Ensembles of Bagged TAO Trees Consistently Improve over Random Forests, AdaBoost and Gradient Boosting

Miguel Á. Carreira-Perpiñán
mcarreira-perpinan@ucmerced.edu
Dept. Computer Science & Engineering
University of California, Merced
Merced, CA, United States

Arman Zharmagambetov
azharmagambetov@ucmerced.edu
Dept. Computer Science & Engineering
University of California, Merced
Merced, CA, United States

ABSTRACT

Ensemble methods based on trees, such as Random Forests, AdaBoost and gradient boosting, are widely recognized as among the best off-the-shelf classifiers: they typically achieve state-of-the-art accuracy in many problems with little effort in tuning hyperparameters, and they are often used in applications, possibly combined with other methods such as neural nets. While many variations of forest methods exist, using different diversity mechanisms (such as bagging, feature sampling or boosting), nearly all rely on training individual trees in a highly suboptimal way using greedy top-down tree induction algorithms such as CART or C5.0. We study forests where each tree is trained on a bootstrapped or random sample but using the recently proposed *tree alternating optimization (TAO)*, which is able to learn trees that have both fewer nodes and lower error. The better optimization of individual trees translates into forests that achieve higher accuracy but using fewer, smaller trees with oblique nodes. We demonstrate this in a range of datasets and with a careful study of the complementary effect of optimization and diversity in the construction of the forest. These *bagged TAO trees* improve consistently and by a considerable margin over Random Forests, AdaBoost, gradient boosting and other forest algorithms in every single dataset we tried.

CCS CONCEPTS

• **Computing methodologies → Classification and regression trees**; **Ensemble methods**.

KEYWORDS

ensemble learning, decision tree optimization, random forests, bagging

ACM Reference Format:
Miguel Á. Carreira-Perpiñán and Arman Zharmagambetov. 2020. Ensembles of Bagged TAO Trees Consistently Improve over Random Forests, AdaBoost and Gradient Boosting. In *Proceedings of the 2020 ACM-IMS Foundations of Data Science Conference (FODS '20), October 19–20, 2020, Virtual Event, USA.* ACM, New York, NY, USA, 12 pages. https://doi.org/10.1145/3412815.3416882

1 INTRODUCTION

We consider ensembles of trees (forests) for classification. These are among the most successful and widely used of all classifiers (in isolation or combined with other models, such as deep nets for feature extraction), and they have received much praise in the statistical and machine learning literature [8, 11, 28]. They are widely used in many applications. For example, in computer vision they are used for body and visual tracking [17, 61], face and object detection [54, 59], shape recognition [1], and others [16].

The success of forests is due to their ability to achieve low bias and low variance by combining weakly correlated trees, for which different ensembling mechanisms exist. Random Forests [7] rely mostly on variance reduction via averaging. Boosting, in its various forms [24, 53], aims at reducing both bias and variance. However, what is common to all these approaches is the way they construct the individual trees: they use a top-down induction algorithm such as CART [9] that recursively splits nodes. It is well known [28] that CART-type algorithms are highly suboptimal tree optimizers: the greedy nature of the splits they create means that suboptimal splits propagate down the tree. In fact, those algorithms ignore the exact form of the objective function altogether while growing the tree. (They do consider it when pruning the tree, but by this time it is too late: the split parameters cannot be changed, one can only remove nodes.) This has led to a perception that decision trees are generally low-accuracy models in isolation [28, p. 352], although combining a large number of trees does produce much more accurate models.

A recent algorithm, *Tree Alternating Optimization (TAO)* [12, 13], may change this situation. TAO monotonically decreases a desired objective function over a decision tree of given structure and finds much better approximate optima than CART-type algorithms. This motivates us to consider how well trees trained with TAO instead of CART would do when ensembled. While a better optimization of individual trees lowers their bias, it may also decrease their diversity and result in worse forests [36, p. 247]. We convincingly show that TAO forests do in fact work very well, even exceeding the accuracy of the state-of-the-art forest classifiers while using smaller forests. We do this by an extensive comparison across multiple datasets with Random Forests, AdaBoost, XGBoost [14] and other forest classifiers in section 5; and by a careful study of several diversity mechanisms that can be used to combine TAO trees in section 6. Before doing that, we review related work (section 2) and TAO in particular (section 3), and discuss the ensembling mechanism we propose (section 4).

2 RELATED WORK

The literature of ensemble learning for classification is huge [36, 65]. The most successful ensembles are forests, made of trees. We review tree learning and tree ensembling.

2.1 Learning a single tree

Finding optimal decision trees or even constant-factor approximations is NP-hard in most formulations of the problem [26, 31]. As any textbook on statistical learning [6, 28, 43] or specialized review of tree induction [9, 40, 46, 47, 50, 51] will show, the established way to learn classification trees from data is top-down induction. This grows the tree greedily and recursively. Starting at the root, a node is split into two children by optimizing a "purity" criterion (typically the Gini index or entropy), which seeks an axis-aligned or hyperplane partition of the space such that the training instances reaching each child belong to one class (or as few classes as possible). This is recursively repeated until each leaf contains instances from the same class (or some other stopping criterion is satisfied). Then, the resulting, usually large tree is optionally pruned in order to reduce overfitting. This is done by removing subtrees to optimize a cost-complexity tradeoff (note that we can remove nodes but we cannot change the node parameters themselves). Various refinements may be combined with this, leading to slightly different algorithms such as CART [9], ID3 [46] or C4.5 and C5.0 [47]. Multiple public and commercial implementations of these exist, e.g. [35, 45, 56].

The vast majority of papers and software focus on axis-aligned trees. Oblique trees, having a hyperplane split, have also been widely researched, involving approximate purity optimization over the hyperplane parameters [9, 44], or other split criteria such as linear discriminant analysis [23, 41], linear perceptrons [58], logistic regression [52, 57] or linear SVMs [3, 4, 10]. While oblique trees provide a more flexible model than axis-aligned ones, their optimization is slower and much less accurate than for axis-aligned splits. The small accuracy improvement that these oblique trees produce does not compensate for the increased complexity of the tree, and indeed oblique trees are rarely used in practice. *The recently proposed TAO algorithm [12, 13] can find much better optima of a regularized loss function over axis-aligned, oblique and other trees. We describe it later.*

2.2 Ensembling trees into a forest

Classification forests usually combine trees by majority voting, where each tree is trained by a CART-style algorithm but without the pruning stage, in order to reduce bias. Various mechanisms are used to introduce diversity among the trees, which is necessary for the forest to improve over a single tree. These involve training each tree independently on different samples (bootstrapped or random subsets) or different feature subsets, both of which are very successfully combined in Random Forests [7] and their variations [5, 25]. Boosting creates a stagewise additive model by training trees sequentially, each one trying to reduce the residual error given all previous trees. This can be done by reweighting the samples, as in AdaBoost [53], or by an approximate functional gradient, as in gradient boosting [14, 24]. While these are the most widely used forests, many other variations exist [16, 17, 34, 54]. As with single tree learning, most forests use axis-aligned trees, although a few have proposed oblique trees [7, 22, 42, 61]; however, their accuracy has not been found uniformly better than that of axis-aligned forests. Forests have also been combined with neural nets [34]. Finally, the high accuracy of forests comes at a computational price: they contain many, very deep trees, particularly for large datasets, which makes storage and inference costly. This has led to various works trying to reduce the size of a trained forest [65].

3 TAO: TREE ALTERNATING OPTIMIZATION

We give a brief description of the TAO algorithm, proposed in [13] in special form and in [12] in general form. TAO operates very differently from CART-style algorithms. There is no recursive growing of a large tree (using a heuristic split criterion) and then pruning it to avoid overfitting. TAO is much closer to the way we typically train a neural net via principled, efficient optimization methods. In a neural net, we select a fixed parametric architecture (number of layers and units, etc.), randomly initialize its parameters, and then iteratively update them so the objective function decreases, usually via a gradient-based method. In TAO, we select a tree structure[1] (e.g. a complete tree of given depth), randomly initialize its parameters (in the decision and leaf nodes), and then iteratively update all the parameters so the objective function decreases. TAO manages this via alternating optimization over the tree nodes, rather than via gradient-based methods, since the tree is nondifferentiable. And, with an appropriate regularization term on the parameters (e.g. ℓ_1), TAO can also prune the tree structure, for example if all the parameters in a node become zero—just as the Lasso "prunes" input features whose weights become zero [27].

In more detail, consider a tree $\mathbf{T}$ with nodes i and parameters $\boldsymbol{\Theta} = \{\boldsymbol{\theta}_i\}$ (e.g. $\boldsymbol{\theta}_i$ can be the weights and bias of a hyperplane decision node), and the objective function E:

$$\min_{\boldsymbol{\Theta}} E(\boldsymbol{\Theta}) = \sum_{n=1}^{N} L_n(y_n, \mathbf{T}(\mathbf{x}_n; \boldsymbol{\Theta})) + \lambda \sum_{\text{nodes } i} \phi_i(\boldsymbol{\theta}_i) \tag{1}$$

where the RHS contains two terms: the first is the loss (on each input instance $\mathbf{x}_n$ with ground-truth label y_n) and the second is a regularization term on the nodes' parameters. In [13], the loss was the 0/1 classification loss and $\phi_i(\boldsymbol{\theta}_i) = \|\boldsymbol{\theta}_i\|_1$ (an ℓ_1 sparsity penalty). However, the argument below readily extends to any loss (supervised or unsupervised) that separates additively over instances and any regularization that separates additively over nodes [12]. TAO is based on two theorems which we describe in English (see [12] for formal statements, proofs and further explanations). Define the *reduced set of node i* (whether decision node or leaf) as the training instances that reach i under the current tree.

Separability condition If nodes i and j in the tree (whether decision or leaves) are not descendants of each other (e.g. all nodes at the same depth), then E can be written *equivalently* as a separable function of the parameters of i and j. (This follows from the fact that the reduced sets of i and j are disjoint, because the tree makes hard, not soft, decisions, by

[1]It is possible to build a (say) CART tree and initialize TAO from it; TAO will then improve this tree. However, we find that using a complete tree of large enough depth Δ (i.e., having 2^Δ leaves) with random parameters at the nodes works better. This also makes TAO self-standing.

sending an instance down exactly one child at each decision node.)

Reduced problem The problem of optimizing E over the parameters $\boldsymbol{\theta}_i$ at a decision node i is *equivalent* to a weighted 0/1 loss binary classification problem over the node's decision function on its reduced set, where each instance is "pseudolabeled" as the child (left or right) that leads to a lower value of E under the current tree. (This follows from the fact that all a decision node can do with an instance is send it down its left or right child, and the ideal choice is the one that results in the best prediction downstream from that node.)
Optimizing E over a leaf i is *equivalent* to training its model parameters $\boldsymbol{\theta}_i$ on its reduced set to optimize E.

The separability condition allows us to optimize E separately (and in parallel) over the parameters of any set of nodes that are not descendants of each other (fixing the parameters of the remaining nodes). This has two advantages: 1) we expect a deeper decrease of the loss, because we optimize over a large set of parameters exactly; 2) the computation is fast: the joint problem over the set of nodes becomes a collection of smaller independent problems each over one node that can be solved in parallel (if so desired). The node sets can be derived via breadth-first or depth-first search. *The reduced problem theorem implies that optimizing E over a decision node's parameters reduces to optimizing a binary classifier (defined by those parameters) with the 0/1 loss with certain "pseudolabels".* While optimizing the 0/1 loss in general is NP-hard [30], we can approximate[2] it by a surrogate loss (e.g. logistic or hinge loss). Regularization terms over the tree parameters carry over to the reduced problem. As an example proposed in [13], *sparse oblique trees* use oblique (hyperplane) decision nodes with an ℓ_1 penalty; the reduced problem can be solved by an ℓ_1-regularized linear SVM or logistic regression [27, sections 3.2 and 3.6], a convex problem for which well-developed code exists, such as LIBLINEAR [21].

Finally, as TAO iterates, the root-leaf path followed by each training instance changes and so does the reduced set at each node. This can cause dead branches (whose reduced set is empty) and pure subtrees (which contain instances from a single class). They can be pruned after convergence. *This means that TAO can actually modify the tree structure, by reducing the size of the tree*; this is very significant with sparsity penalties. By starting with a deep enough tree structure and running TAO over a regularization path over $\lambda \in (0, \infty)$, we obtain a collection of trees with progressively fewer nodes and nonzero parameters at the nodes. We can select λ by cross-validation.

Effectively, what TAO does is *repeatedly train a simple binary classifier at each decision node and a simple K-class classifier at each leaf while monotonically decreasing the objective function E.* After each iteration the reduced set on which each classifier or predictor is trained changes.

[2]We can always guarantee a monotonic decrease in $E(\Theta)$ by not updating the parameters $\boldsymbol{\theta}_i$ in the rare case where the surrogate solution over node i increases E (we only observe this near convergence, and the increase is tiny).

The experiments in [13, 64] with single decision trees convincingly show that TAO does indeed find trees of much lower classification error than traditional algorithms such as CART. Our experiments in this paper confirm this with single trees and, further, show that ensembling such trees yields classifiers of astounding accuracy and small size.

4 TAO FORESTS: ENSEMBLING TAO TREES

We will use the name *TAO tree* to mean a tree trained with TAO and *TAO forest* to mean a forest of TAO trees. We consider two types of TAO trees (and forests), according to the type of classifier at the leaves: TAO-c uses a constant label and TAO-l uses a linear softmax classifier. Both of these use oblique (hyperplane) decision nodes. (TAO can also train axis-aligned trees, regression trees and others [12], but we focus here on oblique trees for classification.)

There are many potentially effective ways to ensemble trees. In this paper we focus on one simple way: each tree is trained independently and the ensemble prediction is obtained by majority voting (TAO-c) or by averaging class probabilities (TAO-l). Diversity among the trees can be introduced via different mechanisms, which we study empirically in section 6. Based on that, we recommend a procedure which achieves close-to-best accuracy and is simple (having few, intuitive hyperparameters). *We train each tree using all available features, taking as initial tree a complete binary tree of depth Δ with random node parameters, on a random sample of around 90% of the training set (or, more simply but usually with slightly lower accuracy, on a bootstrapped sample).* Also, we use a small sparsity penalty λ in order to remove unnecessary parameters without increasing the tree bias[3]. Hence, the TAO forest has two hyperparameters: the number of trees T and the depth Δ of each tree. This is the procedure we use when comparing with other algorithms in section 5.

5 EXPERIMENTS: TAO FORESTS VS STATE-OF-THE-ART FORESTS

We report an extensive comparison across well-known benchmarks of different characteristics (sample size N, dimensionality D, number of classes K; feature vectors that are dense, sparse or from pretrained deep nets; etc.) and types (images, documents, etc.); and multiple forest-based methods: AdaBoost, Random Forests (RF), gradient boosting (XGBoost version), and a few variations proposed in the literature. *In all datasets, TAO forests achieve the highest accuracy of any forest method, often by a large margin, while also using the smallest number of trees and smallest tree depth, and being also competitive in number of parameters and inference FLOPS.* Let us see this in detail.

5.1 Experiment setup

5.1.1 Comparison algorithms. We restrict our comparison to forest-based algorithms:

Random Forests (RF) [7] uses an ensemble of independent trees, each trained on a bootstrap subset of the training data (bagging). Each tree is trained using CART but each node

[3]One can also select λ by, say, cross-validation, but in the context of forests this is probably not worth the effort.

split can only use features from a random subset of size m. We do not restrict the `max_depth` hyperparameter and allow each tree to grow fully, as is recommended for RFs [7].

AdaBoost [53] is one of the earliest boosting frameworks. It uses a set of "weak learners" (typically shallow trees) which are trained sequentially. At each boosting iteration, the training instances are reweighted so the new learner focuses mostly on those instances misclassified by previous classifiers.

Gradient boosting [24] is a generalization of AdaBoost as an approximate gradient optimization in function space of stagewise additive models. We use the the highly optimized XGBoost implementation [14]. We use the Python API provided by the authors (CPU version).

Alternating Decision Forests (ADF) [54]: we compare with their published results (we do not run their algorithm). This algorithm essentially trains a random forest using a combination of boosting and greedy tree growing.

Shallow Neural Decision Forests (sNDF) [34]: we compare with their published results. We do not compare with other versions of the NDF since we consider only oblique or axis-aligned trees. A sNDF is a forest of soft oblique decision trees (i.e., an input instance follows all paths in the tree, each with a different probability).

Oblique Random Forests [42]: we compare with their published results. This algorithm, which is restricted to binary classification only, modifies regular Random Forests so that each hyperplane split (decision node) is given by a logistic regression learnt on the training points reaching that node (a related paper uses a linear SVM instead [61]).

Refined Random Forest (rRF) [48]: this takes as input a pretrained forest and globally optimizes over the parameters on the leaves, which results in an improved prediction error.

Note that, although Random Forests, AdaBoost and gradient boosting are considered to be robust to hyperparameter choice, sometimes they do require some tuning to do their best, depending on the dataset. We explored as best as we could their hyperparameters, often improving over reported results in the literature (e.g. for RF in several datasets in [54]). In particular, we tried different choices of the number of trees and maximum depth (see the tables). In agreement with previous work, we found that for Random Forests it is best to let the trees grow fully, and that the default value of $m = \sqrt{D}$ for the number of features each tree uses worked about optimally. In AdaBoost and XGBoost, we tune the most important hyperparameters `max_depth`, `n_estimators` and `learning_rate` on a subset of the training data for each dataset separately. All other parameters, such as criterion, booster, etc., are set to their default values. For RF and AdaBoost, we use the Python implementation in scikit-learn [45].

We ran our experiments (for all methods) in a computer with an Intel Xeon CPU E5-2699 v3 @ 2.30GHz and 128 GB RAM. We did not use any GPUs.

5.1.2 TAO. We use oblique decision trees (having a hyperplane function at each decision node) with constant leaves (TAO-c) or linear softmax leaves (TAO-l). We take as initial tree a complete binary tree of given depth (Δ in the tables) with random parameters at each node. We train each TAO tree on a 90% random sample of the training data using 40 iterations and a small sparsity penalty of $\lambda = 0.01$. We report the mean error (training and test) and standard deviation over 5 independent runs.

We implemented TAO in Python 2.7.15 with process level parallel processing. In TAO, the node optimization involves an ℓ_1-regularized logistic regression at each decision node and (for TAO-l) either an ℓ_1-regularized logistic regression (for $K = 2$ classes) or an ℓ_1-regularized linear softmax (for $K > 2$ classes) in each leaf. We solve the logistic regression using LIBLINEAR [21] and the linear softmax using SAGA [19], both of which are available inside scikit-learn [45].

Dataset	N_{train}	N_{test}	D	K
Letter	16 000	4 000	16	26
MNIST	60 000	10 000	784	10
Char74k	66 707	7 400	64	62
ImageNet	62 855	12 800	8192	64
R8	5 485	2 189	400	8
RCV1	15 564	518 571	47 236	53
SensIT	78 823	19 705	100	3
SUSY	4.5M	0.5M	18	2

Table 1: Datasets used in our experiments: number of points for training and test (N_{train}, N_{test}), number of features D, number of classes K.

5.1.3 Forest size: number of parameters and FLOPS. For each method's forest, we report its total number of parameters and (estimated) FLOPS for inference:

Total number of parameters We count the parameters for each node of each tree (decision nodes and leaves). In a decision node we count the number of nonzero weights (and the bias), which is 2 for an axis-aligned tree and $D+1$ for an oblique tree (where D is the number of features). In a leaf, we count one for constant-label leaves, $D+1$ for a logistic regression classifier ($K = 2$) and DK for a linear softmax classifier (with $K > 2$).

Inference FLOPS We take the inference time (FLOPS) for one instance along one tree as the number of nonzero parameters it encounters in the root-leaf path it follows. We repeat this for each tree in the forest and average over all training instances.

For the forests created by some algorithms, we do not have access to the actual forest, in which case we report an upper bound and mark it with parentheses in the tables. This is computed as follows:

Total number of parameters For each tree of depth Δ (but not necessarily complete), the maximum number of decision nodes and leaves is $\max(N, 2^{\Delta}) - 1$ and $\max(N, 2^{\Delta})$, respectively, where N is the number of training points. We then count the number of parameters as above and multiply it times the number of trees.

Inference FLOPS We assume that each root-leaf path has a depth exactly equal to Δ.

5.1.4 Datasets. Table 1 summarizes the characteristics of the datasets used in our experiments. Section 8.1 has a description of the datasets.

5.2 Results

Tables 2–6 show the results (sorted by decreasing test error). We include results with the same method but different forest size T (number of trees) and (maximum) depth Δ of each tree. Some results are quoted from published work; we obtained the rest ourselves using existing method implementations. We observe the following.

Our results for AdaBoost, RF and XGBoost are in overall agreement with previous works. Which of them has highest accuracy depends on the dataset, although (with well-set hyperparameters and especially with sufficiently many trees) they generally are close to each other. RFs are simplest to use in terms of hyperparameters and are extremely fast to train. XGBoost (a highly optimized implementation of gradient boosting for trees) takes much longer to train. It also generates forests with many more trees: each iteration (each new additive model) of a gradient boosting forest adds K trees if there are K classes [24].

Let us now look at TAO. First, consider the case of a single TAO tree ($T = 1$ in the tables). It is remarkable that, in many cases, *a single TAO tree already performs very well compared with state-of-the-art forest methods*, exceeding the accuracy of small forests, particularly if the TAO tree has linear leaves (TAO-l). This is extreme in ImageNet, where a single TAO-l tree beats all AdaBoost, RF and XGBoost forests. This amplifies the experimental findings of [13], which showed that TAO could learn constant-label trees of much higher accuracy that traditional tree learning algorithms such as CART. All these results show that TAO is able to find good optima of the tree learning problem, and makes decision trees—long considered as low-accuracy models—serious contenders in terms of accuracy.

Second, consider the case of TAO forests ($T > 1$). TAO trees with constant leaves (TAO-c) already achieve higher accuracy than most forest methods. However, we will focus on TAO trees with linear leaves (TAO-l), which we find always beat TAO-c and use smaller trees. We see that *TAO-l has the lowest test error in all datasets, often by a considerable margin over the other forest methods*. Also, *TAO-l forests have few, shallow trees*. TAO-l forests use few trees (up to 30 in all datasets), much less than the other forest methods, which need 100s or 1000s of trees to achieve their best accuracy and yet cannot match the accuracy of TAO-l. TAO-l forests also use shallow trees (depth up to 7 in all datasets), far shallower than the other forest methods, whose depth can exceed 100. This is because such methods use axis-aligned trees which are typically unbalanced and grown very deep. The TAO-l trees are mostly complete but pruning of nodes does occur during TAO training.

Consider now the number of parameters and inference time. Most forest methods use axis-aligned trees, where each decision node thresholds a single input feature, while we use oblique trees in the TAO forests we report. Compared to axis-aligned trees, oblique trees are shallower and their forests require fewer trees. However, each oblique node uses $D + 1$ parameters (hyperplane weights and bias) while an axis-aligned node uses just 2 (feature index and bias). TAO oblique trees use fewer than $D + 1$ parameters because we run TAO with a tiny sparsity penalty, which does make a significant number of parameters zero with essentially no error increase. Which type of forest has higher number of parameters and inference time is an empirical question. As shown in the tables, *when we compare the most accurate TAO forests with the most accurate AdaBoost, XGBoost or Random forests, the TAO forests usually have lower number of parameters and FLOPS.*

Table 5 uses a large dataset, SUSY [2], containing two highly overlapping classes. Here, TAO-c and TAO-l achieve the same accuracy, beating all other forest methods, and achieving an AUC score very close to the best reported in [2], which used deep neural nets.

Finally, table 6 compares with the "oblique Random Forest" method of [42], which is restricted to binary classification only. It modifies regular RFs so that each hyperplane split (decision node) is given by a logistic regression learnt on the training points reaching that node (a related paper uses a linear SVM instead [61]). We ran TAO-c on binary classification problems derived from MNIST reported in [42]; its advantage is clear.

5.3 Runtime

Training a TAO oblique tree depends on the tree size, dataset size and number of iterations. To give an idea, using our unoptimized Python implementation[4] in a single CPU of a PC, training a TAO-c tree of depth 8 for 20 iterations on MNIST takes 14 minutes; and training a TAO-c tree of depth 13 and a TAO-l tree of depth 3 for 40 iterations on ImageNet take 409 and 295 minutes, respectively. It is hard to compare runtimes with RF and XGBoost because of the difference in accuracy, but if we consider the most accurate forest of each type, then RF (8.4 minutes on ImageNet) is faster by about an order of magnitude than TAO and XGBoost (493 minutes on ImageNet), which have comparable runtimes. We provide a full comparison of runtimes on ImageNet (see table 4) since it was the most time consuming dataset for most of the methods. Note the TAO trees are independent and can be trained in parallel, or even in a distributed system. They are also more suited to hardware accelerations than axis-aligned trees: since they are oblique, they use vector (rather than scalar) operations that are suited to GPUs; since they have a complete binary structure, they can be stored without pointers in an array as a binary heap [15].

6 EXPERIMENTS: STUDY OF DIVERSITY MECHANISMS

Accurate forests aim at low bias and low variance [36, 65]. We achieve low bias by using deep enough oblique trees and optimizing them well with TAO. We train each tree independently in order to decorrelate them with each other and reduce variance. Multiple mechanisms exist to achieve this, some generic and widely used with CART-style trees and some being specific to TAO trees (such as using different initial trees, number of iterations, tree structures or sparsity penalty). We study them systematically, as well as the TAO hyperparameters, in experiments on two datasets, MNIST and Letter. As we will see, some mechanisms work and some do not, and the reasons have to do with how bias and variance interact. We conclude with a recommendation.

[4]We now have a C implementation, which also runs LIBLINEAR with lower accuracy, which is several times faster than the Python code used in this paper.

	Forest	E_{test} (%)	#params.	FLOPS	T	Δ
MNIST (60k,784,10)	CART	12.11±0.04	5 957	(50)	1	50
	TAO-c	5.25±0.20	24k	1 163	1	8
	TAO-l	5.07±0.13	16k	1 194	1	5
	AdaBoost	4.92±0.07	63k	(1 000)	100	10
	AdaBoost	3.18±0.05	611k	(2 955)	100	30
	RF	3.05±0.06	1M	(3 482)	100	46
	AdaBoost	2.96±0.05	5.9M	(29k)	1k	30
	RF	2.84±0.06	10M	(35k)	1k	48
	sNDF [34]	2.80±0.12	22M	22M	80	10
	XGBoost	2.73±0.00	390k	(17k)	1k	30
	ADF [54]	2.71±0.10	(3.6M)	(2 500)	100	25
	TAO-c	2.31±0.08	1.2M	52k	40	8
	XGBoost	2.17±0.00	540k	(57k)	10k	30
	TAO-l	2.11±0.09	469k	33k	30	5
	rRF[48]	2.05±0.02	(160k)	(2 500)	100	25
	TAO-l	2.02±0.06	475k	38k	30	6
Letter (16k,16,26)	CART	13.06±0.15	2 985	(27)	1	27
	TAO-c	9.59±0.31	9 904	111	1	11
	TAO-l	6.60±0.33	6 449	192	1	6
	XGBoost	4.40±0.00	268k	(33k)	2.6k	30
	XGBoost	4.00±0.00	551k	(124k)	26k	30
	RF	3.77±0.06	419k	(2 836)	100	34
	ADF [54]	3.52±0.12	(960k)	(2 500)	100	25
	RF	3.44±0.09	4.2M	(28k)	1k	36
	AdaBoost	3.01±0.06	271k	(2 000)	100	20
	rRF[48]	2.98±0.15	(180k)	(2 500)	100	25
	sNDF [34]	2.92±0.17	2.4M	2.4M	70	10
	TAO-c	2.88±0.09	310k	3 210	30	11
	AdaBoost	2.69±0.04	2.7M	(20k)	1k	20
	TAO-l	2.23±0.09	202k	6 150	30	6
	TAO-l	2.09±0.10	276k	6 310	30	7
Char74k (67k,64,62)	CART	32.14±0.15	20 157	(55)	1	55
	TAO-c	23.94±0.37	46k	432	1	12
	TAO-l	20.82±0.31	42k	2 839	1	4
	XGBoost	18.08±0.00	1.7M	(116k)	6.2k	50
	AdaBoost	17.86±0.15	2.5M	(5 980)	100	60
	RF	17.33±0.14	2.6M	(5 014)	100	65
	XGBoost	17.04±0.00	3.3M	(923k)	62k	50
	AdaBoost	16.93±0.18	25M	(60k)	1k	60
	ADF [54]	16.67±0.21	(4M)	(2 500)	100	25
	RF	16.61±0.14	26M	(51k)	1k	65
	sNDF [34]	16.04±0.20	59M	59M	200	12
	rRF[48]	15.40±0.10	(1.1M)	(2 500)	100	25
	TAO-c	15.19±0.18	1.5M	13k	30	12
	TAO-l	15.00±0.17	1.4M	92k	30	4

Table 2: Comparison on different datasets (MNIST, Letter, Char74k; dataset specs indicated as (N, D, K)) of different forest methods: TAO-c (constant leaves), TAO-l (linear softmax leaves), AdaBoost, XGBoost, Random Forests (RF), Alternating Decision Forest (ADF) [54] and shallow Neural Decision Forest (sNDF) [34]. We report the test error (%, avg±stdev over 5 repeats), number of parameters and FLOPS (numbers in parentheses are estimates), number of trees T and maximum depth of the forest Δ.

	Forest	E_{test} (%)	#params.	FLOPS	T	Δ
SensIT (79k,100,3)	CART	21.44±0.18	12 571	(82)	1	82
	TAO-c	14.57± 0.09	3 480	307	1	7
	TAO-l	14.44± 0.08	2 325	296	1	5
	RF	13.91± 0.06	1.6M	(5 201)	100	71
	AdaBoost	13.83± 0.07	728k	(5 607)	100	60
	XGBoost	13.68± 0.00	1.1M	(8 849)	300	50
	RF	13.63± 0.08	16M	(52k)	1k	76
	AdaBoost	13.18± 0.08	7M	(56k)	1k	60
	XGBoost	13.14± 0.00	2M	(66k)	3k	50
	TAO-c	12.83± 0.06	422k	16k	30	9
	TAO-l	12.79± 0.09	309k	20k	30	5
R8 (5k,400,8)	CART	17.60±0.17	731	(24)	1	24
	TAO-c	6.64± 1.04	3 272	307	1	7
	RF	6.16± 0.35	93k	(1 996)	100	27
	AdaBoost	5.85± 0.07	61k	(1 906)	100	20
	RF	5.57± 0.56	932k	(20k)	1k	27
	XGBoost	5.43± 0.00	44k	(6 048)	800	30
	AdaBoost	5.11± 0.09	593k	(19k)	1k	20
	XGBoost	5.01± 0.00	65k	(12k)	8k	30
	TAO-l	4.93± 0.09	825	286	1	2
	TAO-c	3.98± 0.20	66k	6 913	20	7
	TAO-l	3.61± 0.06	22k	6 380	20	2
RCV1 (16k,47k,53)	CART	29.33±0.13	3 460	(150)	1	150
	RF	19.84± 0.42	1M	(20k)	100	233
	RF	18.78± 0.37	10M	(0.2M)	1k	233
	TAO-c	17.96± 0.03	3.2M	37k	1	12
	AdaBoost	16.81± 0.38	409k	(9 926)	100	100
	AdaBoost	15.95± 0.39	4M	(99k)	1k	100
	TAO-l	15.73± 0.21	14k	3 357	1	4
	XGBoost	14.30± 0.00	244k	(48k)	5.3k	30
	XGBoost	13.84± 0.00	522k	(151k)	53k	30
	TAO-c	13.77± 0.02	84M	1.1M	30	12
	TAO-l	13.29± 0.03	411k	98k	30	4

Table 3: Like table 2 but for SensIT, R8 and RCV1.

Forest	E_{test} (%)	#params.	FLOPS	T	Δ	Time
AdaBoost	>24 hours runtime			100	50	
CART	44.62± 0.32	19 913	(296)	1	296	10.3
TAO-c	14.69± 0.18	3.7M	16k	1	13	409
RF	13.62± 0.32	2.6M	(22k)	100	220	2.2
RF	12.67± 0.13	13M	(109k)	500	218	4.2
RF	12.51± 0.11	25M	(224k)	1k	220	8.4
XGBoost	12.51	596k	(81k)	6.4k	50	417
XGBoost	11.01	782k	(124k)	32k	50	453
XGBoost	10.78	973k	(180k)	64k	50	493
TAO-l	10.77± 0.08	431k	431k	1	1	109
TAO-l	10.45± 0.19	1.1M	254k	1	3	295
TAO-c	08.25± 0.11	100M	0.5M	30	13	516
TAO-l	08.19± 0.13	32M	8.0M	30	3	361

Table 4: Like table 2 but for the ImageNet (= a subset of 64 classes of the full ImageNet, using VGG16 features). Additionally, we report the training time of the entire forest in minutes.

6.1 Training on different samples

Figures 1–2 show the training and test error of a TAO forest where each tree is trained on a subset of the training set (of size N): either a bootstrap sample (N instances sampled with replacement) or a random sample ($M < N$ instances sampled without replacement). We see that the best test accuracy occurs if using $M \approx 90\%$ samples (although the optimal size will depend on the dataset), which is generally quite better than with a bootstrap sample, and also faster to train (since the samples are smaller). However, the bootstrap is a simple, parameterless choice, and likely more useful with small training sets.

Forest	AUC	E_{test}(%)	#params.	FLOPS	T	Δ
AdaBoost	–	21.14	150k	(4 000)	500	8
RF	0.8584	20.97	252k	(4 000)	500	8
BDT [2]	0.8630	–	–	–	–	–
XGBoost	0.8717	19.71	221k	(4 000)	500	8
TAO-c	–	19.65	128k	4 101	30	8
TAO-l	0.8744	19.64	34k	3 210	30	6
sNN [2]	0.8750	–	–	–	–	–
dNN [2]	0.8790	–	–	–	–	–

Table 5: Like table 2 but for SUSY (N=4.5M, D=18, K=2). We also report the AUC score for those methods that output class probabilities. The results of boosted decision trees (BDT) and shallow/deep neural nets (sNN/dNN) are from [2]. Both sNN and dNN are multilayer perceptrons with tanh activation; sNN has a single hidden layer with up to 10k units and dNN has up to 6 hidden layers and up to 500 units each.

MNIST classes	SVM	kNN	AdaBoost	RF	oRF	**TAO-c**
digit-2 vs digit-4	1.3	0.4	2.4	1.7	1.5	0.5
digit-3 vs digit-8	2.3	3.6	3.1	3.2	4.3	0.8
even vs odd	7.4	7.2	9.1	7.8	9.8	1.5

Table 6: Comparison (test error) on MNIST binary classification tasks of TAO, oblique Random Forest (oRF) [42] and other methods (taken from [42]). For TAO we report mean test error on 10-fold cross-validation, as in [42]. AdaBoost, RF and oRF use T = 300 fully grown trees and TAO uses T = 30 trees of depth Δ = 5.

6.2 Training using a different initial tree

Figures 1–2 also show the effect of having the initial tree parameters (the hyperplane weights and bias) be random (the structure is the same for all trees: complete of depth Δ). Clearly, using different random parameters for each tree is better than using the same for each tree. This makes intuitive sense in that different initial trees will likely give rise to different local optima and further increase diversity beyond the use of different samples.

6.3 Training on different feature subsets

Table 7 shows the training and test accuracy if training each tree on a random subset of input features (without replacement). We see that the best accuracy in both training and test occurs if using all features and decreases the fewer features are used (whether picked locally at each node or globally at each tree). This is in stark contrast to what happens with trees in the Random Subspace [29], Random Forest [7] and Extremely Randomized Trees [25] methods, which greatly benefit from using a quite small subset of features for each tree ($\sqrt{D}$ in RF by default). We believe the reason is the TAO optimization, which will achieve a more accurate tree (lower bias) if using all features, even if this may decrease diversity somewhat. CART works very differently: by partitioning the input space ever more finely, it can always achieve low training error even if not using all features, and indeed RF grows trees fully without pruning (although each individual tree is very deep and grossly overfits).

A related diversity mechanism is to train each tree on a random rotation of the features [49]. However, this is ineffective with

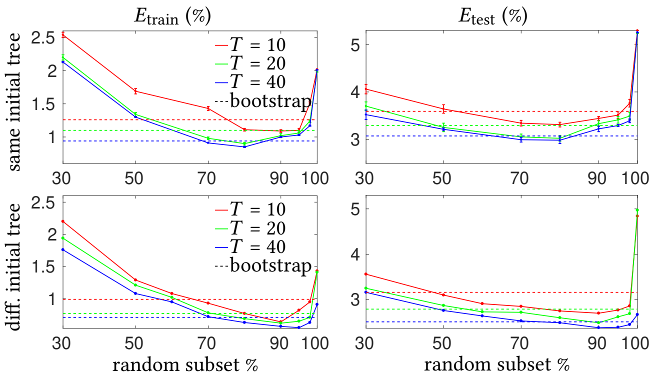

Figure 1: Diversity mechanism (MNIST): training each of the T trees on a bootstrapped (dashed horizontal lines) or random data subset (solid lines, with size given by the X axis as a percentage of the total training set). All trees are complete of depth 8 and their initial parameters are random. The T trees in the forest use the same initial random tree (*top*) or each uses a different initial random tree (*bottom*).

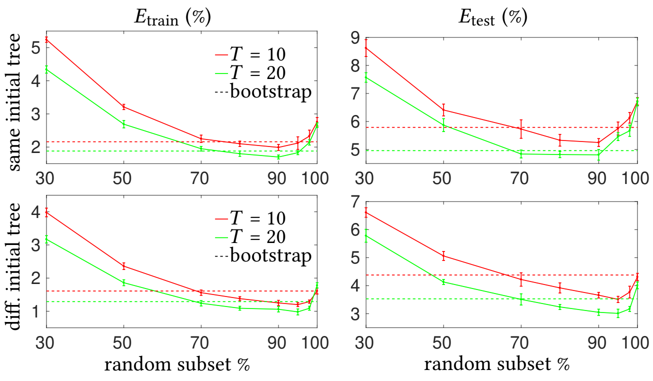

Figure 2: Like fig. 1 but for Letter dataset. All trees have depth 11.

oblique trees because we can absorb the rotation, or indeed any linear transformation, in each decision node hyperplane.

6.4 Number of TAO iterations: interaction with tree depth and number of trees

Figures 3–4 explore the interaction of the number of TAO iterations I that are run to train each tree and the two forest hyperparameters: tree depth Δ (the same for each tree, which is complete) and number of trees T. The results make sense. More iterations monotonically decrease the training error but eventually increase the test error, indicating overfitting (for a model of constant size). The fewer the trees, the fewer TAO iterations are needed for overfitting to occur. *This clearly shows the effectiveness of TAO's optimization and its strong benefit in reducing bias.*

While some papers (e.g. [7]) originally claimed that Random Forests or boosting do not overfit as a function of the number of

	Size $m = D^\alpha$ of feature subset	$T = 10$ E_{train} (%)	E_{test}	$T = 20$ E_{train} (%)	E_{test}
	$\alpha = 1$ (all features)	0.99	3.16	0.77	2.79
Local	$\alpha = 9/10$	1.26	3.18	1.10	2.81
	$\alpha = 8/10$	1.90	3.57	1.51	3.03
	$\alpha = 7/10$	2.88	4.02	2.51	3.86
	$\alpha = 6/10$	4.47	5.32	3.78	4.48
	$\alpha = 5/10$	6.76	7.52	5.50	5.95
Global	$\alpha = 9/10$	1.24	3.20	1.11	2.88
	$\alpha = 8/10$	1.98	3.63	1.63	3.18
	$\alpha = 7/10$	4.19	4.65	3.88	4.21
	$\alpha = 6/10$	5.11	6.13	5.01	5.68
	$\alpha = 5/10$	12.17	13.41	11.79	12.25

Table 7: Diversity mechanism (MNIST): training each of the T trees on a different, random feature subset of size $m = D^\alpha$ where D is the total number of features. "Local" means each node picks m features at random, "global" means each tree picks m features at random, the same for each node. All trees are complete of depth 8 and their initial parameters are random.

trees, in fact they do, as argued in [28, sec. 15.3.4]. This is to be expected as the number of parameters grows. What happens is that overfitting enters very slowly because the trees are poorly optimized, and this produces huge forests. It would be better to achieve forests of the same accuracy but which overfit much faster, so we need fewer trees. This is exactly what happens with TAO forests.

Fig. 5 compares TAO, RF and XGBoost as a function of the number of trees T. All of them reduce both training and test error as T increases; RF and XGBoost even reach zero training error because their trees are so deep. In contrast, TAO uses shallower trees that do not reach zero training error but greatly lower the test error, even with just 10 trees.

6.5 Ensembling the regularization path

The regularization path over the ℓ_1 sparsity hyperparameter $\lambda \in (0, \infty)$ in eq. (1) provides a way to generate a collection of diverse trees spanning a range of depths and number of nodes: from a complete tree of depth Δ for $\lambda = 0$ to a single-node tree for large enough λ. Hence, we considered ensembling trees, each for a different λ value.

Table 8 shows that, while the forest-over-λ improves significantly over the best single tree, it does quite worse compared to training on bootstrap samples. Two possible reasons are that many of the trees are small and the bias increases, and that many trees share subtrees, which reduces diversity.

6.6 Random tree structures

Since TAO uses a given tree structure, this provides another source of diversity: rather than using the same structure for each tree (complete of depth Δ), we can generate a random structure for each tree (and initialize its parameters randomly). The number of tree structures grows faster than exponentially with either the tree depth or number of nodes, which provides considerable potential for diversity among the trained trees. We considered a simple distribution over structures (see section 8.2), where all tree levels are

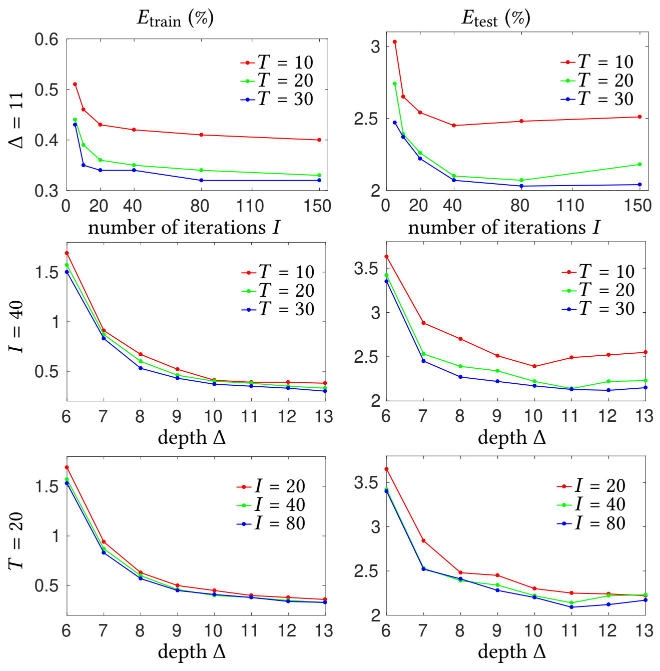

Figure 3: Training (column 1) and test error (column 2) on MNIST for TAO forests as a function of 3 factors: tree depth Δ, number of TAO iterations I and number of trees T. Each row fixes one factor and varies the other two.

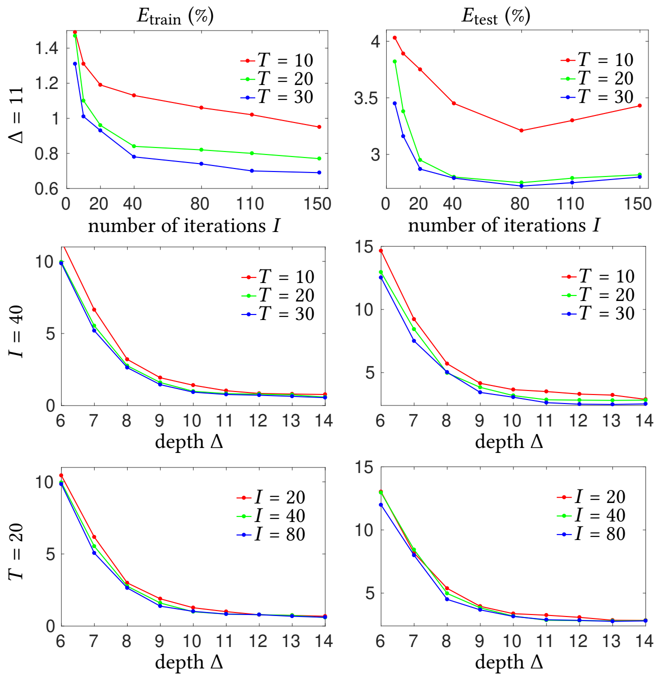

Figure 4: Like fig. 3 but for Letter dataset.

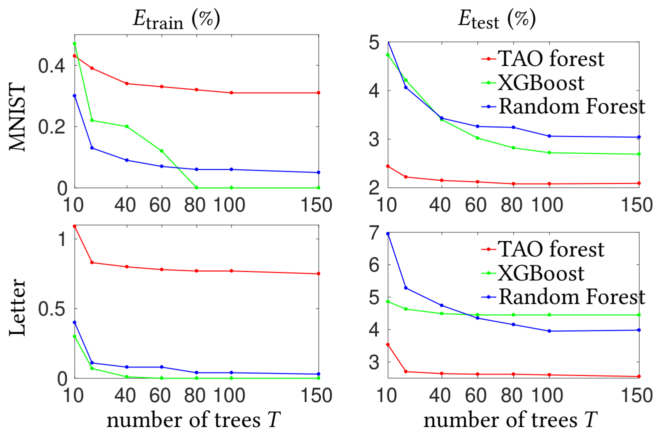

Figure 5: Training (column 1) and test error (column 2) on MNIST (row 1) and Letter (row 2) for TAO forests (80 iterations, tree depth 11), and XGBoost and Random Forests (tree depth 25) as a function of the number of trees T.

Method	$T = 10$		$T = 20$	
	E_{train} (%)	E_{test}	E_{train} (%)	E_{test}
forest-over-λ	1.15	3.71	0.89	3.50
best tree in regul. path	2.49	4.94	2.44	4.84
bootstrap	0.99	3.16	0.77	2.79

Table 8: Diversity mechanism (MNIST): forest-over-λ (row 1), consisting of a forest of T trees, from a regularization path using T values $\lambda \in [0.002, 200]$ (uniformly spaced in log scale). For reference, we show the error of the best tree in the path (row 2) and the error with bootstrap sampling (row 3, from fig. 1).

complete up to depth Δ_1, and then we split nodes recursively with probability p and up to a maximum depth Δ_2. Table 9 shows that this can indeed improve the accuracy over using same-structure trees (with the same number of nodes, for fairness). However, the improvement seems marginal, and is probably not worth the effort. That said, our exploration of structure distributions is limited, and there may be better ones.

6.7 Discussion and recommendation

The previous considerations and experiments suggest that, roughly speaking, the most accurate TAO forests may be achieved by, most importantly, reducing bias, to capitalize on TAO's optimization ability. Hence, we use all input features, and the largest forest (T trees of depth Δ) and number of iterations I possible computationally, unless overfitting is observed. Also, we use a small sparsity penalty $\lambda = 0.01$, which helps to remove unnecessary features with essentially no error increase[5]. We diversify the trees by training them independently on random (or bootstrap) samples and random initial trees (complete of depth Δ but with random parameters). This is what we used in section 5.

[5]This opens interesting possibilities for feature selection and ranking, although we do not consider this here.

$\Delta_1 \backslash \Delta_2$	8	9	10	11	12	13	14
2	–	2.41	2.43	2.45	2.48	2.56	2.79
3	–	2.38	2.42	2.47	2.48	2.52	2.68
4	–	**2.37**	2.40	**2.37**	2.41	2.47	2.46
5	–	**2.31**	2.40	**2.31**	2.39	2.43	2.51
6	–	**2.35**	2.41	**2.36**	2.39	2.42	2.47
7	–	2.40	**2.37**	**2.36**	**2.34**	2.40	**2.34**
8	**2.37**	–	–	–	–	–	–

Table 9: Diversity mechanism (MNIST): random tree structures (using $T = 100$ trees and 40 TAO iterations). An entry at (Δ_1, Δ_2) gives the test error (%) of a forest whose tree structures are complete up to depth Δ_1 and contain more nodes at random up to a depth Δ_2; these nodes split with a probability p, chosen so that the total number of nodes in the tree is 511 on average. The baseline is the entry at (8,8), which is a complete binary tree of depth 8. Entries with lower or equal error are boldfaced.

7 CONCLUSION

Random Forests (closely followed by AdaBoost and gradient boosting) have long been considered the best off-the-shelf classifiers, due to their ease of use and high accuracy. However, these methods rely on heuristic tree learning algorithms such as CART. We show that, by using the recently proposed TAO algorithm to learn each tree instead of CART, and by ensembling trees trained on bootstrapped or random data samples, we obtain smaller forests with consistent, significant improvements in accuracy. This is a remarkable result in that 1) it has immediate practical application in statistics, machine learning, computer vision and other areas, and 2) it opens new research directions in ensemble learning based on better tree optimization. The improvement of TAO is not restricted to classification with bagged trees: we have determined that TAO also improves consistently in regression forests [63] and with boosting instead of bagging (submitted).

REFERENCES

[1] Yali Amit and Donald Geman. 1997. Shape Quantization and Recognition with Randomized Trees. *Neural Computation* 9, 7 (Oct. 1997), 1545–1588.
[2] Pierre Baldi, Peter Sadowski, and Daniel Whiteson. 2014. Searching for Exotic Particles in High-Energy Physics with Deep Learning. *Nature Communications* 5 (2014), 4308.
[3] Kristin P. Bennett. 1992. Decision Tree Construction via Linear Programming. In *Proc. 4th Midwest Artificial Intelligence and Cognitive Sience Society Conference*. 97–101.
[4] Kristin P. Bennett, Donghui Wu, and Leonardo Auslender. 1999. On Support Vector Decision Trees for Database Marketing. In *Int. J. Conf. Neural Networks (IJCNN'99)*. Washington, DC, 904–909.
[5] Gérard Biau and Erwan Scornet. 2016. A Random Forest Guided Tour. *TEST* 25, 2 (June 2016), 197–227 (with comments, pp. 228–268).
[6] Christopher M. Bishop. 2006. *Pattern Recognition and Machine Learning*. Springer-Verlag, Berlin.
[7] Leo Breiman. 2001. Random Forests. *Machine Learning* 45, 1 (Oct. 2001), 5–32.
[8] Leo J. Breiman. 1998. Arcing Classifiers. *Annals of Statistics* 26, 3 (June 1998), 801–824 (with comments, pp. 824–849).
[9] Leo J. Breiman, Jerome H. Friedman, R. A. Olshen, and Charles J. Stone. 1984. *Classification and Regression Trees*. Wadsworth, Belmont, Calif.
[10] Donald E. Brown, Clarence Louis Pittard, and Han Park. 1996. Classification Trees with Optimal Multivariate Decision Nodes. *Pattern Recognition Letters* 17, 7 (June 10 1996), 699–703.
[11] Peter Bühlmann and Sara van de Geer. 2011. *Statistics for High-Dimensional Data*. Springer-Verlag.
[12] Miguel Á. Carreira-Perpiñán. 2020. The Tree Alternating Optimization (TAO) Algorithm: A New Way To Learn Decision Trees and Tree-Based Models. (2020).

arXiv.

[13] Miguel Á. Carreira-Perpiñán and Pooya Tavallali. 2018. Alternating Optimization of Decision Trees, with Application to Learning Sparse Oblique Trees. In *Advances in Neural Information Processing Systems (NEURIPS)*, S. Bengio, H. Wallach, H. Larochelle, K. Grauman, N. Cesa-Bianchi, and R. Garnett (Eds.), Vol. 31. MIT Press, Cambridge, MA, 1211–1221.

[14] Tianqi Chen and Carlos Guestrin. 2016. XGBoost: A Scalable Tree Boosting System. In *Proc. of the 22nd ACM SIGKDD Int. Conf. Knowledge Discovery and Data Mining (SIGKDD 2016)*. San Francisco, CA, 785–794.

[15] Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, and Clifford Stein. 2009. *Introduction to Algorithms* (third ed.). MIT Press, Cambridge, MA.

[16] Antonio Criminisi and Jamie Shotton. 2013. *Decision Forests for Computer Vision and Medical Image Analysis.* Springer-Verlag.

[17] Antonio Criminisi, Jamie Shotton, and Ender Konukoglu. 2012. Decision Forests: A Unified Framework for Classification, Regression, Density Estimation, Manifold Learning and Semi-Supervised Learning. *Foundations and Trends in Computer Graphics and Vision* 7, 2–3 (2012), 81–227.

[18] Teófilo E. de Campos, Bodla Rakesh Babu, and Manik Varma. 2009. Character Recognition in Natural Images. In *Proc. Int. Conf. Computer Vision Theory and Applications (VISAPP 2009)*. Lisbon, Portugal, 273–280.

[19] Aaron Defazio, Francis R. Bach, and Simon Lacoste-Julien. 2014. SAGA: A Fast Incremental Gradient Method with Support for Non-Strongly Convex Composite Objectives. In *Advances in Neural Information Processing Systems (NIPS)*, Z. Ghahramani, M. Welling, C. Cortes, N. D. Lawrence, and K. Q. Weinberger (Eds.), Vol. 27. MIT Press, Cambridge, MA, 1646–1654.

[20] Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Li Fei-Fei. 2009. ImageNet: A Large-Scale Hierarchical Image Database. In *Proc. of the 2009 IEEE Computer Society Conf. Computer Vision and Pattern Recognition (CVPR'09)*. Miami, FL, 248–255.

[21] Rong-En Fan, Kai-Wei Chang, Cho-Jui Hsieh, Xiang-Rui Wang, and Chih-Jen Lin. 2008. LIBLINEAR: A Library for Large Linear Classification. *J. Machine Learning Research* 9 (Aug. 2008), 1871–1874.

[22] Eibe Frank and Stefan Kramer. 2004. Ensembles of Nested Dichotomies for Multi-Class Problems. In *Proc. of the 21st Int. Conf. Machine Learning (ICML'04)*. Banff, Canada, 305–312.

[23] Jerome H. Friedman. 1977. A Recursive Partitioning Decision Rule for Nonparametric Classification. *IEEE Trans. Computers* 26, 4 (April 1977), 404–408.

[24] Jerome H. Friedman. 2001. Greedy Function Approximation: A Gradient Boosting Machine. *Annals of Statistics* 29, 5 (2001), 1189–1232.

[25] Pierre Geurts, Damien Ernst, and Louis Wehenkel. 2006. Extremely Randomized Trees. *Machine Learning* 63, 1 (April 2006), 3–42.

[26] Thomas Hancock, Tao Jiang, Ming Li, and John Tromp. 1996. Lower Bounds on Learning Decision Lists and Trees. *Information and Computation* 126, 2 (May 1 1996), 114–122.

[27] Trevor Hastie, Robert Tibshirani, and Martin Wainwright. 2015. *Statistical Learning with Sparsity: The Lasso and Generalizations.* Chapman & Hall/CRC.

[28] Trevor J. Hastie, Robert J. Tibshirani, and Jerome H. Friedman. 2009. *The Elements of Statistical Learning—Data Mining, Inference and Prediction* (second ed.). Springer-Verlag.

[29] Tin Kam Ho. 1998. The Random Subspace Method for Constructing Decision Forests. *IEEE Trans. Pattern Analysis and Machine Intelligence* 20, 8 (Aug. 1998), 832–844.

[30] K.-U. Hoffgen, H. U. Simon, and K. S. Vanhorn. 1995. Robust Trainability of Single Neurons. *J. Computer and System Sciences* 50, 1 (Feb. 1995), 114–125.

[31] Laurent Hyafil and Ronald L. Rivest. 1976. Constructing Optimal Binary Decision Trees Is NP-Complete. *Inform. Process. Lett.* 5, 1 (May 1976), 15–17.

[32] Donald E. Knuth. 1997. *The Art of Computer Programming Vol. 1: Fundamental Algorithms* (third ed.). Addison-Wesley, Reading, MA.

[33] Donald E. Knuth. 2011. *The Art of Computer Programming Vol. 4A: Combinatorial Algorithms, Part 1.* Addison-Wesley.

[34] Peter Kontschieder, Madalina Fiterau, Antonio Criminisi, and Samuel Rota Buló. 2015. Deep Neural Decision Forests. In *Proc. 15th Int. Conf. Computer Vision (ICCV'15)*. Santiago, Chile, 1467–1475.

[35] Max Kuhn, Steve Weston, Mark Culp, Nathan Coulter, and Ross Quinlan. 2018. C50: C5.0 Decision Trees and Rule-Based Models. R package version 0.1.2. Available online at https://cran.r-project.org/package=C50.

[36] Ludmila I. Kuncheva. 2014. *Combining Pattern Classifiers: Methods and Algorithms* (second ed.). John Wiley & Sons.

[37] Yann LeCun, Léon Bottou, Yoshua Bengio, and Patrick Haffner. 1998. Gradient-Based Learning Applied to Document Recognition. *Proc. IEEE* 86, 11 (Nov. 1998), 2278–2324.

[38] David D. Lewis, Yiming Yang, Tony G. Rose, and Fan Li. 2004. RCV1: A New Benchmark Collection for Text Categorization Research. *J. Machine Learning Research* 5 (April 2004), 361–397.

[39] M. Lichman. 2013. UCI Machine Learning Repository. http://archive.ics.uci.edu/ml.

[40] Wei-Yin Loh. 2014. Fifty Years of Classification and Regression Trees. *International Statistical Review* 82, 3 (Dec. 2014), 329–348 (with discussion, pp. 349–370).

[41] Wei-Yin Loh and Nunta Vanichsetakul. 1988. Tree-Structured Classification via Generalized Discriminant Analysis. *J. Amer. Stat. Assoc.* 83, 403 (Sept. 1988), 715–725.

[42] Bjoern H. Menze, B. Michael Kelm, Daniel N. Splitthoff, Ullrich Koethe, and Fred A. Hamprecht. 2000. On Oblique Random Forests. In *Proc. of the 11st European Conf. Machine Learning (ECML–00)*, Ramon López de Mántaras and Enric Plaza (Eds.). Barcelona, Spain, 453–469.

[43] Kevin P. Murphy. 2012. *Machine Learning: A Probabilistic Perspective.* MIT Press.

[44] S. K. Murthy, S. Kasif, and S. Salzberg. 1994. A System for Induction of Oblique Decision Trees. *J. Artificial Intelligence Research* 2 (1994), 1–32.

[45] Fabian Pedregosa, Gaël Varoquaux, Alexandre Gramfort, Vincent Michel, Bertrand Thirion, Olivier Grisel, Mathieu Blondel, Peter Prettenhofer, Ron Weiss, Vincent Dubourg, Jake Vanderplas, Alexandre Passos, David Cournapeau, Matthieu Brucher, Matthieu Perrot, and Édouard Duchesnay. 2011. Scikit-learn: Machine Learning in Python. *J. Machine Learning Research* 12 (Oct. 2011), 2825–2830. Available online at https://scikit-learn.org.

[46] J. Ross Quinlan. 1986. Induction of Decision Trees. *Machine Learning* 1, 1 (1986), 81–106.

[47] J. Ross Quinlan. 1993. *C4.5: Programs for Machine Learning.* Morgan Kaufmann.

[48] Shaoqing Ren, Xudong Cao, Yichen Wei, and Jian Sun. 2015. Global Refinement of Random Forest. In *Proc. of the 2015 IEEE Computer Society Conf. Computer Vision and Pattern Recognition (CVPR'15)*. Boston, MA, 723–730.

[49] Juan J. Rodríguez, Ludmila I. Kuncheva, and Carlos J. Alonso. 2006. Rotation Forest: A New Classifier Ensemble Method. *IEEE Trans. Pattern Analysis and Machine Intelligence* 28, 10 (Oct. 2006), 1619–1630.

[50] Lior Rokach and Oded Maimon. 2005. Top-Down Induction of Decision Trees Classifiers—A Survey. *IEEE Trans. Systems, Man, and Cybernetics Part C—Applications and Reviews* 35, 4 (Nov. 2005), 476–487.

[51] Lior Rokach and Oded Maimon. 2007. *Data Mining With Decision Trees. Theory and Applications.* Number 69 in Series in Machine Perception and Artificial Intelligence. World Scientific.

[52] Ananth Sankar and Richard J. Mammone. 1993. Growing and Pruning Neural Tree Networks. *j-ieee-tc* 42, 3 (March 1993), 291–299.

[53] Robert E. Schapire and Yoav Freund. 2012. *Boosting. Foundations and Algorithms.* MIT Press.

[54] Samuel Schulter, Paul Wohlhart, Christian Leistner, Amir Saffari, Peter M. Roth, and Horst Bischof. 2013. Alternating Decision Forests. In *Proc. of the 2013 IEEE Computer Society Conf. Computer Vision and Pattern Recognition (CVPR'13)*. Portland, OR, 508–515.

[55] Karen Simonyan and Andrew Zisserman. 2015. Very Deep Convolutional Networks for Large-Scale Image Recognition. In *Proc. of the 3rd Int. Conf. Learning Representations (ICLR 2015)*. San Diego, CA.

[56] Terry Therneau, Beth Atkinson, and Brian Ripley. 2019. rpart: Recursive Partitioning and Regression Trees. R package version 4.1-15. Available online at https://cran.r-project.org/package=rpart.

[57] Alfred Kar Yin Truong. 2009. *Fast Growing and Interpretable Oblique Trees via Logistic Regression Models.* Ph.D. Dissertation. University of Oxford.

[58] Paul E. Utgoff. 1989. Perceptron Trees: A Case Study in Hybrid Concept Representations. *Connection Science* 1, 4 (1989), 377–391.

[59] Paul Viola and Michael J. Jones. 2004. Robust Real-Time Face Detection. *Int. J. Computer Vision* 57, 2 (May 2004), 137–154.

[60] Radim Řehůřek and Petr Sojka. 2010. Software Framework for Topic Modelling with Large Corpora. In *Proc. LREC 2010 Workshop on New Challenges for NLP Frameworks.* Valletta, Malta, 45–50.

[61] Le Zhang, Jagannadan Varadarajan, Ponnuthurai Nagaratnam Suganthan, Narendra Ahuja, and Pierre Moulin. 2017. Robust Visual Tracking Using Oblique Random Forests. In *Proc. of the 2017 IEEE Computer Society Conf. Computer Vision and Pattern Recognition (CVPR'17)*. Honolulu, HI, 5825–5834.

[62] Ziming Zhang, Paul Sturgess, Sunando Sengupta, Nigel Crook, and Philip H. S. Torr. 2012. Efficient Discriminative Learning of Parametric Nearest Neighbor Classifiers. In *Proc. of the 2012 IEEE Computer Society Conf. Computer Vision and Pattern Recognition (CVPR'12)*. Providence, RI, 2232–2239.

[63] Arman Zharmagambetov and Miguel Á. Carreira-Perpiñán. 2020. Smaller, More Accurate Regression Forests Using Tree Alternating Optimization. In *Proc. of the 37th Int. Conf. Machine Learning (ICML 2020)*, Hal Daumé III and Aarti Singh (Eds.). Online, 9762–9772.

[64] Arman Zharmagambetov, Suryabhan Singh Hada, Miguel Á. Carreira-Perpiñán, and Magzhan Gabidolla. 2020. An Experimental Comparison of Old and New Decision Tree Algorithms. (March 20 2020). arXiv:1911.03054.

[65] Zhi-Hua Zhou. 2012. *Ensemble Methods: Foundations and Algorithms.* CRC Publishers.

8 SUPPLEMENTARY MATERIAL

8.1 Datasets

Table 1 summarizes the characteristics of the datasets used in our experiments, whose description is as follows (all classification tasks have dense features unless otherwise stated):

Letter English letter recognition task from UCI dataset [39]. Each letter image was generated by randomly distorting pixel images of the 26 uppercase letters from 20 different commercial fonts. The features are image features such as edge counts and statistical moments. We use the last 4 000 samples as test as in [54].

MNIST Handwritten digits recognition task [37]. The features are the pixel grayscale values in [0,1] of each 28×28 digit image. We use the training/test partition in [37].

Char74k Image classification task [18] where each image contains a character in one 64 classes (0–9, A–Z, a–z). We used the modified version of [54], as described in [62]: this uses images resized into a grayscale image of 8×8 pixels, and the features are the 64 pixel grayscale values. We split the dataset randomly into 7 400 images as test and the rest as training.

ImageNet subset ImageNet is a large-scale object recognition dataset [20]. We use a subset of 64 classes (arbitrarily selected, given in table 10) and use as features the output of 8 192 neurons from a pretrained deep net. Specifically, we fine-tune a batch normalized version of VGG16 on 224×224 RGB images. For each class, we randomly select 100/200 images as validation/test, respectively, and use the rest for training. The VGG16 architecture is the same as in [55], except we replace the last maxpool layer with a pooling of size 3×3 in order to obtain 8 192 dimensional features. We train our modified network using SGD for 30 epochs with fixed learning rate 0.001 and momentum rate 0.9. This network achieves 1.08% train error and 8.79% test error (both top-1).

R8 Top 8 (by frequency) classes of the Reuters-21578 text categorization dataset. We use the preprocessed and stemmed version[6]. We train the Doc2Vec model available from the `gensim` package [60] on all the document collections and obtain 400-dim word-embedding features.

RCV1 Text categorization dataset [38]. We obtained it from the LIBSVM multiclass data collection[7]. The features are sparse normalized log TFIDF vectors.

SensIT Moving vehicles type classification. Features: sensor network data. Also from the LIBSVM multiclass data collection.

SUSY Classification of particle detector collision events, available in the UCI dataset [39]. We randomly select 90% of the instances for training and the rest for test.

8.2 Generating random tree structures

In the main paper, we report results using a diversity mechanism consisting of having the initial tree for each tree in the TAO forest have a random structure. We describe this in detail here.

The motivation for this is the fact that the number of tree structures grows extremely fast with either the depth or the number of nodes in a tree. Hence, we can expect significant diversity in a TAO forest by using a random initial structure for each tree, while still keeping the bias low thanks to TAO's effective optimization. Next, we characterize the number of structures more precisely, and describe our sampling procedure.

8.3 Counting trees

Call n the number of nodes in the tree (including both leaves and internal, or decision, nodes). The number of tree structures having n nodes is known in several cases and can be found in [32], to which the section and equation references below correspond. Section 2.3.4.4 considers the following types of trees:

- *Ordered trees*, where the relative order of the subtrees of each node matters (e.g. whether exchanging the left and right child matters in a binary node). The number of binary trees with n nodes is given in eq. 14 as $b_n = \frac{1}{n+1}\binom{2n}{n}$ (this is also known as the nth Catalan number, C_n). Asymptotically (using the Stirling approximation) this is $4^n n^{-3/2}/\sqrt{\pi}$. It exceeds 10 thousand for $n = 10$ and 1 million for $n = 14$. More generally, the number of t-ary trees (where each decision node has t children) with n nodes is given in exercise 2.3.4.4.11 as $\frac{1}{(t-1)n+1}\binom{tn}{n}$.
- *Oriented (or rooted or unordered) trees*, where the relative order of the subtrees of each node does not matter. The exact formula is complicated (section 2.3.4.4 and exercises 2.3.4.4.1–2.3.4.4.2), but there is a simpler asymptotic expression (exercise 2.3.4.4.4, also [33] section 7.2.1.6 eq. (46)): $A_n = c\alpha^n n^{-3/2}$ where $\alpha \approx 2.9558$ and $c \approx 0.4399$. This exceeds 10 thousand for $n = 13$ and 1 million for $n = 18$.

In the context of decision trees, ordered vs oriented trees work as follows. Imagine a binary tree where we exchange the left and right child of a given node (and adjust the decision function so the tree predictive function remains unchanged). Then, as ordered trees, the resulting tree is considered different from the original; as oriented trees, both are considered the same tree. In parameter space both trees are indeed different because they correspond to different weight values and decision functions. But whether we count ordered or oriented trees, both grow much faster than 2^n with the number of nodes n, which generates an enormous source of diversity. With so many different structures, we can simply sample some of them to generate a forest with sufficiently many trees for any practical purpose.

8.3.1 *Sampling trees.*

We have to define a distribution over tree structures. We do this as follows. First, define a *split probability profile* $P_d \in (0, 1)$ for $d = 0, 1, 2 \dots$ which gives the probability of splitting a node that is at depth d. For example, we can take $P_d = p$ with $p \in [0, 1)$ (independent of the depth), $P_d = p^d$, etc. This induces a probability distribution over tree structures, from which we can then sample a tree as follows:

- Start from a root node (we do not consider empty trees) at depth $d = 0$.
- Split the node with probability P_d.

[6]https://www.cs.umb.edu/~smimarog/textmining/datasets
[7]http://csie.ntu.edu.tw/~cjlin/libsvmtools/datasets/multiclass.html

anemone fish	gasmask	packet	skunk
barn	goldfish, Carassius auratus	panpipe	soup bowl
Bernese mountain dog	Gordon setter	passenger car	space heater
bolete	hartebeest	pedestal	spotlight
borzoi	Ibizan hound	pill bottle	stretcher
bow	iron	platypus	sunglasses
briard	Italian greyhound	pot	swing
brown bear	jack-o-lantern	prairie chicken	television
carousel	jigsaw puzzle	purse	thatch
carpenters kit	junco	rain barrel	tiger cat
cash machine	knot	ram, tup	trolleybus
Chihuahua	kuvasz	red-backed sandpiper	waffle iron
chocolate sauce	lawn mower	rubber eraser	wall clock
cleaver	leopard	safety pin	washing machine
dalmatian	monastery	saxophone	white stork
flute	mud turtle	screwdriver	yawl

Table 10: The 64 object classes we used in our subset of the ImageNet dataset.

- If we do split the node, repeat the procedure recursively for each child at depth $d + 1$.

It is of interest to pick the profile $\{P_d\}_{d \geq 0}$ so that the expected number of nodes of the previous distribution equals a desired value. Hence, let us now compute the *expected number of nodes* $\overline{n}$ (decision and leaf nodes) in the tree, for binary trees, where each split produces two children (for t-ary splits, replace 2 with t in the equations below). Call $\overline{n}_d$ the expected number of nodes in a subtree of a node at depth $d \geq 0$ (counting the node itself), so that $\overline{n}_0 = \overline{n}$ is the expected number of nodes in the whole tree. Then, at a node at depth d we have the node itself, which we either split into two children (with probability P_d), or do not split (with probability $1 - P_d$). By symmetry, the expected number of nodes of each child is the same and equal to $\overline{n}_{d+1}$. Hence we have that $\overline{n}_d$ must satisfy the following equation:

$$\overline{n}_d = 1 + 0(1 - P_d) + (\overline{n}_{d+1} + \overline{n}_{d+1})P_d = 1 + 2P_d\overline{n}_{d+1},$$

for $d = 0, 1, 2 \ldots$ Applying this $M + 1$ times results in the expression

$$\overline{n} = \overline{n}_0 = 1 + 2P_0\overline{n}_1 = 1 + 2P_0(1 + 2P_1\overline{n}_2) = \cdots = 1 + \sum_{i=1}^{M} \prod_{d=0}^{i-1} (2P_d) + \overline{n}_{M+1} \prod_{d=0}^{M} (2P_d).$$

Assume the rightmost term tends to 0 as $M \longrightarrow \infty$. Then the final expression is

$$\overline{n} = 1 + \sum_{i=1}^{\infty} \prod_{d=0}^{i-1} (2P_d). \tag{2}$$

This series converges under some conditions:

- A necessary condition is that $\prod_{d=0}^{i-1} (2P_d) \longrightarrow 0$ as $i \longrightarrow \infty$. This happens if there exist $d_0 \geq 0$ and $Q \in (0, \frac{1}{2})$ satisfying $P_d \leq Q \; \forall d \geq d_0$.
- A sufficient condition is that the limit of the ratio of consecutive terms, if it exists, be smaller than 1. Since $\prod_{d=0}^{i} (2P_d) / \prod_{d=0}^{i-1} (2P_d) = 2P_i$, this implies $\lim_{i \to \infty} P_i < \frac{1}{2}$.

We can use those conditions to construct split probability profiles and generate an enormous variety of distributions. Consider the following profiles as simple examples:

(1) $P_d = p \in (0, \frac{1}{2})$: $\overline{n} = 1/(1 - 2p)$ (by summing the geometric series in eq. (2)).
(2) $P_d = p^d$ with $p \in (0, 1)$: $\overline{n} = \sum_{i=0}^{\infty} 2^i p^{i(i-1)/2}$.
(3) $P_d = p^{d+1}$ with $p \in (0, 1)$: $\overline{n} = \sum_{i=0}^{\infty} 2^i p^{i(i+1)/2}$.

Profile 1 is not very useful because it generates very unbalanced trees with a large depth variation. Profiles 2 and 3 decay quickly with the depth so they generate less unbalanced trees, with a more constrained depth variation. Profile 2 differs from profile 3 in that profile 2 always splits the root. This is convenient since a single-node tree is a constant or linear classifier (depending on the leaf predictor type), which is useful to have in an ensemble but only once (they are very redundant).

A more flexible and convenient profile has the form:

$$P_d = \begin{cases} 1, & 0 \leq d \leq \Delta_1 \\ p, & \Delta_1 < d \leq \Delta_2 \\ 0, & \Delta_2 < d. \end{cases} \tag{3}$$

This generates trees with depth between Δ_1 and Δ_2, having all levels complete up to Δ_1. This eliminates small trees, which are not useful in forests, since they have high bias. If p is large enough but not too large, then the trees will have all depth Δ_2 with high probability, but will not be complete and will differ widely in structure and number of nodes. These are just some possibilities. In practice, we can pick the profile manually depending on the type of nodes in the tree (axis-aligned or oblique), number of classes K, etc.

Given a profile $\{P_d\}_{d \geq 0}$ from eq. (3), we can invert eq. (2) (analytically or using the bisection method) to obtain values of (Δ_1, Δ_2, p) that achieve a desired expected number of nodes $\overline{n}$.