Learning Optimal Decision Trees using Constraint Programming (Extended Abstract*)

Hélène Verhaeghe[1†], **Siegfried Nijssen**[1], **Gilles Pesant**[2], **Claude-Guy Quimper**[3] and **Pierre Schaus**[1]

[1] UCLouvain, ICTEAM, Place Sainte Barbe 2, 1348 Louvain-la-Neuve, Belgium

[2] Polytechnique Montréal, Montréal, Canada

[3] Université Laval, Québec, Canada

{helene.verhaeghe,siegfried.nijssen,pierre.schaus}@uclouvain.be, gilles.pesant@polymtl.ca, claude-guy.quimper@ift.ulaval.ca

Abstract

Decision trees are among the most popular classification models in machine learning. Traditionally, they are learned using greedy algorithms. However, such algorithms have their disadvantages: it is difficult to limit the size of the decision trees while maintaining a good classification accuracy, and it is hard to impose additional constraints on the models that are learned. For these reasons, there has been a recent interest in exact and flexible algorithms for learning decision trees. In this paper, we introduce a new approach to learn decision trees using constraint programming. Compared to earlier approaches, we show that our approach obtains better performance, while still being sufficiently flexible to allow for the inclusion of constraints. Our approach builds on three key building blocks: (1) the use of AND/OR search, (2) the use of caching, (3) the use of the CoverSize global constraint proposed recently for the problem of itemset mining. This allows our constraint programming approach to deal in a much more efficient way with the decompositions in the learning problem.

1 Introduction

Decision trees are popular classification models in machine learning. Benefits of decision trees include that they are relatively easy to interpret and that they provide good classification performance on many datasets.

Several methods have been proposed in the literature for learning decision trees. The greedy methods are the most popular ones. These methods recursively partition a dataset into two subsets based on a greedily selected attribute until some stopping criterion is reached (such as a minimum number of examples in a leaf, or a unique class label in these examples). While in practice these methods obtain a good prediction accuracy for many types of data, unfortunately, they provide little guarantees. As a result, the trees learned using these methods may be unnecessarily complex, may be less accurate than possible, and it is hard to impose additional constraints on the trees.

To address these weaknesses, researchers have studied the inference of *optimal* decision trees under constraints [Nijssen and Fromont, 2007; Bertsimas and Dunn, 2017; Aglin *et al.*, 2020]. These approaches ensure that under well-defined constraints and optimization criteria, an optimal tree is found.

Our paper proposes a new, more scalable approach based on Constraint Programming (CP) for learning decision trees with a fixed maximum depth minimizing the classification error. Our approach combines these key ideas: the use of branch-and-bound in a CP solver, the use of the CoverSize global constraint [Schaus *et al.*, 2017], the use of an AND/OR search tree [Dechter and Mateescu, 2007] and the use of caching as introduced in DL8 [Nijssen and Fromont, 2007]. This allows our constraint programming approach to deal in a much more efficient way with the decompositions in the learning problem. We will show that the combination of these different ideas leads to a model that is more efficient than other approaches proposed in the literature.

Moreover, the fact that our approach is integrated in a general constraint programming system means that it is relatively easy to extend the approach with other constraints in the future.

2 Related Work

Related to this work are the alternative approaches for finding optimal decision trees. There are a number of alternative definitions for the problem of finding optimal decision trees, each using different constraints and optimization criteria.

The most popular setting studied in recent papers [Aghaei *et al.*, 2019; Bertsimas and Dunn, 2017; Verwer and Zhang, 2019; Aglin *et al.*, 2020] is the one in which a decision tree of bounded depth is learned by maximizing the accuracy on a given training dataset. The limit on depth allows modeling the problem as a MIP problem with a fixed number of variables. Constraints can be added, as long as they are linear; this includes constraints on fairness [Aghaei *et al.*, 2019] or

*This paper is an extended abstract of *Learning Optimal Decision Trees using Constraint Programming* presented at The 25th International Conference on Principles and Practice of Constraint Programming (CP2019) [Verhaeghe *et al.*, 2019].

†Contact Author

on the number of examples in the leafs [Bertsimas and Dunn, 2017]. We will use this problem setting in this work.

A slightly different setting was studied in the DL8 algorithm [Nijssen and Fromont, 2010]. DL8 builds on top of itemset mining algorithms to find decision tree paths, and uses dynamic programming to build a decision tree from these paths. Effectively, it uses itemsets as the key of a caching data structure. As a consequence of the use of itemset mining, DL8 does not require a specific constraint on the depth of the decision tree; it uses a minimum support constraint to limit the size of the search space. This approach can be used on constraints that are not linear in nature. From this approach, we will adapt its link to itemset mining, and its use of caching.

To the best of our knowledge, CP has not yet been used in the setting where accuracy is optimized. Two earlier studies [Bessiere *et al.*, 2009; Narodytska *et al.*, 2018] did, however, study the setting in which one finds the smallest decision tree consistent with a training dataset (i.e. the error of the decision tree has to be zero). As training data can be noisy and inconsistent, and hence finding a tree of zero error can be either impossible or undesirable, this setting is less common in the machine learning literature.

Similar to DL8, we will rely in this work on the fact that decision tree learning problems have many decompositions. We will exploit these using AND/OR search, which was studied extensively by Dechter et al. [Dechter and Mateescu, 2004]. AND/OR search is not common in CP systems yet, and has not been used in decision tree learning yet; it has recently been exploited in the context of stochastic CP however [Babaki *et al.*, 2017].

3 Technical Background

3.1 Definition of the Problem

To simplify the exposition we restrict our attention to binary data. Continuous data can be discretized and binarized as proposed by Breiman et al. [Breiman, 1984]; this observation was also exploited in earlier studies [Nijssen and Fromont, 2007; Verwer and Zhang, 2019].

We represent our data using an $n \times m$ binary matrix D. D_i represents the ith row of the data, or, following itemset mining terminology, the ith *transaction* of D. The number of transactions is thus n. The columns of the matrix represent the m *features* or *items* of the transactions. We assume in this work that each transaction belongs to one of two classes, represented by 0 and 1. The classes are stored in a vector v of size n. Hence, the database can be split into D^+, a matrix of size $n^+ \times m$, containing all the transactions from D associated to class 1, and D^-, a matrix of size $n^- \times m$, containing the ones associated to class 0.

In this work we are interested in finding decision trees. Each internal node w of a decision tree is associated to a feature (called the decision of the node) $d[w] \in \{1, \ldots, m\}$; each leaf is associated to a Boolean $b[w]$, representing the prediction for that leaf. We will use the function $F(r, t)$ to represent the predicted class for transaction t on a tree with root r, defined recursively as

$$F(w,t) = \begin{cases} b[w] & \text{if } w \text{ is a leaf;} \\ F(\text{left}(w), t) & \text{if } D_{t,d[w]} = 1; \\ F(\text{right}(w), t) & \text{if } D_{t,d[w]} = 0. \end{cases} \quad (1)$$

Here $\text{left}(w)$ (resp. $\text{right}(w)$) returns the left-hand (resp. right-hand) subtree of node w.

We define the *depth* of a decision tree to be the maximum number of features on any path from the root of the tree towards a leaf. Given a maximum depth, our goal is to find a decision tree that minimizes the number of misclassified transactions (i.e. transactions t where $v[t] \neq F(r, t)$):

$$\min \sum_{t=1}^{n} [F(r,t) \neq v[t]]. \quad (2)$$

We allow for the additional specification of a constraint on the minimum number of examples $N_{\min}$ in each leaf of the tree [Bertsimas and Dunn, 2017; Nijssen and Fromont, 2010].

An extension of the problem is to consider more than two classes (multi-class decision trees). We will limit our discussion to binary classes, but the extension towards data with more than two classes is relatively straightforward.

3.2 The CoverSize Constraint

To determine the accuracy of a decision tree, we need to decide in which nodes of the decision tree a transaction ends up. A correspondence can be drawn here with the *cover* of itemsets in itemset mining [Nijssen and Fromont, 2007; Nijssen and Fromont, 2010]. We exploit this correspondence by adapting the CoverSize global constraint [Schaus *et al.*, 2017] to the context of learning decision trees. The original CoverSize has the following parameters: an array of Boolean variables (one variable for each feature), the database, and a counter variable, and is defined as follows:

$$\text{CoverSize}([I_1, \ldots, I_m], D, c) \iff c = \left| \bigcap_{I_i=1} \{t \in \{1, \ldots, n\} \mid D_{t,i} = 1\} \right|$$

The goal of the constraint is to link an *itemset* to the number of transactions containing the itemset. The itemset is represented by the Boolean array $[I_1, \ldots, I_m]$: Boolean I_i is true if and only if feature i is included in the itemset. A transaction contains an itemset if and only if every feature in the itemset has value 1 in the transaction.

In decision trees, we need to be able to test whether an item is absent in a transaction. Also, we know in advance how many decisions are at most selected for each leaf. For these reasons we needed to adapt the standard CoverSize constraint. The adaptation, called CoverSizeSR, takes two sparse sets of decisions (features), a counter variable and a database as input. The two sets represent the sets of features respectively required to be included in and excluded from a transaction.

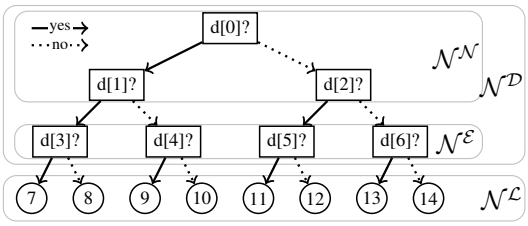
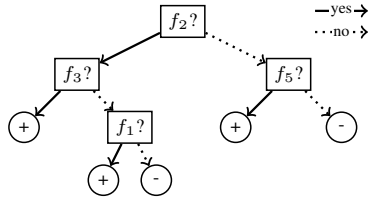
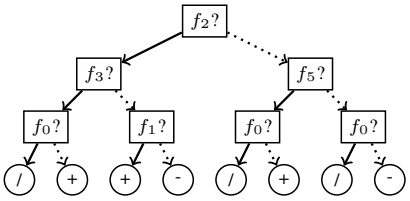


Figure 1: Representation of a perfect decision tree of depth 3



(a) Proper binary tree



(b) Equivalent perfect binary tree

Figure 2: Example of the use of the dummy feature f_0 to transform the proper binary tree into a perfect binary tree

4 CP Modeling of the Problem

4.1 Model of the Problem

In this section, we introduce the variables and constraints used in our model. Fig. 1 shows a visualization of our model for trees of a maximum depth of 3.

Note that in our model we assume that a decision tree is a perfect tree. This assumption is motivated by the existence of a mapping of any proper binary tree (i.e. a tree where each node has exactly 0 or 2 children) into a perfect one (i.e. a proper binary tree with all the leaves at the same level): we add a dummy feature f_0, not belonging to any of the transactions, to the model for unused decision nodes. A node with this value therefore has no transaction from the database that ends up in its left branch. Figure 2 shows how a proper tree can be made perfect by the use of the dummy feature.

The nodes ($\mathcal{N}$) of a perfect decision tree can be partitioned into two groups: the decision nodes ($\mathcal{N}^{\mathcal{D}}$), which are associated to a decision and which have children, and the leaves ($\mathcal{N}^{\mathcal{L}}$), which do not have children. The decision nodes ($\mathcal{N}^{\mathcal{D}}$) can be further partitioned into the end-nodes $\mathcal{N}^{\mathcal{E}}$, which do not have decision nodes as children, and the nodes $\mathcal{N}^{\mathcal{N}}$, which do. Variables and constraints are defined accordingly.

In our model, the number of variables and constraints are independent from the number of transactions in the database and the number of features. In fact, the number of variables and constraints only depend on the number of nodes in the tree.

Variables

In our model we have variables with the following domains. Each decision node has a decision variable d to model the decision feature. Its value can be 0 (representing the dummy feature f_0) or between 1 and m (representing one of the actual features f_1 to f_m). Two counters, c^+ and c^-, are defined for each node of the tree. They are used to keep track of how many transactions respectively from D^+ and D^- match the decisions of the ancestors of the node. A third counter c, defined at the leaves, tracks the total number of transactions. The minimum number of transactions in each leaf is enforced by constraining the domain of c from $N_{\min}$ to $|D|$. Value 0 also belongs to the domain and is meant to be used only when the parent of the node is inactive (i.e. when its decision is f_0). An additional variable e, defined for each node, keeps track of the error of the sub-tree rooted at that node. Our model does not have an explicit variable for the class of the leaves. However, this can be easily deduced from the solution by taking the class associated with the highest counter.

Constraints

On these variables, we define the following constraints.

First, $\forall i \in \mathcal{N}^{\mathcal{L}}$, constraint $c^+[i] + c^-[i] = c[i]$ links the counters. Second, $\forall i \in \mathcal{N}^{\mathcal{D}}$, the counters at the decision nodes are linked to the counters of their children ($c^+[i] = c^+[\text{left}(i)] + c^+[\text{right}(i)]$, $c^-[i] = c^-[\text{left}(i)] + c^-[\text{right}(i)]$). Third, the value of $e[i]$ is defined, $\forall i \in \mathcal{N}^{\mathcal{L}}$, to be the minimum between the class counters ($e[i] = \min\{c^+[i], c^-[i]\}$) and, $\forall i \in \mathcal{N}^{\mathcal{D}}$, to be the sum of the errors from the children of i ($e[i] = e[\text{left}(i)] + e[\text{right}(i)]$). To compute the values of the counters, we need to know which transactions match the decisions of the ancestors of the leaf. To this end, $\forall i \in \mathcal{N}^{\mathcal{L}}$, two CoverSizeSR global constraints, are added, one for each class (D^+ and D^-). The decision variables of the ancestors (an ancestor is either the parent of the node or the parent of an ancestor) are divided into the two required disjoint sets.

The next two constraints ensure the decision tree has no useless node. A node is useless if the decision taken in it was already taken in one of the ancestor nodes. An AllDifferentExcept0 is used at each end-node ($\forall i \in \mathcal{N}^{\mathcal{E}}$) on the ancestors and the end-node to avoid this. A node is also useless if all the leaves below have the same class. This is avoidable if we constrain the error at the node to be strictly higher than the error of the subtree ($d[i] \neq 0 \Rightarrow \min\{c^+[i], c^-[i]\} > e[i]$, $\forall i \in \mathcal{N}^{\mathcal{D}}$). Finally, when a decision node is inactive, all the decision nodes below should be inactive as well ($d[i] = 0 \Rightarrow (d[\text{left}(i)] = 0 \wedge d[\text{right}(i)] = 0)$, $\forall i \in \mathcal{N}^{\mathcal{N}}$).

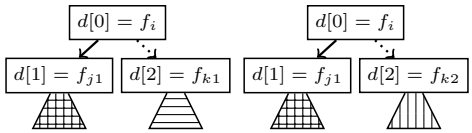
These constraints are enough to guarantee an optimal, well-formed tree (with no dummy decision feature being a parent from a non-dummy decision and with no decision leading to only one classification).

Objective

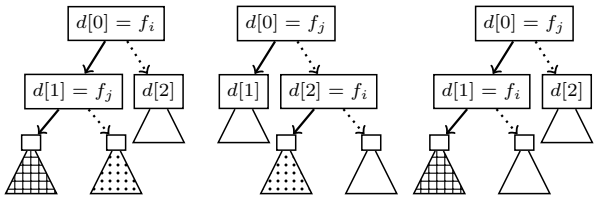
The objective is to minimize the sum of the errors at the leaves, which is stored in $e[\text{root}]$.

Redundant Constraints

We add a number of redundant constraints to make the search more efficient.



(a) Independence of subtrees



(b) Redundant subtrees; identically shaded subtrees are identical

Figure 3: Decompositions

4.2 Search

The motivation behind the use of a specific search strategy is to exploit the tree-decomposition into subproblems. During search each node of the search tree is associated to a subtree of the decision tree being built. This subtree, identified by the node id `currProblem`, is always rooted on a decision node. The assignment of the decision variables occurs in top-down fashion. Therefore in a given node of the search tree, we can always assume every node in $ancestors($`currProblem`$)$ has been assigned.

Big Picture

Our search combines three techniques: AND/OR search trees, branch-and-bound optimization, and memorization. Each of them aims to answer one of the specificities of the problem.

Subtree Independence

Given a subtree with its root decision assigned, its two children are totally independent from one another. Any solution from the left child combined with any solution from the right child leads to a solution of the initial subtree (Fig. 3a). However our goal is to find the best solution and not one solution. Moreover our objective function is the sum of a cost computed in each of the leaves, independently. Therefore, the optimal solution, given a root and ancestors' decisions already assigned, can be computed independently by computing the optimal left child, then the optimal right child and finally combine them. The AND/OR search tree [Dechter and Mateescu, 2007; Marinescu and Dechter, 2004] framework is well suited for this kind of decomposable problem. The search is composed of two types of search nodes: the OR nodes and the AND nodes.

The AND node is responsible for computing the optimal value of the left child, then the right child, and finally returns the composed solution. The OR node tests all the possible values for the root decision variable of `currProblem` and returns the best subtree.

Subtree Equality

Two subproblems are equivalent whenever the set of decisions on the paths towards these nodes (the itemsets corresponding to the sets of decisions) are identical. Figure 3b shows how some subtrees can be the same in two different solutions due to paths that represent the same itemset. This is taken care of by using a caching system similar to the one used in the DL8 dynamic programming approach [Nijssen and Fromont, 2007]. Two subtrees are equivalent if they share the same assigned prefix. The prefix of node i is composed of the values assigned to the decisions of the ancestors. A hash is computed from these decisions and serves as a key to store and retrieve the optimal subtree from storage (hashMap). In addition to the decision in the root of the subtree, its cost is also stored, easing the computation. The search for an already computed solution happens at the beginning of an OR node. A new solution is stored when a new complete optimal subtree is computed, i.e. at the end of the OR node.

Minimization

In order to decrease the number of explored search nodes, bound-based pruning is added to the search. At each of the search nodes, the upper bound of the allowed cost is propagated from node to node. During an OR node, this upper bound is decreased each time a better solution is found and the best cost found so far is set as upper bound of the error of the subtree. During an AND node, the propagated upper bound is first propagated to the computation of the first child. If the result of this first child is above this propagated upper bound, then there is no need to compute the right child since any solution would be above the propagated upper bound.

5 Results

We compared our methods to DL8 [Nijssen and Fromont, 2007] and BinOCT [Verwer and Zhang, 2019], two methods addressing the same problem. Our method outperforms these two others on most of the instances. It could find and prove optimality on roughly 75% of the instances within the time limit. The best solution found was reached by our method in almost all cases. However, DL8 performs better on small instances. The large difference between BinOCT and our method can be explained by the benefits of the AND/OR search that is not used by BinOCT. The gap with DL8 can be partially explained by the cost pruning. It can potentially also be explained by the itemset mining algorithms used: DL8 lacks the optimizations found in the CoverSize constraint. Our experiments also evaluate the effects of some of the techniques used to solve the problem, such as the use of a cache.

6 Conclusion

In summary, our paper presents a new approach for efficiently creating an optimal decision tree of limited depth. This approach based on CP combines the CoverSize global constraint, the concept of AND/OR tree and caching. On most of the benchmarks, it gives the best solution within the allocated time and is the fastest to prove optimality. We believe our approach can be extended in many different ways (e.g. multiclass, continuous features through binarization, side constraints). More details can be found in the original paper [Verhaeghe *et al.*, 2019] and in the extended Constraint Journal version (currently under review).

References

[Aghaei *et al.*, 2019] Sina Aghaei, Mohammad Javad Azizi, and Phebe Vayanos. Learning optimal and fair decision trees for non-discriminative decision-making. 2019.

[Aglin *et al.*, 2020] Gaël Aglin, Siegfried Nijssen, and Pierre Schaus. Learning optimal decision trees using caching branch-and-bound search. In *Thirty-Fourth AAAI Conference on Artificial Intelligence*, 2020.

[Babaki *et al.*, 2017] Behrouz Babaki, Tias Guns, and Luc De Raedt. Stochastic constraint programming with and-or branch-and-bound. In *Proceedings of the Twenty-Sixth International Joint Conference on Artificial Intelligence, IJCAI 2017, Melbourne, Australia, August 19-25, 2017*, pages 539–545, 2017.

[Bertsimas and Dunn, 2017] Dimitris Bertsimas and Jack Dunn. Optimal classification trees. *Machine Learning*, 106(7):1039–1082, 2017.

[Bessiere *et al.*, 2009] Christian Bessiere, Emmanuel Hebrard, and Barry O'Sullivan. Minimising decision tree size as combinatorial optimisation. In *International Conference on Principles and Practice of Constraint Programming*, pages 173–187. Springer, 2009.

[Breiman, 1984] Leo Breiman. *Classification and regression trees*. Routledge, 1984.

[Dechter and Mateescu, 2004] Rina Dechter and Robert Mateescu. The impact of AND/OR search spaces on constraint satisfaction and counting. In *Principles and Practice of Constraint Programming - CP 2004, 10th International Conference, CP 2004, Toronto, Canada, September 27 - October 1, 2004, Proceedings*, pages 731–736, 2004.

[Dechter and Mateescu, 2007] R. Dechter and R. Mateescu. And/or search spaces for graphical models. *Artificial intelligence*, 171(2-3):73–106, 2007.

[Marinescu and Dechter, 2004] Radu Marinescu and Rina Dechter. And/or tree search for constraint optimization. In *Proc. of the 6th International Workshop on Preferences and Soft Constraints*. Citeseer, 2004.

[Narodytska *et al.*, 2018] N. Narodytska, A. Ignatiev, F. Pereira, J. Marques-Silva, and IS RAS. Learning optimal decision trees with sat. In *IJCAI*, pages 1362–1368, 2018.

[Nijssen and Fromont, 2007] S. Nijssen and E. Fromont. Mining optimal decision trees from itemset lattices. In *Proceedings of the 13th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 530–539. ACM, 2007.

[Nijssen and Fromont, 2010] Siegfried Nijssen and Élisa Fromont. Optimal constraint-based decision tree induction from itemset lattices. *Data Min. Knowl. Discov.*, 21(1):9–51, 2010.

[Schaus *et al.*, 2017] P. Schaus, J. Aoga, and T. Guns. Coversize: a global constraint for frequency-based itemset mining. In *International Conference on Principles and Practice of Constraint Programming*, pages 529–546. Springer, 2017.

[Verhaeghe *et al.*, 2019] Hélène Verhaeghe, Siegfried Nijssen, Gilles Pesant, Claude-Guy Quimper, and Pierre Schaus. Learning optimal decision trees using constraint programming. In *The 25th International Conference on Principles and Practice of Constraint Programming (CP2019)*, 2019.

[Verwer and Zhang, 2019] Sicco Verwer and Yingqian Zhang. Learning optimal classification trees using a binary linear program formulation. In *33rd AAAI Conference on Artificial Intelligence*, 2019.