Tree in Tree: from Decision Trees to Decision Graphs

Anonymous Author(s)
Affiliation
Address
email

Abstract

Decision trees have been widely used as classifiers in many machine learning applications thanks to their lightweight and interpretable decision process. This paper introduces Tree in Tree decision graph (TnT), a framework that extends the conventional decision tree to a more generic and powerful directed acyclic graph. TnT constructs decision graphs by recursively growing decision trees inside the internal or leaf nodes instead of greedy training. The time complexity of TnT is linear to the number of nodes in the graph, and it can construct decision graphs on large datasets. Compared to decision trees, we show that TnT achieves better classification performance with reduced model size, both as a stand-alone classifier and as a base estimator in bagging/AdaBoost ensembles. Our proposed model is a novel, more efficient, and accurate alternative to the widely-used decision trees.

1 Introduction

Decision trees (DTs) and tree ensembles are widely used in practice, particularly for applications that require few parameters [1–4], fast inference [5, 6], and good interpretability [7, 8]. In a DT, the internal and leaf nodes are organized in a binary structure, with internal nodes defining the routing function and leaf nodes predicting the class label. Although DTs are easy to train by recursively splitting leaf nodes, the tree structure can be suboptimal for the following reasons: (1) DTs can grow exponentially large as the depth of the tree increases. Yet, the root-leaf path can be short even for large DTs, limiting the predictive power. (2) In a DT, the nodes are not shared across different paths, reducing the efficiency of the model.

Decision trees are similar to neural networks (NNs) in that both models are composed of basic units. A possible way to enhance the performance of DTs or NNs is to replace the basic units with more powerful models. For instance, "Network in Network" builds micro NNs with complex structures within local receptive fields to achieve state-of-the-art performances on image recognition tasks [9]. As for DTs, previous work replaced the axis-aligned splits with logistic regression or linear support vector machines to construct oblique trees [1, 3, 5, 10–12]. The work in [4] further incorporates convolution operations into DTs for improved performance on image recognition tasks, while [1] replaces the leaf predictors with linear regression to improve the regression performance. Unlike the greedy training algorithms used for axis-aligned trees (e.g., Classification and Regression Trees or CART [13]), oblique trees are generally trained by gradient-based [3, 11, 12] or alternating [1, 5] optimization algorithms.

Inspired by the concepts of Network in Network [9] and oblique trees [5, 10], we propose a novel model, Tree in Tree (TnT), to recursively replace the internal and leaf nodes with micro decision trees. In contrast to a conventional tree structure, the nodes in a TnT form a Directed Acyclic Graph (DAG) to address the aforementioned limitations and construct a more efficient model. Unlike previous oblique trees that were optimized on a predefined tree structure [1, 4], TnT can learn graph connections from scratch. **The major contributions of this work are as follows: (1) We extend decision trees to decision graphs and propose a scalable algorithm to construct large decision graphs. (2) We**

Submitted to 35th Conference on Neural Information Processing Systems (NeurIPS 2021). Do not distribute.

show that the proposed algorithm outperforms existing decision trees/graphs, either as a stand-alone classifier or base estimator in an ensemble, under the same model complexity constraints. (3) Rather than relying on a predefined graph/tree structure, the proposed algorithm is capable of learning graph connections from scratch (i.e., starting from a single leaf node) and offers a fully interpretable decision process.

2 Related work

Decision graph (DG) is a generalization of the conventional decision tree algorithm, extending the tree structure to a directed acyclic graph [14, 15]. Despite similarity in using a sequential inference scheme, training and optimizing DGs is more challenging due to the large search space for the graph structure. The work in [14] proposed a greedy algorithm to train DGs by tentatively joining pairs of leaf nodes at each training step (NDG, Algorithm 1). Alternatively, in this work, we revisit the concept of decision graphs by exploiting recent advances in non-greedy tree optimization algorithms [5, 10, 12, 16]. Our proposed Tree in Tree algorithm can construct DGs as a more accurate and efficient alternative to the widely-used decision trees, both as stand-alone classifiers and as weak learners in the ensembles.

Algorithm 1: Naive decision graph (NDG) [14]

$G \leftarrow$ initialize graph with a leaf node;
for $i \leftarrow 1$ **to** N **do**
  for *each leaf node* $(l_i) \in G$ **do**
    Find the maximum gain (g_i) if we split l_i ;
  for *each pair of leaf nodes* $(l_i, l_j) \in G$ **do**
    Record gain $(g_{i,j})$ if we merge l_i and l_j;
  Split/merge nodes to maximize gain;

Note: The split operation has a model complexity penalty (C) for creating an internal node.

Conventional decision tree learning algorithms such as CART [13] and its variations follow a greedy top-down growing scheme. Recent work has focused on optimizing the structure of the tree [16–18]. However, constructing an optimal binary DT is NP-hard [19] and optimal trees are not scalable to large datasets with many samples and features [16–18]. Recent studies have further developed scalable algorithms for non-greedy decision tree optimization, with no guarantee on tree optimality [1, 3, 5, 10–12]. Such scalable approaches can be categorized into two groups: tree alternating optimization (TAO) [1, 5] and gradient-based optimization [3, 10–12].

TAO decomposes the tree optimization problem into a set of reduced problems imposed at the node levels. The work in [5] applied the alternating optimization to both axis-aligned trees and sparse oblique trees. Later, [1] extended TAO to regression tasks and ensemble methods. Unlike TAO, gradient-based optimization requires a differentiable objective function, which can be obtained by different methods. For example, [10] derived a convex-concave upper bound of the empirical loss. [11] and [3] considered a soft (i.e., probabilistic) split at the internal nodes and formulated a global objective function. The activation function for soft splits was refined in [12] to enable conditional inference and parameter update. Both TAO and gradient-based optimization operate on a predefined tree structure and optimize the parameters of the internal nodes.

The proposed Tree in Tree algorithm aims to optimize the graph/tree structure by growing micro decision trees inside current nodes. Compared to the greedy top-down tree induction [13], Tree in Tree solves a reduced optimization problem at each node, which is enabled via non-greedy tree alternating optimization techniques [5]. Compared to NDG, TnT employs a non-greedy process to construct decision graphs, which leads to an improved classification performance (discussed in later sections). Compared to axis-aligned decision trees (e.g., TAO [1, 5], CART [13]), TnT extends the tree structure to a more accurate and compact directed acyclic graph, in which nodes are shared across multiple paths.

3 Methods

In this work, we consider a classification task with input and output spaces denoted by $\mathcal{X} \subset \mathbb{R}^D$ and $\mathcal{Y} = \{1, ..., K\}$, respectively. Similar to conventional decision trees, a decision graph classifier G consists of internal nodes and leaf nodes. Each internal node is assigned a binary split function $s(\cdot;\theta) : \mathcal{X} \rightarrow [left_child, right_child]$ parametrized by θ, which defines the routing function of a graph. For axis-aligned splits, θ indicates a feature index and a threshold. The terminal nodes (with no children) are named leaf nodes and indicate the class labels.

3.1 Decision graph

As an extension to the tree structure, decision graphs organize the nodes into a more generic directed acyclic graph. In this work, we limit our discussion to axis-aligned binary DTs/DGs in which each internal node compares a feature value to a threshold to select one of the two child nodes. Similar to the sequential inference process in DTs, the test samples in a DG start from the root and successively select a path at the internal nodes until a leaf node is reached. The main differences between binary DTs and DGs are the following: (1) In DTs, each child node has one parent node. However, DGs allow multiple parent nodes to share the same child node. Therefore, DG can combine the nodes with similar behaviors (e.g., similar split functions) to reduce model complexity. (2) In binary DTs, the number of leaf nodes is always greater than the internal nodes by one. In DGs, however, $\#Leaves \leq \#Internals + 1$, since multiple internal nodes can share the same leaf node. Furthermore, there exists a unique path to reach each leaf node in a tree structure, which does not hold within DGs. (3) The model complexity of a DT is often quantified by the number of internal or leaf nodes. However, we can post-process a DG by merging the leaf nodes with the same class label. As a result, DGs have a minimum leaf node count equal to the number of classes. Therefore, we use the number of splits (i.e., internal nodes) to quantify the model complexity of a DG.

3.2 Tree in Tree

We propose a novel algorithm named Tree in Tree as a scalable method to construct large decision graphs. Conventional DT training algorithms (e.g., CART) are greedy and recursively split the leaf nodes to grow a deep structure, without optimizing the previously learned split functions. **The key difference between the proposed TnT model and conventional approaches lies in the optimization of the internal nodes. TnT fits new decision trees in place of the internal/leaf nodes and employs such micro DTs to construct a directed acyclic graph.** Overall, the proposed TnT model is a novel extension to the conventional decision trees and generates accurate predictions by routing samples through a directed acyclic graph.

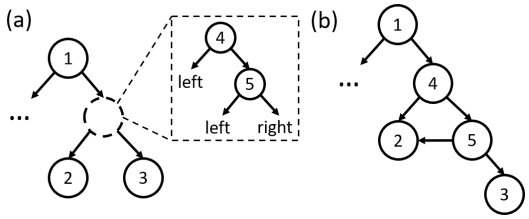

Figure 1: (a) The growing phase of TnT. The micro decision tree (in dashed box) replaces an internal node (dashed circle). Compared to a single node, the substitute micro tree can provide a more powerful split function. (b) The merging phase of TnT. We merge the fitted micro tree into the current structure to create a directed acyclic graph.

Figure 1 shows the high-level procedure for training a decision graph with the proposed TnT algorithm. Assuming a starting decision graph (e.g., a decision tree or a single leaf node), our goal is to grow a larger model with improved predictive power. In the growing phase of TnT (Fig. 1(a)), we replace a node (dashed circle) with a micro decision tree with multiple splits to enable more accurate decision boundaries. In the merging phase (Fig. 1(b)), the micro decision tree is merged into the starting model to construct a TnT decision graph, in which a child node (node 2) may have multiple parent nodes (node 4 and 5).

Growing the graph from internal nodes We consider the training of a decision graph as an optimization problem with the aim of minimizing the loss function on the training data:

$$\min \sum_{x,y \in \mathbb{X},\mathbb{Y}} L\left(y, G(x;\Theta)\right). \tag{1}$$

TnT grows the decision graph $G(\cdot;\Theta)$ from an arbitrary internal node $n_i \in G$ with the split function $s(\cdot;\theta_i)$. θ_i denotes the trainable parameters of n_i including a feature index and a threshold for axis-aligned splits. The overall goal is to replace n_i with a decision tree t_i and minimize the loss function as indicated in (1). All other nodes remain unchanged as we train t_i.

Let us consider a subset of samples $(\mathbb{X}_{subset}, \mathbb{Y}_{subset})$ that is sensitive to the split function $s(\cdot;\theta_i)$, as defined by the following expression:

$$G_{n_i \to left}(\mathbb{X}_{subset};\Theta\backslash\theta_i) \neq G_{n_i \to right}(\mathbb{X}_{subset};\Theta\backslash\theta_i), \tag{2}$$

where $\Theta\backslash\theta_i$ denotes the parameters of all nodes in G excluding n_i. Growing the graph from n_i does not change $\Theta\backslash\theta_i$ since all other nodes are fixed as we solve the reduced optimization problem at

n_i. $G_{n_i \to left}$ sends the samples to the left child at n_i (i.e., $s(\cdot;\theta_i) \to left_child$) while $G_{n_i \to right}$ routes the samples to the right child at n_i. With $\Theta \backslash \theta_i$ being fixed, the output of decision graph only depends on θ_i (i.e., $s(\cdot;\theta_i)$)

$$G(x;\Theta) = \begin{cases} G_{n_i \to left}(x;\Theta\backslash\theta_i) & \text{if } s(x;\theta_i) \to left_child \\ G_{n_i \to right}(x;\Theta\backslash\theta_i) & \text{if } s(x;\theta_i) \to right_child. \end{cases} \tag{3}$$

Having (1) and Equation (3), the reduced optimization problem at node n_i is given by

$$\sum_{x,y \in \mathbb{X},\mathbb{Y}} \min(L(y, G_{n_i \to left}(x;\Theta\backslash\theta_i)), L(y, G_{n_i \to right}(x;\Theta\backslash\theta_i))). \tag{4}$$

Since $L(y, G_{n_i \to left}(x;\Theta\backslash\theta_i)) \neq L(y, G_{n_i \to right}(x;\Theta\backslash\theta_i))$ only if the inequality (2) holds, we train the micro decision tree $t_i(x)$ based on the subset $(\mathbb{X}_{subset}, \mathbb{Y}_{subset})$ instead of using the entire training set. The optimization problem (4) has a closed-form solution as follows:

$$t_i^*(x) \coloneqq \begin{cases} left_child & \text{if } L(y, G_{n_i \to left}(x;\Theta\backslash\theta_i)) < L(y, G_{n_i \to right}(x;\Theta\backslash\theta_i)) \\ right_child & \text{if } L(y, G_{n_i \to right}(x;\Theta\backslash\theta_i)) < L(y, G_{n_i \to left}(x;\Theta\backslash\theta_i)). \end{cases} \tag{5}$$

Equation (5) defines the optimal split function at the internal node n_i which is used to fit the micro decision tree t_i. With other nodes being fixed, we show that the overall loss function of G can be minimized by pursuing the optimal spilt function at an arbitrary internal node n_i. Rather than using a simple axis-aligned split, the proposed TnT algorithm learns a complexity-constrained decision tree to better approximate the optimal split function (Equation (5)).

Growing the graph from leaf nodes Growing from the leaf nodes is a standard practice in greedy training algorithms, where we recursively split the leaf nodes to achieve a deeper tree with a better fit on the training data [13]. In TnT, we replace the leaf predictors with decision trees. Let $G(\cdot;\Theta)$ be a decision graph and $n_l \in G$ an arbitrary leaf node with a constant class label $l(\cdot;\theta_l) = c$. Our goal is to minimize the overall loss function $L(\mathbb{Y}, G(\mathbb{X};\Theta))$ by replacing the leaf predictor $l(\cdot;\theta_l)$ with a micro decision tree $t_l(x)$.

Consider the subset of samples $(\mathbb{X}_{subset}, \mathbb{Y}_{subset})$ that visit the leaf node n_l. Minimization of the loss function (1) can be expressed as

$$\min \sum_{\substack{x \in \mathbb{X}_{subset} \\ y \in \mathbb{Y}_{subset}}} L(y, t_l(x)), \tag{6}$$

where the minimum is simply achieved at $t_l^*(x) \coloneqq y$ for $x, y \in \mathbb{X}_{subset}, \mathbb{Y}_{subset}$ (i.e., the ideal leaf predictor). We build a decision tree to approximate the ideal leaf predictor.

Following the growing phase (Fig. 1(a)), the micro decision trees are merged into the decision graph (Fig. 1(b)). The nodes of the TnT decision graph are similar to those in the decision trees, where an internal node makes a single axis-aligned split and each leaf node contains a class label. In this paper, we construct TnT decision graphs using axis-aligned splits. However, we do not limit the form of split functions ($s(\cdot;\theta)$) or leaf predictors $l(\cdot;\theta)$ in the TnT training process. For example, we could use logistic regression as the split function of the decision graph and micro trees to construct oblique TnT. In this case, θ refers to the trainable weights in logistic regression. **Therefore, various tree-based models could potentially benefit from the proposed TnT framework.**

3.3 Learning procedure

Unlike the learning procedures in [1, 5] which require a predefined tree structure, our proposed TnT algorithm grows a decision graph from a single leaf node. The training of TnT decision graphs is an iterative process that follows a *grow-merge-grow-* $\cdots$ *-merge* alternation. Algorithm 2 shows the pseudocode to train a TnT decision graph. Lines 7-15 find the subset of data samples $\mathcal{X}_{subset}, \mathcal{Y}_{subset}$ that is sensitive to the internal split functions or leaf predictors at each node, and grow micro decision trees. In the internal nodes, $\mathcal{Y}_{subset}$ represents binary labels for the left or right child (i.e., not the label of the training set). Line 16 grows micro decision trees according to the growing phase of the TnT. Line 17 merges the trees into the graph structure.

Regularization Regularization is critical to limit model complexity and prevent overfitting of a decision tree and it is similarly required for TnT decision graphs. In the growing phase of a TnT

(either from internal or leaf nodes), the subsets of samples $\mathcal{X}_{subset}, \mathcal{Y}_{subset}$ at different nodes may have various sizes. Therefore, we need a robust regularization technique to operate across all nodes of the TnT and to train the micro decision trees without overfitting on small subsets. In this work, we propose to use the sample-weighted cost complexity pruning approach [20, 21]. We prune micro decision trees by minimizing $R(t_i) + C_i|t_i|$, where $R(t_i)$ is the misclassification measurement and $|t_i|$ denotes the tree complexity. We calculate $R(t_i)$ using Gini impurity and measure $|t_i|$ by counting the number of splits [13]. C_i is the sample-weighted regularization coefficient calculated by

$$C_i = C\frac{\#\mathcal{X}}{\#\mathcal{X}_{subset,i}}, \tag{7}$$

where $\#\mathcal{X}_{subset,i}$ is the sample count of subset at node n_i. C is a hyperparameter of the TnT and is used to control the pruning strength and tune the model complexity (# splits). For a smaller subset, we need to apply a stronger cost complexity pruning to prevent overfitting.

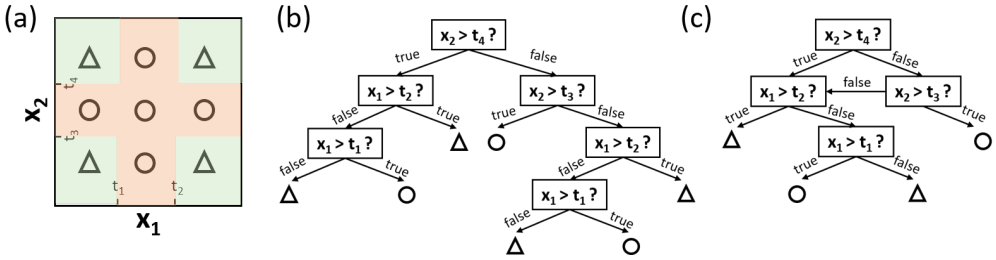

Figure 2: Comparison of DT and TnT decision graph on synthetic data; (a) A toy classification task with desired axis-aligned boundaries. x_1, x_2 and $t_1 - t_4$ denote two features and four thresholds, respectively. Different markers represent binary class labels. (b) A decision tree requires at least six splits to classify the data. (c) A TnT decision graph only requires four binary splits on the same task.

Fine-tune and post pruning The TnT decision graphs are compatible with Tree Alternating Optimization (TAO [5]), previously proposed to optimize decision trees. We used TAO to fine-tune the TnT decision graphs, which led to slight improvements in classification accuracy. The pseudocode for TnT fine-tune algorithm is provided in the supplementary materials. A post pruning process is further applied to TnT decision graphs to remove the dead nodes. A node is pruned if no training samples travel through that node. Post pruning can result in a more compact decision graph and reduce the number of splits without affecting the training accuracy.

Algorithm 2: Tree in Tree (TnT)

Data: Training set $\mathcal{X}, \mathcal{Y}$
Result: TnT decision graph G fitted on the training set

$G \leftarrow$ initialize graph with a leaf node;
$infer(n_t, \mathcal{X}_t)$ denotes the forward inference of data $\mathcal{X}_t$ starting from node n_t ;
for $i_1 \leftarrow 1$ **to** N_1 **do**
  for $i_2 \leftarrow 1$ **to** N_2 **do**
    for *each node* $(n_i) \in G$ **do**
      Samples that visit n_i: $\mathcal{X}_i, \mathcal{Y}_i \subset \mathcal{X}, \mathcal{Y}$;
      if n_i *is an internal node* **then**
        $\mathcal{Y}_{i,left} \leftarrow infer(n_i.left_child, \mathcal{X}_i)$;
        $\mathcal{Y}_{i,right} \leftarrow infer(n_i.right_child, \mathcal{X}_i)$;
        $index_left \leftarrow (\mathcal{Y}_i = \mathcal{Y}_{i,left}$ **and** $\mathcal{Y}_i \neq \mathcal{Y}_{i,right})$;
        $index_right \leftarrow (\mathcal{Y}_i = \mathcal{Y}_{i,right}$ **and** $\mathcal{Y}_i \neq \mathcal{Y}_{i,left})$;
        $\mathcal{X}_{subset}, \mathcal{Y}_{subset} \leftarrow$ copy samples from $\mathcal{X}_i, \mathcal{Y}_i$ at $(index_left$ **or** $index_right)$;
        $\mathcal{Y}_{subset}[index_left] \leftarrow 0$, $\mathcal{Y}_{subset}[index_right] \leftarrow 1$;
      else if n_i *is a leaf node* **then**
        $\mathcal{X}_{subset} \leftarrow \mathcal{X}_i$, $\mathcal{Y}_{subset} \leftarrow \mathcal{Y}_i$;
      Grow a micro tree t_i on subset $\mathcal{X}_{subset}, \mathcal{Y}_{subset}$ in place of n_i;
  Merge t_i into the current decision graph G for all nodes $(n_i \in G)$

Time complexity Compared to decision trees, decision graphs offer an enriched model structure, which increases the complexity of learning the graph structure. Previous work constructed decision graphs by tentatively merging two leaf nodes at each training step, with a time complexity of $O(N_l^2)$, where N_l is the number of leaf nodes [14]. Since the proposed TnT algorithm generates new splits by growing micro decision trees inside the nodes, the dataset is initially sorted in $O(mklog(m))$ for m samples and k features. The time complexity for creating a new split depends on the dataset (i.e., $O(mk)$) and not on the size of the graph. As the graph grows larger, the TnT algorithm optimizes each node for $N_1 * N_2$ times in the worst case (Algorithm 2). Since N_1 and N_2 are hyperparameters that were fixed in this work ($N_1 = 2, N_2 = 5$, the choice of N_1 and N_2 will be discussed in the following section), TnT exhibits a linear time complexity to the number of nodes, $O(nmk + mklog(m))$ with n being the number of nodes. Testing our Python implementation on an Intel i7-9700 CPU, it took 325.3 seconds to build a TnT of 1k splits on the MNIST dataset (60k samples, 784 features, 10 classes).

Synthetic data We first construct a synthetic classification dataset to show the potential benefits of TnT over conventional decision tree algorithms (e.g., CART). Figure. 2(a) visualizes the two-dimensional data distribution with one class on the corners and the other class elsewhere. To achieve optimal decision boundaries, a conventional decision tree requires six splits (Fig. 2(b)), whereas TnT only requires four splits to generate the same decision boundaries (Fig. 2(c)). **By sharing nodes among different decision paths in a graph, TnT enables a more compact model with fewer splits compared to a conventional DT.**

4 Experiments: TnT as a stand-alone classifier

We test the TnT decision graph as a stand-alone classifier and benchmark it against several state-of-the-art decision tree/graph algorithms with axis-aligned splits, including classification and regression trees (CART [13]), tree alternating optimization (TAO [5]), and the naive decision graph (NDG [14]). We also implement the TnT algorithm in two different settings: with or without fine-tuning. We observe that the proposed TnT algorithm consistently achieves a superior performance under similar complexity constraints on multiple datasets.

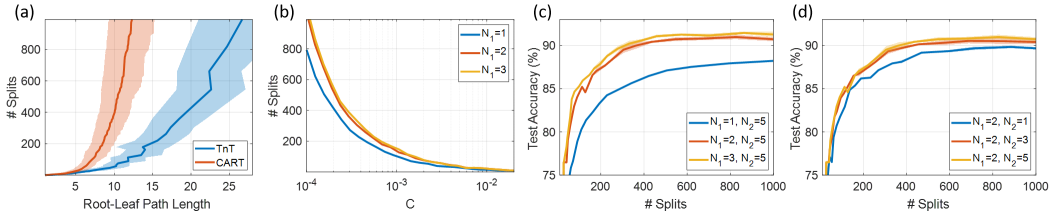

Figure 3: (a) The number of splits as a function of the root-leaf path length. The standard deviation across different samples is shown by shaded areas. (b) The number of splits vs. regularization coefficient C. (c, d) Test performance using different hyperparameter settings on the MNIST dataset. The default setting ($N_1 = 2, N_2 = 5$) is plotted in both figures for comparison.

In the worst-case scenario, the number of nodes increases exponentially with the depth of a tree, which prevents DTs from growing very deep. However, this limitation does not apply to TnT decision graphs. Figure 3(a) illustrates the average length of the root-leaf path as a function of model complexity for TnT and CART. With 1000 splits, the average decision depth of the best-first CART is 12.3, whereas the TnT decision graph has a mean depth of 27.3. In the best-first decision tree induction, we add the best split in each step to maximize the objective [22]. **Therefore, TnT can achieve a much "deeper" model without significantly increasing the number of splits.** The regularization coefficient C is used to control the complexity of decision graphs in TnT. The number of splits decreases as we increase the pruning strength C (Fig. 3(b)). Figures 3(c, d) compare the effect of different hyperparameter settings (N_1, N_2). **We note that the proposed TnT decision graph is a superset of decision trees and that TnT can reduce to a DT learning algorithm under certain conditions.** With $N_1 = 1$, Algorithm 2 replaces a single leaf node with a decision tree, which is equivalent to training a CART with cost complexity pruning. In general, higher values of N_1 and N_2 can lead to a better classification performance. In the following experiments, we set the hyperparameters as $N_1 = 2, N_2 = 5$. A marginal improvement in classification performance can be obtained by increasing N_1 and N_2, at the cost of increased training time.

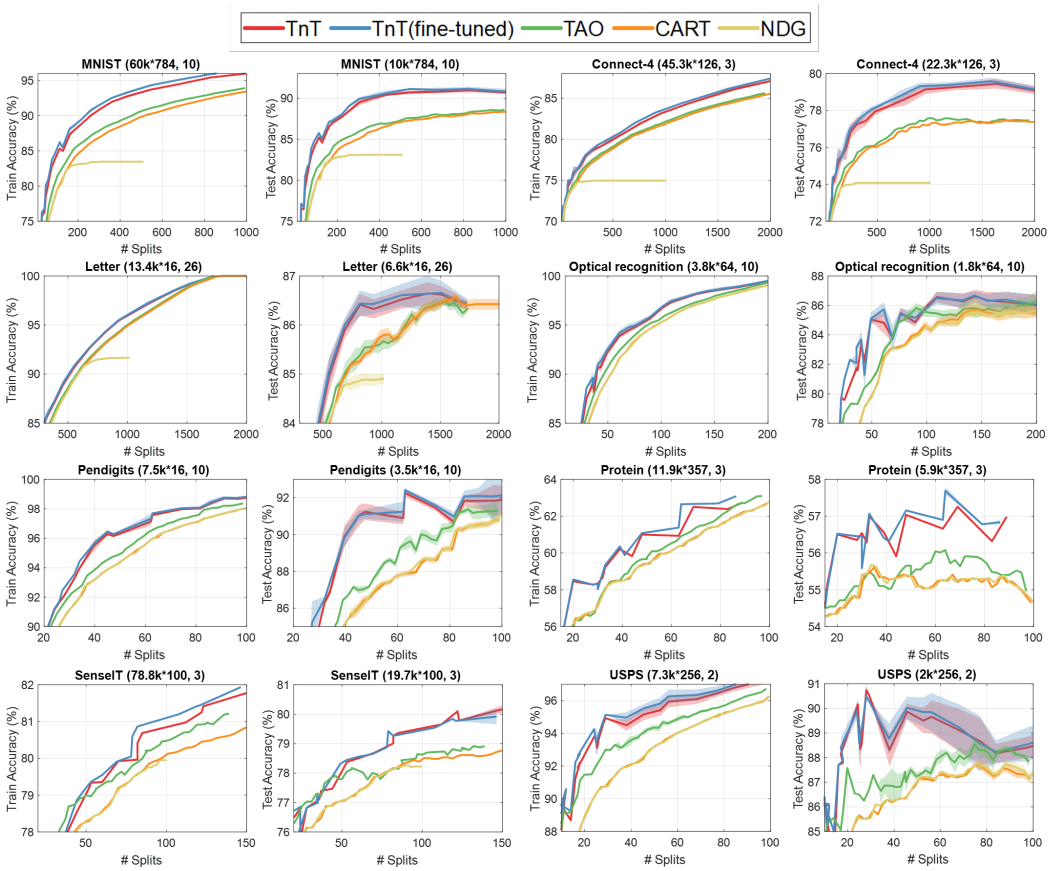

Figure 4: Model comparison in terms of train and test accuracy on multiple classification tasks. The following axis-aligned decision trees/graphs are included: **TnT (this work):** We implement the proposed TnT decision graph at various complexity levels. Hyperparameters are fixed at $N_1 = 2, N_2 = 5$ on all tasks. **TnT (fine-tuned):** The alternating optimization algorithm is used to fine-tune the TnT. **TAO:** The tree alternating optimization algorithm is applied to axis-aligned decision trees [23]. **CART:** Classification and regression trees trained in a best-first manner to assess the optimal tree structure under certain complexity constraint [13, 22]. **NDG:** The naive decision graph trained with Algorithm 1 [14]. The complexity penalty is fixed at $C = 3e - 4$ on all tasks. Dataset statistics are indicated on top of each figure with the following format (# Train/Test samples * # Features, # Classes).

Figure 4 compares the proposed TnT decision graphs with axis-aligned decision trees/graphs previously reported. We include the following datasets: MNIST, Connect-4, Letter, Optical reconstruction, Pendigits, Protein, SenseIT, and USPS from the UCI machine learning repository [24] and LIBSVM Dataset [25] under Creative Commons Attribution-Share Alike 3.0 license. The statistics of datasets including the number of train/test instances, number of attributes, and number of classes are shown in Fig. 4. If a separate test set is not available for some tasks, we randomly partition 33% of the entire data as test set. For all models, we repeat the training procedure five times with different random seeds. The mean classification accuracy is plotted in Fig. 4 with shaded area indicating the standard deviation across trials. The proposed Tree in Tree (TnT) algorithm outperforms axis-aligned decision trees such as TAO [5, 23] and CART [13], as well as NDG which is also based on axis-aligned decision graphs [14]. We also present the results for TnT(fine-tuned), which employs alternating optimization to fine-tune the TnT and slightly improve the classification performance.

Visualization Similar to decision trees, TnT decision graphs enjoy a fully interpretable and visualizable decision process. Figures 5(a-c) visualize the TnT decision graphs with 20, 129, and 1046 splits, respectively. We use different node colors to indicate the dominant class labels. A node will have a dominant class if most samples at that node belong to the same class. We show the nodes in blue if class labels are mixed (i.e., no class label contributes to greater than 50% of the samples

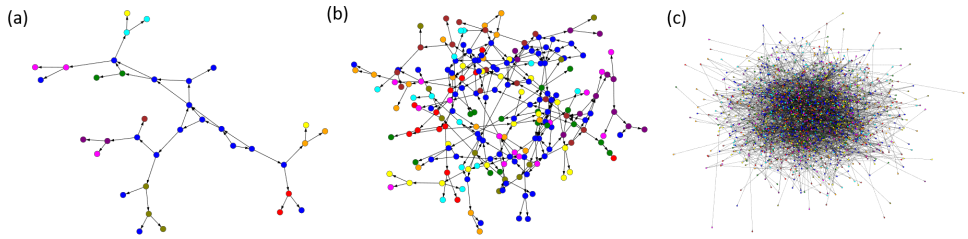

Figure 5: Visualization of TnT decision graphs at various complexity levels. (a) TnT with 20 internal nodes and 16 leaf nodes (train/test accuracy: 70.41%/71.75% on MNIST classification task). (b) 129 internals and 75 leaves (train/test accuracy: 85.54%/85.49%). (c) 1046 internals and 630 leaves (train/test accuracy: 96.04%/90.56%). Different node colors represent dominant class labels (more than 50% of samples belong to the same class). Nodes are shown in blue if no dominant class is found.

visiting that node). As the graph grows larger, TnT performs better on the MNIST dataset, achieving improved classification accuracy on both training and testing sets.

5 Experiments: TnT in the ensemble

Decision trees are widely used as base estimators in ensemble methods such as bagging and boosting. Random Forests apply a bagging technique to decision trees to reduce variance [26], in which each base estimator is trained using a randomly drawn subset of data with replacement [27]. As opposed to bagging, boosting is used as a bias reduction technique where base estimators are incrementally added to the ensemble to correct the previously misclassified samples. Popular implementations of the boosting methods include AdaBoost [28] and gradient boosting [29, 30]. Both AdaBoost and bagging use classifiers as base estimators, whereas the gradient boosting methods require regressors [29, 30]. Although we argue that the proposed TnT algorithm can be applied to regression tasks with a slight modification in the objectives, it is beyond the scope of this paper to demonstrate TnTs as regressors.

Here, we used the TnT decision graphs as base estimators in the bagging (TnT-bagging) and AdaBoost (TnT-AdaBoost) ensembles. **Our goal is to replace decision trees with the proposed TnT classifiers in ensemble methods and compare the performance under various model complexity constraints.** The ensemble methods are implemented using the scikit-learn library in Python (under the 3-Clause BSD license) [31]. We change the ensemble complexity by tuning the number of base estimators (# E) and the total number of splits (i.e., internal nodes, # S). Note that TnT has additional hyperparameters that do not apply to decision trees, such as N_1 and N_2. We set the hyperparameters as $N_1 = 2, N_2 = 5$ throughout the experiments so that the TnT and tree ensembles share a similar hyperparameter exploration space.

Table 1 compares the performance of TnT ensembles with that of decision tree ensembles on two datasets. A complete comparison table on eight datasets is included in the supplementary materials. Since the bagging method can effectively reduce variance, we use large models (i.e., TnTs/decision trees with many splits) as the base estimator. On the contrary, TnTs/decision trees with few splits are used in the AdaBoost ensemble, given that boosting can decrease the bias error. According to Table 1, TnT-bagging is almost strictly better than Random Forest under the same model complexity constraints, indicating that TnT decision graphs outperform decision trees as base estimators. TnT-AdaBoost also outperforms AdaBoost in most cases, showing the advantage of TnT over decision trees. However, we observe a few exceptions in the TnT-AdaBoost vs. AdaBoost comparison, as weak learners with high bias (e.g., decision stumps) are also suitable for boosting ensembles. Overall, the TnT ensembles (TnT-bagging, TnT-AdaBoost) achieve a higher classification accuracy compared to decision trees when used in similar ensemble methods (Random Forest, AdaBoost).

6 Discussions

Broader impact Recently, the machine learning community has seen different variations of decision trees [1, 3–5, 10–12]. In this paper, we present the TnT decision graph as a more accurate and efficient alternative to the conventional axis-aligned decision tree. However, the core idea of TnT (i.e., growing micro trees inside nodes) is generic and compatible with many existing algorithms.

Table 1: Comparison of TnT-based ensembles with conventional random forest and AdaBoost. Mean train and test accuracy (± standard deviation) are calculated across 5 independent trials. We tune the ensemble size (# E, the number of base estimators) and splits count (# S) to change the complexity of the ensemble. Dataset statistics are given in the format: Dataset name (# Train/Test samples * # Features, # Classes). Six additional datasets are included in the supplementary materials.

model		# E	# S	train	test		# E	# S	train	test
TnT-bagging	MNIST (60k/10k*784, 10)	5	4.8k	**97.46±0.16**	**93.65±0.24**	Connect-4 (45.3k/22.3k*126, 3)	5	4.6k	**84.42±0.19**	**80.61±0.18**
Random Forest		5	4.8k	96.55±0.36	92.31±0.57		5	4.6k	83.60±0.12	79.21±0.19
TnT-AdaBoost		5	640	**90.26**	88.38		5	450	**77.75±0.16**	**77.39±0.19**
AdaBoost		5	640	89.75	**88.61**		5	450	77.28	76.74
TnT-bagging		10	9.6k	**98.28±0.06**	**94.92±0.20**		10	9.2k	**85.11±0.05**	**81.44±0.14**
Random Forest		10	9.6k	97.44±0.18	93.64±0.38		10	9.2k	84.21±0.12	79.85±0.20
TnT-AdaBoost		10	1.4k	**95.09±0.09**	**92.36±0.13**		10	940	**80.10±0.23**	**78.94±0.29**
AdaBoost		10	1.4k	94.28	91.49		10	940	79.69	78.37
TnT-bagging		20	19.2k	**98.64±0.06**	**95.57±0.14**		20	18.3k	**85.66±0.12**	**81.93±0.13**
Random Forest		20	19.2k	97.90±0.12	94.36±0.19		20	18.3k	84.57±0.08	80.39±0.09
TnT-AdaBoost		20	2.9k	**98.03±0.11**	**94.49±0.21**		20	1.8k	82.46±0.41	80.53±0.50
AdaBoost		20	2.9k	97.70	94.04		20	1.8k	**82.77**	**81.14**

For example, linear-combination (oblique) splits can be easily incorporated into the proposed TnT framework. Specifically, we can grow oblique decision trees inside the nodes to construct an oblique TnT decision graph (Line 16 of Algorithm 1). In addition to oblique TnTs, the proposed TnT framework is also compatible with regression tasks. As suggested in [1], we may grow decision tree regressors (rather than DT classifiers) inside the leaf nodes to construct TnT regressors, which remains as our future work. Overall, our results show the benefits of extending the tree structure to directed acyclic graphs, which may inspire other novel tree-structured models in the future.

Limitations The proposed TnT decision graph is scalable to large datasets and has a linear time complexity to the number of nodes in the graph. However, the training of TnT is considerably slower than CART. The current TnT algorithm is implemented in Python. It takes about 5 minutes to construct a TnT decision graph with ∼1k splits on the MNIST classification task (train/test accuracy: 95.9%/90.4%). Training a CART with the same number of splits requires 12.6 seconds (train/test accuracy: 93.6%/88.3%). TnT has a natural disadvantage in terms of training time since each node is optimized multiple times (in this work $N_1 * N_2 = 10$), similar to other non-greedy tree optimization algorithms (e.g., 1-4 minutes for TAO [5]). The Python implementation may also contribute to the slow training, and we expect that the training time would significantly improve with a C implementation. We also observe that TnT decision graphs have longer decision paths compared to CART (Figure 3(a)), which may raise a concern on increased inference time.

Parallel implementation Algorithm 2 presents a sequential algorithm to construct TnT decision graphs by visiting the nodes in the breadth-first order. However, it is also possible to concurrently grow micro decision trees inside multiple nodes, which could lead to a parallel implementation of TnT. Specifically, only those nodes in the graph that are non-descendant of each other can be optimized in parallel. Parallel optimization is not applicable to the nodes on the same decision path, since the parent node optimization may alter the samples visiting the child node. The parallel optimization of non-descendant nodes follows the separability condition of TAO [1, 5]. The separability condition also holds for the proposed TnT decision graph, enabling a parallel implementation.

7 Conclusion

In this paper, we propose the Tree in Tree decision graph as an effective alternative to the widely used decision trees. Starting from a single leaf node, the TnT algorithm recursively grows decision trees to construct decision graphs, extending the tree structure to a more generic directed acyclic graph. We show that the TnT decision graph outperforms the axis-aligned decision trees on a number of benchmark datasets. We also incorporate TnT decision graphs into popular ensemble methods such as bagging and AdaBoost, and show that in practice, the ensembles could also benefit from using TnTs as base estimators. Our results suggest the use of decision graphs rather than conventional decision trees to achieve superior classification performance, which may potentially inspire other novel tree-structured models in the future.

References

[1] Arman Zharmagambetov and Miguel Carreira-Perpinan. Smaller, more accurate regression forests using tree alternating optimization. In *International Conference on Machine Learning*, pages 11398–11408. PMLR, 2020.

[2] Ashish Kumar, Saurabh Goyal, and Manik Varma. Resource-efficient machine learning in 2 kb ram for the internet of things. In *International Conference on Machine Learning*, pages 1935–1944. PMLR, 2017.

[3] Bingzhao Zhu, Masoud Farivar, and Mahsa Shoaran. Resot: Resource-efficient oblique trees for neural signal classification. *IEEE Transactions on Biomedical Circuits and Systems*, 14(4):692–704, 2020.

[4] Ryutaro Tanno, Kai Arulkumaran, Daniel Alexander, Antonio Criminisi, and Aditya Nori. Adaptive neural trees. In *International Conference on Machine Learning*, pages 6166–6175. PMLR, 2019.

[5] Miguel A Carreira-Perpinán and Pooya Tavallali. Alternating optimization of decision trees, with application to learning sparse oblique trees. *Advances in Neural Information Processing Systems*, 31:1211–1221, 2018.

[6] Charles Mathy, Nate Derbinsky, José Bento, Jonathan Rosenthal, and Jonathan Yedidia. The boundary forest algorithm for online supervised and unsupervised learning. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 29, 2015.

[7] Andrew Silva, Matthew Gombolay, Taylor Killian, Ivan Jimenez, and Sung-Hyun Son. Optimization methods for interpretable differentiable decision trees applied to reinforcement learning. In *International Conference on Artificial Intelligence and Statistics*, pages 1855–1865. PMLR, 2020.

[8] Scott Lundberg and Su-In Lee. A unified approach to interpreting model predictions. *arXiv preprint arXiv:1705.07874*, 2017.

[9] Min Lin, Qiang Chen, and Shuicheng Yan. Network in network. *arXiv preprint arXiv:1312.4400*, 2013.

[10] Mohammad Norouzi, Maxwell D Collins, Matthew Johnson, David J Fleet, and Pushmeet Kohli. Efficient non-greedy optimization of decision trees. *arXiv preprint arXiv:1511.04056*, 2015.

[11] Peter Kontschieder, Madalina Fiterau, Antonio Criminisi, and Samuel Rota Bulo. Deep neural decision forests. In *Proceedings of the IEEE international conference on computer vision*, pages 1467–1475, 2015.

[12] Hussein Hazimeh, Natalia Ponomareva, Petros Mol, Zhenyu Tan, and Rahul Mazumder. The tree ensemble layer: Differentiability meets conditional computation. In *International Conference on Machine Learning*, pages 4138–4148. PMLR, 2020.

[13] Dan Steinberg and Phillip Colla. Cart: classification and regression trees. *The top ten algorithms in data mining*, 9:179, 2009.

[14] Jonathan Oliver. *Decision graphs: an extension of decision trees*. Citeseer, 1992.

[15] Hiroki Sudo, Koji Nuida, and Kana Shimizu. An efficient private evaluation of a decision graph. In *International Conference on Information Security and Cryptology*, pages 143–160. Springer, 2018.

[16] Jimmy Lin, Chudi Zhong, Diane Hu, Cynthia Rudin, and Margo Seltzer. Generalized and scalable optimal sparse decision trees. In *International Conference on Machine Learning*, pages 6150–6160. PMLR, 2020.

[17] Haoran Zhu, Pavankumar Murali, Dzung T Phan, Lam M Nguyen, and Jayant R Kalagnanam. A scalable mip-based method for learning optimal multivariate decision trees. *arXiv preprint arXiv:2011.03375*, 2020.

[18] Xiyang Hu, Cynthia Rudin, and Margo Seltzer. Optimal sparse decision trees. *Advances in Neural Information Processing Systems (NeurIPS)*, 2019.

[19] Hyafil Laurent and Ronald L Rivest. Constructing optimal binary decision trees is np-complete. *Information processing letters*, 5(1):15–17, 1976.

[20] Jeffrey P Bradford, Clayton Kunz, Ron Kohavi, Cliff Brunk, and Carla E Brodley. Pruning decision trees with misclassification costs. In *European Conference on Machine Learning*, pages 131–136. Springer, 1998.

[21] B Ravi Kiran and Jean Serra. Cost-complexity pruning of random forests. In *International Symposium on Mathematical Morphology and Its Applications to Signal and Image Processing*, pages 222–232. Springer, 2017.

[22] Haijian Shi. *Best-first decision tree learning*. PhD thesis, The University of Waikato, 2007.

[23] Arman Zharmagambetov, Suryabhan Singh Hada, Miguel Á Carreira-Perpiñán, and Magzhan Gabidolla. An experimental comparison of old and new decision tree algorithms. *arXiv preprint arXiv:1911.03054*, 2019.

[24] Uc irvine machine learning repository. `http://archive.ics.uci.edu/ml/index.php`. Accessed: 2021-05-02.

[25] Libsvm data. `https://www.csie.ntu.edu.tw/~cjlin/libsvmtools/datasets/multiclass.html`. Accessed: 2021-05-02.

[26] Leo Breiman. Random forests. *Machine learning*, 45(1):5–32, 2001.

[27] Leo Breiman. Bagging predictors. *Machine learning*, 24(2):123–140, 1996.

[28] Yoav Freund and Robert E Schapire. A decision-theoretic generalization of on-line learning and an application to boosting. *Journal of computer and system sciences*, 55(1):119–139, 1997.

[29] 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, 2016.

[30] Guolin Ke, Qi Meng, Thomas Finley, Taifeng Wang, Wei Chen, Weidong Ma, Qiwei Ye, and Tie-Yan Liu. Lightgbm: A highly efficient gradient boosting decision tree. *Advances in neural information processing systems*, 30:3146–3154, 2017.

[31] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830, 2011.

Checklist

1. For all authors...
   (a) Do the main claims made in the abstract and introduction accurately reflect the paper's contributions and scope? [Yes] See lines 37-43.
   (b) Did you describe the limitations of your work? [Yes] See Section 6, Limitations.
   (c) Did you discuss any potential negative societal impacts of your work? [N/A] This paper introduces a new classifier. We do not see any potential negative societal impacts.
   (d) Have you read the ethics review guidelines and ensured that your paper conforms to them? [Yes]
2. If you are including theoretical results...
   (a) Did you state the full set of assumptions of all theoretical results? [N/A]
   (b) Did you include complete proofs of all theoretical results? [N/A]
3. If you ran experiments...
   (a) Did you include the code, data, and instructions needed to reproduce the main experimental results (either in the supplemental material or as a URL)? [Yes] We include a code in the supplementary material. Datasets are publicly available on UCI repository and LIBSVM Data.
   (b) Did you specify all the training details (e.g., data splits, hyperparameters, how they were chosen)? [Yes] Data splits are discussed in Section 4. Choice of hyperparameters is discussed in the supplementary materials Section B.
   (c) Did you report error bars (e.g., with respect to the random seed after running experiments multiple times)? [Yes] Figure 4 and Table 1 report standard deviations across different trials.
   (d) Did you include the total amount of compute and the type of resources used (e.g., type of GPUs, internal cluster, or cloud provider)? [Yes] Platform and training time are reported in Section 3, Time complexity.
4. If you are using existing assets (e.g., code, data, models) or curating/releasing new assets...
   (a) If your work uses existing assets, did you cite the creators? [Yes] See reference [24], [25], [31]
   (b) Did you mention the license of the assets? [Yes] The scikit-learn library is under the 3-Clause BSD license. Some datasets (e.g., MNIST) are under Creative Commons Attribution-Share Alike 3.0 license.
   (c) Did you include any new assets either in the supplemental material or as a URL? [Yes]

(d) Did you discuss whether and how consent was obtained from people whose data you're using/curating? [N/A] All datasets are publicly available on UCI repository and LIBSVM Data.

(e) Did you discuss whether the data you are using/curating contains personally identifiable information or offensive content? [N/A]

5. If you used crowdsourcing or conducted research with human subjects...

(a) Did you include the full text of instructions given to participants and screenshots, if applicable? [N/A]

(b) Did you describe any potential participant risks, with links to Institutional Review Board (IRB) approvals, if applicable? [N/A]

(c) Did you include the estimated hourly wage paid to participants and the total amount spent on participant compensation? [N/A]