

# Mining Optimal Decision Trees from Itemset Lattices

*Siegfried Nijssen*  
*Elisa Fromont*

*Report CW 476, March 2007*

Katholieke Universiteit Leuven  
Department of Computer Science  
Celestijnenlaan 200A – B-3001 Heverlee (Belgium)

# Mining Optimal Decision Trees from Itemset Lattices

*Siegfried Nijssen*

*Elisa Fromont*

*Report CW 476, March 2007*

Department of Computer Science, K.U.Leuven

We present an exact algorithm for finding a decision tree that optimizes a ranking function under size, depth, accuracy and leaf constraints. Because the discovery of optimal trees has high theoretical complexity, until now no efforts have been made to compute such trees for real-world datasets. An exact algorithm is of both scientific and practical interest. From the scientific point of view, it can be used as a gold standard to evaluate the performance of heuristic decision tree learners, and it can be used to gain new insight in traditional decision tree learners. From the application point of view, it can be used to discover trees that cannot be found by heuristic decision tree learners. The key idea behind our algorithm is the relation between constraints on decision trees and constraints on itemsets. We propose to exploit lattices of itemsets, from which we can extract optimal decision trees in linear time. We give several strategies to efficiently build these lattices and show that the test set accuracies of C4.5 compete with the test set accuracies of optimal trees.

## **Abstract**

**Keywords :** Decision tree learning, Frequent itemset mining, Itemset lattice, Constraint based mining

## 1. INTRODUCTION

Decision trees are among the most popular prediction models in machine learning and data mining, because there are efficient, relatively easily understandable learning algorithms and the models are easy to interpret. From this perspective, it is surprising that mining decision trees under constraints has not been given much attention. For the problems listed below, currently no broadly applicable algorithm exists even though steps in this direction were made in [7] for the last problem:

- given a dataset  $D$ , find the most accurate tree on training data in which each leaf covers at least  $n$  examples;
- given a dataset  $D$ , find the  $k$  most accurate trees on training data in which the majority class in each leaf covers at least  $n$  examples more than any of the minority classes;
- given a dataset  $D$ , find the most accurate tree on training data in which each leaf has a high statistical correlation with the target class according to a  $\chi^2$  test;
- given a dataset  $D$ , find the smallest decision tree in which each leaf contains at least  $n$  examples, and the expected accuracy is maximized on unseen examples;
- given a dataset  $D$ , find the shallowest decision tree which has an accuracy higher than  $minacc$ ;
- given a dataset  $D$ , find the smallest decision tree which has an accuracy higher than  $minacc$ .

Clearly, in the interactive process that knowledge discovery in databases is, the ability to pose *queries* that answer these questions can be very valuable. In this paper we present an algorithm for answering these queries exactly.

Most known algorithms for building decision trees, for instance C4.5, rely on top-down induction and choose splits heuristically. These heuristics impose an additional, implicit constraint on the decision trees that can be learned. If traditional heuristic algorithms do not find a tree that satisfies a set of explicit constraints (for instance, on size or accuracy), this does not mean that such a tree does not exist—it only means that the chosen heuristic imposes an additional constraint that does not allow the algorithm to find it. An exact algorithm could be desirable to answer queries without uncertainty about the influence of heuristics, and could allow us to study the influence of constraints in a much more fundamental way than will ever be possible using heuristic decision tree learners. For instance, we could determine for a sufficiently large number of datasets and constraints what the optimum under given constraints is—with and without heuristics—and learn from this in which cases heuristics have a positive effect or a negative effect.

Until now, no attempt has been made to compute exact optimal trees for many datasets; most people have not seriously considered the problem as it is known to be NP complete [17], and therefore, an efficient algorithm can most likely not exist. The data mining community, however, has an interesting track record of dealing with exponential problems with reasonable computation times. In particular, for many years, the problem of frequent itemset mining, which is exponential in its nature, has attracted a lot of research [1, 28, 10] and although this problem is not always efficiently

solvable in theory, in practice many algorithms have been applied successfully.

In this paper, we propose DL8, an exact algorithm for building decision trees that does not rely on the traditional approach of heuristic top-down induction, and addresses the problem of finding exact optimal decision trees under constraints. Its key feature is that it exploits a relation between constraints on itemsets and decision trees. Even though our algorithm is not expected to work on all possible datasets, we will provide evidence that for a reasonable number of datasets, our approach is feasible and therefore a useful addition to the data mining toolbox.

The paper is organized as follows. In Section 2, we introduce the concepts of decision trees and itemsets. In Section 3, we describe precisely which optimal trees we consider. In Section 4, we motivate the use of such optimal trees. In Sections 5 and 6, we present our algorithm and its connection to frequent itemset mining. In Section 7, we evaluate the efficiency of our algorithm; we compare the accuracy and size of the trees computed by our system with the trees learned by C4.5. Section 8 gives related work. We conclude in Section 9.

## 2. ITEMSET LATTICES FOR DECISION TREE MINING

Let us first introduce some background information about *frequent itemsets* and *decision trees*.

### 2.1 Itemset Mining

Let  $\mathcal{I} = \{i_1, i_2, \dots, i_m\}$  be a set of items and let  $D = \{T_1, T_2, \dots, T_n\}$  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)  $t(I) \subseteq \{1, 2, \dots, n\}$  of an itemset  $I \subseteq \mathcal{I}$  is the set of all identifiers of 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.  $freq(I) = |t(I)|$ ; 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,  $freq(I) \geq minfreq$ ).

DEFINITION 1. A complete lattice is a partially ordered set in which all elements have both a least upper bound and a greatest lower bound.

The set  $2^{\mathcal{I}}$  of all possible subsets of  $\mathcal{I}$ , together with the inclusion operator  $\subseteq$ , constitutes a lattice. The least upper bound of two sets is computed by the  $\cap$  operator, the greatest lower bound by the  $\cup$  operator. There exists a lower bound  $\perp = \emptyset$  and a higher bound  $\top = \mathcal{I}$  for this lattice. Usually, lattices are depicted as in Figure 1, where edges denote a subset relation between sets; sets are depicted as nodes. The figure only shows parts of the lattice for the 6 items  $\mathcal{I} = \{A, \neg A, B, \neg B, C, \neg C\}$ .

In this work, we are interested in finding frequent itemsets for databases that contain examples labeled with classes  $c \in C$ . If 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. The resulting rule  $I \rightarrow c(I)$ , where  $c(I) = argmax_{c' \in C} freq_{c'}(I)$  is called a *class association rule*.

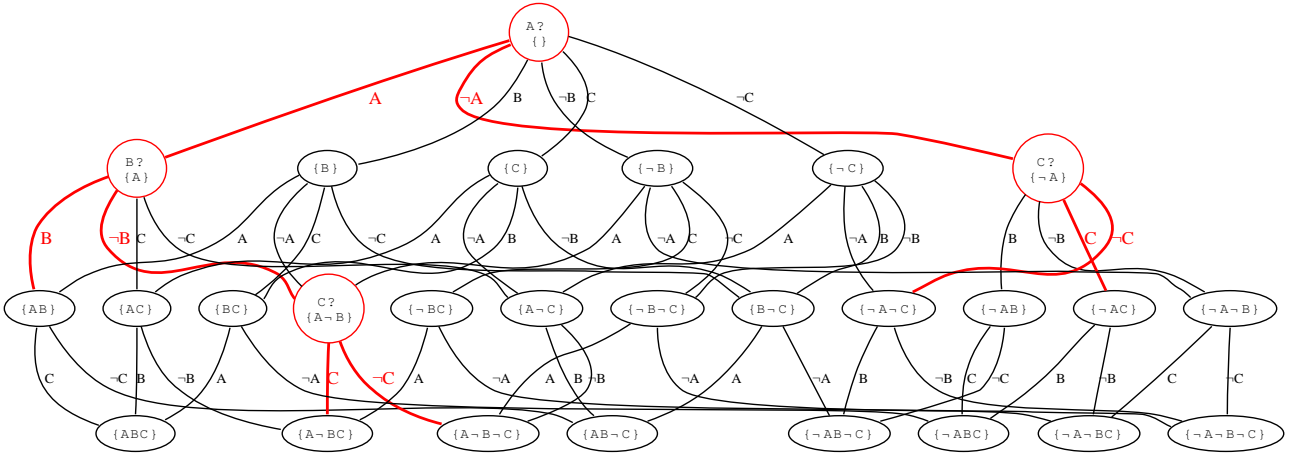


Figure 1: An itemset lattice for items  $\{A, -A, B, -B, C, -C\}$ ; binary decision tree  $A(B(C(1,1),1),C(1,1))$  is hidden in this lattice

## 2.2 Decision trees

A decision tree aims at classifying examples by sorting them down a tree. The leaves of a tree provide the classifications of examples [15]. Each node of a 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; nominal attributes are transformed into boolean attributes by mapping each possible value to a separate attribute. The input of a decision tree learner is then a binary matrix  $B$ , where  $B_{ij}$  contains the value of attribute  $i$  of example  $j$ .

Our results are based on the following observation.

**OBSERVATION 1.** *Let us transform a binary table  $B$  into transactional form  $D$  such that  $T_j = \{i | B_{ij} = 1\} \cup \{-i | B_{ij} = 0\}$ . Then the examples that are sorted down every node of a decision tree for  $B$  are characterized by an itemset of items occurring in  $D$ .*

For example, consider the decision tree in Figure 2. We can determine the leaf to which an example belongs by checking which of the itemsets  $\{B\}$ ,  $\{\neg B, C\}$  and  $\{\neg B, \neg C\}$  it includes. We denote the set of these itemsets with  $leaves(T)$ . Similarly, the itemsets that correspond to paths in the tree are denoted with  $paths(T)$ . In this case,  $paths(T) = \{\emptyset, \{B\}, \{\neg B\}, \{\neg B, C\}, \{\neg B, \neg C\}\}$ .

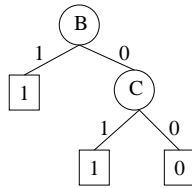


Figure 2: An example tree

The leaves of a decision tree correspond to class association rules, as leaves have associated classes. In decision tree learning, it is common to specify a minimum number of examples that should be covered by each leaf. For association rules, this would correspond to giving a support threshold.

The accuracy of a decision tree is derived from the number of misclassified examples in the leaves:  $accuracy(T) = \frac{|D| - e(T)}{|D|}$ , where

$$e(T) = \sum_{I \in leaves(T)} e(I) \quad \text{and} \quad e(I) = freq(I) - freq_{c(I)}(I).$$

A further illustration of the relation between itemsets and decision trees is given in Figure 1. In this figure, every node represents an itemset; an edge denotes a subset relation. Highlighted is one possible decision tree, which is nothing else than a set of itemsets. The branches of the decision tree correspond to subset relations.

In this paper we present DL8, an algorithm for mining Decision trees from Lattices.

## 3. QUERIES FOR DECISION TREES

The problems that we address in this paper, can be seen as *queries* to a database. These queries consist of three parts. The first part specifies the constraints on the nodes of the decision trees.

$$1. \mathcal{T} := \{T | T \in DecisionTrees, \forall I \in leaves(T), p(I)\}$$

The set  $\mathcal{T}_1$  is called the set of *locally constrained decision trees* and  $DecisionTrees$  is the set of all possible decision trees. Predicate  $p(I)$  expresses a constraint on paths. In the simplest setting,  $p(I) := (freq(I) \geq minfreq)$ . In general,  $p$  can be a formula constructed from the following atoms.

- $freq_i(I) \geq minfreq_i$ , to express a constraint on the minimum number of examples for a class;
- $freq(I) \geq minfreq$ , to express a constraint on the total number of examples in each leaf;
- $\chi^2(I) \geq mincorr$ , to express that every leaf should have a  $\chi^2$  correlation of at least  $mincorr$  with the target attribute of the prediction problem;
- $diff(I) \geq mindiff$ , where

$$diff(I) = freq_m(I) - \max_{c \neq m} freq_c(I),$$

and  $m = \text{argmax}_c \text{freq}_c(I)$ ; this constraint expresses that in every leaf there should be more examples for the majority class than for any of the minority classes.

These atoms may be used in disjunctions and conjunctions, but may not be negated.

The first two predicates are well-known in data mining, as they are *anti-monotonic*. A predicate  $p(I)$  on itemsets  $I \subseteq \mathcal{I}$  is called *anti-monotonic* iff  $p(I) \wedge I' \subseteq I \Rightarrow p(I')$ . Consequently, if  $p$  is an anti-monotonic predicate, the constraint  $(\forall I \in \text{leaves}(T), p(I))$  can equivalently be replaced with the constraint  $(\forall I \in \text{paths}(T), p(I))$ .

The second two predicates are not anti-monotonic, but *boundable*. A predicate  $p(I)$  on itemsets  $I \subseteq \mathcal{I}$  is *boundable* if there exists an anti-monotonic predicate  $q(I)$  such that  $p(I) \Rightarrow q(I)$ . Let us illustrate this for the atom  $\text{diff}(I) \geq \text{mindiff}$ . For a leaf to have  $\text{diff}(I) \geq \text{mindiff}$ , it should at least have one class  $c$  for which  $\text{freq}_c(I) \geq \text{mindiff}$ , or, in other words, a disjunction of minimum frequency constraints must be satisfied.

It is known that disjunctions and conjunctions of anti-monotonic predicates are also anti-monotonic [4]. We will denote the local constraint  $p$  in which  $\chi^2$  and  $\text{diff}$  have been replaced by their anti-monotonic bounds with  $p_b$ .

The second (optional) part of the query expresses constraints that refer to the tree as a whole.

$$2. \mathcal{T}_2 := \{T \mid T \in \mathcal{T}_1, q(T)\}$$

Set  $\mathcal{T}_2$  is called the set of *globally constrained decision trees*. Formula  $q(T)$  is a conjunction of constraints of the form  $f(T) \leq \theta$ , where  $f(T)$  can be

- $e(T)$ , to constrain the error of a tree on a training dataset;
- $ex(T)$ , to constrain the *expected error* on unseen examples, according to some predefined estimate;
- $size(T)$ , to constrain the number of nodes in a tree;
- $depth(T)$ , to constrain the length of the longest root-leaf path in a tree.

In the mandatory third step, we express a preference for a tree in the set  $\mathcal{T}_2$ .

$$3. \text{output } \text{argmin}_{T \in \mathcal{T}_2} [r_1(T), r_2(T), \dots, r_n(T)]$$

The tuples  $\mathbf{r}(T) = [r_1(T), r_2(T), \dots, r_n(T)]$  are compared lexicographically and define a *ranked set of globally constrained decision trees*;  $r_i \in \{e, ex, size, depth\}$ . Our current algorithm requires that at least  $e$  and  $size$  or  $ex$  and  $size$  be used in the ranking; If  $depth$  (respectively  $size$ ) is used in the ranking before  $e$  or  $ex$ , then  $q$  must contain an atom  $depth(T) \leq \text{maxdepth}$  (respectively  $size(T) \leq \text{maxsize}$ ).

We do not constrain the order of  $size(T)$ ,  $e(T)$  and  $depth(T)$  in  $\mathbf{r}$ . We are minimizing the ranking function  $\mathbf{r}(T)$ , thus, our algorithm is an optimization algorithm. The trees that we search for are *optimal* in terms of the problem setting that is defined in the query.

To illustrate our querying mechanism we will now give several examples.

QUERY 1. *Small Accurate Trees with Frequent leaves.*

$$\mathcal{T} := \{T \mid T \in \text{DecisionTrees}, \\ \forall I \in \text{paths}(T), \text{freq}(I) \geq \text{minfreq}\} \\ \text{output } \text{argmin}_{T \in \mathcal{T}} [e(T), size(T)].$$

In other words, we have  $p(T) := (\text{freq}(I) \geq \text{minfreq})$ ,  $q(T) := \text{true}$  and  $r(T) := [e(T), size(T)]$ . This query investigates all decision trees in which each leaf covers at least *minfreq* examples of the training data. Among these trees, we find the smallest most accurate one.

In some cases, one is not interested in large trees, even if they are more accurate.

QUERY 2. *Accurate Trees of Bounded Size.*

$$\mathcal{T}_1 := \{T \mid T \in \text{DecisionTrees}, \\ \forall I \in \text{paths}(T), \text{freq}(I) \geq \text{minfreq}\} \\ \mathcal{T}_2 := \{T \mid T \in \mathcal{T}_1, size(T) \leq \text{maxsize}\} \\ \text{output } \text{argmin}_{T \in \mathcal{T}_2} [e(T), size(T)].$$

One possible scenario in which DL8 can be used, is the following. Assume that we have already applied a heuristic decision tree learner, such as C4.5, and we have some idea about decision tree error (*maxerror*) and size (*maxsize*). Then we can run the following query:

QUERY 3. *Accurate Trees of Bounded Size and Accuracy.*

$$\mathcal{T}_1 := \{T \mid T \in \text{DecisionTrees}, \\ \forall I \in \text{paths}(T), \text{freq}(I) \geq \text{minfreq}\} \\ \mathcal{T}_2 := \{T \mid T \in \mathcal{T}_1, size(T) \leq \text{maxsize}, \\ e(T) \leq \text{maxerror}\} \\ \text{output } \text{argmin}_{T \in \mathcal{T}_2} [size(T), e(T)].$$

This query finds the smallest tree that achieves at least the same accuracy as the tree learned by C4.5.

The previous queries aim at finding compact models that maximize training set accuracy. Such trees might however overfit training data. Another application of DL8 is to obtain trees with high *expected accuracy*. Several algorithms for estimating test set accuracy have been presented in the literature. One such estimate is at the basis of the *reduced error pruning* algorithm of C4.5. Essentially, C4.5 computes an additional penalty term  $x(\text{freq}_1(I), \dots, \text{freq}_n(I))$  for each leaf  $I$  of the decision tree, from which we can derive a new estimated number of errors

$$ex(T) = \sum_{I \in \text{leaves}(T)} e(I) + x(\text{freq}_1(I), \dots, \text{freq}_n(I)).$$

We can now also be interested in answering the following query.

QUERY 4. *Small Accurate Pruned Trees.*

$$\mathcal{T} := \{T \mid T \in \text{DecisionTrees}, \\ \forall I \in \text{paths}(T), \text{freq}(I) \geq \text{minfreq}\} \\ \text{output } \text{argmin}_{T \in \mathcal{T}} [ex(T), size(T)].$$

This query would find the most accurate tree after pruning such as done by C4.5. Effectively, the penalty terms make sure that trees with less leaves are sometimes preferable even if they are less accurate.

Another query could be :

QUERY 5. *Small Accurate Trees with Correlated leaves.*

$$\mathcal{T} := \{T \mid T \in \text{DecisionTrees}, \\ \forall I \in \text{leaves}(T), \chi^2(I) \geq 10\} \\ \text{output } \text{argmin}_{T \in \mathcal{T}} [ex(T), size(T)].$$

In this query, we restrict ourselves to trees that perform well in terms of C4.5's pruning measure. Each leaf in the tree should have a significant correlation with the target class.

## 4. MOTIVATING EXAMPLES

To motivate our work, we will first provide some examples of situations in which traditional heuristic decision tree learners are not able to find trees that are both small and accurate, but which can be solved by an optimal decision tree learner.

A	B	Class	Repeated
1	1	0	5×
0	1	0	5×
1	0	1	40×
0	0	2	50×

Figure 3: Example database 1

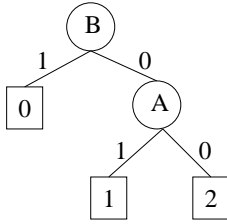


Figure 4: The most accurate tree for example database 1

Given the database in Figure 3, in which we have 3 target classes, and instances are repeated several numbers of times, we are interested in finding the most accurate tree in which every leaf has at least 10 examples. A tree that predicts all examples accurately exists and is given in Figure 4. However, if information gain [23, 6] is used to find a split, we will not be able to find this tree: the information gain of  $A$  is 0.89, while the information gain of  $B$  is 0.47; a test on attribute  $A$  would be preferred in the root of the tree. Only by using a different heuristic—information gain ratio—we are able to find the optimal tree, as  $A$ 's information gain ratio is 0.90, but  $B$ 's ratio is 1.0.

If we would use information gain without size constraint, we could find an accurate tree, but this tree would be larger than necessary.

We can learn from this example that information gain concentrates on the classes that have most examples, as splits for these classes will yield the largest entropy reductions. If smaller classes could easily be separated, this may remain unnoticed. Contrary to traditional heuristic methods, an optimal decision tree learner would not need a different heuristic to find a tree that satisfies the constraints.

A	B	C	Class	Repeated
1	1	0	1	40×
1	1	1	1	40×
1	0	1	1	5×
0	0	0	0	10×
0	0	1	1	5×

Figure 5: Example database 2

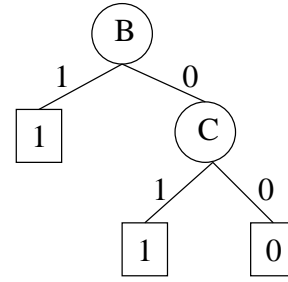


Figure 6: The most accurate tree for example database 2

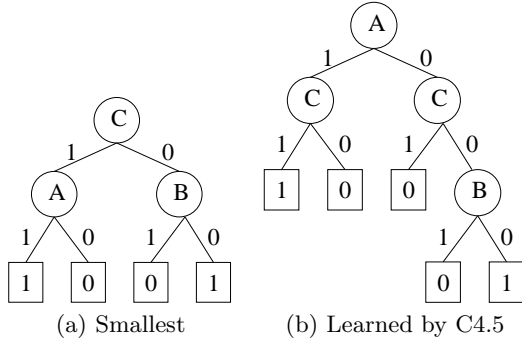
A	B	C	Class	Repeated
1	1	1	1	30×
1	1	0	0	20×
0	1	0	0	8×
0	1	1	0	12×
0	0	0	1	12×
0	0	1	0	18×

Figure 7: Example database 3

As a second example consider the database in Figure 5, in which we have 2 target classes. Assume that we have reasons to believe that only a difference of 10 examples between a majority and a minority class provides significant evidence to prefer one class above the other in a leaf; therefore, we are interested in finding an optimal decision tree in which there is a difference of at least 10 examples between majority and minority classes. A tree satisfying these constraints exists, and is given in Figure 6. Clearly in this example  $A$  and  $B$  have approximately the same predictive power. However, C4.5 will prefer attribute  $A$  in the root, as it has information gain 0.33 (resp. ratio 0.54), while  $B$  only has information gain 0.26 (resp. ratio 0.37). C4.5, however, by making this choice, takes away examples that could have been used to find a significant next split; no further splits can be performed. On the other hand, an optimal decision tree learner can prefer initially sub-optimal splits, if this is useful to build statistical support for tests deeper down the tree.

10	0	0	0	0	1	1	1	1	1	1	
9	0	0	0	0	1	1	1	1	1	1	
8	0	0	0	0	1	1	1	1	1	1	
7	0	0	0	0	1	1	1	1	1	1	
6	0	0	0	0	1	1	1	1	1	1	A
5	0	0	0	0	0	0	0	0	0	0	
4	0	0	0	0	0	0	0	0	0	0	B
3	1	1	1	1	0	0	0	0	0	0	
2	1	1	1	1	0	0	0	0	0	0	
1	1	1	1	1	0	0	0	0	0	0	
	1	2	3	4	5	6	7	8	9	10	
					C						

Figure 8: Visualization of example database 3; every element of the matrix corresponds to one example, and contains its class label;  $A$  is true if  $y \geq 6$ ,  $B$  is true if  $y \geq 4$ , and  $C$  is true if  $x \geq 5$



**Figure 9: Two accurate trees for example database 3**

As a third example consider the database in Figure 7, which is a variation of the XOR problem, as depicted in Figure 8. Assume that we are interested in finding the smallest (or shallowest) most accurate decision tree, and we do not have a constraint on the number of examples in a leaf. Then the optimal tree is given in Figure 9(a), but C4.5 will find the tree in Figure 9(b), as the information gain (cq. ratio) of  $A$  is 0.098 (resp. 0.098), while the information gain of  $C$  is 0.029 (resp. 0.030). What these examples illustrate, is that heuristic decision tree learners can be ‘fooled’ into taking certain decisions due to the proportions with which examples are given. Optimal decision tree learners may expose such behavior less quickly.

Of course, these examples are all artificial. An interesting question is to what extent similar situations occur in real world data. By using our algorithm, we are able to find this out.

## 5. BUILDING OPTIMAL DECISION TREES FROM LATTICES

We will now present the DL8 algorithm for answering decision tree queries. Pseudo-code of the algorithm is given in Algorithm 1.

Parameters of DL8 are the local constraint  $p$  and its anti-monotonic bound  $p_b$ , the ranking function  $\mathbf{r}$ , and the global constraints; each global constraint is passed in a separate parameter; global constraints that are not specified, are assumed to be set to  $\infty$ . The most important part of DL8 is its recursive search procedure. Given an input itemset  $I$ , DL8-RECURSIVE computes one or more decision trees for the transactions  $t(I)$  that contain the itemset  $I$ . More than one decision tree is returned only if a depth or size constraint is specified. Let  $\mathbf{r}(T) = [r_1(T), \dots, r_n(T)]$  be the ranking function, and let  $k$  be the index of the obligatory error function in this ranking. If  $r_1, \dots, r_{k-1} \in \{\text{depth}, \text{size}\}$  then, for every allowed value of depth  $d$  and size  $s$ , DL8-RECURSIVE outputs the best tree  $T$  that can be constructed for the transactions  $t(I)$  according to the ranking  $[r_k(T), \dots, r_n(T)]$ , such that  $\text{size}(T) \leq s$  and  $\text{depth}(T) \leq d$ .

In DL8-RECURSIVE, we use several functions:  $l(c)$ , which returns a tree consisting of a single leaf with class label  $c$ ;  $n(i, T_1, T_2)$ , which returns a tree that contains test  $i$  in the root, and has  $T_1$  and  $T_2$  as left-hand and right-hand branches;  $e_t(T)$ , which computes the error of tree  $T$  when only the transactions in TID-set  $t$  are considered; and fi-

---

### Algorithm 1 DL8( $p, p_b, \text{maxsize}, \text{maxdepth}, \text{maxerror}, \mathbf{r}$ )

---

```

1: if  $\text{maxsize} \neq \infty$  then
2:    $S \leftarrow \{1, 2, \dots, \text{maxsize}\}$ 
3: else
4:    $S \leftarrow \{\infty\}$ 
5: if  $\text{maxdepth} \neq \infty$  then
6:    $D \leftarrow \{1, 2, \dots, \text{maxdepth}\}$ 
7: else
8:    $D \leftarrow \{\infty\}$ 
9:  $\mathcal{T} \leftarrow \text{DL8-RECURSIVE}(\emptyset)$ 
10: if  $\text{maxerror} \neq \infty$  then
11:    $\mathcal{T} \leftarrow \{T \mid T \in \mathcal{T}, e(T) \leq \text{maxerror}\}$ 
12: if  $\mathcal{T} = \emptyset$  then
13:   return undefined
14: return  $\text{argmin}_{T \in \mathcal{T}} \mathbf{r}(T)$ 
15:
16: procedure DL8-RECURSIVE( $I$ )
17:   if DL8-RECURSIVE( $I$ ) was computed before then
18:     return stored result
19:   if  $p(I) = \text{true}$  then
20:      $\mathcal{C} \leftarrow \{l(c(I))\}$ 
21:   else
22:      $\mathcal{C} \leftarrow \emptyset$ 
23:   if  $\text{pure}(I)$  then
24:     store  $\mathcal{C}$  as the result for  $I$  and return  $\mathcal{C}$ 
25:   for all  $i \in \mathcal{I}$  do
26:     if  $p_b(I \cup \{i\}) = \text{true}$  and  $p_b(I \cup \{\neg i\}) = \text{true}$  then
27:        $T_1 \leftarrow \text{DL8-RECURSIVE}(I \cup \{i\})$ 
28:        $T_2 \leftarrow \text{DL8-RECURSIVE}(I \cup \{\neg i\})$ 
29:       for all  $T_1 \in \mathcal{T}_1, T_2 \in \mathcal{T}_2$  do
30:          $\mathcal{C} \leftarrow \mathcal{C} \cup \{n(i, T_1, T_2)\}$ 
31:     end if
32:    $\mathcal{T} \leftarrow \emptyset$ 
33:   for all  $d \in D, s \in S$  do
34:      $\mathcal{L} \leftarrow \{T \in \mathcal{C} \mid \text{depth}(T) \leq d \wedge \text{size}(T) \leq s\}$ 
35:      $\mathcal{T} \leftarrow \mathcal{T} \cup \{\text{argmin}_{T \in \mathcal{L}} [r_k = e_{t(I)}(T), \dots, r_n(T)]\}$ 
36:   end for
37:   store  $\mathcal{T}$  as the result for  $I$  and return  $\mathcal{T}$ 
38: end procedure

```

---

nally, we use a predicate  $\text{pure}(I)$ ; predicate  $\text{pure}$  blocks the recursion if all examples  $t(I)$  belong to the same class.

The algorithm is most easily understood if  $\text{maxdepth} = \infty$ ,  $\text{maxsize} = \infty$ ,  $\text{maxerror} = \infty$  and  $\mathbf{r}(T) = [e(T)]$ ; in this case, DL8-RECURSIVE combines only two trees for each  $i \in \mathcal{I}$ , and returns the single most accurate tree in line 37.

The correctness of the DL8 algorithm is essentially based on the fact that the left-hand branch and the right-hand branch of a node in a decision tree can be optimized independently. In more detail, the correctness follows from the following observations.

(line 1-8) the valid ranges of sizes and depths are computed here if a size or depth constraint was specified;

(line 11) for each depth and size, for so far necessary, DL8-RECURSIVE finds the most accurate tree possible. Some of the accuracies might be too low for the given constraint, and are removed from consideration.

(line 19) we check the leaf constraint here; only if the leaf constraint is satisfied, a candidate decision tree for

classifying the examples  $t(I)$  consists of a single leaf; this candidate is initialized in line 20.

- (line 23) if all examples in a set of transactions belong to the same class, continuing the recursion is not necessary; after all, any larger tree will not be more accurate than a leaf, and we require that size is used in the ranking. More sophisticated pruning is possible in some special cases, for example the *loose-pure* predicate presented in Section 6.3 can be sometimes be used instead of the *pure* predicate.
- (line 26) in this line the anti-monotonic property of the predicate  $p_b(I)$  is used: an itemset that does not satisfy the predicate  $p_b(I)$  cannot be part of a tree, nor can any of its supersets; therefore the search is not continued if  $p_b(I \cup \{i\}) = \text{false}$  or  $p_b(I \cup \{\neg i\}) = \text{false}$ .
- (line 25–36) these lines make sure that each tree that should be part of the output  $\mathcal{T}$ , is indeed returned. We can prove this by induction. Assume that for the set of transactions  $t(I)$ , tree  $T$  should be part of  $\mathcal{T}$  as it is the most accurate tree that is smaller than  $s$  and shallower than  $d$  for some  $s \in S$  and  $d \in D$ ; assume  $T$  is not a leaf, and contains test  $i$  in the root. Then  $T$  must have a left-hand branch  $T_1$  and a right-hand branch  $T_2$ . Tree  $T_1$  must be the most accurate tree that can be constructed for  $t(I \cup \{i\})$  with size at most  $\text{size}(T_1)$  and depth at most  $\text{depth}(T_1)$ ; similarly,  $T_2$  must be the most accurate tree that can be constructed for  $t(I \cup \{\neg i\})$  under depth and size constraints. We can inductively assume that trees with these constraints are found by DL8-RECURSIVE( $I \cup \{i\}$ ) and DL8-RECURSIVE( $I \cup \{\neg i\}$ ) as  $\text{size}(T_1), \text{size}(T_2) \leq \text{maxsize}$  and  $\text{depth}(T_1), \text{depth}(T_2) \leq \text{maxdepth}$ . Consequently  $T$  (or a tree with equal statistics) must be among the trees found by combining results from the two recursive procedure calls in line 30.

A key feature of DL8-RECURSIVE is that in line 37 it stores every result that it computes. Consequently, DL8 avoids that optimal decision trees for any itemset are computed more than once. We do not need to store entire decision trees with every itemset. It is sufficient to store their roots and statistics (error, possibly size and depth), as left-hand and right-hand subtrees can be recovered from the stored results for the left-hand and right-hand itemsets if necessary.

Note that in our algorithm, we output the best tree according to the ranking. The  $k$ -best trees could also be straightforwardly output.

To efficiently index the itemsets  $I$  that are considered by DL8, a trie data structure can be used. In the next section we will consider several strategies for building this data structure.

## 6. MINING STRATEGIES

As with most data mining algorithms, the most time consuming operations are those that access the data. In the following, we will provide four related strategies to obtain the frequency counts that are necessary to check the constraints and compute accuracies: the simple single-step approach, the frequent itemset mining (FIM) approach, the constraint frequent itemset mining approach, and the closure based single-step mining approach.

### 6.1 The Simple Single-Step Approach

The most straightforward approach, referred to as DL8-SIMPLE, computes the itemset frequencies while DL8 is executing. In this case, once DL8-RECURSIVE is called for an itemset  $I$ , we obtain the frequencies of  $I$  in a scan over the data, and store the result in a trie to avoid later recomputations.

### 6.2 The FIM Approach

An alternative approach is based on the observation that every itemset that occurs in a tree, must satisfy the local constraint  $p_b$ . If  $p_b$  is a minimum frequency constraint, we can use a frequent itemset miner to obtain the frequencies in a preprocessing step. DL8 then operates on the resulting set of itemsets, annotating every itemset with optimal decision trees.

Many frequent itemset miners have been studied in the literature; all of these can be used with small modifications to output the frequent itemsets in a convenient form and determine frequencies in multiple classes [1, 28, 10, 25].

We implemented an extension of APRIORI that first computes and stores in a trie all frequent itemsets, and then runs DL8 on the trie. This approach is abbreviated with APRIORI-FREQ+DL8. Compared to other itemset miners, like ECLAT or LCM, we expect that the additional runtime to store all itemsets in APRIORI is the lowest, as APRIORI already builds a trie of candidate itemsets itself.

If we assume that the output of the frequent itemset miner consists of a graph structure such as Figure 1, then DL8 operates in time linear in the number of edges of this graph.

### 6.3 The Constrained FIM Approach

Unfortunately, the frequent itemset mining approach may compute frequencies of itemsets that can never be part of a decision tree. For instance, assume that  $\{A\}$  is a frequent itemset, but  $\{\neg A\}$  is not; then no tree answering example Query 1 will contain a test for attribute  $A$ ; itemset  $\{A\}$  is redundant. In this section, we show that an additional local, anti-monotonic constraint can be used in the frequent itemset mining process to make sure that no such redundant itemsets are enumerated.

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

- the  $\neg\text{pure}(I')$  predicate is false in line (23);
- the conjunction  $p_b(I' \cup \{i_{k_{m+1}}\}) \wedge p_b(I' \cup \{\neg i_{k_{m+1}}\})$  is false in line (26).

It is helpful to negate the *pure* predicate, as one can easily see that  $\neg\text{pure}$  is an anti-monotonic predicate (every superset of a *pure* itemset, must also be *pure*). From now on, we will refer to  $\neg\text{pure}$  as a *leaf constraint*, as it defines a property that is only allowed to hold in the leaves of a tree.

We can now formalize the principle of itemset *relevancy*.

DEFINITION 2. Let  $p_1$  be a local anti-monotonic tree constraint and  $p_2$  be an anti-monotonic leaf constraint. Then

the relevancy of  $I$ , denoted by  $rel(I)$ , is defined by

$$rel(I) = \begin{cases} p_1(I) \wedge p_2(I) & \text{if } I = \emptyset & \text{(Case 1)} \\ true & \text{if } \exists i \in I \text{ s.t.} \\ & rel(I - i) \wedge p_2(I - i) \wedge \\ & p_1(I) \wedge p_1(I - i \cup \neg i) & \text{(Case 2)} \\ false & \text{otherwise} & \text{(Case 3)} \end{cases}$$

**THEOREM 1.** Let  $\mathcal{L}_1$  be the set of itemsets stored by DL8-SIMPLE, 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-SIMPLE, there must be an order of the items in which each prefix satisfies the constraints as defined in line (23) and line (26). 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.  $\square$

Relevancy is a property that can be pushed in a frequent itemset mining process.

**THEOREM 2.** Itemset relevancy 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  $X', X$  upto 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  $j$ :

- $i = j$ : certainly  $Y - i = X$  is not relevant;
- $i \neq j$ : 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);
  - $p_1(X) = false$ : in this case  $p_1(Y) = false$  (anti-monotonicity of  $p_1$ );
  - $p_1(X - j \cup \neg j) = false$ : in this case  $p_1(Y - j \cup \neg j) = p_1(X - j \cup \neg j \cup i) = false$  (anti-monotonicity of  $p_1$ );
  - $p_2(X - j) = false$ : in this case  $p_2(Y - j) = p_2(X - j \cup i) = false$  (anti-monotonicity of  $p_2$ ).

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

$\square$

It is relatively easy to integrate the computation of relevancy in 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. Assume that we have already computed all relevant itemsets

that are a subset of an itemset  $I$ . Then we can determine for each  $i \in I$  if the itemset  $I - i$  is part of this set, and if so, we can derive the class frequencies of  $I - i \cup \neg i$  using the formula  $freq_k(I - i \cup \neg i) = freq_k(I - i) - freq_k(I)$ . If for each  $i$  either  $I - i$  is not relevant, or the predicate  $p(I - i \cup \neg i)$  fails, we can prune  $I$ .

Pruning of this kind can be integrated in both depth-first and breadth-first frequent itemset miners.

### Loose-Pureness

Up until now, we have considered a pureness constraint that states that no internal node should be pure. If we are interested in finding a small accurate tree under a minimum support constraint, this definition of purity can be relaxed.

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

Let us illustrate this definition of loose-purity on an example. Given a prediction problem with 4 classes and  $minfreq = 5$ , and assume that for an itemset (i.e. a node) we have :

- Class 1: 10 examples
- Class 2: 1 examples
- Class 3: 1 examples
- Class 4: 1 examples

It does not make sense to split this node to increase the global accuracy of the tree. Indeed, the best split would separate all current errors (from class 2, 3 and 4) from the majority class, but since any leaf must contain at least 5 examples, and since none of these classes (in particular not the second most frequent class) have enough examples to take a majority in a leaf ( $5 - 3 = 2 > 1$ ) compared to the first class, the error (3 in this example) in the tree cannot decrease. As a further example, if class 2 would have 2 examples, the node could still be split: we could create a split such that 2 examples of class 2 become classified correctly, 2 examples of class 3 and 4 remain classified incorrectly, and 1 example of class 1 becomes classified incorrectly; in total, a decrease in error of 1 could be obtained in the example.

Obviously, loose-purity is monotonic, and is a stronger constraint than purity. We prove now that splitting a loose-pure node will never improve accuracy; consequently, when looking for the smallest most accurate tree, loose-purity can be used instead of purity.

**THEOREM 3.** If  $freq(I) \geq minfreq$ ,  $loose-pure(I) = true$  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  $I$ . 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 loose-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  $I'$ .  $\square$

### Integration in Frequent Itemset Miners

In case  $depth$  is the first ranking function, level-wise algorithms such as APRIORI have an important benefit: after each level of itemsets is generated, we could run DL8 to obtain the most accurate tree up to that depth. APRIORI can stop at the lowest level at which a tree is found that fulfils the constraints.

We implemented two versions of DL8 in which the relevancy constraints are pushed in the frequent itemset mining process: DL8-APRIORI, which is based on APRIORI [1], and DL8-ECLAT, which is based on ECLAT [28].

## 6.4 The Closure-Based Single-Step Approach

In the simple direct approach, we computed an optimal decision tree for every relevant itemset. However, if the local constraint is only coverage based, it is easy to see that for two itemsets  $I_1$  and  $I_2$ , if  $t(I_1) = t(I_2)$ , the result of DL8-RECURSIVE( $I_1$ ) and DL8-RECURSIVE( $I_2$ ) must be the same. To reduce the number of results that we have to store, we should avoid storing such duplicate sets of results.

The solution that we propose is to compute for every itemset its *closure*. Let  $i(t)$  be the function which computes

$$i(t) = \bigcap_{k \in t} T_k$$

for a TID-set  $t$ , then the *closure* of itemset  $I$  is the itemset  $i(t(I))$ . An itemset  $I$  is called closed iff  $I = i(t(I))$ . If  $t(I_1) = t(I_2)$  it is easy to see that also  $i(t(I_1)) = i(t(I_2))$ . Thus, in the trie data-structure that is used in the direct approach, we can index the results on  $i(t(I))$  instead of  $I$  itself.

This means that we need to modify Algorithm 1 such that it searches for the closure of  $I$  in line 17. Similarly, in line 37, we associate computed decision tree(s) to the closure of  $I$  instead of to  $I$  itself.

Our implementation of DL8-SIMPLE which relies on closed itemsets is called DL8-CLOSED. Obviously, DL8-CLOSED will never consider more itemsets than the naive direct implementation, DL8-APRIORI or DL8-ECLAT; itemsets stored by DL8-CLOSED may however be longer as they contain all items in their closure.

### Relevancy and DL8-CLOSED

We can characterize the closed itemsets that are considered by DL8-CLOSED as follows.

**THEOREM 4.** *After running DL8-CLOSED the set of closed itemsets  $\mathcal{C}$  contains exactly all itemsets  $C \subseteq \mathcal{I}$  such that  $C$  is closed, and there is at least one  $I \subseteq C$  with  $t(I) = t(C)$  and  $rel(I) = true$ .*

**PROOF.** This follows from the fact that DL8-SIMPLE considers only relevant itemsets.  $\square$

It follows from this theorem that DL8-CLOSED will never compute a larger set of itemsets than DL8-SIMPLE. By combining closed itemset mining with relevancy, DL8-CLOSED searches for  $\delta$ -free-like itemsets within the concept lattice instead of the original lattice [3].

## Implementation Details

As we perform most experiments with DL8-CLOSED, and DL8-CLOSED is not based on a previously published frequent itemset mining algorithm, we provide some more details about its implementation here.

The most time consuming operations in the algorithm are the computation of closures of itemsets, and the computation of frequencies. To speed up these operations, we combine these operations as follows. 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  $i$ ;
- a set of *active transaction identifiers* storing  $t(I - \{i\})$ ;
- the set of all items  $C$  that are in the closure of  $I$ , 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 17 of DL8-RECURSIVE is executed) we scan each active transaction, and test if it contains the 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  $t(I)$ . Those active items of which the frequency equals that of  $I$ , are added to the closure  $C$ . After this scan of the data, we build a new set of active items. For every item we determine if  $p_b(I \cup \{i\})$  and  $p_b(I \cup \{-i\})$  are true; if so, and if not  $i \notin C$ , we add the item to the new set of active items. In line 25 we only traverse the set of active items; the test of line 26 is then redundant. Finally, in line 27 and 28 the updated sets of active transactions and active items are passed to the recursive calls.

This mechanism for maintaining sets of active transactions is akin to the idea of maintaining projected databases that is implemented in ECLAT and FP-GROWTH. 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  $|t(I)| = |t(I \cup i)| + |t(I \cup \neg i)|$ , it is less beneficial to maintain TID-sets as in ECLAT, and we preferred this solution.

The main reason for calling DL8-RECURSIVE with the set of active transactions  $t(I - \{i\})$  instead of  $t(I)$ , is that we can optimize the memory use of the algorithm. Instead of repeatedly allocating new memory for storing sets of active items and transactions, we can now maintain a single array to store these sets across all recursive calls. A projection is obtained by reordering the items in this array. Consequently, the memory use of our algorithm is entirely determined by the amount of memory that is needed to store the database and the closed itemsets; the memory use is  $\theta(|D| + |\mathcal{C}|)$ .

We store the closed itemsets in a trie data-structure, as is common in many other frequent itemset mining algorithms. Which information is stored for every itemsets, depends on the query. If we are looking only for the smallest, most accurate decision tree, we associated to fields for storing *error*, *size* and *root* (representing the test in the root of the tree for  $t(I)$ ); in our implementation, we assume that each of these elements can be stored in 32 bits. Thus the amount of memory used for each closed itemset is independent of the characteristics of the database.

Datasets	#Ex	#Test	Datasets	#Ex	#Test
anneal	812	36	tumor	336	18
a-credit	653	56	segment	2310	55
balance	625	13	soybean	630	45
breast	683	28	splice	3190	3466
chess	3196	41	thyroid	3247	36
diabetes	768	25	vehicle	846	55
g-credit	1000	77	vote	435	49
heart	296	35	vowel	990	48
ionosphere	351	99	yeast	1484	23
mushroom	8124	116	zoo	101	15
pendigits	7494	49			

Figure 10: Datasets description

Algorithm	Uses relevancy	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	

Figure 12: Properties of the algorithms used in the experiments

## 7. EXPERIMENTS

In this section we compare the different versions of DL8 in terms of efficiency; furthermore, we compare the quality of the constructed trees with those found by the J48 decision tree learner implemented in WEKA [26]. All experiments were performed on Intel Pentium 4 machines with in between 1GB and 2GB of main memory, running Linux. DL8 and the frequent itemset miners were implemented in C++.

The experiments were performed on UCI datasets [19]. Numerical data were discretized before applying the learning algorithms using WEKA’s unsupervised discretization method with a number of bins equal to 4. We limited the number of bins in order to limit the number of created attributes. Figure 10 gives a brief description of the datasets that we used in terms of the number of examples and the number of attributes after binarization.

### 7.1 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. We first evaluate experimentally how these factors are influenced by constraints and properties of the data. Furthermore, we determine how the different approaches for computing the itemset lattices compare. A summary of the algorithms can be found in Figure 12. Besides DL8-APRIORI, DL8-ECLAT and DL8-CLOSED, we also include unmodified implementations of the frequent itemset miners APRIORI [1], ECLAT [28] and LCM [25] in the comparison. These implementations were obtained from the FIMI website [2]. The inclusion of unmodified algorithms allows us to determine how well relevancy pruning works, and allows us to determine the trade-off between relevancy pruning and trie construction.

Results for twelve datasets are listed in Figure 11. We aborted runs of algorithms that lasted for longer than 1800s. The results clearly show that in all cases the number of

closed relevant itemsets is the smallest. The difference between the number of relevant itemsets and the number of frequent itemsets becomes smaller for lower minimum frequency values. 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. Bare in mind, however, that LCM does not output the itemsets in a form that can be used efficiently by DL8.

In most cases, DL8-CLOSED is faster than DL8-APRIORI or DL8-ECLAT, with the exception of the Mushroom, Chess and Splice datasets, which are the largest datasets used in the experiments. The experiments reveal that for the high minimum support values that we had to use, the differences between closed relevant itemsets and relevant itemsets are small; the overhead of DL8-CLOSED seems too large.

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 relevancy pruning, the resulting algorithm is usually faster than the original itemset miner.

In the datasets shown here, the number of attributes is relatively small. For the datasets with larger number of attributes, 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%.

### 7.2 Accuracy

In this section, we compare the accuracies of the decision trees learned by DL8 and J48 on the twenty different UCI datasets. J48 is the Java implementation of  $C4.5$  [23] in WEKA. We used a stratified 10-fold cross-validation to compute the training and test accuracies of both systems. The bottleneck of our algorithm is the in-memory construction of the lattice, and, consequently, the application of our algorithm is limited by the amount of memory available for the construction of this lattice. For the results of Figure 13, we used minimum frequency as local constraint. The frequency was lowered to the lowest value that still allowed the computation to be performed within the memory of our computers. We also give results for higher frequency values to evaluate the influence of frequency on the accuracy of the trees.

For J48, results are provided for pruned trees and unpruned trees; for DL8 results are provided in which the  $e$  (unpruned) and  $ex$  (pruned) error functions are optimized (cf. Queries 1 and 4 of Section 3). Both algorithms were applied with the same minimum frequency setting. We used a corrected two-tailed t-test [18] with a significance threshold of 5% to compare the test accuracies of both systems. A test set accuracy result is in bold when it is significantly better than its counterpart result on the other system. In the last three columns, we also give results for J48 with its default  $minfreq = 2$  setting. The test accuracies of J48 are compared to the test accuracies given by DL8 using pruning. The results of the significance test are given in the “S” column: “+” means that J48 is significantly better, “-” that it is significantly worse and “0” not significantly different.

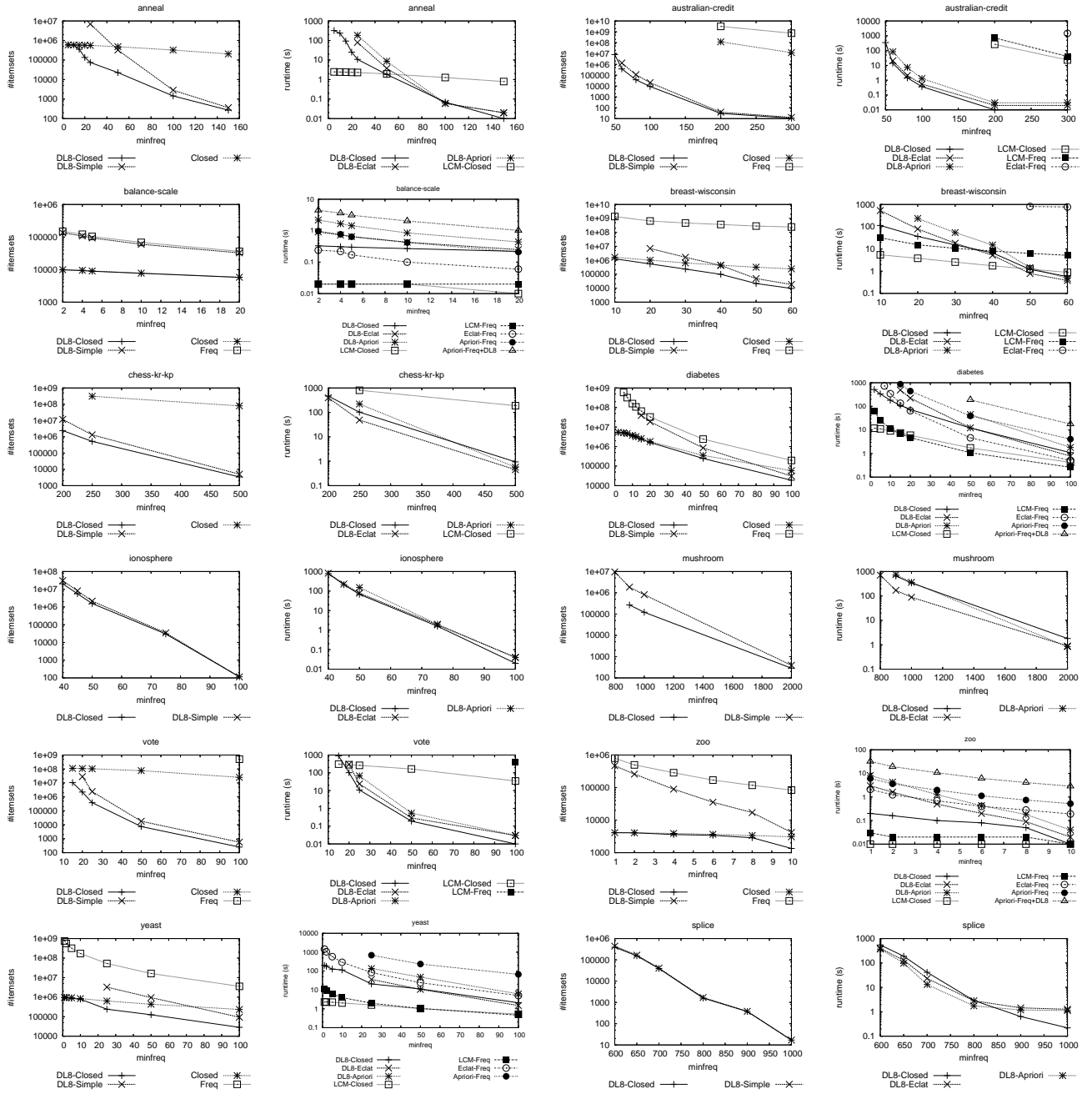


Figure 11: Comparison of the different miners on 12 UCI datasets

The experiments show that both with and without pruning the optimal trees computed by DL8 have a better training accuracy than the trees computed by J48 with the same frequency values. Furthermore, on the test data, in both cases DL8 is significantly better than J48 on 9 of the 20 datasets; only on two datasets is one result significantly worse. The experiments also show that when pruned trees are compared to unpruned ones, the sizes of the trees are on average 1.75 times smaller for J48 and 1.5 times smaller for DL8. After pruning, DL8's trees are still 1.5 times larger than J48's ones. In one case (*vehicle*), the test accuracy result of DL8 is significantly better than the one given by J48

for a smaller tree. The trees computed by the pruned version of J48 and DL8 on the *vehicle* dataset for  $minfreq = 80$  are given in Figure 14. In the other cases where DL8's accuracy is significantly better, the pruned trees of DL8 are 3 to 9 nodes larger than those of J48. This confirms earlier findings which show that smaller trees are not always desirable [22].

If we decrease the frequency threshold down to a certain value, the training accuracy increases, but the experiments in which we were able to reach a  $minfreq$  of 2 (yeast, p-tumor, balance and diabetes), indicate that for testing accuracy, low thresholds are not always the best option.

Datasets	Freq		Train acc		Unpruned Test acc		Size		Train acc		Pruned Test acc		Size		Pruned Freq = 2		
	#	%	J48	DL8	J48	DL8	J48	DL8	J48	DL8	J48	DL8	J48	DL8	Test acc J48	S	size J48
anneal	50	6.1	0.78	0.78	0.77	0.75	4.0	12.0	0.78	0.78	0.77	0.78	3.4	4.2	0.82	+	44.4
anneal	15	1.8	0.83	0.85	0.79	0.81	31.8	39.4	0.82	0.84	0.80	0.82	13.6	25.4	"	0	"
anneal	10	1.2	0.85	0.87	0.82	0.81	43.2	56.6	0.83	0.85	0.81	0.82	22.2	31.4	"	0	"
anneal	5	0.6	0.86	0.89	0.81	0.83	61.6	85.6	0.85	0.86	0.81	0.82	31.4	38.0	"	0	"
anneal	2	0.2	0.89	0.89	0.82	0.82	106.6	87.8	0.86	0.87	0.82	0.82	44.4	45.6	"	0	"
a-credit	50	7.6	0.87	0.88	0.85	<b>0.87</b>	5.0	9.8	0.86	0.88	0.86	0.87	3.0	9.8	0.84	0	36.4
a-credit	45	6.8	0.87	0.88	0.86	0.87	6.2	11	0.86	0.88	0.86	0.88	3.6	11	-	-	"
balance	20	3.2	0.80	0.84	0.76	0.78	19.6	34.8	0.81	0.84	0.78	0.79	14.8	24.6	0.80	0	72.4
balance	15	2.4	0.81	0.85	0.76	<b>0.79</b>	23.8	40.8	0.81	0.85	0.79	0.80	17.2	31	"	0	"
balance	10	1.6	0.83	0.87	0.79	0.80	23.8	52.6	0.83	0.86	0.79	0.79	28.2	38.4	"	0	"
balance	5	0.8	0.86	0.89	0.82	0.82	84.8	86.2	0.85	0.87	0.81	0.80	42.2	48.6	"	0	"
balance	2	0.3	0.90	0.90	0.82	0.81	99.0	114.4	0.89	0.89	0.80	0.80	72.4	65.4	"	0	"
breast-w	40	5.8	0.93	0.97	0.93	<b>0.95</b>	3.4	9.6	0.93	0.97	0.93	<b>0.95</b>	3.4	7.6	0.96	0	15.6
breast-w	30	4.3	0.96	0.96	0.95	0.95	6.8	7.0	0.96	0.97	0.95	0.95	6.8	9.4	"	0	"
chess	200	6.2	0.91	0.91	0.91	0.90	9.0	13	0.91	0.95	0.90	<b>0.95</b>	8.6	13.0	0.99	+	54.4
diabetes	100	7.6	0.75	0.76	0.76	0.75	8.4	9.0	0.75	0.76	0.74	0.74	4.8	8.4	0.74	0	69
diabetes	15	1.9	0.79	0.83	0.75	0.72	26.4	55.4	0.79	0.82	0.74	0.74	20.4	32.4	"	0	"
diabetes	5	0.6	0.84	0.92	<b>0.73</b>	0.65	88	174.0	0.82	0.88	0.75	0.72	38.2	78.2	"	0	"
diabetes	2	0.2	0.90	0.99	0.68	0.66	200.2	288.4	0.84	0.92	0.74	0.71	69	135.2	"	0	"
g-credit	150	15	0.72	0.74	0.71	0.73	6.4	7.0	0.72	0.74	0.71	0.73	5.8	6.8	0.71	0	163
g-credit	100	10	0.73	0.75	0.70	0.70	6.4	11.6	0.73	0.75	0.70	0.71	6.2	9.6	"	0	"
heart-c	30	10.1	0.77	0.84	0.74	0.77	4.4	11.8	0.77	0.84	0.73	<b>0.78</b>	3.6	11.8	0.78	0	31.6
heart-c	10	3.3	0.85	0.91	0.80	0.75	14	22.7	0.85	0.90	0.80	0.81	11.2	22.2	"	0	"
heart-c	5	1.6	0.88	0.94	0.77	0.75	30.8	70	0.87	0.94	0.76	0.80	16.8	35.4	"	0	"
ionosph	50	14.2	0.83	0.86	0.79	<b>0.84</b>	4.0	7.4	0.83	0.86	0.79	<b>0.84</b>	4	6.8	0.86	0	34.6
ionosph	40	11.3	0.89	0.89	0.88	0.88	5	6.8	0.89	0.89	0.88	0.88	5	5.6	"	0	"
mushro	800	9.8	0.92	0.97	0.92	<b>0.97</b>	5.0	11.0	0.92	0.97	0.92	<b>0.97</b>	5.0	11.0	1.0	+	16.8
pendigits	800	10.6	0.51	0.67	0.50	<b>0.68</b>	11.6	15	0.51	0.67	0.51	<b>0.67</b>	11.6	15.0	0.95	+	340
p-tumor	20	5.9	0.42	0.47	0.39	0.37	15.6	20.6	0.42	0.46	0.39	0.38	14.2	17.8	0.40	0	81.2
p-tumor	15	4.4	0.44	0.49	0.38	0.37	22.6	26.8	0.44	0.49	0.39	0.37	19.2	22.2	"	0	"
p-tumor	10	2.9	0.46	0.53	0.38	0.39	23.2	37.2	0.46	0.51	0.37	0.38	30.0	28.8	"	0	"
p-tumor	5	1.4	0.53	0.60	0.40	0.37	56.6	74.2	0.52	0.58	0.39	0.39	42.6	55.4	"	0	"
p-tumor	2	0.5	0.63	0.71	0.40	0.36	116.4	41.5	0.60	0.67	0.40	0.40	81.2	105.2	"	0	"
segment	300	12.9	0.55	0.72	0.55	<b>0.72</b>	7.0	11.0	0.55	0.72	0.55	<b>0.72</b>	7.0	11.0	0.95	0	112.6
segment	200	8.6	0.72	0.83	0.73	<b>0.83</b>	12.6	15.0	0.73	0.83	0.73	<b>0.83</b>	12.6	15.0	"	0	"
soybean	70	11.1	0.51	0.51	0.49	0.49	12.0	12.0	0.51	0.51	0.49	0.49	11.8	11.8	0.82	+	88
soybean	60	9.5	0.51	0.55	0.50	<b>0.55</b>	13.0	15.0	0.51	0.55	0.50	<b>0.55</b>	11.2	15.0	"	+	"
soybean	50	7.9	0.55	0.59	0.52	<b>0.59</b>	14.6	16.8	0.55	0.59	0.51	<b>0.58</b>	14.2	16.8	"	+	"
splice	700	21.9	0.74	0.74	0.74	0.73	5.0	5.0	0.74	0.74	0.74	0.73	5.0	5.0	0.94	+	126.8
thyroid	80	2.4	0.91	0.92	0.91	0.91	1.0	13.4	0.91	0.91	0.91	0.91	1.0	3.0	0.91	+	34.2
vehicle	100	11.8	0.60	0.64	0.58	<b>0.63</b>	10.8	10.0	0.60	0.64	0.57	<b>0.63</b>	11.0	9.0	0.70	+	138
vehicle	80	9.4	0.61	0.67	0.58	<b>0.65</b>	11.4	12.6	0.61	0.67	0.58	<b>0.64</b>	10.6	12.6	"	+	"
vehicle	60	7.0	0.63	0.69	0.59	<b>0.66</b>	16	17	0.63	0.69	0.60	<b>0.66</b>	13.4	15.2	"	+	"
vehicle	50	5.9	0.63	0.71	0.59	<b>0.67</b>	17	22.4	0.63	0.71	0.59	<b>0.67</b>	14.8	22.4	"	0	"
vote	20	4.5	0.96	0.97	<b>0.96</b>	0.94	3.0	15.0	0.96	0.96	0.96	0.94	3.0	7.6	0.96	0	12.6
vote	15	3.4	0.96	0.97	0.95	0.94	3.4	18.0	0.96	0.97	0.96	0.95	3.0	9.2	"	0	"
vowel	100	10.1	0.36	0.39	0.34	0.33	11.0	14.4	0.36	0.39	0.34	0.33	11.0	14.4	0.78	+	290
vowel	70	7.0	0.39	0.46	0.35	0.40	17.0	20.6	0.39	0.45	0.35	0.40	16.8	20.2	"	+	"
yeast	100	6.7	0.53	0.55	0.50	<b>0.53</b>	13.8	15.4	0.53	0.55	0.51	0.53	11.4	13.8	0.53	0	186.0
yeast	20	1.3	0.58	0.62	0.54	0.52	49	84	0.57	0.61	0.54	0.53	32.2	59.0	"	0	"
yeast	10	0.6	0.61	0.67	0.52	0.50	107.2	171.0	0.60	0.65	<b>0.54</b>	0.52	56.0	104.6	"	0	"
yeast	2	0.1	0.74	0.82	0.49	0.48	501.2	724.2	0.68	0.75	<b>0.53</b>	0.50	186.0	307.2	"	+	"

Figure 13: Comparison of J48 and DL8, with and without pruning

If we compare the best results of DL8 with those given by J48 with minimum frequency 2, we see that J48’s test accuracy is significantly better on 9 of the 20 datasets used. However, for all datasets (even if there are no significant difference), the sizes of the trees are a lot smaller for DL8.

There are many possible explanations for the worse results of DL8. One possible explanation is that in many of the datasets with bad results, the number of classes is very high. In some datasets, the lowest possible minimum frequency threshold is higher than the number of examples in the smallest classes, which makes it impossible to classify all examples correctly. To test this hypothesis, we decided to change the minimum frequency constraint into a disjunction of minimum support constraints; every class is given the same minimum support constraint. In this way, we allow that a leaf covers a small number of examples if all

Datasets	%sup	Trainacc	Testacc	Size	Depth
pendigits	50	0.770	0.770	19	7
segment	30	0.878	0.878	27	8
soybean	40	0.756	0.734	33.2	10.3
splice	40	0.740	0.731	5	3
vowel	20	0.610	0.584	52.8	7.8
yeast	10	0.600	0.530	74	11.2
yeast	5	0.633	0.527	126.6	12.9

Figure 15: Results for a disjunction of class minimum support constraints (unpruned)

examples belong to the same class, but we do not allow that a leaf contains a small number of examples if they belong to many different classes. The results of these experiments are listed in Figure 15. As we can see, the accuracies increase in all cases, although not enough to beat J48.

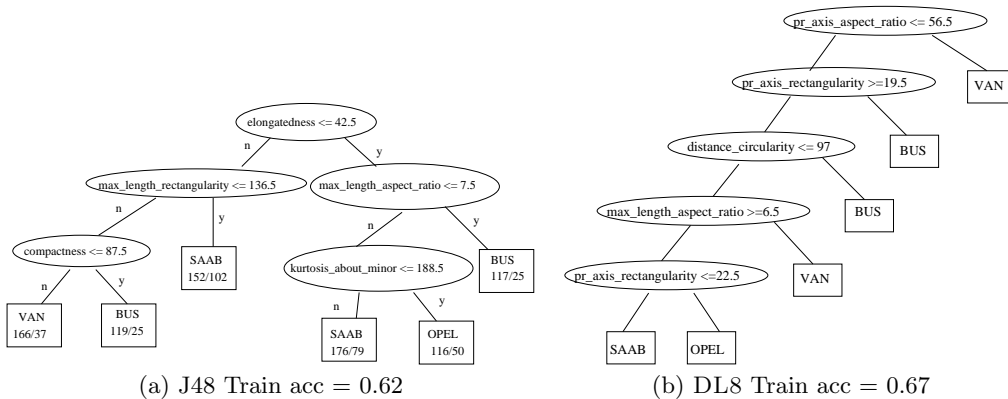


Figure 14: Trees computed by J48 and DL8 on the “vehicle” dataset for support = 80

Datasets	Sup	Max size DL8	Test acc		Size	
			J48	DL8	J48	DL8
balance	2	100	0.82	0.81	99.0	96.6
diabetes	15	27	0.75	0.74	26.4	27.0
g-credit	100	7	0.70	0.72	6.7	7.0
heart-c	10	14	0.80	0.80	14.0	13.0
vote	15	4	0.95	0.96	3.4	3.0
yeast	10	108	0.52	0.52	107.2	107.0

Figure 16: Influence of the size constraint on the test accuracy of DL8 (unpruned)

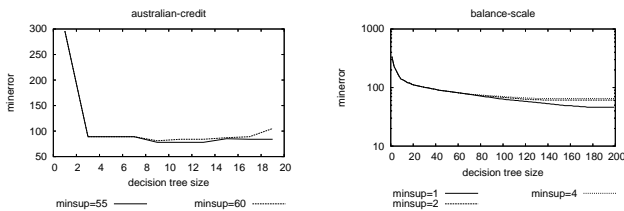


Figure 17: Sizes of decision trees

One of the strengths of DL8 is that it allows to explicitly restrict the size or accuracy. We therefore studied the relation between decision tree accuracies and sizes in more detail. In Figure 16, we show results in which the average size of trees constructed by J48, is taken as a constraint on the size of trees mined by DL8. None of the results given by DL8 are significantly better nor significantly worse than those given by J48.

DL8 can also compute, for every possible size of a decision tree, the smallest error on training data that can possibly be achieved. For two datasets, the results of such a query are given in Figure 17. 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. If we lower the frequency threshold, we can obtain more accurate trees, but only if we allow a sufficient amount of nodes in the tree.

Figures such as Figure 17 are of practical interest, as they allow a user to trade-off the interpretability and the accuracy of a model.

The most surprising conclusion that may however be drawn from all our experiments, is that optimal trees perform remarkably well on most datasets. Our algorithm investigates a vast search space, and its results are still competitive in all

cases. The experiments indicate that the constraints that we employed, either on size, or on minimum support, are sufficient to reduce model complexities and achieve good predictive accuracies.

## 8. RELATED WORK

The search for optimal decision trees dates back to the 70s, when several algorithms for building such trees were proposed. Their applicability was however limited. Research in this direction was therefore almost abandoned in the last two decades. Heuristic tree learners were believed to be nearly optimal for prediction purposes. In this section, we will provide a brief overview of the early work that has been done on searching optimal decision trees. To clarify the relation of these old results with our work, we will summarize them in more modern terminology.

In 1972, Garey [8] proposed an algorithm for constructing an optimal *binary identification procedure*. In this setting, a binary database is given in which every example belongs to a different class. Furthermore, every example has a weight and every attribute has a cost. The target 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. First, a dynamic programming solution is proposed in which bottom-up all subsets of the examples are considered; then, an optimization is introduced in which a distinction is made between two types of attributes: attributes that uniquely identify one example, and attributes that separate classes of examples.

Meisel et al. [14] studied a slightly different setting in 1973, which we would call more common today: multiple examples can have the same class labels, and even numerical attributes are allowed. The first step of this approach builds an overfitting decision tree that classifies all examples correctly; all possible boundaries in the discretization achieved by this tree partition the feature space. The probability of an example is assumed to be the fraction of examples in its corresponding partition in feature space. The task is to find a 100% accurate decision tree with lowest expected cost, where every test has unit cost and examples are distributed according to the previously determined probabilities. The problem is solved by dynamic programming over all possible subsets of tests.

In 1977, it was shown by Payne and Meisel [21] 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. Furthermore, techniques were presented to deal with the situation that some partitions do not contain any examples.

The dynamic programming algorithm of Payne and Meisel is very similar to our algorithm. The main difference is that we build a lattice of tests under a different set of constraints, and choose different datastructures to store the intermediate results of the dynamic programming. The clear link between the work of Payne and Meisel, and results obtained in the data mining and formal concept analysis communities, has not been observed before.

Independently, Schumacher [24] 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, and (2) a probability for every example. The problem is not to learn a predictor for unseen examples —there are no unseen examples—, but to compress the decision table into a compact representation that allows to retrieve the class of an example as quickly as possible. Again, a dynamic programming solution was proposed to compute an optimal tree.

Lew’s algorithm of 1978 [13] is a variation of the algorithm of Schumacher; the main difference is that in Lew’s algorithm it is possible to specify the input decision table in a condensed form, for instance, by using wildcards as attributes. Other extensions allow entries in the decision table to overlap, and to deal with data that is not binary.

More recently, pruning strategies of decision trees have been studied by Garafolakis et al. [9]. DL8, and its early predecessors, can be conceived as an application of Garafolakis’ pruning strategy to another type of datastructure.

Related is also the work of Moore and Lee on the *ADtree* data structure [16]. Both ADtrees and itemset lattices aim at 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. For our bottom-up procedure to work it is necessary that all itemsets that fulfil the given constraints, are stored with associated information. This is not straightforwardly achieved in ADtrees.

The *tree*-relevancy constraint is closely related to the condensed representation of  $\delta$ -free itemsets [3]. Indeed, for  $\delta = \text{minsup} \times |D|$  and  $p(I) := (\text{freq}(I) \geq \text{minfreq})$ , it can be shown that if an itemset is  $\delta$ -free, it is also *tree*-relevant. DL8-CLOSED employs ideas that have also been exploited in the formal concept analysis (FCA) community and in closed itemset miners [25].

A popular topic in data mining is currently the selection of itemsets from a large set of itemsets found by a frequent itemset mining algorithm [27, 12, 5]. DL8 can be seen as one such algorithm for selecting itemsets. It is however the first algorithm that outputs a well-known type of model, and provides accuracy guarantees for this model.

## 9. CONCLUSIONS

We presented DL8, an algorithm for finding decision trees that maximize an optimization criterion under constraints,

and successfully applied this algorithm on a large number of datasets.

We showed that there is a clear link between DL8 and frequent itemset miners, which means that it is possible to apply many of the optimizations that have been proposed for itemset miners also when mining decision trees under constraints. The investigation that we presented here is only a starting point in this direction; it is an open question how fast decision tree miners could become if they were thoroughly integrated with algorithms such as LCM or FP-Growth. 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.

In experiments we compared the test set accuracies of trees mined by DL8 and C4.5. Under the same frequency thresholds, we found that the trees learned by DL8 are often significantly more accurate than trees learned by C4.5. When we compare the best settings of both algorithms, J48 performs significantly better in 45% of the datasets. Efficiency considerations prevented us from applying DL8 on the thresholds where C4.5 performs best, but preliminary results indicate that the best accuracies are not always obtained for the lowest possible frequency thresholds.

Still, our conclusion that trees mined under declarative constraints perform well both on training and test data, means that constraint-based tree miners deserve further study. Many open questions regarding the instability of decision trees, the influence of size constraints, heuristics, pruning strategies, and so on, may be answered by further studies of the results of DL8. Future challenges include extensions of DL8 to other types of data, constraints and optimization criteria. DL8’s results could be compared to many other types of decision tree learners [20, 22].

Given that DL8 can be seen as a relatively cheap type of post-processing on a set of itemsets, 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 [11] that contain both patterns and data.

## Acknowledgments

Siegfried Nijssen was supported by the EU FET IST project “Inductive Querying”, contract number FP6-516169. Éliisa Fromont was supported through the GOA project 2003/8, “Inductive Knowledge bases”, and the FWO project “Foundations for inductive databases”. 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 pre-processing the data that we used in our experiments.

## 10. REFERENCES

- [1] R. Agrawal, H. Mannila, R. Srikant, H. Toivonen, and A. I. Verkamo. Fast discovery of association rules. In *Advances in Knowledge Discovery and Data Mining*, pages 307–328. AAAI/MIT Press, 1996.
- [2] R. J. Bayardo, B. Goethals, and M. J. Zaki, editors. *FIMI ’04, Proceedings of the IEEE ICDM Workshop*

- on Frequent Itemset Mining Implementations, volume 126 of *CEUR Workshop Proceedings*. 2004.
- [3] J.-F. Boulicaut, A. Bykowski, and C. Rigotti. Free-sets: A condensed representation of boolean data for the approximation of frequency queries. *Data Min. Knowl. Discov.*, 7(1):5–22, 2003.
- [4] L. De Raedt. Towards Query Evaluation in Inductive Databases Using Version Spaces. In *KR 2004*, 438–446.
- [5] L. De Raedt and A. Zimmermann. Constraint-Based Pattern Set Mining. In *SIAM International Conference on Data Mining*, to appear, 2007.
- [6] Leo Breiman and J. H. Friedman and R. A. Olshen and C. J. Stone. *Classification and Regression Trees Statistics/Probability Series*, Wadsworth Publishing Company, 1984.
- [7] E. Fromont, H. Blockeel, and J. Struyf. Integrating decision tree learning into inductive databases. In *Revised selected papers of the workshop KDID'06*, LNCS, to appear. Springer, 2007.
- [8] M. R. Garey. Optimal binary identification procedures. *SIAM Journal of Applied Mathematics*, 23(2):173–186, September 1972.
- [9] M. N. Garofalakis, D. Hyun, R. Rastogi, and K. Shim. Building decision trees with constraints. *Data Min. Knowl. Discov.*, 7(2):187–214, 2003.
- [10] J. Han, J. Pei, and Y. Yin. Mining frequent patterns without candidate generation. In *2000 ACM SIGMOD Intl. Conference on Management of Data*, pages 1–12. ACM Press, 2000.
- [11] T. Imielinski and H. Mannila. A database perspective on knowledge discovery. *Comm. Of The ACM*, 39:58–64, 1996.
- [12] A. Knobbe and E. Ho. Maximally informative  $k$ -itemsets and their efficient discovery. In *KDD*, pages 237–244, 2006.
- [13] A. Lew. Optimal conversion of extended-entry decision tables with general cost criteria. *Commun. ACM*, 21(4):269–279, 1978.
- [14] W. S. Meisel and D. Michalopoulos. A partitioning algorithm with application in pattern classification and the optimization of decision trees. *IEEE Trans. Comput.*, 22:93–103, 1973.
- [15] T. Mitchell. *Machine Learning*. McGraw-Hill, New York, 1997.
- [16] A. Moore and M. S. Lee. Cached sufficient statistics for efficient machine learning with large datasets. *Journal of AI Research*, 8:67–91, March 1998.
- [17] B. M. Moret, M. G. Thomason, and R. C. Gonzalez. Optimization criteria for decision trees. Technical Report CS81-6, Dep. of Computer Science, Univ. of New Mexico, Albuquerque, 1981.
- [18] C. Nadeau and Y. Bengio. Inference for the generalization error. *Mach. Learn.*, 52(3):239–281, 2003.
- [19] D. Newman, S. Hettich, C. Blake, and C. Merz. UCI repository of machine learning databases, 1998.
- [20] D. Page and S. Ray. Skewing: An Efficient Alternative to Lookahead for Decision Tree Induction. In: *IJCAI*, pages 601–612, 2003.
- [21] H. J. Payne and W. S. Meisel. An algorithm for constructing optimal binary decision trees. *IEEE Trans. Computers*, 26(9):905–916, 1977.
- [22] F. Provost and P. Domingos. Tree Induction for Probability-based Ranking. *Machine Learning*, 52(3):199–215, 2003.
- [23] J. R. Quinlan. *C4.5: Programs for Machine Learning*. Morgan Kaufmann, 1993.
- [24] H. Schumacher and K. C. Sevcik. The synthetic approach to decision table conversion. *Commun. ACM*, 19(6):343–351, 1976.
- [25] T. Uno, M. Kiyomi, and H. Arimura. *LCM ver. 2: Efficient mining algorithms for frequent/closed/maximal itemsets*. In [2].
- [26] I. H. Witten and E. Frank. *Data Mining: Practical machine learning tools and techniques*. Morgan Kaufmann, San Francisco, 2nd edition edition, 2005.
- [27] X. Yan, H. Cheng, J. Han and D. Xin. Summarizing itemset patterns: a profile-based approach. In *KDD*, pages 314–323, 2005.
- [28] M. J. Zaki, S. Parthasarathy, M. Ogihara, and W. Li. New algorithms for fast discovery of association rules. Technical Report TR651, 1997.