majority class of I'.*

Proof Let class 1 be the majority class in the examples $t = tid(I)$. Then $freq(I) \geq minfreq \Leftrightarrow \sum_{i=1}^{n} freq_i(I) \geq minfreq \Leftrightarrow freq_1 \geq (minfreq - \sum_{i=2}^{n} freq_i(I))$. Since I is almost-pure we know that $(minfreq - \sum_{i=2}^{n} freq_i(I)) > freq_2(I) \geq 0 \Leftrightarrow \forall i, 2 \leq i \leq n, freq_i < minfreq$.

For class k ($k \neq 1$) to be the majority class in I' with $I' \supset I$ and $freq(I') \geq minfreq$, the number of examples of class k in I' should be higher than the minimum number of examples from class 1 that will still be in I'. This number is at least $(minfreq - \sum_{i=2}^{n} freq_i(I))$. So, for k ($k \neq 1$) to be the new majority class in I', we must have $freq_k \geq (minfreq - \sum_{i=2}^{n} freq_i(I))$ which contradicts the definition of loose-purity for I; therefore class 1 must be the majority class in I'. □

Intuitively, the previous theorem states that if the frequency of the examples belonging to the second majority class is low enough, splitting the tree again will never increase its global accuracy. Hence, if we are searching for accurate and small trees, we can pose the (redundant) constraint that every internal node should not be almost-pure to obtain the correct result.

Finding the smallest sufficiently accurate tree Assume we wish to answer the following query:

$$\operatorname*{argmin}_{T \in \mathcal{T}} size(T) \text{ subject to } \varphi(T)$$

where

$$\varphi(T) = error(T) \leq maxerror \wedge size(T) \leq maxsize \wedge (\forall I \in paths(T) : freq(I) \geq minfreq).$$

In principle we could maintain for every itemset a set of trees of size $O(maxerror \times maxsize)$, but such an approach demands the computation of a very large number of trees for every itemset.

Assume now that we execute the following query:

$$\underset{T \in \mathcal{T}}{\text{argmin}}\ error(T) \text{ subject to } \varphi(T)$$

where

$$\varphi(T) = size(T) \leq maxsize \wedge (\forall I \in paths(T) : freq(I) \geq minfreq).$$

Then we can observe that to answer this query, DL8 will find for every possible size up to the maximum size, the most accurate tree of that size. We can easily modify DL8 in line 2 to answer our original query, as it suffices to find the smallest tree in this set which satisfies the accuracy constraint.

6 Experiments

The aim of this section is to answer the following questions :

1. How does an exact tree learner, such as DL8, compare to a well-known heuristic learner in terms of the compromise between size and accuracy?
2. What is the influence of DL8's mandatory constraint (the *minfreq* constraint) on the accuracy of the trees it learns?
3. How well does DL8 perform on non-traditional tree learning problems, such as learning decision trees under cost or size constraints?

With respect to efficiency, we aim to answer the following questions:

4. How much do the constraints on the trees help the local pattern mining phase?
5. How much does restricting the search to closed itemsets improve the overall efficiency?

The experiments were performed on UCI datasets (Asuncion and Newman 2007). Numerical datasets were discretized using WEKA's (Witten and Frank 2005) unsupervised discretization method with an equal-frequency repartition and a number of bins equal to 4 before applying the learning algorithms. We limited the number of bins in order to limit the number of created features. Table 1 gives a brief description of the datasets that we used in terms of the number of examples and the number of attributes after binarization (features). All experiments were performed on Intel Pentium 4 machines with in between 1 GB and 4 GB of main memory, running Linux. DL8 and the frequent itemset miners were implemented in C++.

Table 1 Dataset description

Dataset	#Ex	#Features	Dataset	#Ex	#Features
Anneal	812	36	Pendigits	7494	49
a-credit	653	56	p-tumor	336	18
Balance	625	13	Segment	2310	55
Breast-w	683	28	Soybean	630	45
Chess	3196	41	Splice	3190	3466
Diabetes	768	25	Thyroid	3247	36
g-credit	1000	77	Vehicle	846	55
Heart-c	296	35	Vote	435	49
Ionosphere	351	99	Vowel	990	48
Mushroom	8124	116	Yeast	1484	23

6.1 Quality evaluation

In this section we determine the performance of DL8 when mining trees under constraints. Thus, we aim at answering Questions (1), (2), and (3). Note that no implementation of alternative exact decision tree learners is available. As a baseline to evaluate our results, we decided to use J48, the Java implementation of the heuristic decision tree learner *C4.5* (Quinlan 1993) in WEKA. Without guaranteeing an exact solution, *C4.5* supports some constraints, such as a minimum support on nodes after a split, which corresponds to the anti-monotonic "frequency" constraint in our case. *C4.5* is known to generalize well and give good test accuracy results on many machine learning problems.

To answer Question (1) and, in particular, to evaluate the accuracy of the trees computed by DL8 on test data, we obviously cannot optimize the test set accuracy directly (and thus we cannot guarantee better results on test data): we need to optimize regularized measures on the training data if we wish to avoid overfitting. When using the standard error function (*error*) for *C4.5*, we let *C4.5* continue splitting as long as the training set error is reduced and the frequency constraint is not violated. When using the reduced-error function (f_p) the default pruning criterion in *C4.5* is applied.

We used a stratified 10-fold cross-validation to compute the training and test accuracies of both systems and a corrected two-tailed t-test (Nadeau and Bengio 2003) with a significance threshold of 5% to compare the test accuracies; 0 indicates that no significant difference was observed, $-$ indicates that J48 is significantly better, $+$ that DL8 is significantly better.

As explained in Sect. 5, the bottleneck of our algorithm is the itemset mining phase and the in-memory construction of the lattice which is part of this phase. Therefore, the application of our algorithm is limited by the amount of memory available for the construction of this lattice. In the following experiments, unless specified differently, the *minfreq* threshold used for DL8 for each dataset is the lowest one we could use for the given dataset without running out of memory.

6.1.1 Heuristic versus exact decision tree learning

The main alternative for our approach is the heuristic approach. Results for the heuristic approach can provide us a lower-bound on the accuracy that can be achieved on the datasets that we use in this article. Similarly, in order to study the effects of constraints that we will use later in our experiments, it is of interest to determine the accuracies that DL8 is able to achieve when using the least restrictive constraint possible, which we assume to be the $minfreq = 2$ threshold used in J48. Results for this setting, when applying the f_p optimization criterion, are given in Table 2 for those datasets where the use of this support threshold is feasible for DL8. We used this pruning criterion as both DL8 and J48 overall achieve the best test-set accuracy for this criterion (not shown here).

An important element of our approach is the discretization in binary attributes, which is not needed in J48. To determine the influence of our discretization on the resulting trees, we applied J48 both on the original, undiscretized data and the discretized data that was also used as input to DL8. The table shows that the optimal trees computed by DL8 have a better training accuracy than the trees computed by J48 with the same discretization. When pruned trees are compared to unpruned ones (see Table 3), the trees are on average 1.75 times smaller for J48 and 1.5 time smaller for DL8. After pruning, DL8's trees are still 1.5 times larger than J48's pruned trees. In one case (balance-scale), the tree computed by DL8 is significantly smaller on average (65.4 nodes) than the one computed by J48 (72.4 nodes) for a similar test accuracy. Furthermore, on the test data, there is almost no significant difference (expect for the *yeast* dataset) between J48 and DL8.

This shows that the quality of the trees computed by DL8 remains competitive with J48's one. However, using a simpler heuristic decision tree learner is sufficient to obtain good test-set accuracy and this might yield smaller trees.

Unfortunately, as can be seen in Table 3, the number of datasets for which a minimum support threshold of 2 is feasible in DL8 is limited. Using higher thresholds could allow us to compare our algorithm on a larger number of datasets. The influence of this constraint is hence studied in the next section.

Table 2 Comparison of J48 (with pruning, without constraints) and DL8 (with pruning, without constraints)

	Train acc.		Test acc.			Test significance		Size	
Datasets	DL8	J48	DL8	J48 Disc.	J48 No D.	Disc.	No D.	J48	DL8
Anneal	0.87	0.86	0.82	0.82	0.88	0	–	44.4	45.6
Balance	0.89	0.89	0.80	0.80	0.79	0	0	72.4	65.4
Breast-w	0.98	0.97	0.96	0.96	0.95	0	0	15.6	18.0
Diabetes	0.92	0.84	0.71	0.74	0.73	0	0	69.0	135.2
Heart-c	0.97	0.90	0.77	0.78	0.80	0	0	31.6	50.2
p-tumor	0.67	0.60	0.40	0.40	0.40	0	0	81.2	105.2
Yeast	0.75	0.68	0.50	0.53	0.56	–	–	186.0	307.2

Table 3 Comparison of J48 (without pruning, with constraints) and DL8 (without pruning, with constraints)

	Minfreq		Train acc.		Test acc.			Size	
Datasets	#	%	J48	DL8	J48	DL8	Sign	J48	DL8
Anneal	2	0.2	0.89	0.89	0.82	0.82	0	106.6	87.8
a-credit	40	6.1	0.87	0.89	0.85	0.88	+	6.4	11.0
Balance	2	0.3	0.90	0.90	0.82	0.81	0	99.0	114.4
Breast-w	2	0.3	0.98	1.00	0.95	0.94	0	31.6	48.0
Chess	200	6.2	0.91	0.91	0.91	0.91	0	9.0	8.6
Diabetes	2	0.2	0.90	0.99	0.68	0.66	0	200.2	288.4
g-credit	150	15.0	0.72	0.74	0.71	0.73	0	6.4	7.0
g-credit	100	10.0	0.73	0.75	0.70	0.70	0	6.4	11.6
Heart-c	2	0.6	0.94	1.00	0.76	0.74	0	67.6	74.4
Ionosphere	50	14.2	0.83	0.86	0.79	0.84	+	4.0	7.4
Ionosphere	40	11.3	0.89	0.89	0.88	0.88	0	5.0	6.8
Mushroom	600	7.4	0.92	0.98	0.92	0.98	+	5.0	13.8
Pendigits	470	6.3	0.68	0.75	0.67	0.75	+	21.0	21.0
p-tumor	2	0.5	0.63	0.71	0.40	0.36	0	116.4	152.2
Segment	150	6.5	0.77	0.86	0.76	0.85	+	15.6	16.8
Segment	120	5.2	0.84	0.87	0.84	0.87	+	19.8	25.8
Soybean	40	6.3	0.58	0.65	0.57	0.66	+	17.0	20.6
Splice	700	21.9	0.74	0.74	0.74	0.73	0	5.0	5.0
Thyroid	80	2.4	0.91	0.92	0.91	0.91	0	1.0	13.4
Thyroid	40	1.2	0.92	0.92	0.91	0.91	0	9.2	34.4
Vehicle	50	5.9	0.63	0.71	0.59	0.67	+	17.0	22.4
Vote	10	2.3	0.96	0.98	0.94	0.93	0	4.6	29.6
Vowel	65	6.6	0.40	0.47	0.35	0.43	+	19.2	22.6
yeast	2	0.1	0.74	0.82	0.49	0.48	0	501.2	724.2

In these experiments the *error* optimization criterion is used. The first two columns give the *minf req* threshold in terms of absolute number and percentage of the examples that should be covered by each leaf. The next columns give respectively the training accuracies, test accuracies, and size of the trees built by J48 and DL8 for the given *minfreq* threshold. A significance comparison of the test accuracies of both systems (Sign) is also given

6.1.2 Frequency constraint

As the use of a frequency constraint is an important property of our approach, we determine the influence of this constraint.

As a baseline, we first run both J48 and DL8 on all datasets, for the lowest value for which the execution of DL8 is feasible. We used the training set error (*error*) as optimization criterion. Table 3 shows that, when optimizing *error*, for both training and test accuracies, DL8 is significantly better than J48 on 9 of the 20 datasets, and not worse in the other cases. Hence, we can confirm that our results of the previous section also apply for other values of the frequency constraint.

Secondly, we evaluate how different values of this mandatory constraint influence the accuracy of trees learned by our algorithm. To this aim, Table 3 includes runs of DL8 for multiple minimum frequencies. We gradually decrease the frequency thresholds both when using *error* and f_p as optimisation criteria. As we lower the support thresholds, we observe that the training accuracy increases, but the experiments on the seven datasets for which we were able to reach a *minfreq* of 2, indicate that for test set accuracy, low thresholds are not always the best option. For example, Fig. 3 shows the evolution of the training and test accuracies of both systems when optimizing the reduced-error function f_p and increasing the support. As expected, the training accuracies always decrease when the support increases. However, this behavior is less clear for test accuracies. For example, for the diabetes dataset, the test accuracies clearly increase with the minimum support. The main explanation for this is probably that the support constraint acts as a regularization parameter and helps to avoid overfitting.

These experiments show that when both algorithms try to optimize the same criterion (here the *training set error*) using the same path constraint (support threshold), the exact decision tree learner DL8 gives better results than the heuristic one. Furthermore, the mandatory constraint of DL8 is not necessarily a drawback when looking for decision trees with high test accuracy.

Until now, the results confirm the intuition that high values for the frequency constraint can lead to worse accuracies. Indeed, when in Table 4 we compare the performance of DL8 to J48 on datasets which were not included in Table 2, we observe that the support threshold can have a strong negative influence on the accuracies achieved. For example, for the *pendigits* dataset, the support threshold that we reach for DL8 is 470 (compared to 2 for J48) and the test accuracy drops from 0.95 for J48 to 0.75 for DL8.

In such negative cases, the flexibility of DL8 can be particularly interesting. For example, when the lowest minimum frequency threshold for which we could run our algorithm is much higher than the number of examples in the smallest class of the dataset, we can use a *disjunction of minimum support constraints for each class* instead of the classical *minfreq* to obtain more accurate trees. In this setting, every class is given the same (relative) minimum support constraint. In this way, we allow that a leaf covers a small number of examples if all examples belong to the same small class, but we do not allow that a leaf contains a small number of examples if they belong to many different classes, or to one big class. The results of these experiments on the negative cases of Table 4 are shown in Table 5. We compare the training and test set accuracies and the size of DL8 when optimizing the f_p criterion and using the lowest minimum support threshold that we can reach within the available memory. As we can see, the accuracies of DL8 increase significantly in all cases, which shows the interest of putting such constraints when the class distribution of the data is known beforehand.

6.1.3 Untraditional tasks

This section aims at answering Question (3). By answering this question, we also aim at answering another underlying question: when is it more interesting to use an exact

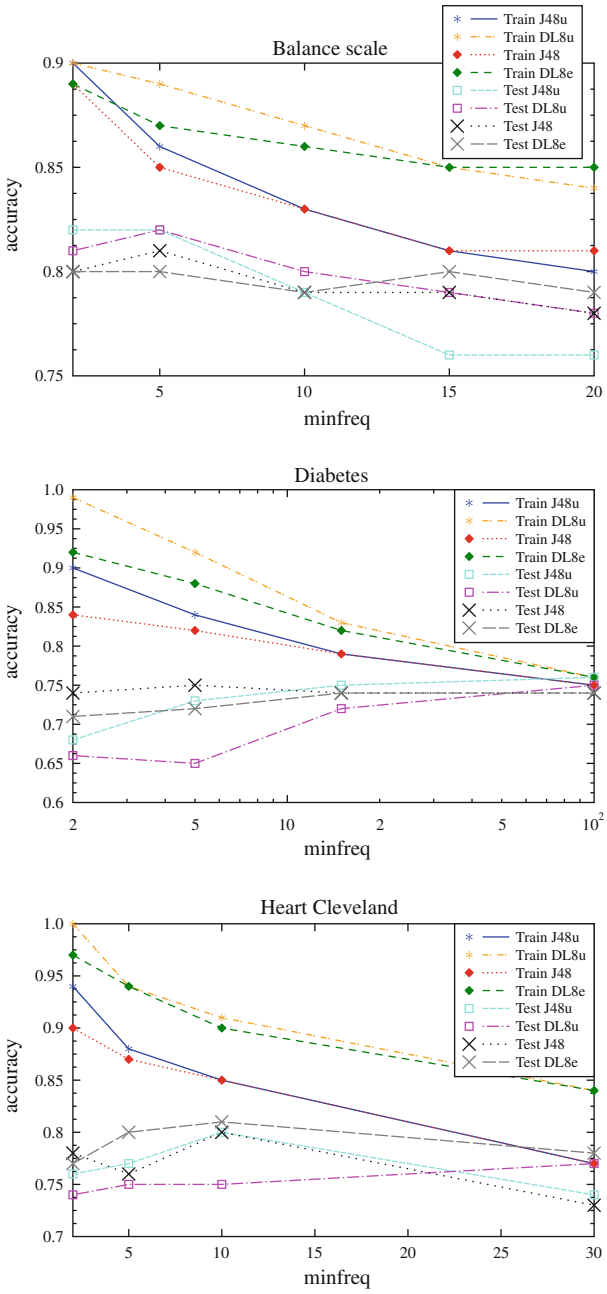


Fig. 3 Evolution of the training and test accuracies of J48 (unpruned and pruned) and DL8 (*error* and f_p optimization functions) for various minimum frequency thresholds

algorithm potentially less efficient and not necessarily more accurate than a heuristic one to learn decision trees?

Table 4 Comparison of J48 (with pruning, without constraints) and DL8 (with pruning, with constraints)

Datasets	Minfreq		Test acc.	
	#	%	J48	DL8
a-credit	40	6.1	0.84	0.88
Chess	200	6.2	0.99	0.91
g-credit	150	15	0.71	0.73
Ionosphere	50	14.2	0.80	0.84
Mushroom	600	7.4	1.00	0.98
Pendigits	470	6.3	0.95	0.75
Segment	150	6.5	0.95	0.85
Soybean	40	6.3	0.82	0.66
Splice	700	21.9	0.94	0.73
Thyroid	80	2.4	0.91	0.91
Vehicle	50	5.9	0.70	0.67
Vote	10	2.3	0.96	0.93
Vowel	65	6.6	0.53	0.43

In these experiments a pruning criterion was used, a frequency threshold of 2 for J48 and higher values for DL8

Table 5 Results of DL8 using a disjunction of frequency constraints

Datasets	One frequency constraint				Disjunction of frequency constraints			
	Minfreq %	Train acc.	Test acc.	Size	Minfreq %	Train acc.	Test acc.	Size
Pendigits	6.3	0.75	0.75	21.2	35.0	0.82	0.82	30.8
Segment	5.2	0.87	0.86	21.2	22.0	0.91	0.90	33.0
Soybean	6.3	0.65	0.66	20.6	35.0	0.80	0.78	38.0
Splice	21.9	0.74	0.73	5.0	32.0	0.78	0.79	7.2
Vowel	6.6	0.47	0.43	21.8	14.0	0.73	0.65	88.6
Yeast	0.1	0.75	0.50	307.2	2.0	0.67	0.52	136.4

In this experiment the f_p optimization function was used with the lowest reachable frequency constraints and a disjunction of class minimum frequency constraints

Learning under size constraints In (Murphy and Pazzani 1997), the authors found that a brute-force enumeration of trees leads can lead to good results for slightly larger trees. To explore this result in more detail, and again to show the flexibility of our algorithm, we investigate how much accuracy is affected if we impose exact size constraints. We use DL8 to compute, for every possible size of a decision tree, the tree with the smallest (pruned) error that can be achieved, and apply this tree on training and test data under ten-fold cross validation. For two datasets, the results of such a query are given in Fig. 4.

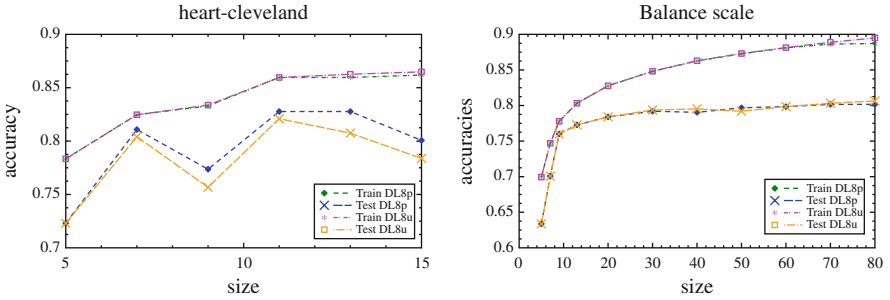


Fig. 4 Errors of decision trees as function of tree size

On the training data, in general, if we increase the size of a decision tree, its accuracy improves quickly at first. Only small improvements can be obtained by further increasing the size of the tree.

On the test data, the effects are less clear. Increasing the tree size can lead to either better or worse results. On balance scale, increasing the size of the tree leads to both better training and test results; the principle of Occam's razor does not seem to apply in this case. On the other hand, on heart-cleveland we observe that an increase in tree size does not lead to significantly better trees on training data, but results in significantly worse trees on test data. In this case, smaller trees would be preferable. Overall, if results on the training data are not significantly different, it seems preferable to choose a smaller tree. Furthermore, we can observe that the figures have a 'tail' in which an increase in allowed tree size leads to increasingly less benefits on the test data.

Answering Question (3), figures such as Fig. 4 are of practical interest, as they allow a user to trade-off the interpretability and the accuracy of a model. Furthermore, when a weighted sum of accuracy and error is used as optimization criterion, as in dyadic decision trees, points on this curve correspond to particular choices for these weights, and can easily be computed by our algorithm.

The two last columns of Table 3 show that the trees computed by DL8, although not less accurate, can be bigger than the trees computed by J48. To investigate the relationship between accuracy and size in more detail for a larger number of datasets, we decided to investigate whether a constraint on size would significantly worsen the performance of DL8. In Table 6, we show results in which the average size of trees constructed by J48 is taken as a constraint on the maximal size of trees mined by DL8. The results given by DL8 are neither significantly better nor significantly worse than those given by J48, in terms of both size and test set accuracy. Furthermore, imposing the size constraint does not significantly affect the accuracy in most cases. This is an indication that for these datasets, we are in the tails that can be seen in Fig. 4, where it makes sense to impose an explicit size constraint, even when a pruning measure is applied.

Learning under cost constraints The second case on which we explore the use of exact decision tree learners is learning trees under cost constraints. We compare DL8 to ICET (Inexpensive Classification with Expensive Tests) (Turney 1995), introduced in Sect. 4.2.4. The aim is to induce decision trees which minimize $f_q(T) = f_{tq}(T) +$

Table 6 Results of DL8 for different size constraints

Dataset	Minfreq	Max size DL8	Test acc			Size		
			J48	DL8 w. size	DL8 w/o size	J48	DL8 w. size	DL8 w/o size
Diabetes	2	69	0.75	0.72	0.71	69.0	68.8	135.2
g-credit	100	7	0.70	0.72	0.71	6.7	7.0	6.8
Heart-c	10	14	0.80	0.80	0.81	14.0	13.0	22.2
Vote	15	4	0.95	0.96	0.95	3.4	3.0	9.2
Yeast	2	186	0.53	0.52	0.50	186.0	185.0	307.2

Shown are test accuracies for DL8 using the f_p optimization criterion

$f_{mq}(T)$; ICET uses a genetic algorithm for this purpose. The comparison is made using the five well known datasets from the UCI repository (Asuncion and Newman 2007) for which test costs are provided (BUPA Liver Disorders, Heart Disease, Hepatitis Prognosis, Pima Indians Diabetes and Thyroid Disease). We binarized the input data before using DL8 similarly as in the previous section. A comparison is given in Fig. 5. The figure shows the *average cost of classification* given by the algorithms as a percentage of the *standard cost* of classification for different misclassification costs.

The *average cost of classification* is computed by dividing the total cost of applying the learned decision tree on all test examples by the number of examples in the test set. The total cost of using a given decision tree on an example occurring in a leaf I is $tq(I) = \sum_{i \in I} tq_i + \sum_{g \in \{g_i | i \in I\}} tq_g$, i.e., the sum of all tests that are chosen and the misclassification cost as specified in the misclassification matrix $Q_{i,j}$. Let $p_c \in [0, 1]$ be the frequency of class c in the given dataset, i.e, the fraction of the examples in the dataset that belong in class c. Let T be the total cost of performing all possible tests (counting only once the additional cost for the tests in the same group). The *standard cost* is $T + \min_c(1 - p_c) \max_{i,j} Q_{i,j}$. The second term is computed from the frequency of the majority class in the dataset and the highest misclassification cost that an algorithm can have if examples are incorrectly classified as the majority class.

In the experiments, we vary the misclassification costs (as specified in the matrix $Q_{i,j}$) from \$10 to \$10,000. For the sake of simplicity, we consider simple cost matrices i.e, all misclassification costs are equal. The lowest frequency threshold we could use for DL8 is 2 for the BUPA Liver Disorders dataset, 16 for the Heart Disease dataset, 5 for the Hepatitis Prognosis dataset, 15 for the Pima Indians Diabetes dataset and 55 for the Thyroid Disease dataset. Note that these supports can be higher than those reported in Tables 2 and 3 because the syntax dependent optimization criterion means we cannot use the smaller set of closed itemsets.

The results show a better performance for DL8 for four of the five datasets. However, for the ann-thyroid dataset, DL8's results are worse for high misclassification costs ($> 10^3$). Further investigations revealed that this behavior is the result of the low number of bins that we used in our discretization, which resulted in an error

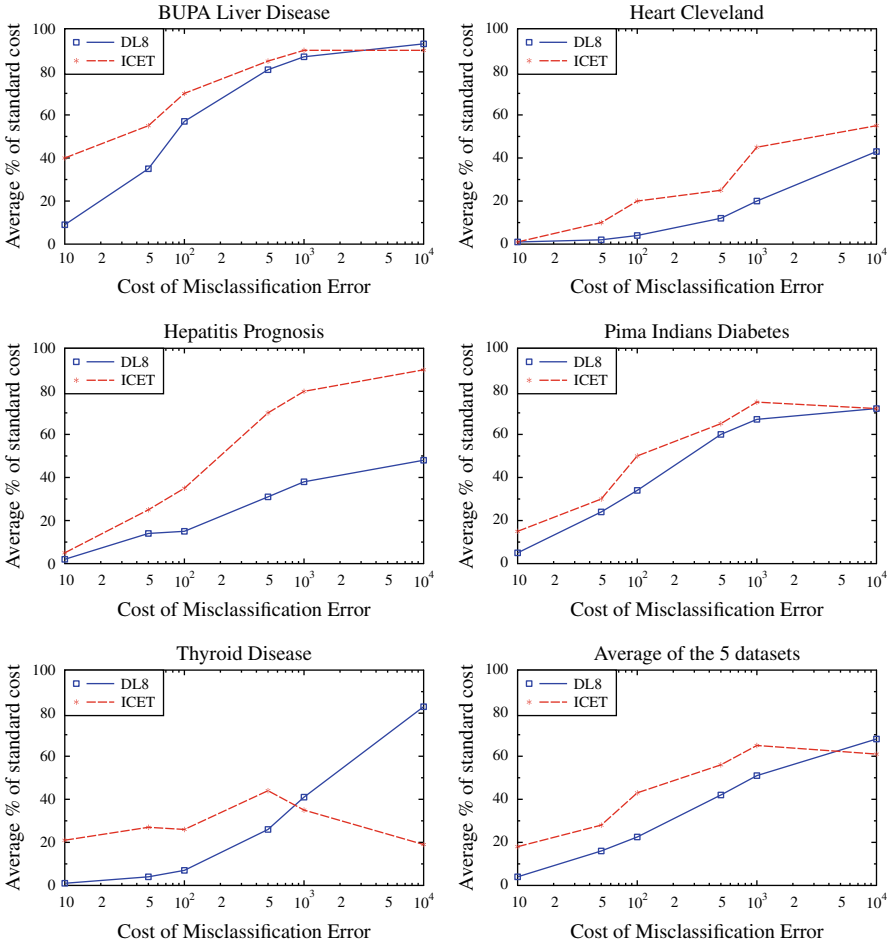


Fig. 5 Comparison of the cost-sensitive decision trees ICET and DL8 with various frequency thresholds (the lower curve, the better)

rate that was close to that of a majority classifier in this very unbalanced dataset (three classes with distribution (93, 191, 3488)). Once the same discretization was used, the error rates were more similar to each other, and the difference in behavior disappeared.

6.2 Efficiency evaluation

We argued that we can construct decision trees both from itemset lattices as from sets of closed itemsets; we can do so while mining itemsets, or by post-processing itemsets. In this section we compare these different versions of DL8 in terms of efficiency.

The applicability of DL8 is limited by two factors: the amount of itemsets that need to be stored, and the time that it takes to compute these itemsets. To answer Ques-

tion (4) we evaluate experimentally how the run time of the pattern mining process is influenced by the relevance constraint and the support constraint. To answer Question (5) we perform this comparison for several alternative approaches for constructing decision trees from patterns, one of which operates on concept lattices. A summary of the algorithms can be found in Table 7. DL8- CLOSED implements the direct mining algorithm of Sect. 5.3. DL8- ECLAT extends the ECLAT algorithm (Zaki et al. 1997a) to search for itemsets and build a decision tree; DL8- APRIORI extends APRIORI with relevance pruning and builds a tree on the resulting lattice. We also include unmodified implementations of the frequent itemset miners APRIORI (Agrawal et al. 1996), ECLAT (Zaki et al. 1997a) and LCM (Uno et al. 2004) in the comparison. These implementations were obtained from the FIMI website (Bayardo et al. 2004). The inclusion of unmodified algorithms allows us to determine how much the search space is reduced by the anti-monotonic relevance pruning, and allows us to determine the trade-off between relevance pruning and trie construction. In APRIORI- FREQ+DL8 we first run traditional APRIORI to construct an itemset lattice without relevance pruning; we run DL8 in a second phase on the constructed lattice.

Results for eight datasets are shown in Figs. 6 and 7. We chose datasets that cover a broad range of dataset properties, including both datasets with large and small numbers of features and transactions. In these runs we computed the most accurate tree given only a minimum frequency constraint. We aborted runs of algorithms that lasted for longer than 1800s.

Answering Question (5), the results clearly show that in all cases the number of closed relevant itemsets is the smallest, which shows the advantage of using closed itemsets. DL8- CLOSED is usually faster than DL8- APRIORI or DL8- ECLAT. For the datasets with larger number of features, such as ionosphere and splice, we found that only DL8- CLOSED managed to run for support thresholds lower than 25%, but still was unable to run for support thresholds lower than 10%. The differences between closed relevant itemsets and non-closed relevant itemsets are smaller for higher minimum support values; the overhead of DL8- CLOSED seems too large in this case.

With respect to Question (4), we can observe that the difference between the number of relevant itemsets and the total number of frequent itemsets becomes smaller for lower minimum frequency values (for good examples, consider the zoo data and

Table 7 Properties of the algorithms used in the experiments

Algorithm	Uses relevance	Closed	Builds tree
DL8- CLOSED	X	X	X
DL8- APRIORI	X		X
DL8- ECLAT	X		X
APRIORI- FREQ			
APRIORI- FREQ+DL8			X
ECLAT- FREQ			
LCM- FREQ			
LCM- CLOSED		X	

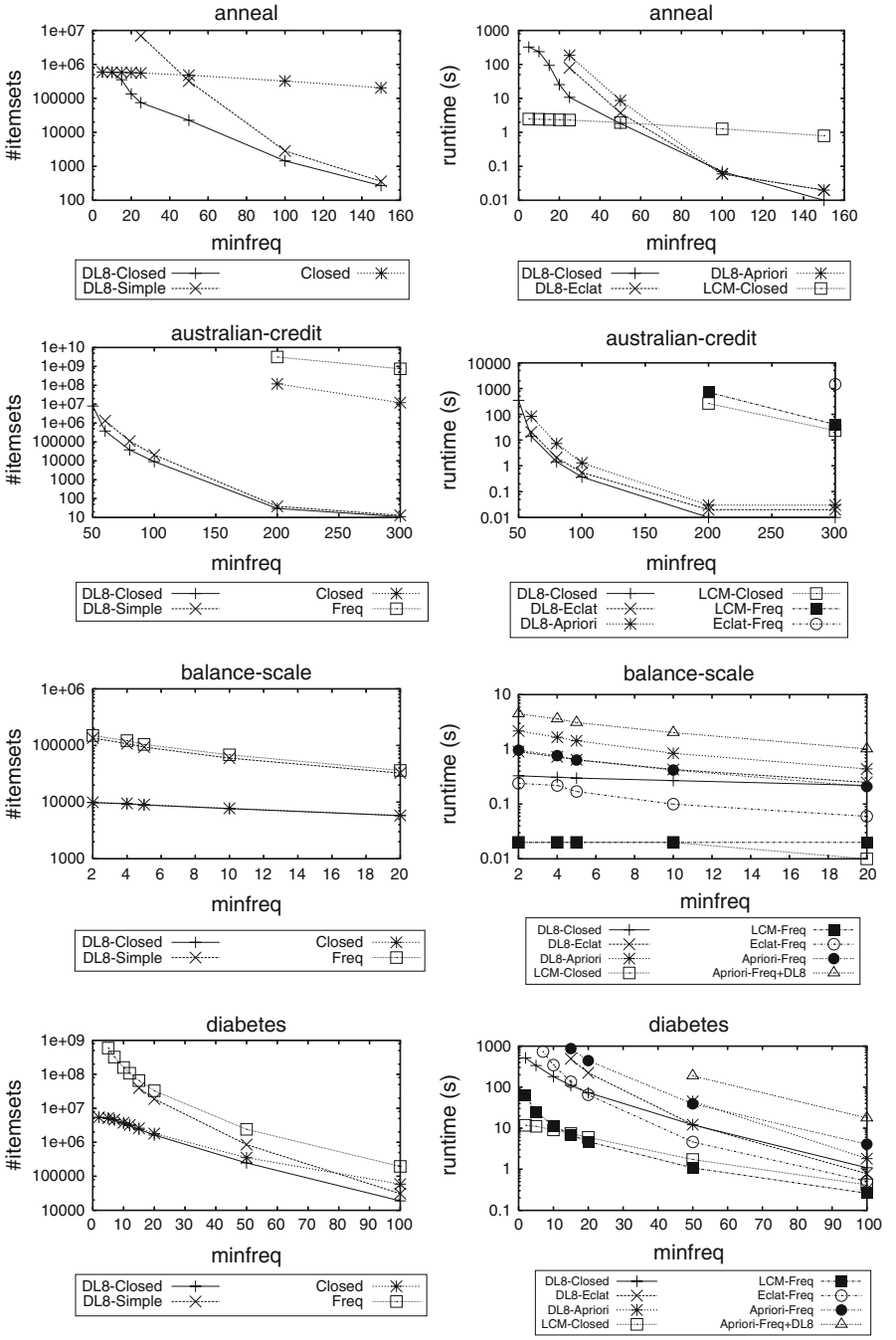


Fig. 6 Comparison of the different miners on 8 UCI datasets (1/2)

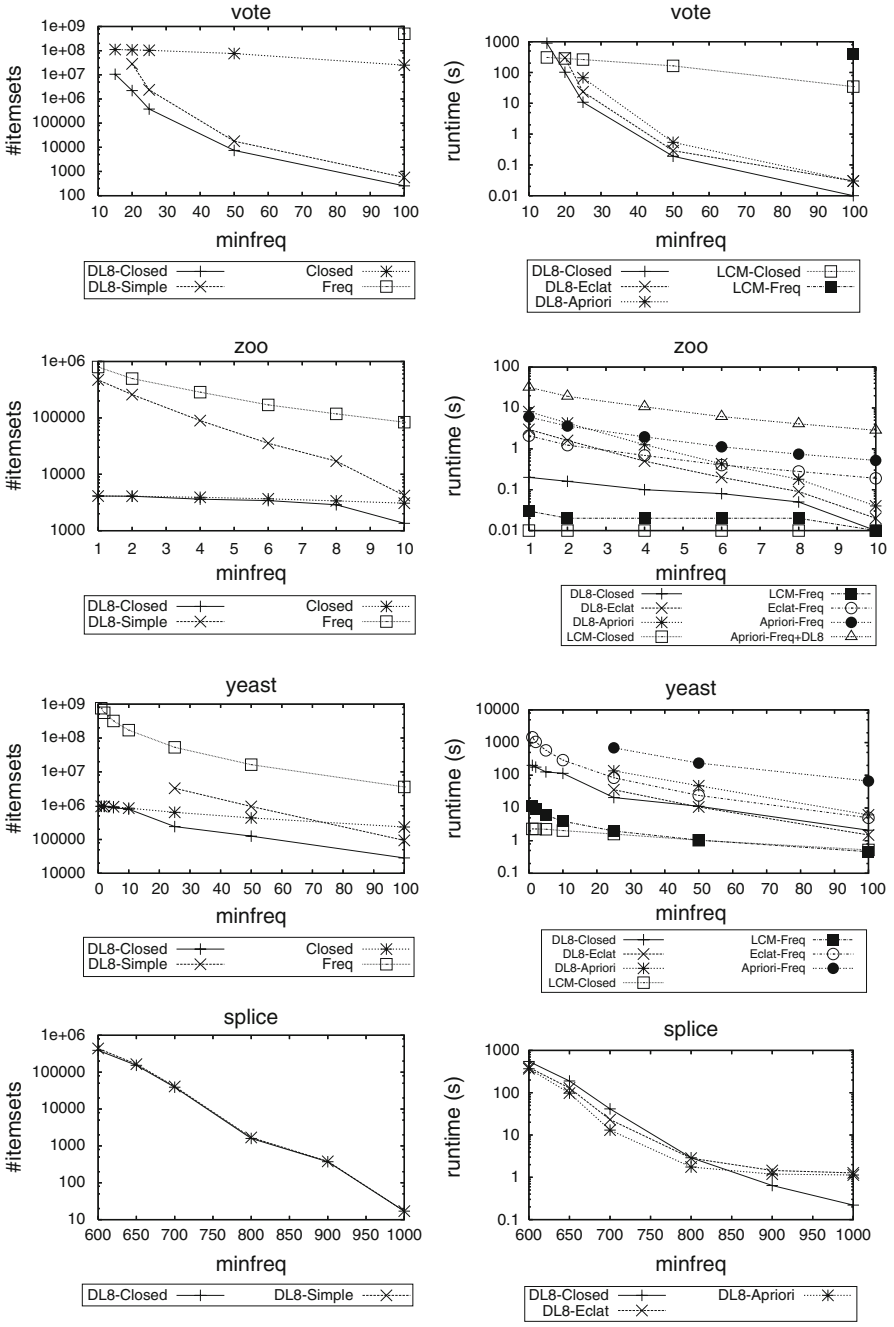


Fig. 7 Comparison of the different miners on 8 UCI datasets (2/2)

the diabetes data). The number of frequent itemsets is so large in most cases, that it is impossible to compute or store them within a reasonable amount of time or space. In those datasets where we can use low minimum frequencies (15 or smaller), the closed itemset miner LCM is usually the fastest; for low frequency values the number of closed itemsets is almost the same as the number of relevant closed itemsets. Bear in mind, however, that LCM does not output the itemsets in a form that can be used efficiently by DL8.

In those cases where we can store the entire output of APRIORI in memory, we see that the additional runtime for storing results is significant. On the other hand, if we perform relevance pruning, the resulting algorithm is usually faster than the original itemset miner.

7 Conclusions

We presented DL8, an algorithm for finding decision trees that minimizes an optimization criterion exactly under a wide range of constraints. This algorithm is based on the relationship between itemsets and decision trees and relies on the construction of an itemset lattice through standard data mining techniques.

With its very general framework, DL8 allows a user to enforce constraints that have never been combined before in a single algorithm. Experiments show that: (i) these constraints can improve the resulting accuracy of a tree; (ii) an exact algorithm can indeed give significantly better results than a heuristic learner if the optimisation criterion is well-defined; (iii) exact results allow to study the behavior of the trees with respect to constraints.

The investigation that we presented here may only be a starting point in this direction; it is an open question how efficient decision tree miners could become if they were thoroughly integrated with algorithms such as LCM, FP-Growth, or algorithms developed within the formal concept analysis community for processing (concept) lattices. Our investigations showed that high runtimes are however not as much a problem as the amount of memory required for storing huge amounts of itemsets. A challenging question for future research is what kind of condensed representations could be developed to represent the information that is used by DL8 more compactly; an alternative could be to trade space and time complexity more carefully.

DL8 can be seen as a relatively cheap type of post-processing on a set of itemsets. In particular, it does not require access to the training data when the model is constructed, in contrast to other approaches that use patterns for classification. Hence DL8 suits itself perfectly for interactive data mining on stored sets of patterns. This means that DL8 might be a key component of inductive databases (Imielinski and Mannila 1996) that contain both patterns and data.

Acknowledgements Siegfried Nijssen was supported by the EU FET IST project "Inductive Querying", contract number FP6-516169, and is currently supported by the Research Foundation – Flanders. During this work, Elisa Fromont was working at the Katholieke Universteit Leuven. Elisa Fromont was partially supported by the BINGO2 project (ANR-07-MDCO 014-02) and the GOA project 2003/8 "Inductive Knowledge bases". The authors thank Luc De Raedt and Hendrik Blockeel for many interesting discussions; Ferenc Bodon and Bart Goethals for putting online their implementations of respectively APRIORI

and ECLAT, which we used to implement DL8, and Takeaki Uno for providing LCM. We also wish to thank Daan Fierens for preprocessing the data that we used in our experiments.

References

Agrawal R, Srikant R (1994) Fast algorithms for mining association rules in large databases. In: Bocca JB, Jarke M, Zaniolo C (eds) VLDB'94, proceedings of 20th international conference on very large data bases. Morgan Kaufmann, pp 487–499

Agrawal R, Mannila H, Srikant R, Toivonen H, Verkamo AI (1996) Fast discovery of association rules. In: Fayyad UM, Piatetsky-Shapiro G, Smyth P, Uthurusamy R (eds) Advances in knowledge discovery and data mining. AAAI/MIT Press, pp 307–328

Angelopoulos N, Cussens J (2005) Exploiting informative priors for Bayesian clas-sification and regression trees. In: Kaelbling LP, Saffiotti A (eds) IJCAI-05, proceedings of the 19th international joint conference on artificial intelligence. Professional Book Center, pp 641–646

Asuncion A, Newman D (2007) UCI machine learning repository. University of California, Irvine, School of Information and Computer Sciences. http://www.ics.uci.edu/~mlearn/MLRepository.html

Baixeries J, Szathmary L, Valtchev P, Godin R (2009) Yet a faster algorithm for building the Hasse diagram of a concept lattice. In: Ferre S, Rudolph S (eds) ICFCA'09, proceedings of the 7th international conference on formal concept analysis. Springer, pp 162–177

Bayardo R, Goethals B, Zaki MJ (eds) (2004) FIMI '04, proceedings ofthe IEEE ICDM workshop on frequent itemset mining implementations. CEUR-WS.org

Blanchard G, Schafer C, Rozenholc Y, Müller KR (2007) Optimal dyadic decision trees. Mach Learn 66(2–3):209–241

Bonchi F, Lucchese C (2007) Extending the state-of-the-art of constraint-based pattern discovery. Data Knowl Eng 60(2):377–399

Boulicaut J-F, Bykowski A, Rigotti C (2000) Approximation of frequency queries by means of free-sets. In: Zighed DA, Komorowski HJ, Zytkow JM (eds) PKDD'00, proceedings ofthe 4th European conference on principles of data mining and knowledge discovery. Springer, pp 75–85

Boulicaut J-F, Bykowski A, Rigotti C (2003) Free-sets: a condensed representation of boolean data for the approximation of frequency queries. Data Mining Knowl Discov 7(1):5–22

Breiman L, Friedman JH, Olshen RA, Stone CJ (1984) Classification and regression trees. Wadsworth Publishing Company, Belmont, California, USA

Bringmann B, Nijssen S, Zimmermann A (2009) Pattern-based classification: a unifying perspective. In: Knobbe A, Furnkranz J (eds) LeGo '09, proceedings of the ECML PKDD 2009 workshop 'From Local Patterns to Global Models'

Bucila C, Gehrke J, Kifer D, White WM (2003) DualMiner: a dual-pruning algorithm for itemsets with constraints. Data Mining Knowl Discov 7(3):241–272

Buntine W (1992) Learning classification trees. Stat Comput 2:63–73

Chipman HA, George EI, McCulloch RE (1998) Bayesian CART model search. J Am Stat Assoc 93(443):935–947

De Raedt L, Zimmermann A (2007) Constraint-based pattern set mining. In: Parthasarathy S, Liu B (eds) SDM'07, proceedings of the seventh SIAM international conference on data mining. SIAM Press, pp 1–12

Dong G, Zhang X, Wong L, Li J (1999) CAEP: classification by aggregating emerging patterns. In: Arikawa S, Furukawa K (eds) DS'99, proceedings of the second international conference on discovery science. Springer, pp 30–42

Esmeir S, Markovitch S (2007a) Anytime induction of cost-sensitive trees. In: Koller D, Schuurmans D, Bengio Y, Bottou L (eds) NIPS'07, proceedings of the 21st conference on neural information processing systems. MIT Press, pp 425–132

Esmeir S, Markovitch S (2007b) Anytime learning of decision trees. J Machine Learn Res 8:891–933

Friedman A, Schuster A, Wolff R (2006) *k*-anonymous decision tree induction. In: Furnkranz J, Scheffer T, Spiliopoulou M (eds) PKDD'06, proceedings ofthe 10th European conference on principles and practice of knowledge discovery in databases. Springer, pp 151–162

Ganter B, Wille R (1999) Formal concept analysis: mathematical foundations. Springer, Berlin

Garey MR (1972) Optimal binary identification procedures. SIAM J Appl Math 23(2):173–186

Garofalakis MN, Hyun D, Rastogi R, Shim K (2003) Building decision trees with constraints. Data Min Knowl Discov 7(2):187–214

Han J, Pei J, Yin Y (2000) Mining frequent patterns without candidate generation. In: Chen W, Naughton J, Bernstein PA (eds) SIGMOD'00, proceedings ofthe 2000 ACM SIGMOD international conference on management of Data. ACM Press, pp 1–12

Hyafil L, Rivest RL (1976) Constructing optimal binary decision trees is NP-complete. Inf Process Lett 5(1):15–17

Imielinski T, Mannila H (1996) A database perspective on knowledge discovery. Commun ACM 39:58–64

Knobbe A, Ho E (2006) Maximally informative k-itemsets and their efficient discovery. In: Eliassi-Rad T, Ungar LH, Craven M, Gunopulos D (eds) KDD'06, proceeding ofthe 12th ACM SIGKDD international conference on knowledge discovery and data mining. ACM Press, pp 237–244

Knobbe A, Cremilleux B, Furnkranz J, Scholz M (2008) From local patterns to global models: the LeGo approach to data mining. In: Furnkranz J, Knobbe A (eds) LeGo'08, proceedings ofthe ECML PKDD 2008 workshop 'From Local Pat-terns to Global Models', pp 1–16

Lew A (1978) Optimal conversion of extended-entry decision tables with general cost criteria. Commun ACM 21(4):269–279

Li W, Han J, Pei J (2001) CMAR: accurate and efficient classification based on multiple class-association rules. In: Cercone N, Lin TY, Wu X (eds) ICDM'01, proceedings of the 2001 IEEE international conference on data mining. IEEE Computer Society, pp 369–376

Liu B, Hsu W, Ma Y (1998) Integrating classification and association rule mining. In: Agrawal R, Stolorz PE, Piatetsky-Shapiro G (eds) KDD'98, proceedings ofthe 4th international conference on knowledge discovery and data mining. AAAI Press, pp 80–86

Machanavajjhala A, Kifer D, Gehrke J, Venkitasubramaniam M (2007) *l*-diversity: privacy beyond *k*-anonymity. ACM Trans Knowl Discov Data 1(1):3

Mannila H, Toivonen H, Verkamo AI (1994) Efficient algorithms for discovering association rules. In: Fayyad UM, Uthurusamy R (eds) KDD'94, proceedings ofthe AAAI workshop on knowledge discovery in databases. AAAI Press, pp 181–192

Meisel WS, Michalopoulos D (1973) A partitioning algorithm with application in pattern classification and the optimization of decision tree. IEEE Trans Comput C-22:93–103

Moore A, Lee MS (1998) Cached sufficient statistics for efficient machine learning with large datasets. J Artif Intell Res 8:67–91

Murphy PM, Pazzani MJ (1997) Exploring the decision forest: an empirical in-vestigation of Occam's razor in decision tree induction. In: Greiner R, Petsche T, Jose S (eds) Computational learning theory and natural learning systems vol IV: making learning systems practical. MIT Press, pp 171–187

Nadeau C, Bengio Y (2003) Inference for the generalization error. Machine Learn 52(3):239–281

Pasquier N, Bastide Y, Taouil R, Lakhal L (1999) Efficient mining of association rules using closed itemset lattices. Inf Syst 24(1):25–46

Payne HJ, Meisel WS (1977) An algorithm for constructing optimal binary decision trees. IEEE Trans Comput 26(9):905–916

Pei J, Han J, Lakshmanan LVS (2001) Mining frequent item sets with convertible constraints. In: ICDE'01, proceedings of the IEEE international conference on data engineering. IEEE Computer Society, pp 433–142

Quinlan JR (1993) C4.5: programs for machine learning. Morgan Kaufmann, San Francisco

Samarati P (2001) Protecting respondents' identities in microdata release. IEEE Trans Knowl Data Eng 13(6):1010–1027

Schumacher H, Sevcik KC (1976) The synthetic approach to decision table conversion. Commun ACM 19(6):343–351

Sweeney L (2002) *k*-anonymity: a model for protecting privacy. Int J Uncertainty, Fuzziness Knowledge-Based Syst 10(5):557–570

Turney P (1995) Cost-sensitive classification: empirical evaluation of a hybrid genetic decision tree induction algorithm. J Artif Intell Res 2:369–409

Uno T, Kiyomi M, Arimura H (2004) LCM ver. 2: efficient mining algorithms for frequent/closed/maximal itemsets. In: Bayardo R, Goethals B, Zaki MJ (eds) FIMI '04, proceedings of the IEEE ICDM workshop on frequent itemset mining implementations. CEUR-WS.org

Witten IH, Frank E (2005) Data mining: practical machine learning tools and techniques, 2nd edn. Morgan Kaufmann, San Francisco

Yan X, Cheng H, Han J, Xin D (2005) Summarizing itemset patterns: a profile-based approach. In: Grossman R, Bayardo RJ, Bennett KP (eds) KDD'05, proceedings of the 11th ACM SIGKDD international conference on knowledge discovery and data mining. ACM Press, pp 314–323

Zaki MJ, Parthasarathy S, Ogihara M, Li W (1997a) New algorithms for fast discovery of association rules. In: Heckerman D, Mannila H, Pregibon D (eds) KDD'97, proceedings of the 3rd international conference on knowledge discovery and data mining. AAAI Press, pp 283–286

Zaki MJ, Parthasarathy S, Ogihara M, Li W (1997b) Parallel algorithms for discovery of association rules. Data Mining Knowl Discov 1(4):343–373