Non-Greedy Algorithms for Decision Tree Optimization: An Experimental Comparison

Arman Zharmagambetov[§], Suryabhan Singh Hada[§], Magzhan Gabidolla, Miguel Á. Carreira-Perpiñán
Dept. of Computer Science and Engineering, University of California, Merced, CA, USA
Email: {azharmagambetov,shada,mgabidolla,mcarreira-perpinan}@ucmerced.edu

***Abstract*—Learning decision trees is a difficult optimization problem: nonconvex, nondifferentiable and over a huge number of tree structures. The dominant paradigm in practice, established in the 1980s, are axis-aligned trees trained with a greedy recursive partitioning algorithm such as CART or C5.0. Several non-greedy optimization algorithms have been proposed recently, which optimize all the nodes' parameters jointly, and we compare experimentally some of them in a range of classification and regression datasets, in terms of accuracy, training time and tree size. The non-greedy algorithms do not improve over CART significantly with one exception, *tree alternating optimization (TAO)*. TAO scales to large datasets and produces axis-aligned and especially oblique trees that consistently outperform all other algorithms, often by a large margin. TAO makes oblique trees preferable to axis-aligned ones in many cases, since they are much more accurate while remaining small and interpretable. This suggests a change in paradigm in practical applications of decision trees.**

I. Introduction

Decision trees occupy a special place in machine learning and statistics. They are among the most interpretable models (and can be understood as rules), they are very fast for inference (since a single root-leaf path is followed), and they handle nonlinear and multiclass problems naturally. Unfortunately, they are also among the hardest models to learn: they define a nonconvex, nondifferentiable objective function over a huge space of tree structures; even the simplest case (axes-aligned with binary inputs and output) is NP-hard [1]. While many algorithms have been proposed, learning a tree from data in practice has long been dominated by greedy recursive partitioning algorithms (such as CART [2] or C5.0 [3]) applied to axes-aligned (univariate) trees, and this is also true of tree ensembles (forests), in spite of their known suboptimality, as noted by multiple reviews [2], [4]–[6] and experimental comparisons of trees and other algorithms [7], [8].

While CART and C5.0 date from the 1980s, many more algorithms for learning trees have been proposed since. While many of these are variations of CART having dubious merits, in the last few years several non-greedy algorithms have been proposed that try to optimize an objective function over all the tree parameters. Do they make a difference in practice? We explore this here on 8 representative tree learning algorithms on 20 classification and 7 regression datasets of different type, dimensionality and sample size, and report the accuracy, size and training time of the resulting trees. We conclude that, in spite of the sometimes exaggerated claims of some papers, these new algorithms do not significantly improve over CART with one exception: tree alternating optimization (TAO).

Next, we briefly describe the algorithms (section II), summarize our experimental results (sections III–IV, and discuss them in light of how each algorithm works (section V). A preliminary version of this paper appeared in [9].

[§]Equal contribution

II. The algorithms

Due to the large and increasing amount of literature on decision trees, it is impossible to cover all algorithms and perform exhaustive evaluations between them. Therefore, we focus on 1) popular, long-established methods that have stood the test of time, and 2) recent representative methods. We consider algorithms that train a single decision tree having constant-label leaves and decision nodes that are either axis-aligned (univariate) or oblique (linear, or multivariate). Below we briefly describe the algorithms we evaluated. More details can be found in the cited papers.

A. Greedy recursive partitioning algorithms

These are considered as the most popular approaches to learn a tree; their roots are in the 1950s but became popular in the early 80s. The basic idea in such algorithms is to induce a decision tree from the top down according to some heuristic splitting rule, and optionally prune it back afterwards. The algorithm is greedy since it never looks back once a node is split. When pruning the fully-grown tree, some nodes are removed to optimize a cost-complexity criterion, but the node parameters remain unchanged. Greedy algorithms are generally very fast but they produce suboptimal trees [6].

- **CART** [2] is one of the most widely used algorithms for training axis-aligned decision trees. It applies a greedy recursive partitioning which optimizes a purity measure (Gini index) at each node. When splitting a given node, it enumerates all the features and thresholds to find the split that maximally reduces the Gini index. It continues to grow a tree up to a maximum depth and then prunes nodes one by one until a cost-complexity criterion is met.
- **C5.0** [3] is also widely used for axis-aligned trees. It has undergone multiple versions (ID3, C4.5, C5.0) and differs from CART in relatively minor details, such as the purity measure (information gain).
- **Oblique Classifier 1 (OC1)** [10] considers a linear combination of the input features (oblique trees). CART

[2] also did consider such trees and used coordinate descent to optimize the purity measure at each split, although this did not work robustly. OC1 adds some improvements to this, such as random restarts.

- **GUIDE** (Generalized, Unbiased, Interaction Detection and Estimation) is the most sophisticated of a series of algorithms developed by Wei-Yin Loh [5], [11]. Like CART, GUIDE grows a tree greedily and recursively, but it adds many heuristics, in particular in how features are selected (involving various statistical tests aimed at eliminating variable selection bias) and the choice of the split point. GUIDE can also learn oblique trees.

B. Non-greedy, global optimization algorithms

In contrast to greedy approaches, this type of algorithms focuses on a joint optimization of the decision tree nodes under a global objective function. One approach is to ignore (during training) the discrete nature of the decision tree and make it probabilistic (or soft), where an input is routed to each leaf with a certain probability. Then the optimization becomes amenable to gradient-based methods. The resulting tree is then made hard again. We do not consider this since it is quite suboptimal. Instead, we consider two methods that directly optimize the decision tree with discrete decision nodes.

- **CO2** [12] formulates a convex-concave upper bound on the tree's empirical loss and optimize it using stochastic gradient descent. The initial tree structure must be provided (e.g. from CART). The use of SGD enables efficient optimization for large scale datasets, however each iteration is not guaranteed to decrease the objective.
- **TAO** (Tree Alternating Optimization) [13], [14] optimizes a decision tree with a predetermined structure. It applies very generally: it handles different classification or regression loss functions, both axis-aligned and oblique decision trees (and other types of trees), and various regularization terms such as ℓ_1.

We describe TAO in more detail. Suppose we have an initial tree with some structure and parameter values (possibly random). TAO minimizes the following objective function jointly over the parameters $\mathbf{\Theta} = \{\boldsymbol{\theta}_i\}$ of all nodes i of the tree:

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

where $\{(\mathbf{x}_n, y_n)\}_{n=1}^N$ is a training set of D-dimensional real-valued instances and their ground truth labels, $L(\cdot,\cdot)$ is the loss function (e.g. cross-entropy, mean squared error, 0/1 loss, etc.), $T(\mathbf{x}; \mathbf{\Theta})$ is the predictive function of the tree, and ϕ_i are regularization terms with hyperparameter $\lambda \geq 0$. The basis of the TAO algorithm is given by a *separability condition*: for any nodes i and j (decision or leaves) that are not descendants of each other (e.g. all nodes at the same depth) the error $E(\mathbf{\Theta})$ in eq. (1) separates over $\boldsymbol{\theta}_i$ and $\boldsymbol{\theta}_j$ (assuming parameters of all other nodes are fixed). Therefore, all such nodes can be trained independently. The optimization over a single node is made possible by equivalence to a *reduced problem*:

- Optimizing over a *decision node* i is done by training a weighted binary classifier (with regularization $\lambda\, \phi_i(\boldsymbol{\theta}_i)$) over $\boldsymbol{\theta}_i$ on the training instances $\{(\mathbf{x}_n, \overline{y}_n)\}$ that currently reach i (the reduced set). Each such instance $\mathbf{x}_n$ is assigned a weight and pseudolabel $\overline{y}_n \in \{-1, +1\}$ based on the child whose subtree gives the better prediction for $\mathbf{x}_n$. A weight per sample is assigned because the loss of the best child is different for each instance. This step can be solved exactly for axis-aligned trees by enumeration over all (feature, threshold) combinations, or approximately for oblique trees via a surrogate loss.
- Optimizing over a *leaf* reduces to fitting the leaf's model to solve the original problem (1) on the reduced set. For example, for constant-label leaves in classification, the solution is the majority vote.

One pass over all nodes (e.g. in breadth-first order) defines a TAO iteration, which is guaranteed to decrease or leave unchanged the objective function (1). The computational complexity per iteration is comparable to that of one CART run (for axis-aligned trees) or to training Δ linear SVMs (for complete oblique trees of depth Δ).

TAO optimizes jointly all node parameters rather than greedily. It works in a way that is similar to how most other machine learning models are trained: one first fixes the model architecture (the tree structure, e.g. a complete binary tree of given depth) and iteratively optimizes its parameters by monotonically decreasing $E(\mathbf{\Theta})$. With most models this is done via gradient-based methods, but this is not possible with decision trees, which are nondifferentiable. Instead, TAO applies alternating optimization (over the parameters at each node). Besides, with an ℓ_1 penalty over the nodes' parameters, nodes are automatically pruned when all its weights become 0. Hence, it is convenient to use a deep enough complete tree structure and let TAO prune it as necessary.

C. Approximate brute force search via branch-and-bound

These approaches (e.g. [15]–[18]) try to optimize an objective such as (1) by brute force to find the globally optimum tree (for special types of trees), for example using branch-and-bound to eliminate parts of the search space (using running bounds on the optimal objective function value). This still has a worst-case exponential complexity. Since running it to completion is intractable, the branch-and-bound search is stopped early with an approximate solution.

- **OCT** (Optimal Classification Trees) [17] formulates the optimization as a mixed-integer optimization (MIO) problem by using binary variables to encode it. This is then solved via commercial MIO solvers. It handles axis-aligned and oblique trees.
- **GOSDT** (Generalized and Scalable Optimal Sparse Decision Trees) [18] uses a custom branch-and-bound algorithm instead of MIO solvers, including some accelerations. It is restricted to axis-aligned trees having binary inputs and outputs only (i.e., boolean functions). This means any continuous features need to be discretized in some way.

III. Experimental setup

A. Algorithms

- **CART(R):** we use a popular R implementation of CART, `rpart` [19]. We let the tree grow up to a depth of 30 (the maximum `rpart` allows), by setting the "minsplit" parameter to 1 and the complexity parameter "cp" to 0. We then prune the tree by using `rpart`'s internal k-fold cross-validation ($k = 10$) to get the list of pruning parameters and choosing the best pruning parameter based on the SE-1 rule (as suggested by the `rpart` documentation).
- **CART(Py):** we also use the CART Python implementation in scikit-learn [20, version 0.22.2]. We let the tree grow fully by setting the "minsplit" parameter to 1 and the complexity parameter "ccp_alpha" to 0. Next, we find the best pruning parameter using k-fold cross-validation.
- **C5.0:** we use the single-threaded Linux version of the authors' C5.0 implementation (in C)[1]. For each of the datasets, we apply a grid search on the k-fold validation set to get the best parameters. Specifically, we tune "-c CF", which controls the pruning severity, and "-m cases", which is the minimum number of points to perform a node split. We use the default options for all other parameters. We have found that in many cases the tuned parameters are not far away from the default settings.
- **GUIDE:** we use GUIDE version 32.3 provided by the author in the form of executables[2]. For axis-aligned classification, we specify option (1), which gives univariate splits a higher priority. For oblique trees, we choose option (0), which gives linear splits a higher priority. GUIDE for regression uses only axis-aligned splits. We do not use the kernel and nearest neighbor node models for classification and variants of linear regression used in regression trees, because they are beyond the scope of the conventional trees considered in this paper. We cross-validated the following hyperparameters: estimated or equal priors for a class distribution, mean or median based cross validation, standard error for pruning, maximum number of splits, and minimum node size. The latter two had the largest effect on the tree structure and accuracy, particularly for large datasets.
- **OC1:** we use the authors' C implementation[3]. Since it only supports classification, we do not report results for regression with OC1. Its default purity measure is the twoing criterion, and we did not try others (using other purity measures requires changing and recompiling the code). We choose the option where only oblique hyperplanes are considered. During cross-validation, we tune the number of random restarts and the number of random jumps, since we found these hyperparameters to have the most important effect (although, in some datasets, we were not able to improve over their default values).
- **GOSDT:** we use the C++ implementation and the Python interface provided by the authors[4]. Although it supports several different losses, we use only the misclassification error. GOSDT has many different configuration options, but because of its long training time, we could not explore those during cross-validation. Therefore, for all the experiments we use only the default hyperparameters. One important parameter is the regularization term λ, and its default value in the configuration documentation was set to $\lambda = 0.05$.
- **TAO:** we implement both axis-aligned and oblique trees, with constant leaves, in Python (version 3.5), without parallel processing in a single CPU.
  - Oblique: we take as initial tree a deep enough, complete binary tree with random parameters at each node. We use an ℓ_1-regularized logistic regression to solve the decision node optimization (implemented in LIBLINEAR [21]). We use a grid search on the k-fold validation set to tune the λ parameter, which controls the sparsity of the tree ($C = 1/\lambda$ in LIBLINEAR).
  - Axis-aligned: we use as initial tree structure a CART tree. We use the scikit-learn implementation of the CART algorithm and perform hyperparameter tuning on a cross-validation set as for other methods.

  We ran TAO until we reach 30 iterations or there is no more training error decrease up to a threshold of 10^{-5}.
- **OCT** and **CO2**: we report the results from the original papers ([17] and [12], respectively), since we could not find any implementation available online.

We ran all experiments in a single Linux PC with the following specifications: OS, Ubuntu 18.04 LTS; CPU, 8 $\times$ Intel Core i7-7700 3.60GHz; memory, 16 GiB DDR4 3600 MHz.

B. Datasets

Table I summarizes the datasets we used. All are publicly available. "MNIST-LeNet5" consists of 800 non-negative real-valued features extracted by the "conv2" layer of the pre-trained LeNet5 neural network [23] for all MNIST images. For some of the UCI collection datasets which do not have a separate test set, we shuffle the entire dataset and keep 20% of the entire data as the test set. We repeat the training procedure 10 times for each dataset, reshuffling the training data each time. We set a timeout of 2 hours' runtime for all algorithms on all datasets except for MNIST (both pixels and LeNet5), for which we allow up to 8 hours to train a tree. Note that we include datasets that are quite larger (in sample size and/or dimensionality) than many existing works on decision trees.

IV. Results

A. Classification

Tables II-III show the train/test average accuracy (in %) and stdev over 10 repeats. Although we report the training error

[1] https://rulequest.com/download.html
[2] http://pages.stat.wisc.edu/~loh/guide.html
[3] http://ccb.jhu.edu/software/oc1/oc1.tar.gz
[4] https://github.com/Jimmy-Lin/GeneralizedOptimalSparseDecisionTrees

TABLE I

DATASETS USED IN OUR EXPERIMENTS (ALL FROM THE UCI COLLECTION [22] UNLESS INDICATED BY A FOOTNOTE). N IS SAMPLE SIZE, D IS FEATURE DIMENSIONALITY AND K IS OUTPUT DIMENSION FOR REGRESSION AND NUMBER OF CLASSES FOR CLASSIFICATION.

	Dataset	N_{train}	N_{test}	D	K
classification	Iris	120	30	4	3
	Wine	142	36	13	3
	Dermatology	293	73	34	6
	Balance scale	500	125	4	3
	Breast Cancer	559	140	9	2
	Blood Trans	598	150	4	2
	German	800	200	20	2
	Banknote auth	1098	274	4	2
	Contraceptive	1178	295	9	3
	Car Eval	1382	346	6	4
	Segment	1848	462	19	7
	Spambase	3681	920	57	2
	Optical recog	3823	1797	64	10
	Landsat	4435	2000	36	6
	Pendigits	7494	3498	16	10
	Letter[a]	16000	4000	16	26
	Connect4	54046	13511	126	3
	MNIST-pixels[a]	60000	10000	784	10
	MNIST-LeNet5[b]	60000	10000	800	10
	SensIT[a]	78823	19705	100	3
regression	concrete	687	343	8	1
	airfoil	1002	501	5	1
	abalone	2506	1671	8	1
	cpuact[c]	4915	3277	21	1
	ailerons[d]	7154	6596	40	1
	CT slice	42800	10700	384	1
	YearPredictionMSD	463715	51630	90	1

[a]https://www.csie.ntu.edu.tw/~cjlin/libsvmtools/datasets/multiclass.html
[b]https://faculty.ucmerced.edu/mcarreira-perpinan/teaching/CSE176/Labs/datasets/
[c]http://www.cs.toronto.edu/~delve/data/comp-activ/desc.html
[d]https://www.dcc.fc.up.pt/~ltorgo/Regression/DataSets.html

for reference, *the important error to consider is the test error*. This is because the training error can be trivially made zero with a tree by growing it deep enough that each leaf contains instances of the same class, at the cost of overfitting.

The most important conclusions are as follows. First, both for axis-aligned and oblique trees, TAO nearly always produces the most accurate trees. In the very few datasets where TAO is not the winner, the difference is very small, and it is due to a handful of instances, since these are small datasets in sample size (and dimension). Note that, in such cases, we could always further improve TAO by using the best-performing tree as initial tree. This is unlike traditional top-down induction algorithms such as CART or C5.0, which build the tree from scratch. However, we did not use this advantage, and in all our experiments we initialize TAO oblique trees randomly.

Second, TAO's improvement over the other methods grows as the dataset complexity grows. Note for example the results for "MNIST-pixels" and "MNIST-LeNet5", having 60k training points and around 800 features.

Finally, oblique trees with TAO are far more accurate than axis-aligned trees, particularly when the features have a higher dimension. This, of course, is to be expected, since they are more powerful models, but it does not generally hold with previous algorithms such as OC1 or GUIDE (which are also slower to train). This explains the historically lack of widespread use of oblique trees.

For axis-aligned trees, CART (both versions) and C5.0—the most popular algorithms—perform similarly in terms of test accuracy, with a little favor to C5.0. GUIDE also performs similarly. These algorithms are based on greedy recursive partitioning and differ in minor details, such as the choice of purity criterion. The brute-force branch-and-bound algorithms show a disappointing performance in spite of their purported optimality because they simply do not scale beyond tiny problem and tree sizes. OCT and GOSDT do well only on very small datasets like Iris or Wine, for which a very shallow tree is sufficient. For larger datasets, deeper trees are necessary to achieve good accuracy, but then the branch-and-bound search has to be stopped early, typically producing a tree that is far from optimal, often even worse than with CART or C5.0. The GOSDT code does not actually support early stopping, so for the vast majority of datasets it was not able to finish within our timeout limit (set generously to 2 hours). Similar comments apply to the oblique trees.

Decision trees are considered as interpretable models, but this need not be true if the tree is very deep or has many nodes. Moreover, a larger tree has larger inference time and needs more space. Hence, it is also important to compare the size of the trained trees. Table IV shows the average maximum depth and average number of leaves for both axis-aligned and oblique trees. As one would expect, larger datasets require larger trees (in depth and number of nodes). Oblique trees generate much more compact trees than axis-aligned trees, but with a more complex node structure (hyperplane vs single feature). OCT generated very small trees because [17] limited the maximum depth due to the long computation time. GOSDT also generated very small trees, using its default regularization parameter (but, as noted, had a very long training time). C5.0 usually generated larger trees compared to CART. In general, the decision trees obtained by TAO have comparable or smaller size than those of the other methods. In some datasets (like Letter), the number of leaves and the maximum depth are quite large for the TAO axis-aligned tree, presumably due to the large number of classes; a more compact tree can be obtained by using an oblique tree. Note that in TAO the learned tree must be a subset of the initial tree; because of the ℓ_1 regularization (with both oblique and axis-aligned trees), a decision node may end up using no features, which leads to pruning it. Hence the final tree structure can be quite smaller than the initial one.

B. Regression

Table V shows the root mean squared error (RMSE) $E = \sqrt{\frac{1}{NK}\sum_{n=1}^{N}\|\mathbf{y}_n - \hat{\mathbf{y}}_n\|^2}$, where N is the sample size, K is the output dimension, and $\mathbf{y}$ and $\hat{\mathbf{y}}$ are the ground truth and predicted vectors, respectively. Note that GUIDE for regression uses only axis-aligned splits. Again, the winner by far are TAO oblique trees, followed at a distance by TAO axis-aligned trees, and then the other methods. Moreover, TAO

TABLE II

CLASSIFICATION DATASETS WITH AXIS-ALIGNED TREES. WE REPORT TRAIN AND TEST ACCURACY (%, AVERAGE ± STDEV OVER 10 REPEATS) AND DATASET SPECS (N_{TRAIN},D,K). WITHIN EACH ROW WE COLOR AS GREEN AND BLUE THE BEST AND SECOND BEST METHOD ON THE TEST SET. "–" MEANS THE RESULT WAS NOT REPORTED IN THE ORIGINAL PAPER; "TIMEOUT" MEANS IT EXCEED THE TRAINING TIMEOUT.

Dataset	(N_{train}, D, K)		TAO	CART(R)	CART(Py)	C5.0	GUIDE	GOSDT	OCT
Iris	(120, 4, 3)	train	97.03±0.65	97.08±1.19	97.02±0.99	99.03± 1.25	97.00±1.35	96.75±0.87	—
		test	95.41±3.81	93.33±3.15	94.75±3.20	92.64± 8.00	93.67±4.58	90.00±4.22	93.5
Wine	(142, 13, 3)	train	96.01±1.49	96.97±2.63	95.86±1.22	96.68± 1.58	96.76±2.28	timeout	—
		test	91.21±3.64	90.00±3.97	90.10±2.21	85.19±15.95	91.94±5.04		94.2
Dermatology	(293, 34, 6)	train	98.61±0.90	97.19±1.02	98.54±0.88	97.14± 0.72	97.33±0.68	timeout	—
		test	96.14±2.72	93.51±1.66	94.44±3.41	90.44± 4.78	95.54±3.08		89.2
Balance scale	(500, 4, 3)	train	85.95±1.28	85.94±0.42	82.66±0.92	88.38± 1.43	86.42±2.81	72.48±2.45	—
		test	82.21±3.36	78.96±0.34	79.62±2.09	78.19± 1.43	77.44±3.26	65.52±4.24	71.6
Breast Cancer	(559, 9, 2)	train	95.39±2.44	96.10±0.01	95.26±1.22	97.36± 0.61	96.62±0.79	93.19±0.42	—
		test	95.91±1.54	94.57±0.02	93.64±2.80	94.83± 0.90	94.57±1.75	91.17±1.28	91.5
Blood Trans	(598, 4, 2)	train	79.03±1.49	76.45±0.01	79.42±2.31	79.69± 0.59	80.90±1.97	76.34±0.71	—
		test	78.95±3.47	75.20±0.02	76.45±2.77	78.40± 2.45	78.80±3.26	75.67±2.82	77.0
German	(800, 20, 2)	train	78.34±1.72	75.96±2.41	78.88±2.04	84.13± 2.73	69.48±1.19	timeout	—
		test	73.35±4.14	72.21±3.35	72.29±3.23	69.13±10.96	69.95±2.59		69.8
Banknote auth	(1098, 4, 2)	train	99.53±0.14	99.45±0.02	99.30±0.24	99.63± 0.12	99.38±0.08	timeout	—
		test	98.16±1.02	97.93±0.06	96.33±2.19	98.70± 0.68	98.25±1.69		90.7
Contraceptive	(1178, 9, 3)	train	54.35±1.33	57.53±1.05	53.88±1.27	76.15± 1.41	58.79±1.89	timeout	—
		test	56.95±3.51	54.37±1.83	54.25±4.43	49.39± 4.79	55.19±2.13		53.3
Car Eval	(1382, 6, 4)	train	99.94±0.12	98.36±1.41	99.91±0.12	92.71± 3.85	69.97±0.48	70.36±0.46	—
		test	96.67±2.67	96.35±1.90	96.51±2.69	87.59± 4.19	70.23±1.93	68.67±1.84	78.8
Segment	(1848, 19, 7)	train	98.39±0.54	98.87±0.69	98.34±0.58	98.42± 0.88	98.31±0.44	timeout	—
		test	96.18±1.41	95.91±0.11	92.17±3.85	94.86± 1.69	95.37±1.11		—
Spambase	(3681, 57, 2)	train	94.13±1.30	94.96±0.01	93.18±2.71	96.18± 0.38	95.36±0.85	timeout	—
		test	93.19±1.05	91.92±0.01	89.62±3.28	92.85± 0.84	92.77±0.64		86.1
Optical recog	(3823, 64,10)	train	98.48±0.06	95.33±2.18	97.85±0.12	96.76± 0.84	94.64±1.10	timeout	—
		test	86.08±0.22	84.26±1.03	85.19±0.11	80.33± 5.32	84.56±1.32		54.7
Landsat	(4435, 36, 6)	train	96.97±0.26	91.13±0.49	97.37±0.12	95.19± 0.71	90.05±0.86	timeout	—
		test	86.10±0.25	85.94±0.27	85.21±0.23	84.12± 1.09	85.24±0.58		78.0
Pendigits	(7494, 16,10)	train	98.87±0.26	99.09±0.29	98.79±0.13	98.98± 0.09	97.90±0.50	timeout	—
		test	92.52±0.24	91.62±0.55	90.19±0.38	91.32± 0.71	89.93±0.77		—
Letter	(16000, 16,26)	train	96.38±0.03	94.30±0.01	95.93±0.11	97.97± 0.14	93.54±0.59	timeout	—
		test	86.39±0.12	86.04±0.04	86.07±0.14	85.26± 0.33	83.93±0.48		—
Connect4	(54046,126, 3)	train	86.99±1.74	82.94±1.08	83.63±0.13	82.60± 0.93	71.76±0.24	timeout	—
		test	79.88±1.53	78.29±0.21	77.55±1.18	77.84± 1.41	71.66±0.32		—
MNIST-pixels	(60000,784,10)	train	93.53±0.24	92.54±0.03	93.12±0.15	94.52± 0.23	75.76±0.40	timeout	—
		test	88.52±0.19	88.03±0.07	88.05±0.02	88.31± 0.35	78.52±0.20		—
MNIST-LeNet5	(60000,800,10)	train	96.94±0.05	95.71±0.04	96.64±0.08	97.89± 0.14	84.15±0.28	timeout	—
		test	93.52±0.03	93.31±0.05	93.32±0.07	93.48± 0.21	85.25±0.28		—
SensIT	(78823,100, 3)	train	83.56±0.12	84.38±0.01	82.82±0.23	86.66± 0.11	79.05±0.23	timeout	—
		test	81.84±1.11	81.71±0.01	81.00±0.19	81.41± 0.04	78.52±0.20		—
wins		train	4 (of 20)	3	2	10	1	0	0
wins		**test**	18 (of 20)	0	0	1	0	0	1

oblique trees are small (see Table VI), which makes them easier to interpret.

C. Runtime

It is commonly accepted that tree induction algorithms are fast, especially those which use greedy recursive partitioning, such as CART and C5.0. We did not apply any parallel processing in our experiments and we observe the following[5]:

- For small UCI datasets (Balance Scale, Breast Cancer, etc.), all axis-aligned and oblique trees that we ran are quite fast (around 0.5–4.0 sec.).
- For larger datasets (MNIST, SensIT, etc.), we observe some fluctuations. In general, axis-aligned trees are still fast to train: for instance, CART and C5.0 took about 200–400 sec. to train on MNIST-pixels, whereas GUIDE and TAO axis-aligned took a little longer than that (about 1300–1500 sec.). The exception is GOSDT since, as we described earlier, it ran out of time in most datasets. Runtime for OCT was not provided in [17]. As for oblique trees, we provide the following runtimes as reference: TAO took about 1200-1400 sec., OC1 about 1800 sec. and GUIDE took extremely long (about 26000 sec.).

[5]Some of these algorithms do benefit significantly from parallelism, in particular TAO. For example, running TAO on a depth-8 oblique tree for 20 iterations on MNIST-pixels takes about 1 minute using our C implementation (in the same 8-core PC on which we ran all our experiments).

V. DISCUSSION

A. TAO

Our experiments show that trees can indeed be accurate models if one optimizes their parameters jointly rather than greedily. *What makes TAO work so well?* Alternating optimization (iteratively optimizing over a group of parameters

TABLE III
SIMILAR TO TABLE II BUT FOR DECISION TREES WITH OBLIQUE SPLITS.

Dataset	$(N_{\text{train}},\ D,\ K)$		TAO	OC1	GUIDE	OCT	CO2
Iris	(120, 4, 3)	train	96.89±9.05	85.42±15.84	98.42±0.58	—	—
		test	94.40±5.12	85.67±14.53	94.33±3.00	95.1	—
Wine	(142, 13, 3)	train	98.22±5.33	89.30±10.25	97.75±1.50	—	—
		test	92.00±9.38	84.45± 8.89	93.33±5.00	91.6	—
Dermatology	(293, 34, 6)	train	100.00±0.00	91.58± 6.10	98.60±0.50	—	—
		test	97.97±1.25	84.46± 7.79	97.84±1.73	92.6	—
Balance scale	(500, 4, 3)	train	99.82±0.47	93.22± 2.17	91.72±3.94	—	—
		test	94.72±1.89	88.96± 2.29	85.60±5.68	87.6	—
Breast Cancer	(559, 9, 2)	train	98.21±0.79	82.99±12.16	96.82±0.51	—	—
		test	97.71±1.04	81.07±12.75	95.64±1.61	94.0	—
Blood Trans	(598, 4, 2)	train	81.54±0.59	80.33± 2.69	81.02±0.63	—	—
		test	80.42±3.13	77.93± 3.72	79.87±3.33	77.4	—
German	(800, 20, 2)	train	82.90±0.71	78.44± 5.85	70.18±1.07	—	—
		test	81.24±0.87	68.65± 4.17	70.15±2.17	71.0	—
Banknote auth	(1098, 4, 2)	train	99.83±0.33	94.12±12.91	99.63±0.27	—	—
		test	99.18±0.14	91.64±13.57	98.80±0.59	98.7	—
Contraceptive	(1178, 9, 3)	train	66.04±7.24	61.44± 5.71	57.58±0.93	—	—
		test	57.47±3.18	49.66± 3.06	56.58±2.58	53.3	—
Car Eval	(1382, 6, 4)	train	99.98±0.05	97.35± 2.92	69.97±0.48	—	—
		test	98.03±1.02	95.49± 2.32	70.23±1.93	87.5	—
Segment	(1848, 19, 7)	train	99.48±0.21	91.61± 8.84	98.41±0.41	—	97
		test	96.48±1.31	88.53± 7.47	95.48±1.02	—	96
Spambase	(3681, 57, 2)	train	95.55±0.47	80.72±16.51	95.74±0.99	—	—
		test	93.31±1.22	78.20±15.48	92.24±0.59	86.6	—
Optical recog	(3823, 64,10)	train	97.68±0.59	72.50±19.62	94.54±1.36	—	—
		test	91.27±1.74	62.00±17.60	79.19±1.20	54.3	—
Landsat	(4435, 36, 6)	train	94.45±0.49	80.25± 2.20	91.54±1.29	—	—
		test	87.81±0.88	73.54± 2.00	85.97±0.80	78.2	—
Pendigits	(7494, 16,10)	train	99.81±0.13	91.72± 7.81	98.85±0.14	—	96
		test	96.80±0.70	84.42± 7.02	91.80±0.69	—	92
Letter	(16000, 16,26)	train	95.43±0.29	75.85± 3.80	90.80±1.05	—	94
		test	90.41±0.31	65.81± 4.83	82.65±0.90	—	87
Connect4	(54046,126, 3)	train	82.40±0.53	79.02± 1.45	72.11±0.31	—	81
		test	81.09±0.39	75.42± 0.64	72.01±0.36	—	78
MNIST-pixels	(60000,784,10)	train	98.43±0.07	78.62± 9.62	73.02±0.79	—	94
		test	94.74±0.11	74.34± 9.94	73.79±0.91	—	90
MNIST-LeNet5	(60000,800,10)	train	99.98±0.01	89.52±14.76	84.15±0.28	—	—
		test	98.22±0.18	87.97±14.24	85.25±0.28	—	—
SensIT	(78823,100, 3)	train	85.97±0.13	76.10±12.69	79.64±0.28	—	83
		test	85.44±0.27	73.70±11.31	79.25±0.33	—	82
wins		train	18 (of 20)	0	2	0	0
wins		**test**	18 (of 20)	0	1	1	0

given the rest are fixed) is a simple but solid algorithm which works very well *if groups of parameters exhibit significant separability with respect to each other*; this is proven by TAO's separability condition [13], [14]. Optimizing over one node is not trivial but can be shown to be equivalent to a certain binary classification problem (reduced problem theorem [13], [14]), *for which efficient approximate algorithms exist*.

It is instructive to compare tree learning with (say) neural net learning. The latter can be optimized via gradient-based algorithms which guarantee a monotonic decrease of the objective function (and indeed are responsible for the success of deep learning). Alternating optimization works badly with neural nets because their weights do not exhibit separability (although such separability can be artificially introduced [24], [25]). With decision trees we have the opposite situation: we cannot use gradients, but alternating optimization applies and is effective. CART-type algorithms grow the tree structure (greedily fixing node parameters). This can also be applied to neural nets, but *it is far more effective to fix the structure and optimize the weights*—just as TAO does.

In TAO the exploration over tree structures happens implicitly: weights become 0 because of the ℓ_1 penalty, and when all the weights in a node become 0 that node is effectively redundant and can be pruned at the end.

The better optimization also makes oblique trees much better than axis-aligned trees, which was to be expected (particularly with correlated high-dimensional features) but did not occur with CART-type algorithms. This is because optimizing a purity criterion at a node can be done exactly with univariate nodes (by enumeration over all features and thresholds) but is a difficult problem with multivariate nodes.

B. *Approximate brute force search via branch & bound*

The OCT [17] and GOSDT [18] papers claim their algorithms are optimal (indeed, "optimal" is part of each algorithm's name) and have provable guarantees of optimality. This

TABLE IV
CLASSIFICATION DATASETS: AVERAGE MAXIMUM DEPTH (Δ) AND AVERAGE NUMBER OF LEAVES (L) OVER 10 REPEATS FOR BOTH AXIS-ALIGNED AND OBLIQUE TREES, REPORTED AS Δ \L. "–" MEANS NOT REPORTED IN THE ORIGINAL PAPER OR EXCEEDED TRAINING TIMEOUT.

Dataset	axis-aligned							oblique				
	TAO	CART(Py)	CART(R)	C5.0	GUIDE	GOSDT	OCT	TAO	OC1	GUIDE	OCT	CO2
Iris	3\4.4	3\4.4	2.5\3.6	1.5\3.5	2.3\3.3	2\3	4\–	3\8.1	1.6\2.6	2.3\3.3	4\–	–\–
Wine	2.8\5.6	2.8\5.8	3\5.5	1.2\3.5	2.7\4.6	–\–	4\–	5\16	2\3.2	2.4\3.5	4\–	–\–
Dermatology	7\9.6	7\9.6	6.1\7.2	4.7\6.7	6.7\8.6	–\–	4\–	5\13.5	3\5	5.1\6.2	4\–	–\–
Balance scale	7.2\24.6	7.2\24.6	6.7\22.6	7.1\27.8	6.3\18.1	2.4\3.4	4\–	4\9.9	2.9\4.3	5.8\8.3	4\–	–\–
Breast Cancer	3.4\5.4	3.4\5.6	3.2\5.5	4\9	4\7.6	1\2	4\–	3\7.8	2.5\4	1.8\3.2	4\–	–\–
Blood Trans	7.4\14	7.4\20.8	0\1	2.5\4.6	4.8\8.2	0\1	4\–	6.3\19.1	3.9\5.7	2.4\3.5	4\–	–\–
German	5\6.6	5\7.6	4.9\6.6	5.2\19.10	4.5\5.9	–\–	4\–	3\3.4	2.4\4.9	3.5\6.4	4\–	–\–
Banknote auth	6\17.8	6\20.2	5.8\14	5.8\14.3	6.7\19	–\–	4\–	3\7.4	4.2\9.3	5.1\8.3	4\–	–\–
Contraceptive	4.6\6.6	4.6\7.4	4.3\7.6	11.1\89.6	5.6\12.7	–\–	4\–	5\24.4	5.1\9.3	4\7.3	4\–	–\–
Car Eval	12.7\68	12.2\68	11.8\56.7	3.6\41.2	4.6\6.7	0\1	4\–	7\38.3	3.5\5.6	3.6\5	4\–	–\–
Segment	14\38.2	14\39.2	13.8\41.3	7.3\21	13.2\39.7	–\–	–\–	8\135	5.3\11	15.2\31.2	–\–	8\–
Spambase	14.2\50.2	14.4\55	10.7\41.7	14.7\68.6	12.3\53.5	–\–	4\–	4\14.8	3.56\5.1	15.4\48	4\–	–\–
Optical recog	12\193.8	12\198.2	11.7\107.2	10.8\72.6	13.4\126.6	–\–	4\–	7\57.4	4.2\9.6	13.7\138.7	4\–	–\–
Landsat	12\231.4	12\236.4	11.8\57.6	11.1\67.1	9.9\50.8	–\–	4\–	7\70.6	6.3\13.4	17\50.5	4\–	–\–
Pendigits	15.2\177.2	15.2\183.6	13.8\153.6	13.6\129	14\161.5	–\–	–\–	8\146	5.9\19.4	20.1\135.5	–\–	12\–
Letter	27\1550	27\1579	26\920	17\1343	23.4\994.8	–\–	–\–	11\1077	10.2\88.7	28\673.1	–\–	12\–
Connect4	33.2\5336	33.2\5744	27.7\1213	16.8\813	16.7\38.4	–\–	–\–	8\210	8.6\32.8	17.9\26.5	–\–	16\–
MNIST-pixels	19\899.5	19\899.5	18.3\805.4	19\941.6	8.5\58.8	–\–	–\–	8\177.8	5\12.8	14.9\38.5	–\–	14\–
MNIST-LeNet5	17\484	17\484	18.6\363.2	15.2\582	9.3\42.4	–\–	–\–	8\166.8	4.4\11.8	9.3\42.4	–\–	–\–
SensIT	12\152	12\152	14\239.5	15.2\410	9.8\41.6	–\–	–\–	7\69.2	8.1\21.8	10.8\24.3	–\–	6\–

TABLE V
REGRESSION DATASETS (AXIS-ALIGNED AND OBLIQUE TREES). WE REPORT TRAIN AND TEST ROOT MEAN SQUARED ERROR (RMSE), AS IN TABLE II.

Dataset	$(N_{\text{train}},\ D, K)$		oblique	axis-aligned			
			TAO	TAO	CART(R)	CART(Py)	GUIDE
concrete	(687, 8, 1)	train	3.41 ±0.23	3.06 ±5.36	3.93 ±2.67	3.07 ±2.31	5.46 ± 0.37
		test	7.17 ±0.43	7.20 ±3.17	7.23 ±3.08	7.22 ±3.13	7.50 ± 0.27
airfoil	(1002, 5, 1)	train	3.01 ±0.29	0.47 ±0.10	0.72 ±0.12	0.52 ±0.10	2.44 ± 0.10
		test	3.13 ±0.38	2.73 ±0.62	2.77 ±0.86	2.75 ±0.62	3.20 ± 0.19
abalone	(2506, 8, 1)	train	2.11 ±0.02	2.29 ±0.12	2.31 ±0.25	2.32 ±0.11	2.21 ± 0.05
		test	2.18 ±0.05	2.32 ±0.58	2.38 ±0.31	2.34 ±0.59	2.30 ± 0.09
cpuact	(4915, 21, 1)	train	2.47 ±0.07	2.68 ±0.69	2.91 ±0.71	2.71 ±0.65	10.00 ± 0.88
		test	2.71 ±0.04	3.26 ±0.51	3.36 ±1.32	3.28 ±0.44	10.99 ± 1.54
ailerons $(E \times 10^{-4})$	(7154, 40, 1)	train	1.65 ±0.02	2.39 ±0.00	1.81 ±0.12	2.83 ±0.23	1.86 ± 0.02
		test	1.76 ±0.02	2.55 ±0.00	2.21 ±0.63	2.85 ±0.57	2.06 ± 0.02
CT slice	(42800, 384, 1)	train	1.42 ±0.04	1.01 ±0.04	1.12 ±0.15	1.06 ±0.06	8.12 ± 0.17
		test	1.54 ±0.05	2.66 ±0.04	2.91 ±0.95	2.69 ±0.03	8.23 ± 0.20
YearPredictionMSD	(463715, 90, 1)	train	8.91 ±0.03	9.71 ±0.31	9.73 ±0.20	9.71 ±0.24	9.78 ± 0.01
		test	9.11 ±0.05	9.76 ±0.11	9.81 ±0.31	9.79 ±0.54	9.83 ± 0.01
wins		train	4 (of 7)	3	0	0	0
wins		**test**	6 (of 7)	1	0	0	0

TABLE VI
REGRESSION DATASETS: AVERAGE MAXIMUM DEPTH (Δ) AND AVERAGE NUMBER OF LEAVES (L) OVER 10 REPEATS.

Dataset	oblique		axis-aligned							
	TAO		TAO		CART(R)		CART(Py)		GUIDE	
	Δ	L	Δ	L	Δ	L	Δ	L	Δ	L
concrete	9.0	192.0	11.2	113.0	11.8	104.4	11.2	113.0	9.0	59.9
airfoil	8.0	147.1	15.0	479.8	15.6	457.2	15.0	479.8	10.6	95.3
abalone	6.0	58.6	5.0	12.8	4.8	11.0	5.0	12.8	6.0	18.1
cpuact	6.0	52.7	9.0	57.2	8.7	52.8	9.0	57.2	8.4	21.8
ailerons	6.0	60.2	7.0	15.0	7.8	66.6	7.0	15.0	8.5	66.1
CT slice	7.0	74.8	36.0	700.0	30.0	691.6	36.0	700.0	12.7	83.2
YearPredictionMSD	8.0	157.9	12.0	135.0	11.8	121.0	12.0	135.0	10.3	111.9

misrepresents the reality and is in stark contrast with their practical performance. The catch is that brute-force search, while obviously optimal in theory, can perform terribly under realistic computation time limits. Alternating optimization and gradient methods actively and efficiently improve the current iterate until convergence to a (local) optimum. Branch & bound (B&B) works in a far less aggressive way, exploring an exponential number of subproblems (most of which are useless) and hence having an exponential worst-case complexity (which is not significantly improved by using running bounds to prune problems). Exact B&B is impractical unless one limits the tree to tiny sizes because the number of tree structures is huge (it grows much faster than 2^n where n is the number of nodes [14]), and besides one has to optimize over the nodes' parameters as well. Stopping B&B early may unfortunately provide meaningless estimates, because high-quality solutions may not appear until deep into the exploration and may never be reached in a practical runtime. Also, B&B cannot recognize that a tree is optimal until the entire search is finished, while alternating optimization will terminate at an optimal tree instantly (since it cannot be improved).

It is true that mixed-integer optimization commercial solvers have drastically improved in the last decades, as emphasized (somewhat hyperbolically) by [17], but the empirical results show that approximate brute-force search is still limited to unacceptably small trees in practice (depth up to around 4 and small datasets in dimensionality and sample size). Few real-world problems can be modeled with so small models. Indeed, with TAO we observe that the accuracy improves significantly with the depth until it stagnates. Although this depth depends on the difficulty of the problem, it is rarely smaller than 4, even for oblique trees, and usually quite bigger. Importantly, note that the number of leaves and decision nodes grows exponentially with the depth; each additional layer doubles that number. Thus, increasing the depth a bit causes enormous extra computation for B&B methods. Hence, significant algorithmic improvements will be necessary for brute force search via branch & bound to become ever competitive with methods such as TAO that monotonically decrease the objective function at each iteration.

VI. Conclusion

For a long time, decision trees have been unfairly considered as second-class models in terms of accuracy, which has limited their practical application. Rather than a lack of modeling power, the problem has been a lack of an effective algorithm to learn trees from data—until now. Non-greedy optimization jointly over all the nodes' parameters makes it possible to train highly accurate trees that can also remain interpretable, in particular with the TAO algorithm, as shown here. This should make oblique trees much more useful in practice. Beyond this, TAO has also shown marked improvements on forests [26]–[28] and can also train more complex tree-based models [29].

Acknowledgments. Work partially supported by NSF award IIS–2007147.

References

[1] L. Hyafil and R. L. Rivest, "Constructing optimal binary decision trees is NP-complete," *Information Processing Letters*, vol. 5, 1976.

[2] L. J. Breiman, J. H. Friedman, R. A. Olshen, and C. J. Stone, *Classification and Regression Trees*. Belmont, Calif.: Wadsworth, 1984.

[3] J. R. Quinlan, *C4.5: Programs for Machine Learning*. Morgan Kaufmann, 1993.

[4] S. K. Murthy, "Automatic construction of decision trees from data: A multi-disciplinary survey," *Data Mining and Knowledge Discovery*, vol. 2, 1998.

[5] W.-Y. Loh, "Fifty years of classification and regression trees," *International Statistical Review*, vol. 82, 2014.

[6] T. J. Hastie, R. J. Tibshirani, and J. H. Friedman, *The Elements of Statistical Learning*, 2nd ed., Springer, 2009.

[7] D. Michie, D. J. Spiegelhalter, and C. C. Taylor, *Machine Learning, Neural and Statistical Classification*. Ellis Horwood, 1994.

[8] T.-S. Lim, W.-Y. Loh, and Y.-S. Shih, "A comparison of prediction accuracy, complexity, and training time of thirty-three old and new classification algorithms," *Machine Learning*, vol. 40, 2000.

[9] A. Zharmagambetov, S. S. Hada, M. Á. Carreira-Perpiñán, and M. Gabidolla, "An experimental comparison of old and new decision tree algorithms," Mar. 20 2020, arXiv:1911.03054.

[10] S. K. Murthy, S. Kasif, S. Salzberg, and R. Beigel, "OC1: A randomized algorithm for building oblique decision trees," in *AAAI*, 1993.

[11] W.-Y. Loh, "Improving the precision of classification trees," *Annals of Applied Statistics*, vol. 3, 2009.

[12] M. Norouzi, M. Collins, M. A. Johnson, D. J. Fleet, and P. Kohli, "Efficient non-greedy optimization of decision trees," in *NIPS*, 2015.

[13] M. Á. Carreira-Perpiñán and P. Tavallali, "Alternating optimization of decision trees, with application to learning sparse oblique trees," in *NEURIPS*, 2018.

[14] M. Á. Carreira-Perpiñán, "The Tree Alternating Optimization (TAO) algorithm: A new way to learn decision trees and tree-based models," 2021, arXiv.

[15] K. P. Bennett, "Global tree optimization: A non-greedy decision tree algorithm," *Computing Science and Statistics*, vol. 26, 1994.

[16] S. Nijssen and E. Fromont, "Optimal constraint-based decision tree induction from itemset lattices," *Data Mining and Knowledge Discovery*, vol. 21, 2010.

[17] D. Bertsimas and J. Dunn, "Optimal classification trees," *Machine Learning*, vol. 106, 2017.

[18] J. Lin, C. Zhong, D. Hu, C. Rudin, and M. Seltzer, "Generalized and scalable optimal sparse decision trees," in *ICML*, 2020.

[19] T. Therneau, B. Atkinson, and B. Ripley, "rpart: Recursive partitioning and regression trees," R package version 4.1-15, Apr. 12 2019, available online at https://cran.r-project.org/package=rpart.

[20] F. Pedregosa et al., "Scikit-learn: Machine learning in Python," *J. Machine Learning Research*, vol. 12, 2011, available online at https://scikit-learn.org.

[21] R.-E. Fan, K.-W. Chang, C.-J. Hsieh, X.-R. Wang, and C.-J. Lin, "LIBLINEAR: A library for large linear classification," *J. Machine Learning Research*, vol. 9, 2008.

[22] M. Lichman, "UCI machine learning repository," http://archive.ics.uci.edu/ml, 2013.

[23] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner, "Gradient-based learning applied to document recognition," *Proc. IEEE*, vol. 86, 1998.

[24] M. Á. Carreira-Perpiñán and W. Wang, "Distributed optimization of deeply nested systems," Dec. 24 2012, arXiv:1212.5921.

[25] ——, "Distributed optimization of deeply nested systems," in *AISTATS*, 2014.

[26] M. Á. Carreira-Perpiñán and A. Zharmagambetov, "Ensembles of bagged TAO trees consistently improve over random forests, AdaBoost and gradient boosting," in *FODS*, 2020.

[27] A. Zharmagambetov and M. Á. Carreira-Perpiñán, "Smaller, more accurate regression forests using tree alternating optimization," in *ICML*, 2020.

[28] A. Zharmagambetov, M. Gabidolla, and M. Á. Carreira-Perpiñán, "Improved boosted regression forests through non-greedy tree optimization," in *IJCNN*, 2021.

[29] A. Zharmagambetov and M. Á. Carreira-Perpiñán, "Learning a tree of neural nets," in *ICASSP*, 2021.