Scalable Computation of Abductive Explanations for Tree Ensembles

Teemu Koukkari
University of Helsinki, Finland

Jeremias Berg
University of Helsinki, Finland

Abstract

Tree ensembles such as random forests and boosted trees are classifiers that achieve state-of-the-art performance especially on classification over tabular data. The high efficiency comes, however, at the cost of interpretability. While individual decision trees are considered interpretable, combining many trees obscures the ensemble's reasoning, motivating methods for explaining their decisions.

We contribute to the field of formal explainable AI by proposing two new instantiations of a deletion-based procedure for computing abductive explanations (also known as sufficient reasons or PI explanations) for tree ensembles, one based on the implicit-hitting-set paradigm, and the other on branch-and-bound search. Informally speaking, an abductive explanation is a minimal subset of the features of the classification task that is sufficient for reaching the decision. We describe both methods and their variants, and show experimentally that they significantly improve the scalability of exact abductive explanation computation for tree ensembles.

2012 ACM Subject Classification Computing methodologies → Machine learning; Computing methodologies → Artificial intelligence; Theory of computation → Logic

Keywords and phrases abductive explanations, sufficient reasons, PI explanations, tree ensembles, random forests, boosted trees, IHS, B&B, formal explainable AI

Digital Object Identifier 10.4230/LIPIcs.CVIT.2016.23

1 Introduction

Tree ensembles like random forests [6] and boosted trees [7] are central machine learning (ML) models for tackling classification tasks in various application domains. Tree ensembles work well especially on classification tasks over tabular data on which they have been shown to achieve state-of-the-art performance [5]. A tree ensemble is a collection of decision trees that classifies data points by aggregating the classifications made by individual decision trees in various ways; a random forest with majority voting for example assigns a datapoint the same class as the majority of the trees in the ensemble.

The well-recognized need for trustworthy AI, with trustworthiness typically equated with the ability to explain the decisions made by models [20] has motivated research into explainable AI (XAI) and ways of computing explanations for the decisions made by commonly used ML models. The field XAI can further be divided into heuristic approaches like SHAP [14] and Anchors [19], and formal XAI [16] that aims to ground the correctness of the computed explanations in formal logic to avoid the ambiguities and inaccuracies of heuristic explanations [8].

Our work contributes to the field of formal XAI by proposing two new approaches to computing so-called abductive explanations [10] (also known as sufficient reasons [2] or prime implicant explanations) for a general model of a tree ensemble that encompasses both random forests and boosted trees. An abductive explanation for the decision $\mathcal{M}(\mathbf{x})$ made by a classifier $\mathcal{M}$ on a datapoint $\mathbf{x}$ is a subset-minimal set of features in the classification problem whose values in $\mathbf{x}$ are sufficient for $\mathcal{M}$ to make the same decision. While a single decision tree can be considered an interpretable model [20] in the sense that the computation of an

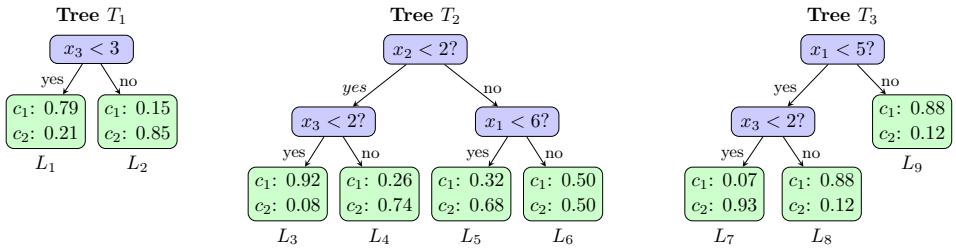

Figure 1 Example random forest used as a running example, consisting of three decision trees, the features $\mathcal{F} = \{1, 2, 3\}$, and the classes $\mathcal{K} = \{c_1, c_2\}$. The leaves are labelled $L_1, \ldots, L_9$ for reference.

abductive explanation for a single tree can be done in linear time with respect to its size [15] the same does not hold for tree ensembles [6]. The computation of abductive explanations for random forests was shown to be DP-complete in [13]. Thus, the main challenge facing current approaches to computing minimal abductive explanations is scaling the algorithmic methods to ensemble sizes encountered in practical applications.

Prior to our work, the state-of-the-art tools for computing subset-minimal abductive explanations for tree ensembles are XReason [9] for boosted trees and RFXpl [13] for random forests, both of which are based on core-guided optimization of a MaxSAT problem [18]. Related work includes majoritary [1] and tree-specific [2] explanations, which are weak abductive explanations computable in polynomial time by restricting attention to most (but not all) trees in the ensemble, as well as inflated and cardinality-minimal explanations [12, 11], which extend the notion of abductive explanations.

As our main contribution, we propose two new algorithms for computing minimal abductive explanations of tree ensembles. Both make use of the straightforward deletion-based minimization algorithm [2, 9, 11]) and differ in the instantiation of the check for whether a given set of features constitutes a potentially non-minimal explanation, which is the most challenging part of the approach. The first of our methods is inspired by the implicit hitting set (IHS) paradigm for optimization [17], and the second is a branch and bound (B&B) search with domain specific heuristics. We implement both approaches and report on an empirical evaluation showing that our tool scales significantly better than previous methods for computing subset-minimal abductive explanations.

2 Tree ensembles and their explanations

A classification problem is defined by a set of features $\mathcal{F} = \{1, \ldots, m\}$ and a set of classes $\mathcal{K} = \{c_1, \ldots, c_K\}$. Each feature $k \in \mathcal{F}$ takes values in a continuous interval $I_k \subset \mathbb{R}$, and these intervals define an m-dimensional feature space $\prod_{i=1}^{m} I_i \subset \mathbb{R}^m$. We represent an arbitrary point in the feature space as a vector $\mathbf{x} = (x_1, \ldots, x_m)$. A classifier $\mathcal{T}$ assigns each point $\mathbf{x}$ in the feature space a class $\mathcal{T}(\mathbf{x}) \in \mathcal{K}$. In this work, all considered classifiers are *tree ensembles*. We use the following general definition of a tree-ensemble, similar to the one used in [11].

A decision tree T is a binary tree in which each internal node n is associated with a splitting value $d_{i,m}$ for the feature $i \in \mathcal{F}$. Each leaf l of T is associated with a score function score_l that assigns a non-negative real value to each class in $\mathcal{K}$. The set $\text{path}(T, l) = \{r, n_2, \ldots, n_k\}$ contains the root r and all internal nodes on the unique path from r to the leaf l. Each datapoint $\mathbf{x} = (x_1, \ldots, x_m)$ corresponds to a unique leaf $l_{\mathbf{x}}$ of T that is found by, starting from the root, comparing the splitting value $d_{i,k}$ of each internal node n with the value x_i of the corresponding feature in $\mathbf{x}$, traversing left if $x_i < d_{i,k}$ and right otherwise, until reaching

a leaf. Overloading notation, we say that the score $\text{score}(T, c, \mathbf{x})$ of $\mathbf{x}$ for the class c in T is $\text{score}_{l_x}(c)$, i.e. the score that $l_{\mathbf{x}}$ assigns c.

A tree ensemble $\mathcal{T} = \{T_1, \ldots, T_k\}$ is a collection of decision trees. The set LEAVES-OF($\mathcal{T}$) contains all of the leaf nodes in $\mathcal{T}$. For a datapoint $\mathbf{x}$, the set $\mathcal{L}(\mathcal{T}, \mathbf{x})$ contains from each tree $T \in \mathcal{T}$ the leaf of T that $\mathbf{x}$ corresponds to. The score $\text{score}(\mathcal{T}, c, \mathbf{x})$ of $\mathbf{x}$ for the class $c \in \mathcal{K}$ in $\mathcal{T}$ is $\text{score}(\mathcal{T}, c, \mathbf{x}) = \sum_{T \in \mathcal{T}} \text{score}(T, c, \mathbf{x}) = \sum_{l \in \mathcal{L}(\mathcal{T}, \mathbf{x})} \text{score}_l(c)$, i.e. the sum of scores of $\mathbf{x}$ for c in each individual tree of $\mathcal{T}$, or equivalently, the sum of scores assigned to c by each leaf in $\mathcal{L}(\mathcal{T}, \mathbf{x})$. The class $\mathcal{T}(\mathbf{x}) \in \mathcal{K}$ that $\mathcal{T}$ assigns $\mathbf{x}$ maximizes $\text{score}(\mathcal{T}, c, \mathbf{x})$ over all classes c.

The model of a tree ensemble that we use captures a random forest with majority voting by requiring each leaf score function to assign exactly one class a score of 1 and the others 0. Similarly, a boosted tree ensemble is obtained by associating each tree $T \in \mathcal{T}$ with some class c and requiring the leaf score functions of T assign non-zero values only to c.

An abductive explanation [10] (or a sufficient reason [2]) for the prediction $\mathcal{T}(\mathbf{x})$ is a minimal subset $E \subset \mathcal{F}$ of the features that is sufficient for $\mathcal{T}$ to reach that prediction. More precisely, E is an abductive explanation if: i) $\mathcal{T}(\mathbf{x}) = \mathcal{T}(\mathbf{x}')$ for any datapoint $\mathbf{x}'$ that agrees with $\mathbf{x}$ on the features in E, and ii) condition i) does not hold for any proper subset $E_s \subset E$. A superset $E_L \supset E$ of an abductive explanation E is a weak abductive explanation.

Equivalently, E is a weak abductive explanation for $\mathcal{T}(\mathbf{x})$ iff no *counterexample* exists for any class $c \neq \mathcal{T}(\mathbf{x})$, where a counterexample on c is a selection of exactly one leaf per tree, reachable by some datapoint that agrees with $\mathbf{x}$ on E, whose summed score for c exceeds the summed score for $\mathcal{T}(\mathbf{x})$. Both algorithms we propose for checking weak abductive explanations are based on searching for counterexamples.

We compute abductive explanations using the standard deletion-based scheme [11, 10, 9, 2], which maintains a current weak explanation E and tries to shrink it one feature at a time. Starting from the trivial weak explanation $E = \mathcal{F}$, the procedure iterates over each feature $f \in \mathcal{F}$ and removes f from E whenever $E \setminus \{f\}$ is still a weak explanation for $\mathcal{T}(\mathbf{x})$; otherwise f is kept. Once all features have been considered, the resulting E is an abductive explanation for $\mathcal{T}(\mathbf{x})$. The non-trivial step is the *weak-explanation check*: given $\mathbf{x}$, $\mathcal{T}$, and a candidate $E \subseteq \mathcal{F}$, decide whether E is a weak explanation for $\mathcal{T}(\mathbf{x})$. The new instantiations of this check that we present in Section 3 are the main contribution of our work.

▶ **Example 1.** Consider the ensemble $\mathcal{T}$ shown in Figure 1 and a datapoint $\mathbf{x} = (4, 3, 5)$. The root of T_1 is associated with the splitting value $d_{3,1} = 3$. The score function score_{L_1} associated with L_1 assigns $\text{score}_{L_1}(c_1) = 0.79$ and $\text{score}_{L_1}(c_2) = 0.21$. The point $\mathbf{x}$ has $\mathcal{L}(\mathcal{T}, \mathbf{x}) = \{L_2, L_5, L_8\}$ and the scores $\text{score}(\mathcal{T}, c_1, \mathbf{x}) = 1.35$ and $\text{score}(\mathcal{T}, c_2, \mathbf{x}) = 1.65$, so $\mathcal{T}(\mathbf{x}) = c_2$. The set $E = \{1, 3\}$ is an abductive explanation for $\mathcal{T}(\mathbf{x}) = c_2$ since any datapoint $\mathbf{x}' = (4, x_2', 5)$ corresponds to the leaves L_2 and L_8. The sets $\mathcal{L}' = \{L_1, L_3, L_7\}$ and $\mathcal{L}'' = \{L_2, L_6, L_9\}$ are counterexamples for $E' = \{1\}$ and $E'' = \{3\}$ on c_1.

2.1 A SAT-encoding for tree ensembles

We assume familiarity with propositional logic and SAT solving [4]. A formula F in conjunctive normal form (CNF) is a conjunction of clauses, each a disjunction of literals (variables or their negations), and is satisfiable if some 0-1 assignment to its variables satisfies all clauses. Modern SAT-solvers support solving under a set $\mathcal{A}$ of *assumptions* (literals): invoked on F under $\mathcal{A}$, the solver returns either a solution that sets every literal in $\mathcal{A}$ to 1, or a subset $\mathcal{A}_s \subseteq \mathcal{A}$,*unsatisfiable core*, such that $F \wedge \bigwedge_{l \in \mathcal{A}_s} l$ is unsatisfiable.

We use a simplification of the propositional encoding of tree ensembles from [11]. Given an ensemble $\mathcal{T}$ with features $\mathcal{F}$ and classes $\mathcal{K}$, the encoding produces a satisfiable CNF-formula

$F_{\mathcal{T}}$ that represents the feature domains and the internal structure of the trees in $\mathcal{T}$. For each feature $i \in \mathcal{F}$, let $\{d_{i,1}, d_{i,2}, \dots d_{i,k}\}$ be all split values on the feature i used in some internal node of $\mathcal{T}$, sorted in ascending order, i.e. assume $d_{i,j} \leq d_{i,j+1}$ for all j. For each $d_{i,j}$ the formula $F_{\mathcal{T}}$ contains an indicator variable $\langle x_i < d_{i,j} \rangle$ for a datapoint $\mathbf{x}$ having $x_i < d_{i,j}$. For each leaf l in $\mathcal{T}$, $F_{\mathcal{T}}$ contains an indicator p_l for a datapoint corresponding to l.

There are two types of constraints in $F_{\mathcal{T}}$. For all features i and distinct splitting values, the formula contains the clause $(\neg\langle x_i < d_{i,j}\rangle \vee \langle x_i < d_{i,j+1}\rangle)$ (equivalent to $\langle x_i < d_{i,j}\rangle \rightarrow \langle x_i < d_{i,j+1}\rangle$) to ensure consistent ordering of the splitting values. For each leaf l in $\mathcal{T}$ $F_{\mathcal{T}}$ contains clauses equivalent to

$$\left(\bigwedge_{\substack{n \in \text{path}(T,l) \\ x_i < d_i^n}} \langle x_i < d_{i,n} \rangle \wedge \bigwedge_{\substack{n \in \text{path}(T,l) \\ x_i \geq d_i^n}} \neg\langle x_i < d_{i,n} \rangle \right) \leftrightarrow p_l.$$

to ensure that $p_l = 1$ if and only if l is the leaf that can be found by traversing its tree from the root following the feature values of the datapoint.

We mainly use $F_{\mathcal{T}}$ to check whether a fixed set $\mathcal{L}$ of leaves corresponds to some datapoint, i.e. whether $\mathcal{L} \subset \mathcal{L}(\mathcal{T}, \mathbf{x})$ for some point $\mathbf{x}$. This is achieved by invoking a SAT solver on $F_{\mathcal{T}}$ while assuming $p_l = 1$ for all $l \in \mathcal{L}$. If the solver reports SAT, the obtained assignment of the $\langle x_i < d_{i,j}\rangle$ variables can be used to construct such $\mathbf{x}$. If the solver reports UNSAT, the obtained unsatisfiable core corresponds to a subset $\mathcal{L}_s \subseteq \mathcal{L}$ such that no datapoint corresponds to all of the leaves in $\mathcal{L}_s$.

3 Checking candidate weak explanations

We detail the main contribution of our work, two new instantiations of the weak-explanation check used by the deletion-based minimization algorithm. One is inspired by the implicit hitting set (IHS) paradigm, and the other is based on branch and bound (B&B) search.

3.1 IHS for checking weak explanations

The IHS-inspired approach checks whether E is a weak explanation by independently searching, for each class $c \neq \mathcal{T}(\mathbf{x})$, for a counterexample on c. For each such class, the approach uses an integer program (IP) I^c_{opt} over the leaf indicator variables p_l. The constraints $\sum_{l \in T} p_l = 1$ for each $T \in \mathcal{T}$ enforce that exactly one leaf per tree is selected, and the IP maximizes the score difference $\sum_l (\text{score}_l(c) - \text{score}_l(\mathcal{T}(\mathbf{x})))p_l$. A solution with a positive objective value corresponds to a candidate counterexample on c; it is an actual counterexample if its leaves are reachable by some datapoint that agrees with $\mathbf{x}$ on E.

Algorithm 1 details the IHS-inspired weak-explanation check. For each class $c \neq \mathcal{T}(\mathbf{x})$, it first builds the propositional representation $F^c_{\mathcal{T},E}$ (with each feature in E fixed according to $\mathbf{x}$ by unit clauses) and the IP I^c_{opt} on Line 3. Since the check considers one class at a time, both encodings can be restricted to the trees that contain at least one leaf assigning a positive score to c or $\mathcal{T}(\mathbf{x})$. The inner loop on Lines 6-11 then iteratively refines I^c_{opt}: the optimizer proposes a candidate $\mathcal{L}$ with positive objective; the oracle checks via SAT under the assumptions $\{p_l = 1 \mid l \in \mathcal{L}\}$ whether $\mathcal{L}$ is reachable. If so, $\mathcal{L}$ is a counterexample and E is not a weak explanation (Line 10); otherwise, the returned unsatisfiable core $\mathcal{L}_C$ is blocked from future solutions on Line 11. The loop terminates for c when the optimizer is infeasible or all remaining solutions have non-positive objective. If no class yields a counterexample, E is a weak explanation (Line 12).

Algorithm 1 IHS for checking weak explanations.

```
1:  procedure IHS-CHECK(x, E, T, K)
2:      for c ∈ K \ {T(x)} do
3:          (F^c_{T,E}, I^c_opt) ← ENCODEENSEMBLE(x, E, T, c)
4:          oracle ← INITORACLE(F^c_{T,E})
5:          optimizer ← INITOPTIMIZER(I^c_opt)
6:          while true do
7:              (L, obj, feasible?) = optimizer.solve()
8:              if not feasible? or obj ≤ 0 then break
9:              (is_sat?, L_C) = oracle.check(L)
10:             if is_sat? then return false
11:             optimizer.add(Σ_{l∈L_C} p_l < |L_C|)
12:     return true
```

▶ **Example 2.** Invoke Algorithm 1 on the ensemble $\mathcal{T}$ in Figure 1, the datapoint $\mathbf{x} = (4, 3, 5)$, classes $\{c_1, c_2\}$, and a candidate explanation $E = \{2, 3\}$. Consider the iteration of the outer loop for the class $c = c_1$. Assume the optimizer returns $\mathcal{L}_1 = \{L_2, L_3, L_8\}$ on the first iteration of the inner loop. When invoked with the corresponding assumptions, the oracle can return the core $\{L_3\}$, since no datapoint that agrees with $\mathbf{x}$ on feature 3 corresponds to L_3, which adds $p_3 < 1$ to the optimizer. Assume the optimizer next returns $\mathcal{L}_2 = \{L_2, L_6, L_8\}$. The oracle can return the core $\{L_6, L_8\}$ since no datapoint corresponds to both L_6 and L_8 due to conflicting conditions on feature 1. The constraint $p_6 + p_8 < 2$ is then added to the optimizer. On the next iteration, the optimizer can propose $\mathcal{L}_3 = \{L_2, L_6, L_9\}$, for which the oracle returns is_sat?=true, indicating that $\mathcal{L}_3$ is a counterexample for E.

The pure-optimizer variant. We also consider a variant of the IHS approach, referred to as *pure-optimizer*, that eliminates the SAT oracle by encoding reachability directly into a single IP I_{opt}. Unlike Algorithm 1, I_{opt} combines all classes $c \neq \mathcal{T}(\mathbf{x})$ into one optimization problem: in addition to the leaf-selection constraints $\sum_{l \in T} p_l = 1$, it introduces for each class c a 0-1 indicator i_c with the reified constraint $i_c \Rightarrow \sum_l (\text{score}_l(c) - \text{score}_l(\mathcal{T}(\mathbf{x})))p_l > 0$, together with $\sum_{c \neq \mathcal{T}(\mathbf{x})} i_c = 1$, ensuring that any solution beats $\mathcal{T}(\mathbf{x})$ on some class. The propositional structure of $F_{\mathcal{T},E}$ is also encoded as linear constraints, so that any solution is reachable. The candidate E is then a weak explanation iff I_{opt} is infeasible, decided by a single IP solver call. The pure-optimizer variant thus differs from Algorithm 1 both in search structure (one solver call vs. a refinement loop) and in encoding (all classes combined in one IP vs. a separate IP per class).

3.2 Branch and bound for checking explanations

Our B&B-algorithm is based on the existence of a point $\mathbf{x}$ that reaches all of the leaves in $\mathcal{L}$ being equivalent to all bounds implied by the leaves in $\mathcal{L}$ being consistent with each other. In addition, the algorithm keeps track of the score difference between classes c and $\mathcal{T}(\mathbf{x})$.

Algorithm 2 details our B&B-based weak-explanation check. When checking a candidate E for the prediction $\mathcal{T}(\mathbf{x})$ the procedure loops over every class $c \neq \mathcal{T}(\mathbf{x})$ and invokes the DFS function to either compute a counterexample for E on c and return true, or determine that one does not exist and return false. If a counterexample is found, E is determined to not be a weak abductive explanation on Line 4. If the loop finishes without finding counterexamples on any class, E is determined to be a weak abductive explanation on Line 5.

Algorithm 2 Branch and bound for checking weak explanations.

```
1:  procedure BnB-check(x, E, T, K)
2:      for c ∈ K \ {T(x)} do
3:          if DFS(1, init_bounds(x, E), T, c, ∅) then
4:              return false
5:      return true
6:
7:  function DFS(d, bounds, T = {T_1 ..., T_m}, c, L)
8:      for l in leaves-of({T_d}) do
9:          (feas?, new_b) ← up_bounds(T_d, bounds, l)
10:         if not feas? then continue
11:         L_d ← L ∪ {l}
12:         cur_diff ← (Σ_{l'∈L_d} (score_{l'}(c) − score_{l'}(T(x))))
13:         b_diff ← ub_approx(T, c, new_b, T(x), d + 1)
14:         if cur_diff + b_diff < 0 then continue
15:         if |L_d| = m then return true
16:         else if DFS(d + 1, new_b, T, c, L_d) then return true
17:     return false
```

The recursive DFS subprocedure attempts to construct a counterexample $\mathcal{L}$ in a depth-first manner over the trees in $\mathcal{T}$. Each recursive call expects the current depth d, the set $\mathcal{L}$ of leaves selected so far, a map "bounds" that maintains the upper and lower bounds on each feature implied by the leaves in $\mathcal{L}$, the classifier $\mathcal{T}$ and the target class c. The set $\mathcal{L}$ of leaves is initialized to empty, and "bounds" to fix the value of each feature in E to its value in $\mathbf{x}$ and leave the rest unbounded.

On depth d, the DFS procedure loops over each leaf l of the d:th tree $T_d \in \mathcal{T}$ and attempts to extend $\mathcal{L}$ with l. First, the up_bounds checks whether the bounds implied by l on the features are consistent with the bounds implied by the other leaves in $\mathcal{L}$ and by E. If an inconsistency is detected, l is skipped on Line 10. If l is consistent with $\mathcal{L}$, the call also returns the updated bounds on all features stored in the new_b variable and the extended candidate counterexample $\mathcal{L}_\mathrm{d}$ is formed on Line 11. The rest of the loop approximates whether $\mathcal{L}_\mathrm{d}$ can be extended to include a leaf from each tree in a way that results in the sum of scores for c exceeding the sum of scores for $\mathcal{T}(\mathbf{x})$. The cur_dif variable (Line 12) contains the already incurred difference in these scores, and the b_diff variable (Line 13) an upper bound on the difference that can be incurred from the remaining trees. If the sum of the two is negative, $\mathcal{L}_\mathrm{d}$ can not be extended into an actual counterexample, so the branch is cut on Line 14. If $\mathcal{L}_\mathrm{d}$ contains a leaf from each tree in $\mathcal{T}$, it is a counterexample, so the recursion terminates and returns true on Line 16. Otherwise, the procedure recurses to the next tree.

The ub_approx procedure approximates an upper bound on the difference in score between c and $\mathcal{T}(\mathbf{x})$ in the remaining leaves by selecting from each remaining tree a leaf that is consistent with the bounds maintained in new_b and maximizes the difference in score between c and $\mathcal{T}(\mathbf{x})$. Note that the selected leaves need not be consistent with each other, which is why the result is an upper bound rather than the exact value.

▶ **Example 3.** Consider the ensemble shown in Figure 1, the datapoint $\mathbf{x} = (4, 3, 5)$, the target class c_1, and candidate explanation $E = \{2, 3\}$. Invoke the DFS procedure to compute a counterexample of E on c_1. Initially, the bounds for features 2 and 3 are fixed to 3 and 5, respectively. Assume that the leaf L_2 is picked on depth 1.

At depth 2, L_3 and L_4 are incompatible with the bounds fixed by E and $\mathbf{x}$. The leaf L_5 is deemed feasible, and the upper bound of feature 1 is updated to 6. However, as $\mathcal{L}_2 = \{L_2, L_5\}$ has cur_diff $= -1.06$ and b_diff $= 0.76$, the branch is pruned. The leaf L_6 is also deemed feasible, which updates the (branch-specific) lower bound on feature 1 to 6. Since $\mathcal{L}_2 = \{L_2, L_6\}$ has cur_diff $= -0.7$ and b_diff $= 0.76$, the search continues.

4 Experimental evaluation

4.1 Implementation details

We report on an empirical evaluation of three deletion-based algorithms, distinguished by their implementation of weak-explanation check:

- **IHS**: uses IHS-CHECK (Algorithm 1).
- **B&B**: uses BNB-CHECK (Algorithm 2).
- **Optimizer**: uses the pure-optimizer variant of IHS-CHECK.

All algorithms are implemented in C++17. **IHS** and **Optimizer** both use the Gurobi (v12.0.1) IP solver as the optimizer. **IHS** uses the CaDiCaL (v2.1.3) [3] SAT solver as the oracle. We reuse the constraint encoding of paths with similar conditions. In its configuration, **IHS** terminates the inner loop of Algorithm 1 when the optimizer reports an incumbent solution with a positive objective value, rather than running it to optimality. The **B&B** implementation is optimized over the pseudocode of Algorithm 2. Only one copy of the "bounds" array is used and updated between recursion levels. The counterexample is not explicitly constructed, but instead only the bounds array is kept. The results of score computations are passed between recursion depths for reuse, including the approximations computed by UB_APPROX which are only updated when needed.

As a central implementational detail, all our tools have a preprocessing step that prunes subtrees of the ensemble that can be determined to be unreachable based on the features in the candidate explanation and the datapoint to be explained. In practice this also means re-encoding the constraint representations of the ensemble whenever the candidate explanation changes, which requires e.g.. resetting the SAT and IP solvers.

4.2 Benchmarks and experimental setup

We consider two sets of benchmarks, the *medium* set and the *large* set. The medium set is based on 34 datasets featured in [9] and [13]. The number of features in these datasets ranges from 4 to 64, with an average of 24, and the number of classes ranges from 2 to 11, with an average of 3.7. For each dataset, we train a random forest using scikit-learn with 50 trees of depths ranging from 3 to 6, and a boosted tree ensemble using XGBoost with 50 trees of depth 3 or 4. We randomly sample 100 data points from each ensemble, resulting in a total of 6800 explanation tasks.

The large benchmark set is formed from the seven datasets in the medium set that could be used to train (in the number of nodes) larger tree ensembles. In addition, the Letter Recognition dataset from the UCI Machine Learning Repository is included to further diversify the benchmarks. Feature counts range from 5 to 40, with an average of 19, and class counts from 2 to 26, with an average of 6.3. For each of the eight datasets, we train two boosted tree models: one with 50 estimators (trees that can assign positive scores) per class and a maximum depth of 12, and another with 500 estimators per class and a maximum depth of 5. We also train two random forest models with the same respective configurations. For each model 25 datapoints are sampled randomly for 800 explanation tasks total.

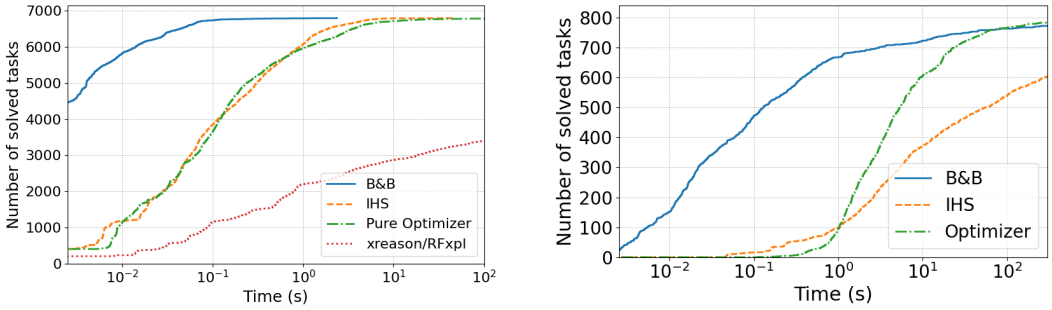

Figure 2 Scalability on the full medium (left) and large (right) benchmark sets. RFxpl was used for random forests and xreason for boosted ensembles, and the results were combined.

All experiments were run on 2.50-GHz Intel Xeon Gold 6248 machines with 381 GB RAM running RHEL, with 4 threads and a memory limit of 8 GB, using a per-instance timeout of 100 (300) seconds for the medium (large) set.

4.3 Results

Figure 2 details the scalability of our main approaches on the two benchmark sets. Overall, **B&B** performs well, being the fastest on 99.9% of medium-set benchmarks and 94.5% of large-set benchmarks. On the large set, **Optimizer** solved 784 out of 800 benchmarks, compared to 772 for **B&B** and 603 for **IHS**. In total, at least one method solved 787 benchmarks. **Optimizer** outperforms **IHS** on 53.5% of medium-set benchmarks and 81.9% of large-set benchmarks that were solved by at least one of the two.

Finally, we compare the tool we developed as part of this work with the existing state-of-the-art (SOTA) open source tools for computing an abductive explanation for tree ensembles: the xreason approach [9] for boosted trees (from https://github.com/alexeyignatiev/xreason/, commit 8240f35), and the RFxpl approach [11] for random forests (from https://github.com/izzayacine/RFxpl/, commit b07883d). To increase fairness in the comparison to the Python-based RFxpl and xreason, we report their runtime after discounting the time required for Python to form the CNF instance. We are unaware of any other purely C++ tools that compute abductive explanations for tree ensembles.

As shown in Figure 2, all variants in our tool significantly outperform xreason and RFxpl. **B&B** finishes 3397 out of the 3400 explanation tasks of boosted trees in the medium dataset in one second and all 3400 in one minute, compared to 2167 and 3272 finished by xreason, respectively. Similarly, **B&B** finishes 3362 of the benchmarks for random forests in the medium set in one second and all 3400 in one minute, compared to 1439 and 3295 finished by RFXpl, respectively.

5 Conclusion

We propose two new methods for computing minimal abductive explanations for tree ensembles, based on the implicit hitting set paradigm and on branch-and-bound search, and report on the results of an empirical evaluation. Our results indicate that the new approaches significantly improve the scalability of computing minimal abductive explanations, with the branch-and-bound method solving the large majority of benchmarks in under a second.

References

1. Gilles Audemard, Steve Bellart, Louenas Bounia, Frédéric Koriche, Jean-Marie Lagniez, and Pierre Marquis. Trading complexity for sparsity in random forest explanations. In *Proc. AAAI*, pages 5461–5469, 2022. doi:10.1609/AAAI.V36I5.20484.
2. Gilles Audemard, Jean-Marie Lagniez, Pierre Marquis, and Nicolas Szczepanski. Computing abductive explanations for boosted trees. In *Proc. AISTATS*, pages 4699–4711, 2023.
3. Armin Biere, Tobias Faller, Katalin Fazekas, Mathias Fleury, Nils Froleyks, and Florian Pollitt. CaDiCaL 2.0. In *Proc. CAV*, pages 133–152, 2024. doi:10.1007/978-3-031-65627-9_7.
4. Armin Biere, Marijn Heule, Hans van Maaren, and Toby Walsh, editors. *Handbook of Satisfiability – Second Edition*. IOS Press, 2021. doi:10.3233/FAIA336.
5. Vadim Borisov, Tobias Leemann, Kathrin Seßler, Johannes Haug, Martin Pawelczyk, and Gjergji Kasneci. Deep neural networks and tabular data: A survey. *IEEE Trans. Neural Networks Learn. Syst.*, 35(6):7499–7519, 2024. doi:10.1109/TNNLS.2022.3229161.
6. Leo Breiman. Random forests. *Mach. Learn.*, 45(1):5–32, 2001. doi:10.1023/A:1010933404324.
7. Jerome H. Friedman. Greedy function approximation: A gradient boosting machine. *Ann. Stat.*, 29(5):1189–1232, 2001.
8. Xuanxiang Huang and João Marques-Silva. On the failings of Shapley values for explainability. *Int. J. Approx. Reason.*, 171:109112, 2024. doi:10.1016/J.IJAR.2023.109112.
9. Alexey Ignatiev, Yacine Izza, Peter J. Stuckey, and João Marques-Silva. Using MaxSAT for efficient explanations of tree ensembles. In *Proc. AAAI*, pages 3776–3785, 2022. doi:10.1609/AAAI.V36I4.20292.
10. Alexey Ignatiev, Nina Narodytska, and João Marques-Silva. Abduction-based explanations for machine learning models. In *Proc. AAAI*, pages 1511–1519, 2019. doi:10.1609/AAAI.V33I01.33011511.
11. Yacine Izza, Alexey Ignatiev, Sasha Rubin, João Marques-Silva, and Peter J. Stuckey. Most general explanations of tree ensembles. In *Proc. IJCAI*, pages 5463–5471, 2025. doi:10.24963/ijcai.2025/608.
12. Yacine Izza, Alexey Ignatiev, Peter J. Stuckey, and João Marques-Silva. Delivering inflated explanations. In *Proc. AAAI*, pages 12744–12753, 2024. doi:10.1609/AAAI.V38I11.29170.
13. Yacine Izza and João Marques-Silva. On explaining random forests with SAT. In *Proc. IJCAI*, pages 2584–2591, 2021. doi:10.24963/IJCAI.2021/356.
14. Scott M. Lundberg and Su-In Lee. A unified approach to interpreting model predictions. In *Proc. NeurIPS*, pages 4765–4774, 2017.
15. João Marques-Silva, Thomas Gerspacher, Martin C. Cooper, Alexey Ignatiev, and Nina Narodytska. Explaining naive Bayes and other linear classifiers with polynomial time and delay. In *Proc. NeurIPS*, 2020.
16. João Marques-Silva and Alexey Ignatiev. Delivering trustworthy AI through formal XAI. In *Proc. AAAI*, pages 12342–12350, 2022. doi:10.1609/AAAI.V36I11.21499.
17. Erick Moreno-Centeno and Richard M. Karp. The implicit hitting set approach to solve combinatorial optimization problems with an application to multigenome alignment. *Oper. Res.*, 61(2):453–468, 2013. doi:10.1287/OPRE.1120.1139.
18. António Morgado, Carmine Dodaro, and João Marques-Silva. Core-guided MaxSAT with soft cardinality constraints. In *Proc. CP*, pages 564–573, 2014. doi:10.1007/978-3-319-10428-7_41.
19. Marco Túlio Ribeiro, Sameer Singh, and Carlos Guestrin. Anchors: High-precision model-agnostic explanations. In *Proc. AAAI*, pages 1527–1535, 2018. doi:10.1609/AAAI.V32I1.11491.
20. Cynthia Rudin. Stop explaining black box machine learning models for high stakes decisions and use interpretable models instead. *Nat. Mach. Intell.*, 1(5):206–215, 2019. doi:10.1038/S42256-019-0048-X.