Apprentissage d'arbres de décision optimaux à partir de treillis d'itemsets

Elisa Fromont, Siegfried Nijssen

K.U.Leuven
Celestijnenlaan 200 A, 3000 Leuven, Belgique
{elisa.fromont, siegfried.nijssen}@cs.kuleuven.be

Résumé : Nous présentons DL8, un algorithme permettant d'apprendre des arbres de décision sous contraintes. Cet algorithme permet d'optimiser des critères de taille, de profondeur et de précision de l'arbre. Un algorithme exact est intéressant du point de vue pratique comme du point de vue purement scientifique. Il peut, par exemple, être utilisé comme référence pour évaluer les performances et comprendre le comportement des systèmes d'apprentissage d'arbres de décision utilisant des heuristiques. Du point de vue applicatif, il peut permettre de découvrir des arbres ne pouvant pas être appris par ces systèmes d'apprentissage. DL8 repose essentiellement sur la relation existant entre les contraintes applicables aux arbres de décision et celles applicables aux itemsets. Nous proposons d'exploiter des treillis d'itemsets pour extraire des arbres de décision optimaux en temps linéaire et donnons différentes stratégies permettant de construire ces treillis efficacement. Nos expériences montrent que la précision en test de DL8 est meilleure que celle de systèmes tel que C4.5 en utilisant les mêmes contraintes, ce qui confirme les résultats stipulant qu'une recherche exhaustive n'entraine pas forcement un sur-apprentissage. Ces expériences prouvent également que DL8 est un outil utile et intéressant pour apprendre des arbres de décision sous contraintes.

Mots-clés : Arbres de décision, recherche d'itemsets fréquents, treillis d'itemsets, analyse de concepts formels, fouille de données sous contraintes.

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 by Fromont *et al.* (2007) 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 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*.

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

Most 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 a tree satisfying the constraints 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 on training data under some given constraints is. This would allow us to gain further insight in the behavior of decision trees under constraints. For instance, Murphy & Pazzani (1997) reported that for small, mostly artificial datasets, small decision trees are not always preferable in terms of generalization ability, while Quinlan & Cameron-Jones (1995) 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.

It is important to point out that we are essentially interested in the interaction between constraints and predictive behavior, as we believe that this may aid users to gain more insight in their data. If there are no constraints on size or support, it is almost always possible to learn a decision tree that is 100% accurate on training data by continuing to grow a tree as long as necessary. Theoretical results have been obtained that bound the size that a tree should have to obtain arbitrarily high accuracy (Kearns & Mansour (1999); Dietterich *et al.* (1996)) given access to an oracle. Nock & Nielsen (2004) also give some results to bound the worst-case reduction in error in each test node of the tree. These results provide valuable insight in the quality of heuristic learners, as they show that under certain circumstances heuristic learners are capable of finding accurate decision trees on training data within a bounded number of steps. These results however do not address any of the example queries listed above. Similarly, interesting results have also been obtained in the framework of PAC learning, where learners are required that can obtain high accuracy on training data in reasonable time ; however, only very limited types of decision trees have been studied in this framework (Auer *et al.*, 1995).

To the best of our knowledge, few attempts have been made to compute exact optimal trees in a setting which optimizes decision trees under a wide range of constraints ; most people have not seriously considered the problem as it is known to be NP hard (Hyafil & Rivest, 1976), and therefore, an efficient algorithm can most likely not exist. In this paper, our hypothesis is that this theoretical limitation does not mean that optimal trees

might not be computable in practice in many datasets. The tool that we intend to use are frequent itemset miners (Agrawal *et al.*, 1996; Zaki *et al.*, 1997). The problem of frequent itemset mining, which has been studied extensively in the data mining community, is also not efficiently solvable in theory, but in practice these algorithms have been applied successfully. We investigate to what extent we can use these algorithms to solve the theoretically hard problem of finding decision trees under constraints. As a result, we propose DL8, an exact algorithm for building decision trees that does not rely on the traditional approach of heuristic top-down induction. 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 consider. 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 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 background information about *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. $\mathit{freq}(I) = |t(I)|$; the support of an itemset is $\mathit{support}(I) = \mathit{freq}/|D|$. An itemset I is said to be frequent if its support is higher than a given threshold *minsup*; this is written as $\mathit{support}(I) \geq \mathit{minsup}$ (or, equivalently, $\mathit{freq}(I) \geq \mathit{minfreq}$).

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 $\mathit{freq}_c(I)$ 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 $c(I) = argmax_{c' \in C} \mathit{freq}_{c'}(I)$ 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 (Quinlan, 1993). 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 :

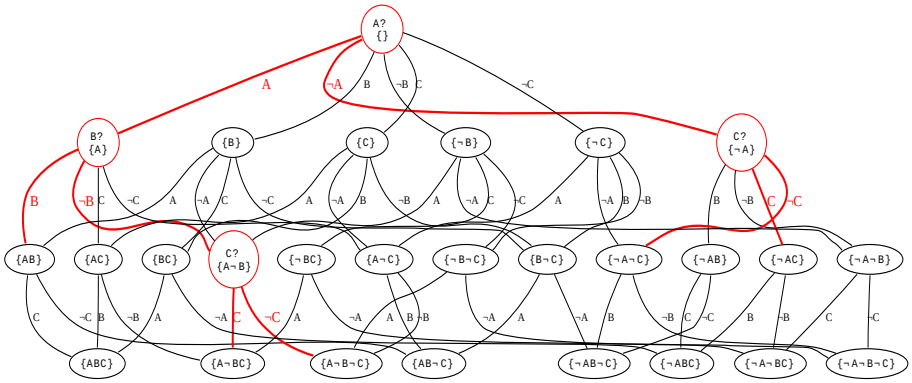


FIG. 1 – An itemset lattice for items $\{A, \neg A, B, \neg B, C, \neg C\}$; binary decision tree A(B(C(l,l),l),C(l,l)) is hidden in this lattice

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 a set of items occurring in D.

For example, consider the decision tree in Figure 4. 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 this set of items with *leaves*(T). Similarly, the itemsets that correspond to paths in the tree are denoted with *paths*(T). In this case, *paths*$(T) = \{\emptyset, \{B\}, \{\neg B\}, \{\neg B, C\}, \{\neg B, \neg C\}\}$. 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

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

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 the figure, A(L,R) denotes a binary decision tree where A is the attribute at the root node and L, R are the left and right children of the tree. *l* denotes a leaf.

From the theory of frequent itemset mining, it is known that itemsets form a *lattice* (these are typically depicted as in Figure 1). 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. The first part specifies the constraints on the nodes of the decision trees.

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

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, $p(I) := (\textit{freq}(I) \geq minfreq)$. 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 itemsets, 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 predefined estimate ;
- $\textit{size}(T)$, to constrain the number of nodes in a tree ;
- $\textit{depth}(T)$, 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* ; $r_i \in \{e, ex, \textit{size}, \textit{depth}\}$. 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 $\textit{depth}(T) \leq \textit{maxdepth}$ (respectively $\textit{size}(T) \leq \textit{maxsize}$).

We do not constrain the order of $\textit{size}(T)$, $e(T)$ and $\textit{depth}(T)$ 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)

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

In other words, we have $p(T) := (\mathit{freq}(I) \geq \mathit{minfreq})$, $q(T) := \mathit{true}$ and $r(T) := [e(T), \mathit{size}(T)]$. 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. To retrieve *Accurate Trees of Bounded Size*, Query 1 can be transformed such that $q(T) := \mathit{size}(T) \leq \mathit{maxsize}$.

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)

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

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 $x(\mathit{freq}_1(I), \ldots \mathit{freq}_n(I))$ for each leaf I of the decision tree, from which we can derive a new estimated number of errors

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

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

Query 3 (Small Accurate Pruned Trees)

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

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.

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	$40\times$
1	1	1	1	$40\times$
1	0	1	1	$5\times$
0	0	0	0	$10\times$
0	0	1	1	$5\times$

FIG. 2 – Database 1

A	B	C	Class	#
1	1	1	1	$30\times$
1	1	0	0	$20\times$
0	1	0	0	$8\times$
0	1	1	0	$12\times$
0	0	0	1	$12\times$
0	0	1	0	$18\times$

FIG. 3 – Database 2

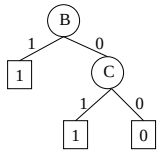
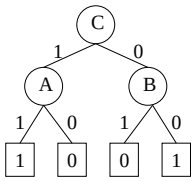
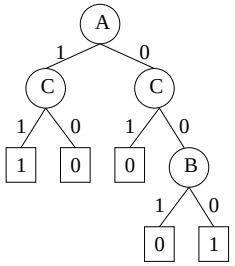


FIG. 4 – An example tree



(a) Smallest



(b) Learned using heuristics

FIG. 5 – Two accurate trees for database 2

As a first example, consider the database in Figure 2, in which we have 2 target classes. Assume that we are interested in answering Query 1 with *minfreq* $= 10$. An optimal tree exists (see Figure 4), 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). The tree that is found by C4.5 contains a single test instead of two, as the examples that contain $\{\neg A\}$ cannot be split further without violating the constraints.

As a second example, consider the database in Figure 3, 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 already noticed by (Page & Ray, 2003). We can also see in these examples that the smallest most accurate tree is not necessarily smaller or larger than the tree found by C4.5.

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.

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 $r_1, \ldots, r_{k-1} \in \{\textit{depth}, \textit{size}\}$ 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 $\textit{size}(T) \leq s$ and $\textit{depth}(T) \leq d$.

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 $\textit{pure}(I)$; predicate *pure* blocks the recursion if all examples $t(I)$ belong to the same class.

The algorithm is most easily understood if $\textit{maxdepth} = \infty$, $\textit{maxsize} = \infty$, $\textit{maxerror} = \infty$ 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 26.

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 1-2) the valid ranges of sizes and depths are computed here if a size or depth constraint was specified;

(line 4) 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 11) a candidate decision tree for classifying the examples $t(I)$ consists of a single leaf.

(line 12) 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.

(line 15) 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 $p(I \cup \{i\}) = \textit{false}$ or $p(I \cup \{\neg i\}) = \textit{false}$.

(line 14–25) 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 $\textit{size}(T_1)$ and depth at most $\textit{depth}(T_1)$; 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

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

```
1: if maxsize ≠ ∞ then ← {1, 2, ..., maxsize} else S ← {∞}
2: if maxdepth ≠ ∞ then D ← {1, 2, ..., maxdepth} else D ← {∞}
3: 𝒯 ←DL8-RECURSIVE(∅)
4: if maxerror ≠ ∞ then {𝒯 ← {T|T ∈ 𝒯, e(T) ≤ maxerror}
5: if 𝒯 = ∅ then return undefined
6: return argmin_{T∈𝒯} r(T)
7:
8: procedure DL8-RECURSIVE(I)
9:     if DL8-RECURSIVE(I) was computed before then
10:        return stored result
11:    𝒞 ← {l(c(I))}
12:    if pure(I) then
13:        store 𝒞 as the result for I and return 𝒞
14:    for all i ∈ ℐ do
15:        if p(I ∪ {i}) = true and p(I ∪ {¬i}) = true then
16:            𝒯_1 ← DL8-RECURSIVE(I ∪ {i})
17:            𝒯_2 ← DL8-RECURSIVE(I ∪ {¬i})
18:            for all T_1 ∈ 𝒯_1, T_2 ∈ 𝒯_2 do
19:                𝒞 ← 𝒞 ∪ {n(i, T_1, T_2)}
20:        end if
21:    𝒯 ← ∅
22:    for all d ∈ D, s ∈ S do
23:        ℒ ← {T ∈ 𝒞|depth(T) ≤ d ∧ size(T) ≤ s}
24:        𝒯 ← 𝒯 ∪ {argmin_{T∈ℒ}[r_k = e_{t(I)}(T), ..., r_n(T)]}
25:    end for
26:    store 𝒯 as the result for I and return 𝒯
27: end procedure
```

constraints are found by DL8-RECURSIVE($I \cup \{i\}$) and DL8-RECURSIVE($I \cup \{\neg i\}$) as $size(T_1), size(T_2) \leq$ *maxsize* and $depth(T_1), depth(T_2) \leq$ *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 19.

A key feature of DL8-RECURSIVE is that in line 26 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.

As with most data mining algorithms, the most time consuming operations are those that access the data. Different strategies can be considered to obtain the frequency counts that are necessary to check the constraints and compute accuracies. The most straightforward approach, referred to as DL8-SIMPLE, would compute the item frequencies while DL8 is executing. In this case, once DL8-RECURSIVE is called for an

itemset I, the frequencies of I are obtained by scanning the data. We can then store the result to avoid later recomputations. Another, less naive, strategy could be used, 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 in a preprocessing step. DL8 can then operate 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 modifi cations to output the frequent itemsets in a convenient form and determine frequencies in multiple classes (Agrawal *et al.*, 1996; Zaki *et al.*, 1997). If we assume that the output of the frequent itemset miner consists of a graph structure such as Figure 1, then DL8 can operate in time linear in the number of edges of this graph.

Early experiments showed that these approaches were not effi cient enough to build the lattice in reasonable runtimes for suffi ciently low supports. We provide now a new approach to compute more effi ciently the information needed to retrieve the trees.

5.1 The Constrained Frequent Itemset Mining Approach

The key observation is that the frequent itemset miners 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 theorems in this section can be found in (Nijssen & Fromont, 2007).

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 prefi xes $I' = [i_{k_1}, i_{k_2}, \ldots, i_{k_m}]$ ($m < n$) of this order :

– the $\neg$*pure*(I') predicate is false in line (12) ;
– the conjunction $p(I' \cup \{i_{k_{m+1}}\}) \wedge p(I' \cup \{\neg i_{k_{m+1}}\})$ is false in line (15).

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 defi nes 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 defi ned by*

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

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

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 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. Consequently, frequent itemset miners that incorporate two additional constraints can be used to obtain exactly those itemsets that are necessary for building decision trees.

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 fulfills the constraints.

5.2 The Closure-Based Direct 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 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 DL8-SIMPLE, we could index the results on $i(t(I))$ instead of I itself.

We incorporate this observation as follows in Algorithm 1. In line 9, we first compute the closure I' in the data. We use this closure to check if DL8-RECURSIVE(I) was already computed earlier by searching for I' in a trie data structure. In line 26, we associate the result to I' instead of I itself.

Our single-step approach which relies on closed itemset indexing is called DL8-CLOSED. 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. More details about this implementation can be found in (Nijssen & Fromont, 2007).

6 Experiments

Experiments that compare the efficiency of the different implementations of DL8 and other itemset miners can be found in (Nijssen & Fromont, 2007). Our investigations showed that high runtimes are not as much a problem as the amount of memory required for storing huge amounts of itemsets. DL8-CLOSED proved to be the most efficient of our implementations and was used for the following experiments.

We compared the decision trees learned by DL8 with the trees learned by C4.5. In particular, we want to answer these questions for exhaustively determined trees :

- how well do they generalize to test data ?
- do the results of Quinlan & Cameron-Jones (1995) that oversearching and overfitting are orthogonal also apply to constrained decision trees ?
- do the results of Murphy & Pazzani (1997) and Provost & Domingos (2003) that small decision trees are not always desirable in terms of predictive accuracy, also apply to constrained decision trees ?

In our experiments, we used J48, which is the Java implementation of *C4.5* (Quinlan, 1993) in WEKA (Witten & Frank, 2005). The experiments shown in Figure 1 were performed on 20 UCI datasets (Newman *et al.*, 1998). The three first columns of the table give a brief description of the datasets in terms of the number of examples and the number of attributes after binarization. Numerical data were discretized before applying our mining algorithms. We used Weka's unsupervised discretization method with a number of bins equal to 4. We limited the number of bins in order to reduce the number of created attributes. We used a stratified 10-fold cross-validation to compute the training and test accuracies of both systems.

In these experiments, we used minimum frequency as the local constraint. We lowered the minimum 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). Both algorithms were applied with the same minimum frequency setting ; for J48 results are also provided for its default $minfreq = 2$ setting. We used a corrected two-tailed t-test (Nadeau & Bengio, 2003) 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.

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. On the test data, in both cases, DL8 is significantly better than J48 on 9 of the 20 datasets ; only on 1 dataset is the result significantly worse. The experiments also show that the pruned trees computed by DL8 are on average 1.5 times larger than those computed by J48 with pruning. In one case (*vehicle*), the test accuracy result of

					Unpruned						Pruned						Pruned Freq = 2	
	Number		Freq		Train acc		Test acc		Size		Train acc		Test acc		Size		Test acc	size
Datasets	Ex	Att	#	%	J48	DL8	J48	DL8	J48	DL8	J48	DL8	J48	DL8	J48	DL8	J48	J48
anneal	812	55	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		
anneal	812	55	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	0.82	44.4
a-credit	653	75	50	7.6	0.87	0.88	0.85	**0.87**	5.0	9.8	0.86	0.88	0.86	0.87	3.0	9.8	0.84	36.4
balance	625	25	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		
balance	625	25	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		
breast-w	683	55	40	5.8	0.93	0.97	0.93	**0.95**	3.4	9.6	0.93	0.97	0.93	**0.95**	3.4	7.6		
breast-w	683	55	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	0.96	15.6
chess	3196	42	200	6.2	0.91	0.91	0.91	0.90	9.0	13	0.91	0.95	0.90	**0.95**	8.6	13.0	**0.99**	54.4
diabetes	768	49	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		
diabetes	768	49	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	0.74	69
g-credit	1000	89	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		
g-credit	1000	89	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	0.71	163
heart-c	296	55	30	10.1	0.77	0.84	0.74	0.77	4.4	11.8	0.77	0.84	0.73	**0.78**	3.6	11.8		
heart-c	296	55	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	0.78	31.6
ionosph	351	196	50	14.2	0.83	0.86	0.79	**0.84**	4.0	7.4	0.83	0.86	0.79	**0.84**	4	6.8	0.86	34.6
mushro	8124	119	800	9.8	0.92	0.97	0.92	**0.97**	5.0	11.0	0.92	0.97	0.92	**0.97**	5.0	11.0	**1.0**	16.8
pendigits	7494	97	800	10.6	0.51	0.67	0.50	**0.68**	11.6	15	0.51	0.67	0.51	**0.67**	11.6	15.0	**0.95**	340
p-tumor	336	32	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		
p-tumor	336	32	2	0.5	0.63	0.71	0.40	0.36	116.4	152.2	0.60	0.67	0.40	0.40	81.2	105.2		
segment	2310	108	300	12.9	0.55	0.72	0.55	**0.72**	7.0	11.0	0.55	0.72	0.55	**0.72**	7.0	11.0		
segment	2310	108	200	8.6	0.72	0.83	0.73	**0.83**	12.6	15.0	0.73	0.83	0.73	**0.83**	12.6	15.0	**0.95**	112.6
soybean	630	51	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		
soybean	630	51	60	9.5	0.51	0.55	0.50	**0.55**	13.0	15.0	0.51	0.55	0.50	**0.55**	11.2	15.0		
soybean	630	51	50	7.9	0.55	0.59	0.52	**0.59**	14.6	16.8	0.55	0.59	0.51	**0.58**	14.2	16.8	**0.82**	88
splice	3190	3466	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	3247	46	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	846	109	100	11.8	0.60	0.64	0.58	**0.63**	10.8	10.0	0.60	0.64	0.57	**0.63**	11.0	9.0	**0.70**	138
vote	435	49	20	4.5	0.96	0.97	**0.96**	0.94	3.0	15.0	0.96	0.96	0.96	0.94	3.0	7.6		
vote	435	49	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	0.96	12.6
vowel	990	80	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		
vowel	990	80	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	**0.78**	290
yeast	1484	42	100	6.7	0.53	0.55	0.50	**0.53**	13.8	15.4	0.53	0.55	0.51	0.53	11.4	13.8		
yeast	1484	42	10	0.6	0.61	0.67	0.52	0.50	107.2	171.0	0.60	0.65	**0.54**	0.52	56.0	104.6		
yeast	1484	42	2	0.1	0.74	0.82	0.49	0.48	501.2	724.2	0.68	0.75	**0.53**	0.50	186.0	307.2	**0.53**	186.0

TAB. 1 – Comparison of J48 and DL8 accuracies and size with and without pruning

DL8 is significantly better than the one given by J48 for a smaller tree. 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. These results confirm earlier findings which show that smaller trees are not always desirable; under the same setting of constraints, we also find that it is not harmful at all to search exhaustively.

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 (Garey, 1972; Meisel & Michalopoulos, 1973; Payne & Meisel, 1977; Schumacher & Sevcik, 1976; Lew, 1978). 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 (Kearns & Mansour, 1999; Dietterich *et al.*, 1996; Nock & Nielsen, 2004). Still, the idea of exhaustively finding optimal decision trees under certain constraints was also studied (Auer *et al.*, 1995; Murphy & Pazzani, 1997), but only for much smaller datasets and smaller types of trees than studied in this paper. Recently Blanchard *et al.* (2007) 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 smaller datasets than our algorithm, and did not investigate the link with data mining algorithms.

Algorithmically, the $tree$-relevancy constraint is closely related to the condensed representation of δ-free itemsets (Boulicaut *et al.*, 2003). Indeed, for $\delta = \mathit{minsup} \times |D|$ and $p(I) := (\mathit{freq}(I) \geq \mathit{minfreq})$, it can be shown that if an itemset is δ-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 (Pasquier *et al.*, 1999).

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, Yan *et al.* (2005)). 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. 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. Furthermore, since memory requirement seems to be the most challenging problem, new condensed representations could be developed to represent the information that is used by DL8 more compactly.

In the 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 signifi cantly more accurate than trees learned by C4.5. When we compare the best settings of both algorithms, J48 performs signifi cantly better in 45% of the datasets. Effi ciency 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 (Page & Ray, 2003; Provost & Domingos, 2003).

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 (Imielinski & Mannila, 1996) that contain both patterns and data.

Références

AGRAWAL R., MANNILA H., SRIKANT R., TOIVONEN H. & VERKAMO A. I. (1996). Fast discovery of association rules. In *Advances in Knowledge Discovery and Data Mining*, p. 307–328. AAAI/MIT Press.

AUER P., HOLTE R. C. & MAASS W. (1995). Theory and applications of agnostic pac-learning with small decision trees. In *ICML*, p. 21–29.

BLANCHARD G., SCHÄFER C., ROZENHOLC Y. & MÜLLER K. R. (2007). Optimal dyadic decision trees. *Mach. Learn.*, **66**(2-3), 209–241.

BOULICAUT J.-F., BYKOWSKI A. & RIGOTTI C. (2003). Free-sets : A condensed representation of boolean data for the approximation of frequency queries. *Data Min. Knowl. Discov.*, **7**(1), 5–22.

DIETTERICH T. G., KEARNS M. J. & MANSOUR Y. (1996). Applying the weak learning framework to understand and improve c4.5. In *ICML*, p. 96–104.

FROMONT E., BLOCKEEL H. & STRUYF J. (2007). Integrating decision tree learning into inductive databases. In *Revised selected papers of the workshop KDID'06*,LNCS Springer, p. to appear : .

GAREY M. R. (1972). Optimal binary identifi cation procedures. *SIAM Journal of Applied Mathematics*, **23**(2), 173–186.

HYAFIL L. & RIVEST R. L. (1976). Constructing optimal binary decision trees is np-complete. *Inf. Process. Lett.*, **5**(1), 15–17.

IMIELINSKI T. & MANNILA H. (1996). A database perspective on knowledge discovery. *Comm. Of The Acm*, **39**, 58–64.

KEARNS M. J. & MANSOUR Y. (1999). On the boosting ability of top-down decision tree learning algorithms. *J. Comput. Syst. Sci.*, **58**(1), 109–128.

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

MEISEL W. S. & MICHALOPOULOS D. (1973). A partitioning algorithm with application in pattern classifi cation and the optimization of decision tree. *IEEE Trans. Comput.*, **C-22**, 93–103.

MURPHY P. M. & PAZZANI M. J. (1997). *Exploring the decision forest : an empirical investigation of Occam's razor in decision tree induction*, In *Computational Learning Theory and Natural Learning Systems*, volume IV : Making Learning Systems Practical, p. 171–187. MIT Press.

NADEAU C. & BENGIO Y. (2003). Inference for the generalization error. *Mach. Learn.*, **52**(3), 239–281.

NEWMAN D., HETTICH S., BLAKE C. & MERZ C. (1998). UCI repository of machine learning databases.

NIJSSEN S. & FROMONT E. (2007). Mining optimal decision trees from itemset lattices. Technical report CW476, Dept. Computer Science, Katholieke Universiteit Leuven, March 2007.

NOCK R. & NIELSEN F. (2004). On domain-partitioning induction criteria : worst-case bounds for the worst-case based. *Theor. Comput. Sci.*, **321**(2-3), 371–382.

PAGE D. & RAY S. (2003). Skewing : An effi cient alternative to lookahead for decision tree induction. In *Proceedings of the Eighteenth International Joint Conference on Artificial Intelligence (IJCAI'03)*, p. 601–612.

PASQUIER N., BASTIDE Y., TAOUIL R. & LAKHAL L. (1999). Effi cient mining of association rules using closed itemset lattices. *Information Systems*, **24**(1), 25–46.

PAYNE H. J. & MEISEL W. S. (1977). An algorithm for constructing optimal binary decision trees. *IEEE Trans. Computers*, **26**(9), 905–916.

PROVOST F. & DOMINGOS P. (2003). Tree induction for probability-based ranking. *Machine Learning*, **52**, 199–215.

QUINLAN J. R. (1993). *C4.5 : Programs for Machine Learning*. Morgan Kaufmann.

QUINLAN J. R. & CAMERON-JONES R. M. (1995). Oversearching and layered search in empirical learning. In *IJCAI*, p. 1019–1024.

SCHUMACHER H. & SEVCIK K. C. (1976). The synthetic approach to decision table conversion. *Commun. ACM*, **19**(6), 343–351.

WITTEN I. H. & FRANK E. (2005). *Data Mining : Practical machine learning tools and techniques*. San Francisco : Morgan Kaufmann, 2nd edition edition.

YAN X., CHENG H., HAN J. & XIN D. (2005). Summarizing itemset patterns : a profi le-based approach. In *Proceeding of the 11th ACM SIGKDD international conference on Knowledge discovery in data mining (KDD'05)*, p. 314–323.

ZAKI M. J., PARTHASARATHY S., OGIHARA M. & LI W. (1997). *New Algorithms for Fast Discovery of Association Rules*. Rapport interne TR651.