arXiv:1906.03849v1 [cs.LG] 10 Jun 2019

Robustness Verification of Tree-based Models

Hongge Chen[*,1] Huan Zhang[*,2] Si Si[3] Yang Li[3] Duane Boning[1] Cho-Jui Hsieh[2]
[1]Department of EECS, MIT
[2]Department of Computer Science, UCLA
[3]Google Research
chenhg@mit.edu, huan@huan-zhang.com, sisidaisy@google.com
liyang@google.com, boning@mtl.mit.edu, chohsieh@cs.ucla.edu
[*]Hongge Chen and Huan Zhang contributed equally

Abstract

We study the robustness verification problem for tree based models, including decision trees, random forests (RFs) and gradient boosted decision trees (GBDTs). Formal robustness verification of decision tree ensembles involves finding the exact minimal adversarial perturbation or a guaranteed lower bound of it. Existing approaches find the minimal adversarial perturbation by a mixed integer linear programming (MILP) problem, which takes exponential time so is impractical for large ensembles. Although this verification problem is NP-complete in general, we give a more precise complexity characterization. We show that there is a simple linear time algorithm for verifying a single tree, and for tree ensembles the verification problem can be cast as a max-clique problem on a multi-partite graph with bounded boxicity. For low dimensional problems when boxicity can be viewed as constant, this reformulation leads to a polynomial time algorithm. For general problems, by exploiting the boxicity of the graph, we develop an efficient multi-level verification algorithm that can give tight lower bounds on robustness of decision tree ensembles, while allowing iterative improvement and any-time termination. On RF/GBDT models trained on 10 datasets, our algorithm is hundreds of times faster than the previous approach that requires solving MILPs, and is able to give tight robustness verification bounds on large GBDTs with hundreds of deep trees.

1 Introduction

Recent studies have demonstrated that neural network models are vulnerable to adversarial perturbations—a small and imperceptible-to-human input perturbation can easily change the predicted label [31, 15, 6, 13]. This has created serious security threats to many real applications so it becomes important to formally verify the robustness of machine learning models. Usually, the robustness verification problem can be cast as finding the minimal adversarial perturbation to an input example that can change the predicted class label. A series of robustness verification algorithms have been developed for neural network models [19, 32, 36, 35, 34, 38, 14, 29], where efficient algorithms are mostly based on relaxation or approximation of nonlinear activation functions of neural networks.

We study the robustness verification problem of tree-based models, including a single decision tree and tree ensembles such as random forests (RFs) and gradient boosted decision trees (GBDTs). These models have been widely used in practice and recent studies have demonstrated that both RFs and GBDTs are vulnerable to adversarial perturbations [18, 12, 8]. It is thus important to develop a formal robustness verification algorithm for tree-based models. Robustness verification requires computing the minimal adversarial perturbation. [18] showed that computing minimal adversarial perturbation for tree ensemble is NP-complete in general, and they proposed a Mixed-Integer Linear

Programming (MILP) approach to compute the minimal adversarial perturbation. Although exact verification is NP-hard, in order to have an efficient verification algorithm for real applications we seek to answer the following questions:

- Can we have efficient polynomial time algorithms for exact verification under some special circumstances?
- For general tree ensemble models with a large number of trees, can we efficiently compute a meaningful lower bounds on robustness while scaling to large tree ensembles?

In this paper, we answer the above-mentioned questions in the affirmative by formulating the verification problem of tree ensemble as a graph problem. First, we show that for a single decision tree, robustness verification can be done exactly in linear time. Then we show that for an ensemble of K trees, the verification problem is equivalent to finding the maximum cliques in a K-partite graph, and the graph is in a special form with boxicity equal to the input feature dimension. Therefore, for low-dimensional problems, verification can also be done in polynomial time with maximum clique searching algorithms. Finally, for large-scale tree ensembles, we propose a multiscale verification algorithm by exploiting the boxicity of the graph, which can give tight lower bounds on robustness. Furthermore, it supports any-time termination: we can stop the algorithm at any time to obtain a reasonable lower bound given a computation time constraint. Our proposed algorithm is efficient and is scalable to large tree ensemble models. For instance, on a large multi-class GBDT with 200 trees robustly trained on the MNIST dataset (using [8]), we obtained 78% verified robustness accuracy on test set with maximum ℓ_∞ perturbation $\epsilon = 0.2$ and the time used for verifying each test example is 12.6 seconds, whereas the MILP method uses around 10 min for each test example.

2 Background and Related Work

Adversarial Robustness For simplicity, we consider a multi-class classification model $f : \mathbb{R}^d \to \{1, \ldots, C\}$ where d is the input dimension and C is number of classes. For an input example x, assuming that $y_0 = f(x)$ is the correct label, the **minimal adversarial perturbation** is defined by

$$r^* = \min_{\delta} \|\delta\|_\infty \quad \text{s.t.} \quad f(x+\delta) \neq y_0. \tag{1}$$

Note that we focus on the ℓ_∞ norm measurement in this paper which is widely used in recent studies [22, 36, 5]. Exactly solving (1) is usually intractable. For example, if $f(\cdot)$ is a neural network, (1) is non-convex and [19] showed that solving (1) is NP-complete for ReLU networks.

Adversarial attacks are algorithms developed for finding a feasible solution $\bar{\delta}$ of (1), where $\|\bar{\delta}\|_\infty$ is an *upper bound* of r^*. Many algorithms have been proposed for attacking machine learning models [15, 20, 6, 22, 9, 10, 16, 3, 12, 24, 21]. Most practical attacks cannot reach the minimal adversarial perturbation r^* due to the non-convexity of (1). Therefore, attacking algorithms cannot provide any formal guarantee on model robustness [1, 33].

On the other hand, **robustness verification algorithms** are designed to find the exact value or a *lower bound* of r^*. An exact verifier needs to solve (1) to the global optimal, so typically we resort to relaxed verifiers that give lower bounds. When a verification algorithm finds a lower bound $\underline{r}$, it guarantees that no adversarial example exists within a radius $\underline{r}$ ball around x. This is important for deploying machine learning algorithms to safety-critical applications such as autonomous vehicles or aircraft control systems [19, 17].

For verification, instead of solving (1) we can also solve the following **decision problem of robustness verification**

$$\textit{Does there exist an } x' \in \mathrm{Ball}(x, \epsilon) \quad \textit{such that } f(x) \neq y_0? \tag{2}$$

Note that in our setting

$$\text{Ball}(x, \epsilon) := \{x' : \|x' - x\|_\infty \leq \epsilon\}$$

. If we can answer this decision problem, a binary search can give us the value of r^*, so the complexity of (2) is in the same order of (1). Furthermore, a safe answer to (2) (always say yes when unsure) will lead to a lower bound of r^*, which is what we want to do in verification. The decision version is also widely used in the verification community since people care about "robustness error at ϵ perturbation" which is defined to be the ratio of number of test samples that satisfy (2). Verification methods for neural networks have been studied extensively in the past few years [36, 37, 35, 38, 29, 14, 30, 2].

Adversarial Robustness of Tree-based Models Unlike neural networks, decision-tree based models are non-continuous step functions, and thus existing neural network verification algorithms cannot be directly applied. To evaluate the robustness of tree-based models, [18] showed that solving (1) for general tree ensemble models is NP-complete, so no polynomial time algorithm can compute r^* unless P=NP. A Mixed Integer Linear Programming (MILP) algorithm was thus proposed in [18] to compute (1) in exponential time. Some hard-label attacking algorithms for neural networks, including the boundary attack [3] and OPT-attack [12], can also be applied since they only require function evaluation of the non-smooth (hard-label) decision function $f(\cdot)$, and can be viewed as faster ways to compute an upper bound of r^*. To the best of our knowledge, **there is no prior existing algorithm for efficient verification, or equivalently, efficiently computing a lower bound of r^* for ensemble trees**.

3 Proposed Algorithm

In this section, we propose the first-ever tree ensemble verification algorithms. The tree ensemble exact verification problem is NP-complete by its nature, and here we propose a series of efficient verification algorithms for real applications. First, we will introduce a linear time algorithm for exactly computing the minimal adversarial distortion r^* for verifying a single decision tree. For an ensemble of trees, we cast the verification problem into a max-clique searching problem in K-partite graphs. For large-scale tree ensembles, we then propose an efficient multi-level algorithm for verifying an ensemble of decision trees.

3.1 Exactly Verifying a Single Tree in Linear Time

Although computing r^* for a tree ensemble is NP-complete [18], we show that a **linear time** algorithm exists for finding the minimum adversarial perturbation and computing r^* for a single decision tree. We assume the decision tree has n nodes and the root node is indexed as 0. For a given example $x = [x_1, \ldots, x_d]$ with d features, starting from the root, x traverses the decision tree model until reaching a leaf node. Each internal node, say node i, has two children and a feature-threshold pair (t_i, η_i) to determine the traversal direction—x will be passed to the left child if $x_{t_i} \leq \eta_i$ and to the right child otherwise. Each leaf node has a value v_i corresponding to the predicted class label for a classification tree, or a real value for a regression tree.

Conceptually, the main idea of our single tree verification algorithm is to compute a ***d*-dimensional box** for each leaf node such that any example in this box will fall into this leaf. Mathematically, the node i's box is defined as the Cartesian product $B^i = (l_1^i, r_1^i] \times \cdots \times (l_d^i, r_d^i]$ of d closed intervals on the real line. By definition, the root node has box $[-\infty, \infty] \times \cdots \times [-\infty, \infty]$ and given the box of an internal node i, its children's boxes can be obtained by changing only one interval of the box based on the split condition (t_i, η_i). More specifically, if p, q are node i's

left and right child node respectively, then we set their boxes $B^p = (l_1^p, r_1^p] \times \cdots \times (l_d^p, r_d^p]$ and $B^q = (l_1^q, r_1^q] \times \cdots \times (l_d^q, r_d^q]$ by setting

$$(l_t^p, r_t^p] = \begin{cases} (l_t^i, r_t^i] & \text{if } t \neq t_i \\ (l_t^i, \min\{r_t^i, \eta_i\}] & \text{if } t = t_i \end{cases}, \quad (l_t^q, r_t^q] = \begin{cases} (l_t^i, r_t^i] & \text{if } t \neq t_i \\ (\max\{l_t^i, \eta_i\}, r_t^i] & \text{if } t = t_i. \end{cases} \tag{3}$$

After computing the boxes for internal nodes, we can also obtain the boxes for leaf nodes using (3). Therefore computing the boxes for all the leaf nodes of a decision tree can be done by a depth-first search traversal of the tree with time complexity $O(nd)$.

With the boxes computed for each leaf node, the minimum perturbation required to change x to go to a leaf node i can be written as a vector $\boldsymbol{\epsilon}(x, B^i) \in \mathbb{R}^d$ defined as

$$\boldsymbol{\epsilon}(x, B^i)_t := \begin{cases} 0 & \text{if } x_t \in (l_t^i, r_t^i] \\ x_t - r_t^i & \text{if } x_t > r_t^i \\ l_t^i - x_t & \text{if } x_t \leq l_t^i. \end{cases} \tag{4}$$

Then the minimal distortion can be computed as $r^* = \min_{i:v_i \neq y_0} \|\boldsymbol{\epsilon}(x, B^i)\|_\infty$, where y_0 is the original label of x, and v_i is the label for leaf node i. To find r^*, we check B^i for all leaves and choose the smallest perturbation. This is a linear-time algorithm for exactly verifying the robustness of a single decision tree.

In fact, this $O(nd)$ time algorithm is used to illustrate the concept of "boxes" that will be used later on for the tree ensemble case. If our final goal is to verify a single tree, we can have a more efficient algorithm by combining the distance computation (4) in the tree traversal procedure, and the resulting algorithm will take only $O(n)$ time. This algorithm is presented as Algorithm 1 in the appendix.

3.2 Verifying Tree Ensembles by Max-clique Enumeration

Now we discuss the robustness verification for tree ensembles. Assuming the tree ensemble has K decision trees, we use $S^{(k)}$ to denote the set of leaf nodes of tree k and $m^{(k)}(x)$ to denote the function that maps the input example x to the leaf node of tree k according to its traversal rule. Given an input example x, the tree ensemble will pass x to each of these K trees independently and x reaches K leaf nodes $i^{(k)} = m^{(k)}(x)$ for all $k = 1, \ldots, K$. Each leaf node will assign a prediction value $v_{i^{(k)}}$. For simplicity we start with the binary classification case, with x's original label being $y_0 = -1$ and we want to turn it into $+1$. For binary classification the prediction of the tree ensemble is computed by $\text{sign}(\sum_k v_{i^{(k)}})$, which covers both GBDTs and random forests, two widely used tree ensemble models. Assume x has the label $y_0 = -1$, that means $\text{sign}(\sum_k v_{i^{(k)}}) < 0$ for x, and our task is to verify if the sign of the summation can be flipped within $\text{Ball}(x, \epsilon)$.

We consider the decision problem of robustness verification (2). A naive analysis will need to check all the points in $\text{Ball}(x, \epsilon)$ which is uncountably infinite. To reduce the search space to finite, we start by defining some notation: let $\mathbb{C} = \{(i^{(1)}, \ldots, i^{(K)}) \mid i^{(k)} \in S^{(k)},\ \forall k = 1, \ldots, L\}$ to be all the possible tuples of leaf nodes and let $\mathcal{C}(x) = [m^{(1)}(x), \ldots, m^{(K)}(x)]$ be the function that maps x to the corresponding leaf nodes. Therefore, a tuple $C \in \mathbb{C}$ directly determines the model prediction $\sum v_C := \sum_k v_{i^{(k)}}$. Now we define a valid tuple for robustness verification:

Definition 1. *A tuple $C = (i^{(1)}, \ldots, i^{(K)})$ is valid if and only if there exists an $x' \in Ball(x, \epsilon)$ such that $C = \mathcal{C}(x')$.*

The decision problem of robustness verification (2) can then be written as:

$$\textit{Does there exist a valid tuple } C \textit{ such that } \sum v_C > 0 \textit{?}$$

Next, we show how to model the set of valid tuples. We have two observations. First, if a tuple contains any node i with $\inf_{x' \in B^i}\{\|x - x'\|_\infty\} > \epsilon$, then it will be invalid. Second, there exists an x such that $C = \mathcal{C}(x)$ if and only if $B^{i^{(1)}} \cap \cdots \cap B^{i^{(K)}} \neq \emptyset$, or equivalently:

$$(l_t^{i^{(1)}}, r_t^{i^{(1)}}] \cap \cdots \cap (l_t^{i^{(K)}}, r_t^{i^{(K)}}] \neq \emptyset, \quad \forall t = 1, \ldots, d.$$

We show that the set of valid tuples can be represented as cliques in a graph $G = (V, E)$, where $V := \{i | B^i \cap \text{Ball}(x, \epsilon) \neq \emptyset\}$ and $E := \{(i, j) | B^i \cap B^j \neq \emptyset\}$. In this graph, nodes are the leaves of all trees and we remove every leaf that has empty intersection with $\text{Ball}(x, \epsilon)$. There is an edge (i, j) between node i and j if and only if their boxes intersect. The graph will then be a K-partite graph since there cannot be any edge between nodes from the same tree, and thus maximum cliques in this graph will have K nodes. We define each part of the K-partite graph as V_k. Here a "part" means a disjoint and independent set in the K-partite graph. The following lemma shows that intersections of boxes have very nice properties:

Lemma 1. *For boxes $B^1, \ldots, B^K$, if $B^i \cap B^j \neq \emptyset$ for all $i, j \in [K]$ then $B^1 \cap B^2 \cap \cdots \cap B^K \neq \emptyset$ and their intersection will also be an nonempty box: $\bar{B} = B^1 \cap B^2 \cap \cdots \cap B^K$.*

The proof can be found in the appendix. Based on the above lemma, each K-clique (fully connected subgraph with K nodes) in G can be viewed as a set of leaf nodes that has nonempty intersection with each other and also have nonempty intersection with $\text{Ball}(x, \epsilon)$, so the intersection of those K boxes and $\text{Ball}(x, \epsilon)$ will be a nonempty box, which implies each K-clique corresponds to a valid tuple of leaf nodes:

Lemma 2. *A tuple $C = (i^{(1)}, \ldots, i^{(K)})$ is valid if and only if nodes $i^{(1)}, \ldots, i^{(K)}$ form a K-clique (maximum clique) in graph G constructed above.*

Therefore the robustness verification problem can be formulated as

$$\textit{Is there a maximum clique } C \textit{ in } G \textit{ such that } \sum v_C > 0 \textit{?} \tag{5}$$

This reformulation indicates that the tree ensemble verification problem can be solved by an efficient maximum clique enumeration algorithm. Some standard maximum clique searching algorithms can be applied here to perform verification:

- **Finding K-cliques in K-partite graphs:** Any algorithm for finding all the maximum cliques in G can be used. The classic B-K backtracking algorithm [4] takes $O(3^{\frac{m}{3}})$ time to find all the maximum cliques where m is the number of nodes in G. Furthermore, since our graph is a K-partite graph, we can apply some specialized algorithms designed for finding all the K-cliques in K-partite graphs [23, 25, 28].
- **Polynomial time algorithms exist for low-dimensional problems:** Another important property for graph G is that each node in G is a d-dimensional box and each edge indicates intersection of two boxes. This implies our graph G is with "boxicity d" (see [7] for detail). [7] proved that the number of maximum cliques will only be $O((2m)^d)$ and it is able to find the maximum weight clique in $O((2m)^d)$ time. Therefore, for problems with very small d, the time complexity for verification is actually polynomial.

Therefore we can exactly solve the tree ensemble verification problem using algorithms for maximum cliques searching in K-partite graph, and its time complexity is found to be as follows:

Theorem 1. *Exactly verifying the robustness of a K-tree ensemble with at most n leaves per tree and d dimensional features takes* $\min\{O(n^K), O((2Kn)^d)\}$ *time.*

This is a direct consequence of the fact that the number of K-cliques in a K-partite graph with n vertices per part is bounded by $O(n^K)$, and number of maximum cliques in a graph with a total

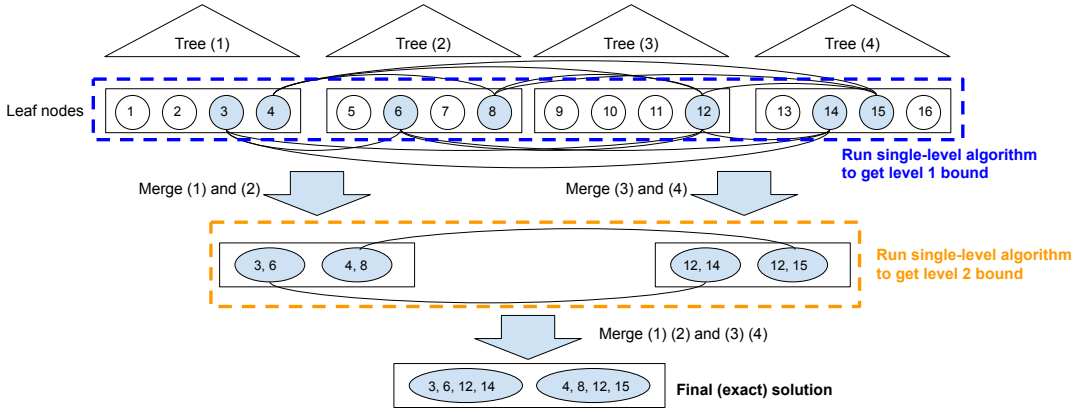

Figure 1: The proposed multi-level verification algorithm. Lines between leaf node i on tree t_1 and leaf node j on t_2 indicate that their ℓ_∞ feature boxes intersect (i.e., there exists an input such that tree 1 predicts v_i and tree 2 predicts v_j).

of m vertices with boxicity d is $O((2m)^d)$. For a general graph, since K and d can be in $O(n)$ and $O(m)$ [27], it can still be exponential. But the theorem gives a more precise characterization for the complexity of the verification problem for tree ensembles.

Based on the nice properties of maximum cliques searching problem, we propose a simple and elegant algorithm that enumerates all K-cliques on a K-partite graph with a known boxicity d in Algorithm 2 in the appendix, and we can use this algorithm for tree ensemble verification when the number of trees or the dimension of features is small.

3.3 An Efficient Multi-level Algorithm for Verifying the Robustness of a Tree Ensemble

Practical tree ensembles usually have tens or hundreds of trees and with large dimensions, so Algorithm 2 will take exponential time and will be too slow. We thus develop an efficient multi-level algorithm for computing verification bounds by further exploiting the boxicity of the graph.

Figure 1 illustrates the graph and how our multilevel algorithm runs. There are four trees and each tree has four leaf nodes. A node is colored if it has nonempty intersection with $\text{Ball}(x, \epsilon)$; uncolored nodes are discarded. To answer question (5), we need to compute the maximum $\sum v_C$ among all K-cliques, denoted by v^*. As mentioned before, for robustness verification we only need to compute an upper bound of v^* in order to get a lower bound of minimal adversarial robustness. In the following, we will first discuss algorithms for computing an upper bound at the top level, and then show how our multi-scale algorithm iterative refines this bound until reaching the exact solution v^*.

Bounds for a single level. To compute an upper bound of v^*, a naive approach is to assume that the graph is fully connected between independent sets (fully connected K-partite graph) and in this case the maximum sum of node values will be the sum of the maximum value of each independent set:

$$\sum_{k=1}^{K} \max_{i \in S^{(k)}} v_i \geq v^*. \tag{6}$$

One can easily show this is an upper bound of v^* since any original K-clique will still be considered when we add more edges to the graph.

Another slightly better approach is to exploit the edge information but only between tree t and $t+1$. If we search over all the length-K paths $[i^{(1)}, \ldots, i^{(K)}]$ from the first to the last group and

define the value of a path to be $\sum_k v_{i^{(k)}}$, then the maximum valued path will be a upper bound of v^*. This can be computed in linear time using dynamic programming. We scan nodes from tree 1 to tree K, and for each node we store a value d_i which is the maximum value of paths from tree 1 to this node. At tree k and node i, the d_i value can be computed by

$$d_i = v_i + \max_{j: j \in S^{(k-1)} \text{ and } (j,i) \in E} d_j. \tag{7}$$

Taking the max d value in the last tree will give us a tighter upper bound of v^*.

Merging T independent sets Now we try to refine our bound. Our approach is to partition the graph into K/T groups, each with T independent sets. Within each group, we find all the T-cliques and use a new "pseudo node" to represent each T-clique. T-cliques in a K-partite graph can be enumerated efficiently if we choose T to be a relative small number (e.g., 2 or 3 in the experiments).

Now we exploit the boxicity property of our graph to form a graph among these cliques (illustrated as the second level nodes in Figure 1). By Lemma 1, we know that the intersection of T boxes will still be a box, so each T-clique is still a box and can be represented as a pseudo node in the level-2 graph. Also because each pseudo node is still a box, we can easily form edges between pseudo nodes to indicate the nonempty overlapping between them and this will be a K/T-partite boxicity graph since no edge can be formed for the cliques within the same group. Thus we get the level-2 graph. With the level-2 graph, we can again run the single level algorithm to compute a upper bound on v^* to get a lower bound of r^* in (1), but different from the level-1 graph, now we already considered all the within-group edges so the value we get will be less or equal to the level-1 bound, which means tighter.

The overall multi-level framework We can run the algorithm level by level until merging all the groups into one, and in the final level the pseudo nodes will correspond to the K-cliques in the original graph, and the maximum value will be exactly v^*. Therefore, our algorithm can be viewed as an anytime algorithm that refines the upper bound level-by-level until reaching the maximum value. Although getting to the final level still requires exponential time, in practice we can stop at any level (denoted as L) and get a reasonable bound. In experiments, we will show that by merging few trees we already get a bound very close to the final solution. Algorithm 3 in the appendix gives the complete procedure.

Handling multi-class tree ensembles For a multiclass classification problem, say C-class classification problem, C groups of tree ensembles and each with K trees are built for the classification task; for the k-th tree in group c, prediction outcome is denoted as $i^{(k,c)} = m^{(k,c)}(x)$ where $m^{(k,c)}(x)$ is the function that maps the input example x to a leaf node of tree k in group c. The final prediction is given by $\text{argmax}_c \sum_k v_{i^{(k,c)}}$. Given an input example x with ground-truth class c and an attack target class c', we extract $2K$ trees for class c and class c', and flip the sign of all prediction values for trees in group c', such that initially $\sum_t v_{i^{(t,c)}} + \sum_t v_{i^{(t,c')}} < 0$ for a correctly classified example. Then, we are back to the binary case with $2K$ trees, and can still apply our multi-level framework to obtain a lower bound $\underline{r}_{(c,c')}$ of $r^*_{(c,c')}$ for this target attack pair (c, c'). Robustness of an untargeted attack can be evaluated by taking $\underline{r} = \min_{c' \neq c} \underline{r}_{(c,c')}$.

3.4 Verification Problems Beyond Ordinary Robustness

The above discussions focus on the decision problem of ℓ_∞ robustness verification (2). In fact, our approach works for a more general verification problem for *any* d-dimensional box B:

$$\textit{Is there any } x' \in B \textit{ such that } f(x) \neq y_0 \textit{?} \tag{8}$$

In typical robustness verification settings, B is defined to be Ball(x, ϵ) but in fact we can allow any boxes in our algorithm. For a general B, Lemma 1 still holds so all of our algorithms and analysis can go through. The only thing we need to change is to compute the intersection between B and each box of leaf node at the first level in Figure 1 to eliminate nodes that have empty intersection with B. So robustness verification is just a special case where we remove all the nodes with empty intersection with Ball(x, ϵ). For example, if we want to identify a set S of unimportant variables, where any change in S cannot alter the prediction for a given sample x, then we can choose B as $B_i = [-\infty, \infty]$ if $i \in S$ and $B_i = \{x_i\}$ otherwise. Similarly, we can also compute a set of anchor features (similar to [26]) such that once a set of features are fixed, any perturbation outside the set cannot change the prediction.

4 Experiments

We evaluate our proposed method for ensemble tree robustness verification on two tasks: binary and multiclass classification on 10 public datasets including both small and large scale datasets[1]. The statistics of the data sets are shown in the appendix. As we defined in Section 2, r^* is the radius of minimum adversarial perturbation that reflects true model robustness, but is hard to obtain; our method finds $\underline{r}$ that is a lower bound of r^*, which guarantees that *no adversarial example* exists within radius $\underline{r}$. A high quality lower bound $\underline{r}$ should be close to r^*. We include the following algorithms in our comparisons:

- Cheng's attack [12] provides results on adversarial attacks on these models, which gives an upper bound of the model robustness r^*. We denote it as $\overline{r}$ and $\overline{r} \geq r^*$.
- MILP: an MILP based method [18] gives the exact r^*. MILP is proposed for adversarial attacks on tree ensembles and thus $r^* \geq \underline{r}$ always holds for all lower bounds. MILP needs to solve a Mixed Integer Linear Program to find r^* and runs in exponential time, and is slow when the number of trees or dimension of the features increases.
- LP relaxed MILP: a Linear Programming (LP) relaxed MILP formulation by directly changing all binary variables to continuous ones. Since the constraints are removed, solving the minimization of MILP gives a lower bound of robustness, $\underline{r}_{LP}$, serving as a baseline method.
- Our proposed multi-level verification framework in Section 3.3 (with pseudo code as Algorithm 3 in the appendix). We are targeting to compute robustness interval $\underline{r}_{ours}$ for tree ensemble verification.

In Tables 1 and 2 we show empirical comparisons on these 10 datasets. We consider ℓ_∞ robustness, and normalize our datasets to [0, 1] such that perturbations on different datasets are comparable. We include both standard (naturally trained) GBDT models (Table 1) and robust GBDT models (Table 2) in [8]. The robust GBDTs try to optimize the model performance under the worst-case perturbation of input features, which leads to a max-min saddle point problem when finding the split at each node. All GBDTs are implemented under XGBoost framework [11]. The number of trees in GBDT and parameters used in training GBDT for different datasets are shown in Table 3 in the appendix. Because we solve the decision problem of robustness verification, we use 10 times binary search to find the largest $\underline{r}$ in all experiments, and the reported time is the total time including all binary search trials. We present the average of $\underline{r}$ or r^* over 500 examples. The MILP based method from [18] is an accurate but very slow method; the results marked with an asterisk ("*") in the table have very long running time and thus we only evaluate 50 examples instead of 500.

From Tables 1 and 2 we can see that our method gives a tight lower bound $\underline{r}$ compared to r^* from MILP, while achieving up to $\sim$ 3000X speedup on large models. The running time of the baseline LP relaxation, however, is on the same order of magnitude as the MILP method, but the

[1] Our code (XGBoost compatible) is available at https://github.com/chenhongge/treeVerification.

Dataset	Cheng's attack [12]		MILP [18]		LP relaxation		Ours				Ours vs. MILP	
	avg. $\bar{r}$	avg. time	avg. r^*	avg. time	avg. $\underline{r}_{LP}$	avg. time	T	L	avg. r^*	avg. time	$\underline{r}_{our}/r^*$	speedup
breast-cancer	.221	2.18s	.210	.012s	.064	.009s	2	1	.208	.001s	.99	12X
covtype	.058	4.76s	.028*	355*s	.005*	154*s	2	3	.022	3.39s	.79	105X
cod-rna	.054	2.13s	.035	.485s	.017	.222s	2	3	.033	.059s	.94	8.2X
diabetes	.064	1.70s	.049	.061s	.015	.026s	3	2	.042	.018s	.86	3.4X
Fashion-MNIST	.048	12.2s	.014*	1150*s	.003*	898*s	2	1	.012	11.8s	.86	97X
HIGGS	.015	3.80s	.0028*	68*min	.00035*	50*min	4	1	.0022	1.29s	.79	3163X
ijcnn1	.047	2.72s	.030	4.64s	.008	2.67s	2	2	.026	.101s	.87	4.6X
MNIST	.070	11.1s	.011*	367*s	.003*	332*s	2	2	.011	5.14s	1.00	71X
webspam	.027	5.83s	.00076	47.2s	.0002	39.7s	2	1	.0005	.404s	.66	117X
MNIST 2 vs. 6	.152	12.0s	.057	23.0s	.016	11.6s	4	1	.046	.585s	.81	39X

Table 1: Average ℓ_∞ distortion over 500 examples and average verification time per example for three verification methods. Here we evaluate the bounds for **standard (natural) GBDT models**. Results marked with an asterisk ("*") are the averages of 50 examples due to long running time. T is the number of independent sets and L is the number of levels in searching cliques used in our algorithm. A ratio $\underline{r}_{our}/r^*$ close to 1 indicates better lower bound quality.

Dataset	Cheng's attack [12]		MILP [18]		LP relaxation		Ours				Ours vs. MILP	
	avg. $\bar{r}$	avg. time	avg. r^*	avg. time	avg. $\underline{r}_{LP}$	avg. time	T	L	avg. r^*	avg. time	$\underline{r}_{our}/r^*$	speedup
breast-cancer	.404	1.96s	.400	.009s	.078	.008s	2	1	.399	.001s	1.00	9X
covtype	.079	.481s	.046*	305*s	.0053*	159*s	2	3	.032	4.84s	.70	63X
cod-rna	.062	2.02s	.055	.607s	.017	.410s	2	3	.052	.104s	.95	5.8X
diabetes	.137	1.52s	.112	.034s	.035	.013s	3	2	.109	.006s	.97	5.7X
Fashion-MNIST	.153	13.9s	.091*	41*min	.009*	34*min	2	1	.071	18.0s	.78	137X
HIGGS	.023	3.58s	.0084*	59*min	.00031*	54*min	4	1	.0063	1.41s	.75	2511X
ijcnn1	.054	2.63s	.036	2.52s	.009	1.26s	2	2	.032	0.58s	.89	4.3X
MNIST	.367	1.41s	.264*	615*s	.019*	515*s	2	2	.253	12.6s	.96	49X
webspam	.048	4.97s	.015	83.7s	.0024	60.4s	2	1	.011	.345s	.73	243X
MNIST 2 vs. 6	.397	17.2s	.313	91.5s	.039	40.0s	4	1	.308	3.68s	.98	25X

Table 2: Average ℓ_∞ distortion over 500 examples and average verification time per example for three verification methods and an attack method. Here we evaluate the bounds for **robustly trained models** introduced in [8]. Results marked with an asterisk ("*") are the averages of 50 examples due to long running time. T is the number of independent sets and L is the number of levels in searching cliques used in our algorithm. A ratio $\underline{r}_{our}/r^*$ close to 1 indicates better lower bound quality.

results are much worse, with $\underline{r}_{LP} \ll r^*$. Our proposed method given as Algorithm 3 is a multi-scale any-time approach, that is, it can stop at any scale to get a reasonable robustness bound. Figure 2 shows how the tightness of our robustness verification lower bounds changes with different size of clique per level (T) and different number of levels (L). We test on a 20-tree standard GBDT model on the diabetes dataset. Here we also show the exact bound r^* by the MILP method. Our verification bound converges to the MILP bound as more levels of clique enumerations are used. Also, when we find larger cliques within each level, the bound becomes tighter.

To show the scalability of our method, we vary the number of trees in GBDT and compare per example running time with MILP method on ijcnn1 dataset in Figure 3. We see that our proposed methods spend much less time on each example compared to the MILP method and their running time grows slower than MILP when the number of trees increases.

In Section 3.4, we showed that the proposed algorithm works for a more general verification problem such as how to identify unimportant features in the data, where any change on those features cannot change the prediction. Here we use MNIST data as an example for this task where we perturb pixels (features) on MNIST test images on both standard and robustly trained

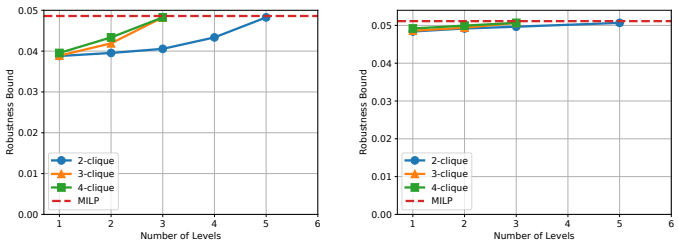

Figure 2: Robustness bounds obtained with different parameters ($T = \{2, 3, 4\}$, $L = \{1, \cdots, 6\}$) on a 20-tree standard GBDT model with the diabetes dataset (left) and a 20-tree robust GBDT model with the ijcnn1 dataset (right). $\underline{r}_{our}$ converges to r^* as L increases.

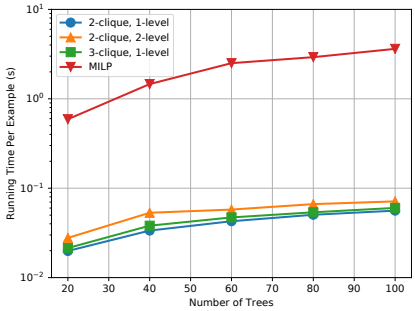

Figure 3: Running time of MILP and our method on GBDT model with different number of trees on the ijcnn1 dataset.

multi-class decision trees with depth 8. In Figure 4, yellow pixels cannot change prediction for any perturbation and a darker pixel represents a smaller lower bound of perturbation to change the model output using that pixel. The standard naturally trained model has some very dark pixels compared to the robust model.

Figure 4: Identifying important pixels on 3 MNIST images. Left: digit image; Middle: standard GBDT model; Right: robust GBDT model. Changing one of any yellow pixels cannot change prediction; pixels in darker colors tend to affect model prediction more than pixels in lighter colors.

References

[1] Anish Athalye, Nicholas Carlini, and David Wagner. Obfuscated gradients give a false sense of security: Circumventing defenses to adversarial examples. In *ICML*, 2018.

[2] Osbert Bastani, Yewen Pu, and Armando Solar-Lezama. Verifiable reinforcement learning via policy extraction. In *Advances in Neural Information Processing Systems*, pages 2494–2504, 2018.

[3] Wieland Brendel, Jonas Rauber, and Matthias Bethge. Decision-based adversarial attacks: Reliable attacks against black-box machine learning models. In *ICLR*, 2018.

[4] Coen Bron and Joep Kerbosch. Algorithm 457: finding all cliques of an undirected graph. *Communications of the ACM*, 16(9):575–577, 1973.

[5] Rudy R Bunel, Ilker Turkaslan, Philip Torr, Pushmeet Kohli, and Pawan K Mudigonda. A unified view of piecewise linear neural network verification. In *Advances in Neural Information Processing Systems*, pages 4790–4799, 2018.

[6] Nicholas Carlini and David Wagner. Towards evaluating the robustness of neural networks. In *2017 IEEE Symposium on Security and Privacy (SP)*, pages 39–57. IEEE, 2017.

[7] L Sunil Chandran, Mathew C Francis, and Naveen Sivadasan. Geometric representation of graphs in low dimension using axis parallel boxes. *Algorithmica*, 56(2):129, 2010.

[8] Hongge Chen, Huan Zhang, Duane Boning, and Cho-Jui Hsieh. Robust decision trees against adversarial examples. In *ICML*, 2019.

[9] Pin-Yu Chen, Yash Sharma, Huan Zhang, Jinfeng Yi, and Cho-Jui Hsieh. Ead: elastic-net attacks to deep neural networks via adversarial examples. In *Thirty-second AAAI conference on artificial intelligence*, 2018.

[10] Pin-Yu Chen, Huan Zhang, Yash Sharma, Jinfeng Yi, and Cho-Jui Hsieh. Zoo: Zeroth order optimization based black-box attacks to deep neural networks without training substitute models. In *Proceedings of the 10th ACM Workshop on Artificial Intelligence and Security*, pages 15–26. ACM, 2017.

[11] Tianqi Chen and Carlos Guestrin. Xgboost: A scalable tree boosting system. In *Proceedings of the 22nd acm sigkdd international conference on knowledge discovery and data mining*, pages 785–794. ACM, 2016.

[12] Minhao Cheng, Thong Le, Pin-Yu Chen, Jinfeng Yi, Huan Zhang, and Cho-Jui Hsieh. Query-efficient hard-label black-box attack: An optimization-based approach. In *ICLR*, 2019.

[13] Kevin Eykholt, Ivan Evtimov, Earlence Fernandes, Bo Li, Amir Rahmati, Chaowei Xiao, Atul Prakash, Tadayoshi Kohno, and Dawn Song. Robust physical-world attacks on deep learning models. *arXiv preprint arXiv:1707.08945*, 2017.

[14] Timon Gehr, Matthew Mirman, Dana Drachsler-Cohen, Petar Tsankov, Swarat Chaudhuri, and Martin Vechev. Ai2: Safety and robustness certification of neural networks with abstract interpretation. In *2018 IEEE Symposium on Security and Privacy (SP)*, pages 3–18. IEEE, 2018.

[15] Ian Goodfellow, Jonathon Shlens, and Christian Szegedy. Explaining and harnessing adversarial examples. In *International Conference on Learning Representations*, 2015.

[16] Andrew Ilyas, Logan Engstrom, Anish Athalye, and Jessy Lin. Black-box adversarial attacks with limited queries and information. In *International Conference on Machine Learning*, pages 2142–2151, 2018.

[17] Kyle D Julian, Shivam Sharma, Jean-Baptiste Jeannin, and Mykel J Kochenderfer. Verifying aircraft collision avoidance neural networks through linear approximations of safe regions. *arXiv preprint arXiv:1903.00762*, 2019.

[18] Alex Kantchelian, JD Tygar, and Anthony Joseph. Evasion and hardening of tree ensemble classifiers. In *International Conference on Machine Learning*, pages 2387–2396, 2016.

[19] Guy Katz, Clark Barrett, David L Dill, Kyle Julian, and Mykel J Kochenderfer. Reluplex: An efficient smt solver for verifying deep neural networks. In *International Conference on Computer Aided Verification*, pages 97–117. Springer, 2017.

[20] Alexey Kurakin, Ian Goodfellow, and Samy Bengio. Adversarial machine learning at scale. *arXiv preprint arXiv:1611.01236*, 2016.

[21] Yanpei Liu, Xinyun Chen, Chang Liu, and Dawn Song. Delving into transferable adversarial examples and black-box attacks. *arXiv preprint arXiv:1611.02770*, 2016.

[22] Aleksander Madry, Aleksandar Makelov, Ludwig Schmidt, Dimitris Tsipras, and Adrian Vladu. Towards deep learning models resistant to adversarial attacks. *arXiv preprint arXiv:1706.06083*, 2017.

[23] Mohammad Mirghorbani and P Krokhmal. On finding k-cliques in k-partite graphs. *Optimization Letters*, 7(6):1155–1165, 2013.

[24] Nicolas Papernot, Patrick McDaniel, Ian Goodfellow, Somesh Jha, Z Berkay Celik, and Ananthram Swami. Practical black-box attacks against machine learning. In *Proceedings of the 2017 ACM on Asia conference on computer and communications security*, pages 506–519. ACM, 2017.

[25] Charles A Phillips, Kai Wang, Erich J Baker, Jason A Bubier, Elissa J Chesler, and Michael A Langston. On finding and enumerating maximal and maximum k-partite cliques in k-partite graphs. *Algorithms*, 12(1):23, 2019.

[26] Marco Tulio Ribeiro, Sameer Singh, and Carlos Guestrin. Anchors: High-precision model-agnostic explanations. In *Thirty-Second AAAI Conference on Artificial Intelligence*, 2018.

[27] Fred S Roberts. On the boxicity and cubicity of a graph. *Recent Progresses in Combinatorics*, pages 301–310, 1969.

[28] Markus Schneider and Burkhard Wulfhorst. Cliques in k-partite graphs and their application in textile engineering. 2002.

[29] Gagandeep Singh, Timon Gehr, Matthew Mirman, Markus Püschel, and Martin Vechev. Fast and effective robustness certification. In *Advances in Neural Information Processing Systems*, pages 10802–10813, 2018.

[30] Gagandeep Singh, Timon Gehr, Markus Püschel, and Martin Vechev. An abstract domain for certifying neural networks. *Proceedings of the ACM on Programming Languages*, 3(POPL):41, 2019.

[31] Christian Szegedy, Wojciech Zaremba, Ilya Sutskever, Joan Bruna, Dumitru Erhan, Ian Goodfellow, and Rob Fergus. Intriguing properties of neural networks. *arXiv preprint arXiv:1312.6199*, 2013.

[32] Vincent Tjeng, Kai Xiao, and Russ Tedrake. Evaluating robustness of neural networks with mixed integer programming. *arXiv preprint arXiv:1711.07356*, 2017.

[33] Jonathan Uesato, Brendan O'Donoghue, Aaron van den Oord, and Pushmeet Kohli. Adversarial risk and the dangers of evaluating against weak attacks. *arXiv preprint arXiv:1802.05666*, 2018.

[34] Shiqi Wang, Yizheng Chen, Ahmed Abdou, and Suman Jana. Mixtrain: Scalable training of formally robust neural networks. *arXiv preprint arXiv:1811.02625*, 2018.

[35] Tsui-Wei Weng, Huan Zhang, Hongge Chen, Zhao Song, Cho-Jui Hsieh, Luca Daniel, Duane Boning, and Inderjit Dhillon. Towards fast computation of certified robustness for relu networks. In *International Conference on Machine Learning*, pages 5273–5282, 2018.

[36] Eric Wong and J Zico Kolter. Provable defenses against adversarial examples via the convex outer adversarial polytope. In *International Conference on Machine Learning*, 2018.

[37] Eric Wong, Frank Schmidt, Jan Hendrik Metzen, and J Zico Kolter. Scaling provable adversarial defenses. In *Advances in Neural Information Processing Systems*, pages 8400–8409, 2018.

[38] Huan Zhang, Tsui-Wei Weng, Pin-Yu Chen, Cho-Jui Hsieh, and Luca Daniel. Efficient neural network robustness certification with general activation functions. In *Advances in Neural Information Processing Systems*, pages 4939–4948, 2018.

A Proof of Lemma 1

Proof. If we have K one dimensional intervals $I_1 = (l_1, r_1], I_2 = (l_2, r_2], \ldots, I_T = (l_K, r_K]$, we want to prove if every pair of them have nonempty overlap $I_1 \cap \cdots \cap I_K \neq \emptyset$. This can be proved by the following. Without loss of generality we assume $l_1 \leq l_2 \leq \cdots \leq l_K$. For each $k < K$, $I_k \cap I_K \neq \emptyset$ implies $l_K < r_k$. Therefore, $(l_T, \min(r_1, r_2, \ldots, r_K)]$ will be a nonempty set that is contained in $I_1, I_2, \ldots, I_K$. Therefore $I_1 \cap I_2 \cap \cdots \cap I_K \neq \emptyset$ and it is another interval.

This can be generalized to d-dimensional boxes. Assume we have boxes $B_1, \ldots, B_K$ such that $B_i \cap B_j \neq \emptyset$ for any i and j. Then for each dimension we can apply the above proof, which implies that $B_1 \cap B_2 \cap \cdots \cap B_K \neq \emptyset$ and the intersection will be another box. □

B Data Statistics and Model Parameters in Tables 1 and 2

Table 3 presents data statistics and parameters for the models in Tables 1 and 2 in the main text. The standard test accuracy is the model accuracy on natural, unmodified test sets.

Dataset	training set size	test set size	# of features	# of classes	# of trees	robust ϵ	depth		standard test acc.	
							robust	natural	robust	natural
breast-cancer	546	137	10	2	4	0.3	8	6	.978	.964
covtype	400,000	181,000	54	7	80	0.2	8	8	.847	.877
cod-rna	59,535	271,617	8	2	80	0.2	5	4	.880	.965
diabetes	614	154	8	2	20	0.2	5	5	.786	.773
Fashion-MNIST	60,000	10,000	784	10	200	0.1	8	8	.903	.903
HIGGS	10,500,000	500,000	28	2	300	0.05	8	8	.709	.760
ijcnn1	49,990	91,701	22	2	60	0.1	8	8	.959	.980
MNIST	60,000	10,000	784	10	200	0.3	8	8	.980	.980
webspam	300,000	50,000	254	2	100	0.05	8	8	.983	.992
MNIST 2 vs. 6	11,876	1,990	784	2	1000	0.3	6	4	.997	.998

Table 3: The data statistics and parameters for the models presented in Tables 1 and 2.

C An $O(n)$ time algorithm for verifying a decision tree

The robustness of a single tree can be easily verified by the following $O(n)$ algorithm, which traverse the whole tree and computes the bounding boxes for each node in a depth-first search fashion.

Algorithm 1: Linear time ℓ_∞ untargeted attack for a decision tree.

Initial $p^* = 0, \ell_t = -\infty, r_t = \infty, \quad \forall t = 1, \dots d$;
`ComputeRecursive`$(0, 0)$;

Function `ComputeRecursive`(i, p)
  if *i is leaf node* **then**
    if $v_i \neq y_0$ **then**
      $p^* \leftarrow \min(p^*, p)$;
  else
    `/* Checking conditions for the left child */`
    $s \leftarrow r_{t_i}$;
    $r_{t_i} \leftarrow \min(r_{t_i}, l_{t_i})$;
    if $l_{t_i} \leq r_{t_i}$ **then**
      if $r_{t_i} < x_{t_i}$ **then**
        `ComputeRecursive`$(i.$*left_ child,* $\max(p, |x_{t_i} - r_{t_i}|))$
      else
        `ComputeRecursive`$(i.$*left_ child,* $p)$;
    $r_{ti} \leftarrow s$;
    `/* Checking conditions for the right child */`
    $s \leftarrow l_{t_i}$;
    $l_{t_i} \leftarrow \max(l_{t_i}, l_{t_i})$;
    if $l_{t_i} \leq r_{t_i}$ **then**
      if $l_{t_i} > x_{t_i}$ **then**
        `ComputeRecursive`$(i.$*right_ child,* $\max(p, |x_{t_i} - l_{t_i}|))$
      else
        `ComputeRecursive`$(i.$*right_ child,* $p)$;
end

D Clique Enumeration Algorithm

The algorithm first looks at any first two parts V_1 and V_2 of the graph and enumerates all 2-cliques in $O(|V_1||V_2|)$ time. Then, each 2-clique found is converted into a "pseudo node" (this is possible due to Lemma 1), and all 2-cliques form a new part V_2' of the graph. Then we replace V_1 and V_2 with V_2', and continue to enumerate all 2-cliques between V_2' and V_3 to form V_3'. A 2-clique between V_2' and V_3 represents a 3-clique in V_1, V_2 and V_3 due to boxicity. Note that enumerating all 3-cliques in a general 3-partite graph takes $O(|V_1||V_2||V_3|)$ time; thanks to boxicity, our algorithm takes $O(|V_2'||V_3|)$ time which equals to $O(|V_1||V_2||V_3|)$ only when V_1 and V_2 form a complete bipartite graph, which is unlikely in common cases. This process continues recursively until we process all K parts and have only V_K' left, where each vertex in V_K' represents a K-clique in the original graph. After obtaining all K-cliques, we can verify each to

compute verification bound.

Algorithm 2: Enumerating all K-cliques on a K-partite graph with a known boxicity d

```
1  Initial V_1, V_2, ..., V_K;
2  for k ← 1, 2, 3, ..., K do
3  |  U_k ← {(A_i, B^{i^(k)}) | i^(k) ∈ V_k, A_i = {i^(k)}};
   |  /* A is a set of the nodes in a clique.                          */
   |  /* Here U_k is a set of 1-cliques.                               */
4  end
5  CliqueEnumerate(U_1, U_2, ..., U_K);

6  Function CliqueEnumerate(U_1, U_2, ..., U_K)
7  |  V'_1 ← U_1;
8  |  for k ← 2, 3, ..., K do
9  |  |  V'_k ← ∅;
10 |  |  for (A, B) ∈ V'_{k-1} do
11 |  |  |  for (Â, B̂) ∈ U_k do
12 |  |  |  |  if B ∩ B̂ ≠ ∅ then
   |  |  |  |  |  /* A clique is found, add it as a pseudo node with the intersection
   |  |  |  |  |     of two boxes                                      */
13 |  |  |  |  |  V'_k ← V'_k ∪ {(A ∪ Â, B ∩ B̂)};
14 |  |  |  end
15 |  |  end
16 |  end
17 |  return V'_K;
18 end
```

E Multi-level Verification Framework

Algorithm 3 presents our multi-level decision tree verification framework describe in Section 3.3.

Algorithm 3: Multi-level verification framework

Initial $V_1,\ V_2,\ ,\ldots,\ V_K$; maximum number of independent sets in a group: T; maximum number of levels: L, $L \leq \log_T(K)$;
`MultiLevelVerify(`$V_1,\ V_2,\ ,\ldots,\ V_K$`)`;

Function `MultiLevelVerify(`$V_1,\ V_2,\ ,\ldots,\ V_K$`)`
  for $k \leftarrow 1,\ 2,\ 3,\ \ldots,\ K$ **do**
    $U_k^{(0)} \leftarrow \{(A_i,\ B^{i^{(k)}})|i^{(k)} \in V_k,\ A_i = \{i^{(k)}\}\}$;
    `/*` A `is a set of the nodes in a clique */`
  end
  for $l \leftarrow 1,\ 2,\ \ldots,\ L$ **do**
    for $k \leftarrow 0,\ 1,\ \ldots,\ \lfloor K/T^l \rfloor$ **do**
      $U_k^{(l)} \leftarrow$ `CliqueEnumerate(`$U_{kT+1}^{(l-1)},\ U_{kT+2}^{(l-1)},\ \ldots,\ U_{(k+1)T-1}^{(l-1)}$`)`
    end
  end
  for $k \leftarrow 0,\ 1,\ \ldots,\ \lfloor K/T^L \rfloor$ **do**
    for $(A,\ B) \in U_k^{(L)}$ **do**
      $v_i \leftarrow$ sum of the values corresponding to all leaves in A.
    end
  end
  Run the dynamic programming method in (7) for v^*.
end