Learning Optimal Decision Trees with SAT

Nina Narodytska1, Alexey Ignatiev2,3, Filipe Pereira2, Joao Marques-Silva2
1 VMware Research, CA, USA
2 LASIGE, Faculdade de Ciências, Universidade de Lisboa, Portugal
3 ISDCT SB RAS, Irkutsk, Russia
nnarodytska@vmware.com, {aignatiev,jpms}@ciencias.ulisboa.pt, fcpereira@lasige.di.fc.ul.pt

Abstract

Explanations of machine learning (ML) predictions are of fundamental importance in different settings. Moreover, explanations should be succinct, to enable easy understanding by human decision makers. Decision trees represent an often used approach for developing explainable ML models, motivated by the natural mapping between decision tree paths and rules. Clearly, smaller trees translate directly to smaller rules, and so one challenge is to devise solutions for computing smallest size decision trees given training data. Although simple to formulate, the computation of smallest size decision trees turns out to be an extremely challenging computational problem, for which no practical solutions are known. This paper develops a SAT-based model for computing smallest-size decision trees given training data. In sharp contrast with past work, the proposed SAT model is shown to scale for publicly available datasets of practical interest.

1 Introduction

For a growing number of applications of Machine Learning (ML) and Data Mining (DM) there is a demand to associate explanations with the predictions made, such that these explanations can be interpreted by a human decision maker. The need for explanations associated with ML and DM models in different settings motivates the general area of explainable AI (XAI). The importance of XAI is illustrated by a range of work in the recent past [Li et al., 2018; Lakkaraju et al., 2017; Angelino et al., 2017; Lakkaraju et al., 2016; Letham et al., 2015; Lou et al., 2012] among others, by a number of recent events dedicated to the topic [IJCAI XAI Workshop, 2017; ICML WHI Workshop, 2017; NIPS IML Symposium, 2017], but also by recent research programs [DARPA, 2016]. [Biran and Cotton, 2017] provides a recent account of work on XAI. Furthermore, the relevance of XAI is implicit in recent EU-level regulations [EU Data Protection Regulation, 2016], which is motivated in part by a demand for greater transparency in algorithmic decision making [Goodman and Flaxman, 2017; Weller, 2017]. Although XAI can also be seen as controversial in some works [Doshi-Velez and Kim, 2017], it is also apparent that the demand for greater transparency in algorithmic decision making represents one of the main driving forces towards devising explainable ML models.

One approach to achieve explainable ML models is to use models that by construction provide explanations. This is the case with decision trees [Bessiere et al., 2009], but also with rule lists and rule sets [Lakkaraju et al., 2016; Angelino et al., 2017]. Decision trees have been actively investigated since the early 80s [Breiman et al., 1984; Quinlan, 1986; Quinlan, 1993; Mitchell, 1997; Cockett and Hierrera, 1990], and find important practical applications in a wide range of domains, including ML and DM [Quinlan, 1993; Murthy, 1998; Rokach and Maimon, 2015]. Decision trees can naturally associate explanations with predictions made, and so represent a natural choice in the general area of XAI. Indeed, decision trees are by construction capable of providing explanations for predictions made, which are often easy to comprehend by human decision makers, provided decision trees (and so their paths) are small in size. Thus, finding optimal (in size) decision trees can enable computing small-size explanations in XAI settings. We emphasize that by optimal decision trees we mean decision trees of the minimum size.

On the negative side, the construction of optimal decision trees is well-known to be NP-hard, for different notions of optimality [Hyafil and Rivest, 1976; Hancock et al., 1996]. As a result, practical algorithms for learning decision trees are heuristic [Breiman et al., 1984; Quinlan, 1986], with no guarantees in terms of size compared to the optimal. One recent exception [Bessiere et al., 2009], based on constraint programming (CP), enables learning optimal decision trees, but existing results provide only approximate solutions. The use of Boolean satisfiability (SAT) for computing optimal decision trees has been shown to scale only for small-size examples [Bessiere et al., 2009], and only for decision tree of fixed structure and size. Not surprisingly, it is generally believed that learning optimal decision trees is impractical except for very small examples [Rokach and Maimon, 2015].

This paper proposes a different take on learning optimal decision trees from examples. The paper identifies a number of fundamental properties of decision trees, which can be exploited when developing propositional models for learning decision trees from examples. Furthermore, the paper develops a novel SAT model for deciding the existence of a decision tree consistent with training data, enabling the learning
of optimal decision trees. The model is amenable to a number of improvements, but also to a number of variants. The experimental results, all obtained on well-known datasets, for which optimal decision trees were not known, show that the proposed approach can learn optimal decision trees for many representative datasets.

2 Preliminaries

2.1 Classification Problems

We follow the notation used in earlier work \cite{Bessiere2009}. We consider a set of features \( \mathcal{F} = \{ f_1, \ldots, f_K \} \), all of which are assumed to be binary, taking a value in \{0, 1\}. When necessary, the fairly standard one-hot-encoding is assumed for handling non-binary categorical features. Numeric features can be handled with standard techniques as well. Furthermore, we start from given training data (or examples) \( \mathcal{E} = \{ e_1, \ldots, e_M \} \). We consider binary classification, and so \( \mathcal{E} \) is partitioned into \( \mathcal{E}^+ \) and \( \mathcal{E}^- \), denoting the examples classified as positive and as negative, respectively. Moreover examples are assumed to be consistent, i.e. a set of features is either associated with the positive or negative clause, but not both. Extensions to this basic formulation, including non-binary features, handling of non-binary classes, and allowing for inconsistent examples are beyond the scope of this paper but are discussed in later sections. Furthermore, in this paper we assume that all features are specified for all examples; the work can be generalized for situations where the value of some features for some examples is left unspecified.

Since all features are binary, a literal on a feature \( f_r \) will be represented as \( f_r \), denoting that the feature takes value 1, i.e. \( f_r = 1 \), or \( \neg f_r \), denoting that the feature takes value 0, i.e. \( f_r = 0 \). Moreover, any example \( e \in \mathcal{E} \) is represented as a 2-tuple \((\mathcal{L}_q, e_q)\), where \( \mathcal{L}_q \) denotes the literals associated with the example and \( e_q \in \{0, 1\} \) is the class to which the example belongs. We have \( e_q = 1 \) if \( e_q \in \mathcal{E}^+ \) and \( e_q = 0 \) if \( e_q \in \mathcal{E}^- \). A literal \( l_r \) on a feature \( f_r \), \( l_r \in \{ f_r, \neg f_r \} \), discriminates an example \( e_q \) iff \( \neg l_r \in \mathcal{L}_q \).

**Example 1.** Table 1 shows a simple classification example.

<table>
<thead>
<tr>
<th>( e_q )</th>
<th>( L_1 )</th>
<th>( L_2 )</th>
<th>( L_3 )</th>
<th>( L_4 )</th>
<th>( L_5 )</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td></td>
</tr>
</tbody>
</table>

Table 1: A classification example.

2.2 Boolean Satisfiability

We assume notation and definitions standard in the area of Boolean Satisfiability (SAT), i.e. the decision problem for propositional logic \cite{Biere2009}. Formulas are represented in Conjunctive Normal Form (CNF) and defined over a set of variables \( X = \{ x_1, \ldots, x_n \} \). A formula \( F \) is a conjunction of clauses, a clause is a disjunction of literals, and a literal is a variable \( x_i \) or its complement \( \neg x_i \). Where appropriate, formulas are viewed as sets of sets of literals.

CNF encodings of cardinality constraints have been studied extensively, and will be assumed throughout \cite{Biere2009}. Moreover, standard classification techniques are assumed \cite{Plaisted1986}.

2.3 Related Work

The complexity of learning optimal decision trees is well-known, for different notions of optimality \cite{Hyafil1976, Hancock1996}. For example, finding a decision tree that minimizes the average number of tests necessary to classify an example is NP-hard \cite{Hyafil1976}. Motivated by these results, and also by existing experimental evidence, the learning of optimal decision trees is often deemed infeasible in practice \cite{Rokach2015}.

As a result, all practical approaches for learning decision trees are heuristic \cite{Breiman1984, Quinlan1986, Quinlan1993, Mitchell1997, Rokach2015}. These heuristic approaches often learn reasonably small decision trees. Furthermore, most ML and DM packages implement some algorithm for learning decision trees \cite{Hall2009, Pedregosa2011}. One concrete example used in the paper is ITI \cite{Utgoff1997}.

Despite the computational difficulty of the problem, there are examples of earlier attempts at learning optimal decision trees \cite{Bessiere2009, Dufour2014}. Earlier work considered SAT and constraint programming (CP) \cite{Bessiere2009} and, more recently what can be viewed as an incomplete variant of A* \cite{Dufour2014}, based on breadth-first search, with backtracking, and different admissible heuristics. The use of SAT was shown unrealistic by earlier results \cite{Bessiere2009}. Moreover, the use of CP was shown to enable obtaining approximate solutions for reasonably sized problems, this at the cost of sacrificing achieving optimal solutions \cite{Bessiere2009}.

3 SAT-Based Learning of Decision Trees

This section proposes a different take on the problem of learning a decision tree using SAT-based methods compared to \cite{Bessiere2009}. Our encoding exploits a structural

\footnote{As we show in the results, for larger data sets learnt decision trees may be significantly larger than the optimal size.}

\footnote{The use of ITI is motivated by also being used in earlier related work \cite{Bessiere2009}.}
property of the learning problem. Namely, one way to learn a decision tree with a given number of nodes is to, first, guess a valid binary tree topology and, second, verify that we can classify all positive and negative examples correctly for this topology. Therefore, our encoding consists of two parts: constraints that encode a valid binary tree and constraints that ensure the decision tree is accurate when classifying examples in the training data. We start with the generation of valid binary trees constraints.

We develop the encoding for a specific number of nodes \( N \) of the decision tree, but such that any binary tree of size \( N \) can be learned. To search for the optimal value, a wealth of approaches can be considered, which have been studied in the context of SAT-based optimization. In this paper we refine upper bounds on the size of the decision tree, until the formula becomes unsatisfiable for a target size \( N \), in which case the optimal size is \( N + 2 \) as \( N \) must be an odd number.

3.1 Encoding Valid Binary Trees

In this section we consider an encoding of a binary tree for a given number of nodes \( N \). We note that we always build a full binary tree where each node has two children. We assume that nodes are numbered in the breadth-first order from left to right. Any decision tree can be represented with this numbering as we can number nodes using the breadth-first order. Namely, the root node of the tree is numbered 1. The two children of a node \( i \) can be numbered in the range from \( i + 1 \) to \( \min(2i + 1, N) \). The number of the left child and the number of the right child are consecutive numbers.

For each node we introduce a propositional variable \( v_i \) to encode information about internal nodes and leaves. To encode the child-parent relationship, we introduce three sets of propositional variables, \( l_{ij}, r_{ij}, p_{ij} \) (see the upper part of Table 2) Note that \( l_{ij} \) and \( r_{ij} \) are defined for even/odd indices as a left/right child must be an even/odd node. Moreover, the shortcuts \( j \in LR(i) \) and \( j \in RR(i) \) are also defined in Table 2.

Example 2. Consider an example of an encoding for a tree with \( N = 5 \) nodes. We introduce four sets of variables: \( \{v_1, \ldots, v_5\} \), \( \{l_{12}, l_{24}, l_{23}, l_{34}\} \), \( \{r_{13}, r_{25}, r_{35}\} \) and \( \{p_{21}, p_{31}, p_{32}, p_{42}, p_{43}, p_{52}, p_{53}, p_{54}\} \).

Next we describe the constraints used. We assume a non-trivial learning problem, and so the root node is not a leaf.

\[
\neg v_1 \quad (1)
\]

If a node is a leaf node, then it has no children:

\[
v_i \rightarrow \neg l_{ij} \quad j \in LR(i) \quad (2)
\]

The left child and the right child of the \( i \)th node are numbered consecutively.

\[
l_{ij} \leftrightarrow r_{ij+1} \quad j \in LR(i) \quad (3)
\]

A non-leaf node must have a child.

\[
\neg v_i \rightarrow \left( \sum_{j \in LR(i)} l_{ij} = 1 \right) \quad (4)
\]

If the \( i \)th node is a parent then it must have a child.

\[
p_{ij} \leftrightarrow l_{ij}, \quad j \in LR(i) \quad (5)
\]

\[
p_{ij} \leftrightarrow r_{ij}, \quad j \in RR(i)
\]

The binary tree must be a tree. Hence, all nodes but the first must have a parent:

\[
\left( \sum_{j = \left\lfloor \frac{i}{2} \right\rfloor}^{\min(j-1, N)} p_{ji} = 1 \right) \quad \text{with } j = 2, \ldots, N \quad (6)
\]

Note that constraints (4) contain the cardinality constraints. We used the sequential counters encoding [Sinz, 2005] to model a cardinality constraint as a Boolean formula.

Example 3. We continue with Example 2. We will have the following constraints to encode a binary tree construction.

First, we enforce that the root is not a leaf by adding \( \neg v_1 \).

Then we encode constraints from (2): \( v_1 \rightarrow \neg l_{12}; v_2 \rightarrow \neg l_{24}; v_3 \rightarrow \neg l_{34}. \) Second, we add constraints from (3) that left and right nodes are numbered consecutively and a non-leaf must have a child (constraints (4)):

\[
l_{1,2} \leftrightarrow l_{12}; l_{24} \leftrightarrow l_{25}; l_{34} \leftrightarrow r_{35};
\]

\[
\neg v_1 \rightarrow (l_{12} = 1); \neg v_2 \rightarrow (l_{24} = 1); \neg v_3 \rightarrow (l_{34} = 1);
\]

Next, we encode parent-child relations using (5) and (6):

\[
p_{21} \leftrightarrow l_{12}; p_{24} \leftrightarrow l_{24}; p_{43} \leftrightarrow l_{34},
\]

\[
p_{31} \leftrightarrow r_{13}; p_{32} \leftrightarrow r_{25}; p_{53} \leftrightarrow r_{35},
\]

\[
p_{21} = 1; p_{31} + p_{32} = 1; p_{42} + p_{43} = 1; p_{52} + p_{53} + p_{54} = 1.
\]

As can be seen from this example, the encoding is both compact and easy to implement. There are only two valid binary trees. The first one has two internal nodes: \( v_1 \) and \( v_2 \). The second one has two internal nodes: \( v_1 \) and \( v_3 \).

We also ensure a simple approach for breaking some symmetries. Concretely, for any internal node of a binary tree, the left branch will always be associated with some feature being assigned value 0, whereas the right branch will always be associated with that feature being assigned value 1.

Table 2: Description of propositional variables.

<table>
<thead>
<tr>
<th>Var</th>
<th>Description of variables</th>
</tr>
</thead>
<tbody>
<tr>
<td>( v_i )</td>
<td>if node ( i ) is a leaf node, ( i = 1, \ldots, N ).</td>
</tr>
<tr>
<td>( l_{ij} )</td>
<td>if node ( i ) has node ( j ) as the left child, with ( j \in LR(i) ), where ( LR(i) = \text{even}([i + 1, \min(2i + 1, N - 1)]) ), ( i = 1, \ldots, N ).</td>
</tr>
<tr>
<td>( r_{ij} )</td>
<td>if node ( i ) has node ( j ) as the right child, with ( j \in RR(i) ), where ( RR(i) = \text{odd}([i + 2, \min(2i + 1, N)]) ), ( i = 1, \ldots, N ).</td>
</tr>
<tr>
<td>( p_{ij} )</td>
<td>if the parent of node ( i ) is node ( j ) and ( i, j = 2, \ldots, N ), ( i = 1, \ldots, N - 1 ).</td>
</tr>
<tr>
<td>( a_{ij} )</td>
<td>if feature ( f_i ) is assigned to node ( j ), ( r = 1, \ldots, K, r = 1, \ldots, N ).</td>
</tr>
<tr>
<td>( u_{ij} )</td>
<td>if feature ( f_i ) is being discriminated against by node ( j ), ( r = 1, \ldots, K, r = 1, \ldots, N ).</td>
</tr>
<tr>
<td>( d_{ij} )</td>
<td>if feature ( f_i ) is discriminated for value 0 by node ( j ) or by one of its ancestors, ( r = 1, \ldots, K, r = 1, \ldots, N ).</td>
</tr>
<tr>
<td>( c_{ij} )</td>
<td>if class of leaf node ( j ) is ( 1, j = 1, \ldots, N ).</td>
</tr>
</tbody>
</table>
examples; otherwise the classification would not be (100%) accurate. If the node’s class is negative, then the path in the tree, and the literals associated with each branch, must discriminate all the positive examples; otherwise the classification would not be (100%) accurate. This section develops the constraints that achieve the target discrimination. (As indicated earlier, it is possible to allow for some examples not to be discriminated, i.e. simply let some examples not to be discriminated, but this is beyond the scope of the paper.)

Besides the variables used to generate valid binary trees, we use additional variables to capture the constraints that the decision tree must correctly classify all examples in \( \mathcal{E} \). (The lower part of Table 2 summarizes these variables.) Five additional sets of variables are used. Variables \( a_{rj} \) and \( u_{rj} \) encode information about the binary features that was branched on at node \( j \). Variables \( a_{rj} \) signal that the \( r \)th feature is branched on in node \( j \). Variables \( u_{rj} \) store information whether the \( r \)th feature is discriminated at any node on the path from this node to the root. Variables \( d_{0rj} \) and \( d_{1rj} \) are used to remember if a feature was discriminated positively or negatively along the path from the root to the \( j \)th node. Concretely, any example exhibiting \( f_r = 0 \) (resp. \( f_r = 1 \)) will be discriminated by node \( j \) or by one of its ancestors iff \( d_{0rj} = 1 \) (resp. \( d_{1rj} = 1 \)).

Next we present the constraints used. To discriminate a feature for value 0 at node \( j, j = 2, \ldots, N \):

\[
d_{0rj} \leftrightarrow \bigvee_{i=[\frac{1}{2}]}^{j-1} \left( (p_{ji} \land d_{irj}) \lor (a_{ri} \land r_{ij}) \right); \quad d_{1rj} = 0. \tag{7}
\]

To discriminate a feature for value 1 at node \( j, j = 2, \ldots, N \):

\[
d_{1rj} \leftrightarrow \bigvee_{i=[\frac{1}{2}]}^{j-1} \left( (p_{ji} \land d_{irj}) \lor (a_{ri} \land l_{ij}) \right); \quad d_{1rj} = 0. \tag{8}
\]

Using a feature \( r \) at node \( j, r = 1, \ldots, K, j = 1, \ldots, N \):

\[
\bigwedge_{i=[\frac{1}{2}]}^{j-1} (u_{ri} \land p_{ji} \land a_{rj})
\]

\[
u_{rj} \leftrightarrow (a_{rj} \lor \bigvee_{i=[\frac{1}{2}]}^{j-1} (u_{ri} \land p_{ji})) \quad \tag{9}
\]

For a non-leaf node \( j \), exactly one feature is used:

\[
\neg v_j \leftrightarrow \left( \sum_{r=1}^{K} a_{rj} = 1 \right) \quad \text{with } j = 1, \ldots, N \tag{10}
\]

For a leaf node \( j \), no feature is used:

\[
v_j \rightarrow \left( \sum_{r=1}^{K} a_{rj} = 0 \right) \quad \text{with } j = 1, \ldots, N \tag{11}
\]

Let \( e_q \in \mathcal{E}^+ \), and let the sign of the literal on feature \( f_r \) for \( e_q \) be \( \sigma(r, q) \in \{0, 1\} \). For every leaf node \( j, j = 1, \ldots, N \):

\[
v_j \land \neg c_j \rightarrow \bigvee_{r=1}^{K} d_{rj}^{\sigma(r, q)} \quad \tag{12}
\]

i.e. any positive example must be discriminated if the leaf node is associated with the negative class.

Let \( e_q \in \mathcal{E}^- \), and let the sign of the literal on feature \( f_r \) for \( e_q \) be \( \sigma(r, q) \). Then, for every leaf node \( j, j = 1, \ldots, N \):

\[
v_j \land c_j \rightarrow \bigvee_{r=1}^{K} d_{rj}^{\sigma(r, q)} \quad \tag{13}\]

i.e. any negative example must be discriminated if the leaf node is associated with the positive class.

**Proposition 1** (Propositional model size). For a DT learning problem with \( K \) binary features, \( M = |\mathcal{E}| \) examples, and a target decision tree with \( N \) nodes, the proposed encoding size (on the number of literals) is in \( O(K \times N^2 + M \times N \times K) \).

**Proof.** (Sketch) By inspection of the constraints proposed in this section, noting that \( r \) ranges from 1 to \( K \), \( i, j \) range from 1 to \( N \), and the the size of \( \mathcal{E} \) is \( M \). The term \( M \times N \) results from (12) and (13), each contains \( O(K) \) literals. The term \( K \times N^2 \) depends on the remaining constraints. \( \square \)

**Remark 1.** The proposed model is far tighter than the one proposed in [Bessiere et al., 2009] in terms of the size of the SAT encoding, i.e. in \( O(K \times N^2 \times M^2 + N \times K^2 + K \times N^3) \), which exhibits the additional drawback of requiring a fixed target binary tree. Moreover, in practice one would in general expect \( M \) to be far larger than either \( K \) or \( N \). Thus, in practice the proposed model will in general be orders of magnitude tighter than the model proposed in earlier work [Bessiere et al., 2009]. Our experimental results confirm this observation (see Table 3).

**Example 4.** Figure 1 shows a binary tree of size 5 that classifies examples from Table 1 correctly. We observe that this tree is one of two valid binary trees (see Example 3), (7)–(13) rule out the second valid binary tree as it does not ensure correct classification of examples.

### 3.3 Additional Inference Constraints

We propose additional constraints that aim at pruning the search space, by filtering as soon as possible tree arrangements that are invalid. During the search, a partial structure of the tree is constructed. Hence, we can reduce the search space by pruning invalid extensions of this partial tree. We recall that, in general, for a node \( i \), the left child ranges from \( i + 1 \) to \( \min(2i, N - 1) \), among even-numbered nodes, and the right child ranges from \( i + 2 \) to \( \min(2i + 1, N) \), among odd-numbered nodes.

For each \( k \), with \( k < i \), such that \( e_k \) holds, then the upper bound on the number of left and right children of \( i \) is reduced. Likewise, if \( e_k \) does not hold, then the lower bound on the number of left and right children of \( i \) is increased. The following example demonstrates our intuition.
Example 5. Suppose we are looking for a binary tree of size $N = 7$. We focus on the third node, $i = 3$. In general, nodes 4, 5, 6 or 7 can be children of this node. If we have a balanced binary tree then nodes 6 and 7 are children of node 3. If we have highly unbalanced binary tree where each left branch is of length one then nodes 4 and 5 are children of node 3.

Suppose that during the search we learn that there exists (at least) one node that is a leaf and it is numbered 3 or less. In this case, node 2 is a leaf and we conclude that only nodes 4 and 5 can be children of node 3.

Let $\lambda_{t,i}$ denote the number of leaf nodes until (and including) node $i$. Clearly, $0 \leq t \leq \lfloor \frac{t}{2} \rfloor$. $\lambda_{t,i}$ is defined inductively:

1. $\lambda_{0,i} = 1$, for $1 \leq i \leq N$.

2. $\lambda_{t,i} \leftrightarrow (\lambda_{t-1,i} \lor \lambda_{t-1,i-1} \land v_i)i = 1..N, t = 1..\lfloor \frac{t}{2} \rfloor$.

Similarly, we can define the number of non-leaf (tree) nodes until node $i$. $\tau_{t,i}$, with $0 \leq t \leq i$. Again, $\tau_{t,i}$ is defined inductively as follows:

1. $\tau_{0,i} = 1$, for $1 \leq i \leq N$.

2. $\tau_{t,i} \leftrightarrow (\tau_{t,i-1} \lor \tau_{t,i-1} \land \neg v_i)$, for $i = 1..N, t = 1..i$.

Proposition 2 (Refine upper bound on descendants’ numbers). If $\lambda_{t,i} = 1$, with $0 < t \leq \lfloor \frac{t}{2} \rfloor$, then $l_{i,2(i-t+1)+1} = 0$ and $r_{i,2(i-t+1)+1} = 0$.

Proposition 3 (Refine lower bound on descendants’ numbers). If $\tau_{t,i} = 1$, with $\lfloor \frac{t}{2} \rfloor < t \leq i$, then $l_{i,2(t-1)+1} = 0$ and $r_{i,2(t-1)+1} = 0$.

Overall, additional inference constraints give 10% to 50% speed up in our experiments.

Extending the model. As indicated earlier, the model can be extended to handle non-binary features, e.g. by exploiting the well-known one-hot-encoding to replace non-binary features with a set of binary features. The model naturally enables non-binary classes, simply by ensuring that a node of a given class discriminates all others. Finally, the model can be extended to accommodate for inconsistent training data, i.e. multiple classes associated with the same assignment of feature. This requires allowing some classes not to be discriminated in some decision tree path. This solution represents a more challenging combinatorial optimization problem.

4 Experimental Results

We perform an experimental evaluation on a set of benchmarks from [Bessiere et al., 2009; Olson et al., 2017]. We ran our experiments on Intel(R) Xeon(R) CPU 3.50GHz. The Glucose3 SAT solver [Audemard and Simon, 2009] was used, with a total timeout of 1000 sec in all experiments. The memory limit is 2GB. We also employed the ITI algorithm to produce decision trees from labeled examples [Utgoff et al., 1997a; Utgoff et al., 1997b]. The ITI algorithm does not guarantee optimality, so the size of the tree produced by ITI serves as an upper bound on the size of the optimal tree in our experiments. We refer to our encoding as DT1.

4.1 Comparison with Existing SAT Encodings

Our first experiment is to compare our SAT encoding (DT1) with results for the SAT model reported in [Bessiere et al., 2009] that we refer to as DT2. The encodings are compared in terms of their size and performance on a subset of benchmarks from [Bessiere et al., 2009] that the authors kindly shared with us. For a given tree size of depth 4, we build a CNF encoding of the problem. The first row shows results reported in [Bessiere et al., 2009]. Note that results for the Income benchmark were reported for 10% of data samples in Table 1 in [Bessiere et al., 2009], so we report on the same amount of data randomly sampled. We would like to point out that our encoding is more general compared to DT2 that only solves the classification problem for a fixed binary tree. In our case, we encode both the tree construction for a given number of nodes and the classification task. The second row shows our results. As can be seen from Table 3, our encoding is more compact compared to the CNF encoding in [Bessiere et al., 2009]. As can be observed, the encoding size can be 1000 times smaller than DT2 on some examples. Next, we discuss the performance of these encodings. DT2 was able to find the optimal only for the two small instances Weather and Mouse with the total time to prove optimality 0.37s and 577.27s, respectively. Using our encoding, the total time to prove optimality are 0.05s and 12.94s. Even though these results were obtained on different machines, it is plain that our encoding is more effective in finding and proving optimality.

4.2 Comparison on Extended Set of Benchmarks

We perform a series of experiments on an extended set of benchmarks. The goal is to demonstrate that for problems that admit relatively small decision trees we can find the optimal decision tree and prove its optimality. We start with toy benchmarks from [Bessiere et al., 2009; Olson et al., 2017] that admit a small DT (less than 15 nodes). Table 4 shows our results. The second column (\#nd\textsuperscript{it}) shows the size of the decision tree produced by ITI. The third column (\#nd\textsuperscript{sat}) shows the size of optimal tree that we find. To find the optimal tree we perform a linear search on the total number of nodes starting from the upper bound provided by the ITI tree. We only consider trees with an odd number of nodes. The ‘time’ column shows the total time in seconds to perform this linear search procedure. As can be seen from the table, our encoding is very efficient and we can improve results of ITI and prove optimality for these benchmarks.

Next we consider larger benchmarks. Finding an optimal tree is a hard combinatorial problem as the search space is huge even for trees of size around 100 nodes. Exploring such search space is a challenging problem. However, [Bessiere et
Table 5: Results on subsampled instances solved to optimality.

<table>
<thead>
<tr>
<th>Name</th>
<th>#n</th>
<th>#d</th>
<th>#o</th>
<th>Rb</th>
<th>Rf</th>
<th>test acc. ITI/DT1</th>
</tr>
</thead>
<tbody>
<tr>
<td>Car</td>
<td>0.05</td>
<td>22.30</td>
<td>18.80</td>
<td>75.41</td>
<td>20</td>
<td>86 / 21</td>
</tr>
<tr>
<td>Cancer</td>
<td>0.2</td>
<td>1.367</td>
<td>23.67</td>
<td>684.12</td>
<td>3</td>
<td>172 / 21</td>
</tr>
<tr>
<td>Shuttle</td>
<td>0.05</td>
<td>10.23</td>
<td>12.86</td>
<td>110.11</td>
<td>14</td>
<td>89 / 89</td>
</tr>
<tr>
<td>Colic</td>
<td>0.1</td>
<td>1.367</td>
<td>23.67</td>
<td>684.12</td>
<td>3</td>
<td>172 / 21</td>
</tr>
<tr>
<td>Appen-tis</td>
<td>0.05</td>
<td>10.23</td>
<td>12.86</td>
<td>110.11</td>
<td>14</td>
<td>89 / 89</td>
</tr>
<tr>
<td>Australian</td>
<td>0.1</td>
<td>1.367</td>
<td>23.67</td>
<td>684.12</td>
<td>3</td>
<td>172 / 21</td>
</tr>
<tr>
<td>Backache</td>
<td>0.05</td>
<td>10.23</td>
<td>12.86</td>
<td>110.11</td>
<td>14</td>
<td>89 / 89</td>
</tr>
</tbody>
</table>

Table 6: Results on reduced instances solved to optimality.

<table>
<thead>
<tr>
<th>Name</th>
<th>#n</th>
<th>#d</th>
<th>#o</th>
<th>Rb</th>
<th>Rf</th>
<th>test acc. ITI/DT1</th>
</tr>
</thead>
<tbody>
<tr>
<td>Appen-tis</td>
<td>0.05</td>
<td>22.30</td>
<td>18.80</td>
<td>75.41</td>
<td>20</td>
<td>86 / 21</td>
</tr>
<tr>
<td>Australian</td>
<td>0.2</td>
<td>1.367</td>
<td>23.67</td>
<td>684.12</td>
<td>3</td>
<td>172 / 21</td>
</tr>
<tr>
<td>Backache</td>
<td>0.05</td>
<td>10.23</td>
<td>12.86</td>
<td>110.11</td>
<td>14</td>
<td>89 / 89</td>
</tr>
<tr>
<td>Colic</td>
<td>0.1</td>
<td>1.367</td>
<td>23.67</td>
<td>684.12</td>
<td>3</td>
<td>172 / 21</td>
</tr>
<tr>
<td>Appen-tis</td>
<td>0.05</td>
<td>10.23</td>
<td>12.86</td>
<td>110.11</td>
<td>14</td>
<td>89 / 89</td>
</tr>
<tr>
<td>Australian</td>
<td>0.1</td>
<td>1.367</td>
<td>23.67</td>
<td>684.12</td>
<td>3</td>
<td>172 / 21</td>
</tr>
<tr>
<td>Backache</td>
<td>0.05</td>
<td>10.23</td>
<td>12.86</td>
<td>110.11</td>
<td>14</td>
<td>89 / 89</td>
</tr>
</tbody>
</table>

5 Conclusions

This paper develops novel SAT-based solutions for computing optimal decision trees. The proposed approach exploits a number of well-established techniques for SAT-based problem solving, including exploiting properties of decision trees, breaking symmetries in the problem formulation, and using tight encodings for standard constraints, including atMost1 constraints. In contrast with earlier work [Bessiere et al., 2009], the paper shows that optimal decision trees can be computed for datasets of reasonable size. To our best knowledge, this paper presents for the first time the size of optimal decision trees for several well-known datasets.

Acknowledgments

Alexey Ignatiev and Joao Marques-Silva were supported by FCT grants SAFETY (SRHR/ BPD/120315/2016), ABSOLV (028986/02/SAICT/2017), and LASIGE Research Unit, ref. UID/CEC/00408/2013.
References


