Mining Optimal Decision Trees from Itemset Lattices

Siegfried Nijssen
K.U.Leuven
Celestijnenlaan 200 A
Leuven, Belgium

Elisa Fromont
K.U.Leuven
Celestijnenlaan 200 A
Leuven, Belgium

{siegfried.nijssen,elisa.fromont}@cs.kuleuven.be

ABSTRACT

We present DL8, 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 few efforts have been made to compute such trees for real-world datasets. An exact algorithm is of both scientific and practical interest. From a scientific point of view, it can be used as a gold standard to evaluate the performance of heuristic constraint-based decision tree learners and 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 that there is a relation between constraints on decision trees and constraints on itemsets. We show that optimal decision trees can be extracted from lattices of itemsets in linear time. We give several strategies to efficiently build these lattices. Experiments show that under the same constraints, DL8 obtains better results than C4.5, which confirms that exhaustive search does not always imply overfitting. The results also show that DL8 is a useful and interesting tool to learn decision trees under constraints.

Categories and Subject Descriptors

H.2.8 [**Database Management**]: Database applications—*Data Mining*; I.2.8 [**Artificial Intelligence**]: Problem Solving, Control Methods, and Search—*Dynamic programming*

General Terms

Algorithms, Experimentation, Performance

Keywords

Decision trees, Frequent itemsets, Formal concepts, Constraint-based mining

KDD'07, August 12–15, 2007, San Jose, California, USA.

1. INTRODUCTION

Decision trees are among the most popular prediction models in machine learning and data mining, because there are efficient, relatively easily understandable 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 [8] 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 χ^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 for unseen examples;
- given a dataset D, find the smallest or shallowest decision tree which has an accuracy higher than *minacc*.

In the interactive process that knowledge discovery in databases is, the ability to pose *queries* that answer these questions can be very valuable.

Most well-known algorithms for building decision trees, for instance C4.5, use a top-down induction paradigm, in which a good split is chosen heuristically. If such algorithms do not find a tree that satisfies the specified constraints, this does not mean that such a tree does not exist—it only means that the chosen heuristic is not good enough to find it. An exact algorithm could be desirable to answer queries without uncertainty. Furthermore, to assess the quality of heuristic learners, it is of interest to know, for a sufficiently large number of datasets, what their true optimum under given constraints is. This would allow us to gain deeper insight in the predictive behavior of decision trees. For instance, [18] reported that for small, mostly artificial datasets, small decision trees are not always preferable in terms of generalization ability, while [27] showed that when learning rules, exhaustive searching and overfitting are orthogonal. An efficient algorithm for learning decision trees under constraints

allows us to investigate these observations for larger datasets and more complex models.

To the best of our knowledge, few attempts have been made to implement a general and exact algorithm for learning decision trees under constraints; most people have not seriously considered the problem as it is known to be NP-complete [12], and therefore, an efficient algorithm most likely does not exist. This theoretical result however does not imply that the problem is unsolvable in all cases. In the data mining literature, several exponential problems have still been shown to be solvable in practice. In particular, the problem of frequent itemset mining has attracted a lot of research [1, 32, 11], and many frequent itemset mining algorithms have been applied successfully despite the exponential nature of this problem.

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.

This 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 are looking for. In Section 4, we motivate the use of such optimal trees. In section 5, we present our algorithm and its connection to frequent itemset mining. In Section 6, 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 7 gives related work. We conclude in Section 8.

2. ITEMSET LATTICES FOR DECISION TREE MINING

Let us first introduce some terminology regarding *frequent itemsets* and *decision trees*.

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) $t(I) \subseteq \{1, 2, \ldots n\}$ 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. $freq(I) = |t(I)|$; 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, $freq(I) \geq minfreq$).

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*.

A decision tree aims at classifying examples by sorting them down a tree. The leaves of a tree provide the classifications of examples [17]. 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 values of the attribute. 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 $T_j = \{i | B_{ij} = 1\} \cup \{\neg i | B_{ij} = 0\}$. 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 1. 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 contains. 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, $paths(T) = \{\emptyset, \{B\}, \{\neg B\}, \{\neg B, C\}, \{\neg B, \neg C\}\}$.

Contrary to what is common in the data mining literature, in this case it is essential that we include *negative items*, such as $\neg B$, in the itemsets.

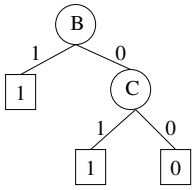


Figure 1: An example tree

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 2. 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.

From the theory of frequent itemset mining, it is known that itemsets form a *lattice* (these are typically depicted as in Figure 2). In this paper we present DL8, an algorithm for mining **D**ecision trees from **L**attices.

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.

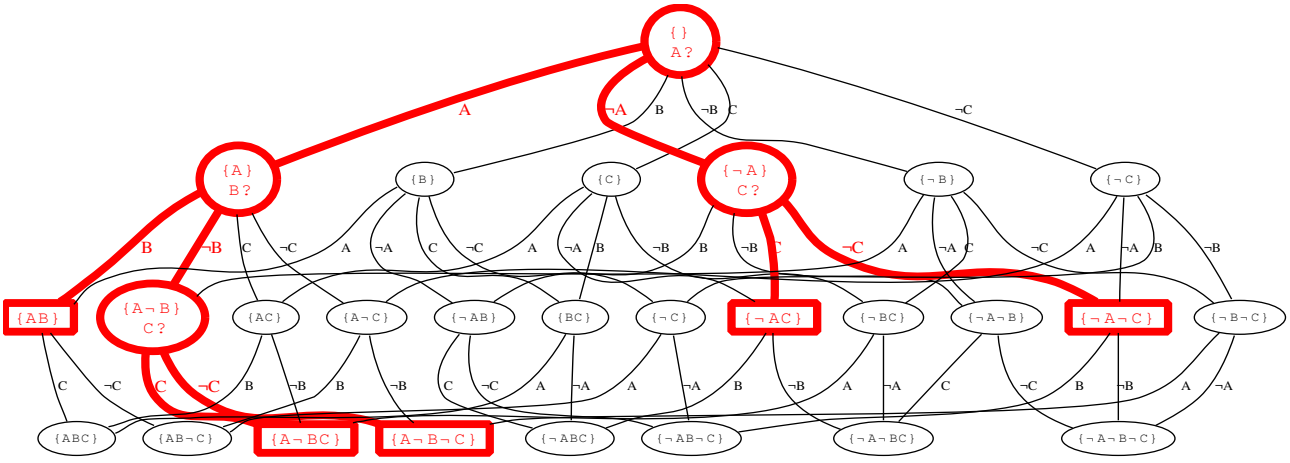


Figure 2: An itemset lattice for items $\{A, \neg A, B, \neg B, C, \neg C\}$; binary decision tree A(B(C(l,l),l),C(l,l)) is hidden in this lattice

The first part specifies the constraints on the nodes of the decision trees.

1. $\mathcal{T}_1 := \{T | T \in DecisionTrees, \forall I \in paths(T), p(I)\}$

The set $\mathcal{T}_1$ is called the set of *locally constrained decision trees* and *DecisionTrees* is the set of all possible decision trees. The predicate $p(I)$ expresses a constraint on paths. In our simplest setting, $p(I) := (freq(I) \geq minfreq)$. The predicate $p(I)$ must fulfill these properties:

- the evaluation of $p(I)$ must be independent of the tree T of which I is part.
- p must be *anti-monotonic*. A predicate $p(I)$ on itemsets $I \subseteq \mathcal{I}$ is called *anti-monotonic* iff $p(I) \wedge (I' \subseteq I) \Rightarrow p(I')$.

We can distinguish two types of local constraints: coverage-based constraints, such as frequency, of which the fulfillment is entirely dependent on $t(I)$, and pattern-based constraints, such as the size of an itemset, of which the fulfillment depends on the properties of the (items in the) itemset itself. In the following, we consider only coverage-based constraints; extensions to pattern-based constraints are possible, but beyond the scope of this paper.

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

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

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 estimation procedure;
- $size(T)$, 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. output $argmin_{T \in \mathcal{T}_2}[r_1(T), r_2(T), \ldots, r_n(T)]$

The tuples $\mathbf{r}(T) = [r_1(T), r_2(T), \ldots, r_n(T)]$ 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 $depth(T) \leq maxdepth$ (respectively $size(T) \leq maxsize$).

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.*

$$\mathcal{T} := \{\ T \quad | \quad T \in DecisionTrees, \forall I \in paths(T), freq(I) \geq minfreq\ \}$$
$$output\ argmin_{T \in \mathcal{T}}[e(T), size(T)].$$

In other words, we have $p(T) := (freq(I) \geq minfreq)$, $q(T) := true$ 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. To retrieve *accurate trees of bounded size*, Query 1 can be extended such that $q(T) := size(T) \leq maxsize$.

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 2. *Accurate Trees of Bounded Size and Accuracy.*

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

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 $x(freq_1(I), \ldots freq_n(I))$ for each leaf I of the decision tree, from which we can derive a new estimated number of errors

$$ex(T) = \sum_{I \in leaves(T)} e(I) + x(freq_1(I), \ldots freq_n(I)).$$

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

QUERY 3. *Small Accurate Pruned Trees.*

$$\mathcal{T} := \{\ T \quad | \quad T \in DecisionTrees, \forall I \in paths(T), freq(I) \geq minfreq\ \}$$
$$output\ argmin_{T \in \mathcal{T}}[ex(T), size(T)].$$

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

4. MOTIVATING EXAMPLES

To motivate our work, it is useful to briefly consider two examples that illustrate what kind of trees cannot be found if the well-known information gain (ratio) heuristic of C4.5 is used to answer Query 1 of Section 3.

A	B	C	Class	#
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 3: Database 1

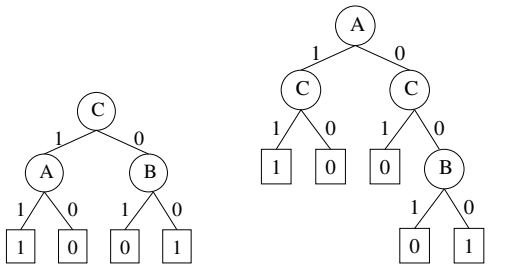
A	B	C	Class	#
1	1	1	1	30×
1	1	0	0	20×
0	1	0	0	8×
0	1	1	0	12×
0	0	0	1	12×
0	0	1	0	18×

Figure 4: Database 2

As a first example, consider the database in Figure 3, in which we have 2 target classes. The last column indicates how many times an example is repeated. Assume that we are interested in answering Query 1 with $minfreq = 10$. An optimal tree exists (see Figure 1), but a heuristic learner will not find it, as it prefers attribute A in the root: A has information gain 0.33 (resp. ratio 0.54), while B only has information gain 0.26 (resp. ratio 0.37).

As a second example, consider the database in Figure 4, which is a variation of the XOR problem. Then the correct answer to Query 1 with $minfreq = 1$ is given in Figure 5(a), but the use of information gain (ratio) would yield the tree in Figure 5(b), as the information gain (resp. ratio) of A is 0.098 (resp. 0.098), while the information gain of C is 0.029 (resp. 0.030).

We learn from these examples that the proportions of examples can 'fool' heuristic decision trees into an suboptimal shape, as also observed in [22]. Optimal learners are less sensitive to such behavior.



(a) Smallest (b) Learned using heuristics

Figure 5: Two accurate trees for database 2

5. THE DL8 ALGORITHM

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

The main idea behind DL8 is that the lattice of itemsets, as depicted in Figure 2, can be traversed bottom-up, and that we can determine the best decision tree(s) for the transactions $t(I)$ covered by an itemset I by combining for all $i \in \mathcal{I}$, the optimal trees of its children $I \cup \{i\}$ and $I \cup \{\neg i\}$ in the lattice. The main property that we exploit is that if a tree is optimal, then also the left-hand and right-hand branch of its root must be optimal; this applies to every subtree of the decision tree.

More formally, the parameters of DL8 are the local constraint p, 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 ∞. 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 $\mathbf{r}(T) = [r_1(T), \ldots, r_n(T)]$ be the ranking function, and let k be the index of the obligatory error function in this ranking. If $r_1, \ldots, r_{k-1} \in \{depth, size\}$ 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 $[r_k(T), \ldots, r_n(T)]$, such that $size(T) \leq s$ and $depth(T) \leq d$.

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 finally, we use a predicate $pure(I)$ which blocks the recursion if all examples $t(I)$ belong to the same class.

The algorithm is most easily understood if $maxdepth = \infty$, $maxsize = \infty$, $maxerror = \infty$ 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 34.

The correctness of the algorithm 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 satisfying the constraints 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) a candidate decision tree for classifying the examples $t(I)$ consists of a single leaf.

(line 20) 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 (see [21] for more details).

(line 23) in this line the anti-monotonic property of the predicate $p(I)$ is used: an itemset that does not satisfy the predicate $p(I)$ cannot be part of a tree, nor can any of its supersets; therefore the search is not continued if $p(I \cup \{i\}) = \mathit{false}$ or $p(I \cup \{\neg i\}) = \mathit{false}$.

(line 22–33) 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 $\mathit{size}(T_1)$ and depth at most $\mathit{depth}(T_1)$; 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 $\mathit{size}(T_1)$, $\mathit{size}(T_2) \leq \mathit{maxsize}$ and $\mathit{depth}(T_1)$, $\mathit{depth}(T_2) \leq \mathit{maxdepth}$. Consequently T (or a tree with the same properties) must be among the trees found by combining results from the two recursive procedure calls in line 27.

A key feature of DL8-Recursive is that in line 34 it stores every results that it computes. Consequently, DL8 avoids that optimal decision trees for any itemset are computed more than once; furthermore, we do not need to store the entire decision trees with every itemset; it is sufficient to store the root and statistics (error, possibly size and depth); 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 $k-\mathit{best}$ trees can also straightforwardly be output.

To efficiently index the itemsets I, a trie data structure can be used [7].

As with most data mining algorithms, the most time consuming operations are those that access the data. DL8 requires frequency counts for itemsets in line 20, 23 and 32. 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 constrained FIM approach, and the closure based single-step approach.

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

```
 1: if maxsize ≠ ∞ then
 2:     S ← {1, 2, ..., maxsize}
 3: else
 4:     S ← {∞}
 5: if maxdepth ≠ ∞ then
 6:     D ← {1, 2, ..., maxdepth}
 7: else
 8:     D ← {∞}
 9: T ←DL8-Recursive(∅)
10: if maxerror ≠ ∞ then
11:     T ← {T|T ∈ T, e(T) ≤ maxerror}
12: if T = ∅ then
13:     return undefined
14: return argmin_{T∈T} r(T)
15:
16: procedure DL8-Recursive(I)
17:     if DL8-Recursive(I) was computed before then
18:         return stored result
19:     C ← {l(c(I))}
20:     if pure(I) then
21:         store C as the result for I and return C
22:     for all i ∈ I do
23:         if p(I∪{i}) = true and p(I∪{¬i}) = true then
24:             T1 ← DL8-Recursive(I ∪ {i})
25:             T2 ← DL8-Recursive(I ∪ {¬i})
26:             for all T1 ∈ T1, T2 ∈ T2 do
27:                 C ← C ∪ {n(i, T1, T2)}
28:         end if
29:     T ← ∅
30:     for all d ∈ D, s ∈ S do
31:         L ← {T ∈ C|depth(T) ≤ d ∧ size(T) ≤ s}
32:         T ← T ∪ {argmin_{T∈L}[r_k = e_{t(I)}(T), ..., r_n(T)]}
33:     end for
34:     store T as the result for I and return T
35: end procedure
```

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 to avoid later recomputations.

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. If p is a minimum frequency constraint, we can use a frequent itemset miner to obtain the frequencies of itemsets 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, 32, 11, 29].

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 referred to as Apriori-Freq+DL8. Compared to other itemset miners, 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 2, then DL8 operates in time linear in the number of edges of this graph.

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. Proofs of the theorems given in this section can be found in [21].

If we consider the DL8-SIMPLE algorithm, an itemset $I = \{i_1, \ldots, i_n\}$ is stored 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 to 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

- the $\neg pure(I')$ predicate is false in line (20);
- the conjunction $p(I' \cup \{i_{k_{m+1}}\}) \wedge p(I' \cup \{\neg i_{k_{m+1}}\})$ is false in line (23).

It is helpful to negate the *pure* predicate, as one can easily see that $\neg pure$ is an anti-monotonic predicate (every superset of a pure itemset, must also be pure). From now on, we will refer to $\neg pure$ 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 1. *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*

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

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 $\mathcal{L}_1 = \mathcal{L}_2$.*

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

THEOREM 2. *Itemset relevancy is an anti-monotonic property.*

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. 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 [32].

The Closure-Based Single-Step Approach

In the simple single-step approach, we stored the optimal decision trees for every itemset separately. 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

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

for a TID-set t, then the *closure* of itemset I is the itemset $i(t(I))$. An itemset I is *c*losed 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 simple single-step approach, we could index the results on $i(t(I))$ instead of I itself.

We incorporate this observation as follows in Algorithm 1. Before executing line 17, we make a pass over the data to determine the closure I' of the itemset I, and to collect the frequencies of all itemsets $\{I \cup \{i\}, I \cup \{\neg i\} | i \in \mathcal{I}\}$. In line 17 we check if DL8-RECURSIVE(I) was already computed earlier by searching for I' in a trie data structure. In line 34, we associate the result to I' instead of I itself.

Our single-step approach which relies on closed itemset indexing is called DL8-CLOSED. Obviously, DL8-CLOSED will never consider more itemsets than DL8-SIMPLE, DL8-APRIORI or DL8-ECLAT; itemsets stored by DL8-CLOSED may however be longer as they contain all items in their closure.

Our implementation of DL8-CLOSED is based on optimization strategies that are common in depth-first frequent itemset miners, such as the use of projected databases, with modifications that make sure that the space complexity of our algorithm is $\theta(n + m)$, where n is the size of the trie that stores all closed itemsets, and m is the size of the binary matrix that contains the data. For more details see [21].

6. 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 J48, the Java implementation of C4.5 [26] in WEKA [30]. 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 [20]. Numerical data were discretized before applying the learning algorithms using WEKA's unsupervised discretization

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 6: 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 8: Properties of the algorithms used in the experiments

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 6 gives a brief description of the datasets that we used in terms of the number of examples and the number of attributes after binarization.

6.1 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 8. Besides DL8-Apriori, DL8-Eclat and DL8-Closed, we also include unmodified implementations of the frequent itemset miners Apriori [1], Eclat [32] and LCM [29] in the comparison. These implementations were obtained from the FIMI website [3]. 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. Furthermore, we also perform frequent and closed itemset mining experiments on data with only the positive items, to estimate the added complexity of using negative items.

Results for four datasets are listed in Figure 7. We aborted runs of algorithms that lasted for longer than 1500s. More results can be found in [21]. We only show datasets here in which frequent itemset miners manage to run within 1500s.

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. Bear in mind, however, that LCM does not output itemsets in a form that can be used efficiently by DL8.

In all cases, DL8-Closed is faster than DL8-Apriori or DL8-Eclat. In those cases where Apriori-Freq+DL8 can store the entire output of Apriori in memory, we see that the additional runtime for storing these results is significant. On the other hand, if we perform relevancy pruning, the resulting algorithm is usually faster than the original 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%.

6.2 Accuracy

Figure 9 provides the results of experiments in which we used stratified 10-fold cross-validation to compute the training and test accuracies of DL8-Closed and J48. For each dataset, we lowered this frequency to the lowest value that still allowed the computation to be performed within the memory of our computers. 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 3 of Section 3). We used a corrected two-tailed t-test [19] 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.

First, both algorithms were applied with the same minimum frequency constraint. 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 and only significantly worse on one dataset. 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 time smaller for DL8. After pruning, DL8's trees are still 1.5 times larger than J48's ones. A closer inspection of these trees reveals a similar phenomenon as we discussed for the data in Figure 3: C4.5's trees are smaller as it creates trees with small numbers of incorrectly classified examples in the leaves, which cannot be split off without violating the constraints. In cases where DL8's accuracy is significantly better, the pruned trees of DL8 are 3 to 9 nodes larger than those of J48. These results confirm earlier findings which show that smaller trees are not always desirable. If we compare the multiple minimum frequency constraints, it turns out that the best test accuracy results are not always obtained for the lowest minimum frequencies.

Second, in the last six columns of Figure 9, we give results for J48 with its default *minfreq*= 2 setting, both when using the discretized data, and when using the original, non-discretized data. The test accuracies of J48 with *minfreq*= 2 are compared with the test accuracies of DL8 for the various *minfreq* values, when using pruning. The results of the significance test are given in the "S" column: "+" means that

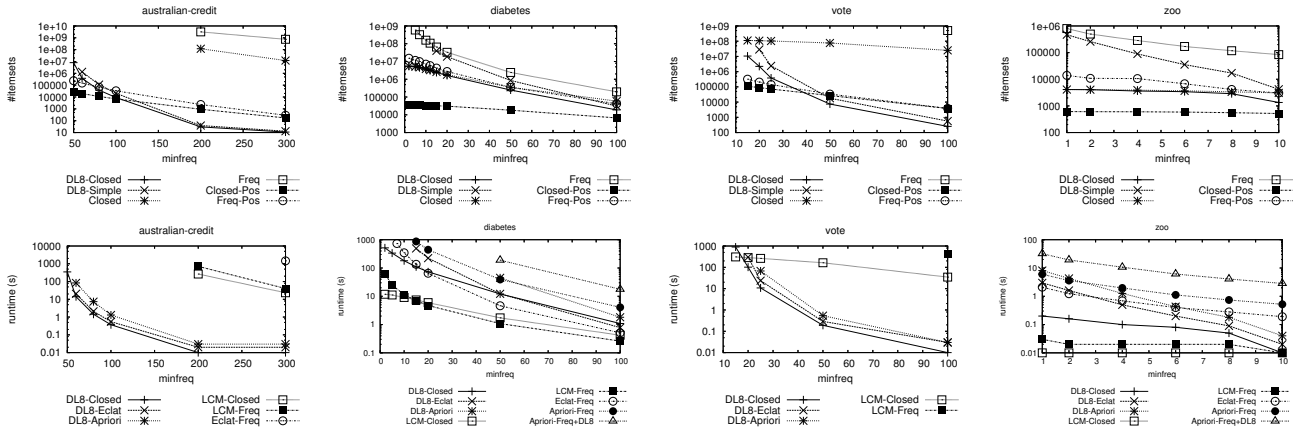


Figure 7: Comparison of the different miners on 4 UCI datasets

	Minfreq		Unpruned Train acc		Unpruned Test acc		Unpruned Size		Pruned Train acc		Pruned Test acc		Pruned Size		Pruned J48, **Minfreq = 2** Test acc		S		Size	
Datasets	#	%	J48	DL8	J48	DL8	J48	DL8	J48	DL8	J48	DL8	J48	DL8	Discr.	No D.	Discr.	No D.	Discr.	No D.
anneal	10	1.2	.85	.87	.82	.81	43	57	.83	.85	.81	.82	22	31	.82	.88	0	+	44	54
	2	.2	.89	.89	.82	.82	107	88	.86	.87	.82	.82	44	46			0	+		
a-credit	45	6.8	.87	.88	.86	.87	6	11	.86	.88	.86	.88	4	11	.84	.86	-	0	36	40
balance	15	2.4	.81	.85	.76	**.79**	24	41	.81	.85	.79	.80	17	31	.80	.79	0	0	72	84
	2	.3	.90	.90	.82	.81	99	114	.89	.89	.80	.80	72	65			0	0		
breast-w	40	5.8	.93	.97	.93	**.95**	3	10	.93	.97	.93	**.95**	3	8	.96	.95	0	0	16	20
	30	4.3	.96	.96	.95	.95	7	7	.96	.97	.95	.95	7	9			0	0		
chess	200	6.2	.91	.91	.91	.90	9	13	.91	.95	.90	**.95**	9	13	.99	1	+	+	54	54
diabetes	15	1.9	.79	.83	.75	.72	26	55	.79	.82	.74	.74	20	32	.74	.73	0	0	69	41
	2	.2	.90	.99	.68	.66	20	288	.84	.92	.74	.71	69	135			0	0		
g-credit	100	10	.73	.75	.70	.70	6	12	.73	.75	.70	.71	6	10	.71	.70	0	0	163	171
heart-c	30	1.1	.77	.84	.74	.77	4	12	.77	.84	.73	**.78**	4	12	.78	.80	0	0	32	37
	5	1.6	.88	.94	.77	.75	4	70	.87	.94	.76	.80	17	35			0	0		
	2	.6	.94	1	.76	.74	68	74	.90	.97	.78	.77	32	50			0	0		
ionosph	50	14.2	.83	.86	.79	**.84**	4	7	.83	.86	.79	**.84**	4	7	.86	.91	0	+	35	27
	40	11.3	.89	.89	.88	.88	5	7	.89	.89	.88	.88	5	6			0	0		
mushro	800	9.8	.92	.97	.92	**.97**	5	11	.92	.97	.92	**.97**	5	11	1	1	+	+	17	17
pendigits	600	8.0	.58	.72	.58	**.72**	14	17	.58	.72	.58	.72	13	15	.95	.96	+	+	340	278
p-tumor	15	4.4	.44	.49	.38	.37	23	27	.44	.49	.39	.37	19	22	.40	.40	0	0	81	81
	2	.5	.63	.71	.40	.36	116	152	.60	.67	.40	.40	81	105			0	0		
segment	200	8.6	.72	.83	.73	**.83**	13	15	.73	.83	.73	**.83**	13	15	.95	.97	+	+	80	113
soybean	60	9.5	.51	.55	.50	**.55**	13	15	.51	.55	.50	**.55**	11	15	.82	.82	+	+	88	88
	50	7.9	.55	.59	.52	**.59**	15	17	.55	.59	.51	**.58**	14	17			+	+		
splice	700	21.9	.74	.74	.74	.73	5	5	.74	.74	.74	.73	5	5	.94	.94	+	+	127	127
thyroid	80	2.4	.91	.92	.91	.91	1	13	.91	.91	.91	.91	1	3	.91	.99	0	+	34	21
vehicle	50	5.9	.63	.71	.59	**.67**	17	22	.63	.71	.59	**.67**	15	22	.70	.72	0	+	139	135
vote	20	4.5	.96	.97	**.96**	.94	3	15	.96	.96	.96	.94	3	8	.96	.96	0	0	13	13
	15	3.4	.96	.97	.95	.94	3	18	.96	.97	.96	.95	3	9			0	0		
vowel	100	1.1	.36	.39	.34	.33	11	14	.36	.39	.34	.33	11	14	.78	.82	+	+	290	191
	70	7.0	.39	.46	.35	.40	17	21	.39	.45	.35	.40	17	20			+	+		
yeast	100	6.7	.53	.55	.50	**.53**	14	15	.53	.55	.51	.53	11	14	.53	.56	0	0	186	318
	2	.1	.74	.82	.49	.48	501	724	.68	.75	**.53**	.50	186	307			+	+		

Figure 9: Comparison of J48 and DL8, with and without pruning

J48 is significantly better, "-" that it is significantly worse and "0" that there is no significant difference. This comparison shows that the constraints have a negative impact on the accuracy for 7 of the 20 datasets. Furthermore, J48 is better on 3 additional datasets if discretization is not performed beforehand, revealing that a good discretization is sometimes beneficial. The results of DL8 under constraints indicate however that the impacts of these constraints on accuracies are not as large as one would suspect if one would only consider the results of J48 using constraints.

One of the strengths of DL8 is that it allows us to explicitly restrict the size or accuracy. We therefore studied the relation between decision tree accuracies and sizes in more detail. In Figure 10, 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 11. In general, if we increase the size of

Datasets	Sup	Max size	Test acc		Size	
		DL8	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 10: Influence of the size constraint on the test accuracy of DL8 (unpruned)

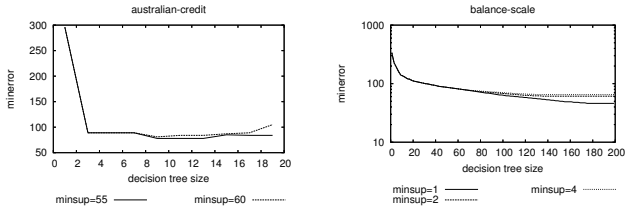


Figure 11: Sizes of decision trees

a decision tree, its accuracy improves quickly at first. Only small improvements can be obtained by further increasing the size of the tree. Figures such as Figure 11 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 could consequently be very sensitive to overfitting. Still, its results are 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.

7. RELATED WORK

The search for optimal decision trees dates back to the 70s, when several dynamic programming algorithms for building such trees were proposed [9, 16, 24, 28, 15]. This early work concentrated on finding small summarizations of input data, and did not study the prediction of unseen examples. Optimization criteria were based on the cost of attributes, and the size or the depth of a tree. Afterwards, attention mostly shifted to heuristic decision tree learners, which were found to obtain satisfactory results for many datasets in a fraction of the runtime; theoretical results were obtained that show to what extent heuristic decision trees can be considered optimal [14, 6]. Still, the idea of exhaustively finding optimal decision trees under certain constraints was also studied [2, 18], but only for much smaller datasets and smaller types of trees than studied in this paper. Recently [4] presented a dynamic programming algorithm that is quite similar to DL8 and its early ancestors. A new optimization criterion was introduced for finding optimal dyadic decision trees, which use a fixed mechanism for discretization of data. This algorithm was only applied on small datasets, and did not investigate the link with data mining algorithms.

More recently, pruning strategies of decision trees have been studied [10]. DL8 can be conceived as the generalization of these pruning strategies to another data structure.

Algorithmically, the *relevancy* constraint presented in Section 5 with $p_2 = true$ is closely related to the condensed representation of δ-free itemsets [5]. For $\delta = minsup \times |D|$ and $p_1(I) := (freq(I) \geq minfreq)$, it can be shown that if an itemset is δ-free, it is also *relevant*. DL8-Closed employs ideas that have also been exploited in the formal concept analysis (FCA) community and in closed itemset miners [23].

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 (see for instance, [31]). 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.

8. 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 for 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 [22, 25].

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 [13] that contain both patterns and data.

Acknowledgements

Siegfried Nijssen was supported by the EU FET IST project "Inductive Querying", contract number FP6-516169. Elisa 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 preprocessing the data that we used in our experiments.

9. 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] P. Auer, R. C. Holte, and W. Maass. Theory and applications of agnostic PAC-learning with small decision trees. In *ICML*, pages 21–29, 1995.

[3] R. J. Bayardo, B. Goethals, and M. J. Zaki, editors. *FIMI '04, Proceedings of the IEEE ICDM Workshop on Frequent Itemset Mining Implementations, Brighton, UK*, volume 126 of *CEUR Workshop Proceedings*. CEUR-WS.org, 2004.

[4] G. Blanchard, C. Schäfer, Y. Rozenholc, and K. R. Müller. Optimal dyadic decision trees. *Machine Learning*, 66(2-3):209–241, 2007.

[5] J.-F. Boulicaut, A. Bykowski, and C. Rigotti. Free-sets: A condensed representation of boolean data for the approximation of frequency queries. *Data Mining and Knowledge Discovery*, 7(1):5–22, 2003.

[6] T. G. Dietterich, M. J. Kearns, and Y. Mansour. Applying the waek learning framework to understand and improve C4.5. In *ICML*, pages 96–104, 1996.

[7] E. Fredkin. Trie memory. *Communications of the ACM*, 3(9):490–499, 1960.

[8] E. Fromont, H. Blockeel, and J. Struyf. Integrating decision tree learning into inductive databases. In *Revised selected papers of the workshop on Knowledge Discovery in Inductive Databases (KDID'06)*. To appear, Springer, 2007.

[9] M. R. Garey. Optimal binary identification procedures. *SIAM Journal of Applied Mathematics*, 23(2):173–186, 1972.

[10] M. N. Garofalakis, D. Hyun, R. Rastogi, and K. Shim. Building decision trees with constraints. *Data Mining and Knowledge Discovery*, 7(2):187–214, 2003.

[11] J. Han, J. Pei, and Y. Yin. Mining frequent patterns without candidate generation. In *2000 ACM SIGMOD International Conference on Management of Data*, pages 1–12. ACM Press, 2000.

[12] L. Hyafil and R. L. Rivest. Constructing optimal binary decision trees is NP-complete. *Information Processing Letters*, 5(1):15–17, 1976.

[13] T. Imielinski and H. Mannila. A database perspective on knowledge discovery. *Communications of the ACM*, 39:58–64, 1996.

[14] M. J. Kearns and Y. Mansour. On the boosting ability of top-down decision tree learning algorithms. *Journal of Computer and System Sciences*, 58(1):109–128, 1999.

[15] A. Lew. Optimal conversion of extended-entry decision tables with general cost criteria. *Communications of the ACM*, 21(4):269–279, 1978.

[16] W. S. Meisel and D. Michalopoulos. A partitioning algorithm with application in pattern classification and the optimization of decision tree. *IEEE Trans. Comput.*, C-22:93–103, 1973.

[17] T. Mitchell. *Machine Learning*. McGraw-Hill, New York, 1997.

[18] P. M. Murphy and M. J. Pazzani. *Exploring the decision forest: an empirical investigation of Occam's razor in decision tree induction*. Computational Learning Theory and Natural Learning Systems, volume IV: Making Learning Systems Practical, pages 171–187. MIT Press, 1997.

[19] C. Nadeau and Y. Bengio. Inference for the generalization error. *Machine Learning*, 52(3):239–281, 2003.

[20] D. Newman, S. Hettich, C. Blake, and C. Merz. UCI repository of machine learning databases, 1998.

[21] S. Nijssen and E. Fromont. Mining optimal decision trees from itemset lattices. Technical report CW476, Dept. Computer Science, Katholieke Universiteit Leuven, March 2007.

[22] D. Page and S. Ray. Skewing: An efficient alternative to lookahead for decision tree induction. In *Proceedings of the Eighteenth International Joint Conference on Artificial Intelligence (IJCAI'03)*, pages 601–612, 2003.

[23] N. Pasquier, Y. Bastide, R. Taouil, and L. Lakhal. Efficient mining of association rules using closed itemset lattices. *Information Systems*, 24(1):25–46, 1999.

[24] H. J. Payne and W. S. Meisel. An algorithm for constructing optimal binary decision trees. *IEEE Trans. Computers*, 26(9):905–916, 1977.

[25] F. Provost and P. Domingos. Tree induction for probability-based ranking. *Machine Learning*, 52:199–215, 2003.

[26] J. R. Quinlan. *C4.5: Programs for Machine Learning*. Morgan Kaufmann, 1993.

[27] J. R. Quinlan and R. M. Cameron-Jones. Oversearching and layered search in empirical learning. In *Proceedings of the Fourteenth International Joint Conference on Artificial Intelligence (IJCAI'95)*, pages 1019–1024, 1995.

[28] H. Schumacher and K. C. Sevcik. The synthetic approach to decision table conversion. *Communications of the ACM*, 19(6):343–351, 1976.

[29] T. Uno, M. Kiyomi, and H. Arimura. LCM ver. 2: Efficient mining algorithms for frequent/closed/maximal itemsets. In [3], 2004.

[30] I. H. Witten and E. Frank. *Data Mining: Practical machine learning tools and techniques*. Morgan Kaufmann, San Francisco, 2nd edition edition, 2005.

[31] X. Yan, H. Cheng, J. Han, and D. Xin. Summarizing itemset patterns: a profile-based approach. In *Proceeding of the conference on Knowledge Discovery and Data Mining (SIGKDD'05)*, pages 314–323, 2005.

[32] M. J. Zaki, S. Parthasarathy, M. Ogihara, and W. Li. New algorithms for fast discovery of association rules. In *Proceeding of the conference on Knowledge Discovery and Data Mining (SIGKDD'97)*, pages 283–286, 1997.