TU Delft

Delft University of Technology

Piecewise Constant and Linear Regression Trees
An Optimal Dynamic Programming Approach

van den Bos, Mim; van der Linden, Jacobus G.M.; Demirović, Emir

Publication date
2024

Document Version
Final published version

Published in
Proceedings of Machine Learning Research

Citation (APA)
van den Bos, M., van der Linden, J. G. M., & Demirović, E. (2024). Piecewise Constant and Linear Regression Trees: An Optimal Dynamic Programming Approach. *Proceedings of Machine Learning Research*, *235*, 48994-49007.

Important note
To cite this publication, please use the final published version (if applicable).
Please check the document version above.

Copyright
Other than for strictly personal use, it is not permitted to download, forward or distribute the text or part of it, without the consent of the author(s) and/or copyright holder(s), unless the work is under an open content license such as Creative Commons.

Takedown policy
Please contact us and provide details if you believe this document breaches copyrights.
We will remove access to the work immediately and investigate your claim.

This work is downloaded from Delft University of Technology.
For technical reasons the number of authors shown on this cover page is limited to a maximum of 10.

Piecewise Constant and Linear Regression Trees: An Optimal Dynamic Programming Approach

Mim van den Bos [*1] **Jacobus G. M. van der Linden** [*1] **Emir Demirović** [1]

Abstract

Regression trees are a human-comprehensible machine-learning model that can represent complex relationships. They are typically trained using greedy heuristics because computing optimal regression trees is NP-hard. Contrary to this standard practice, we consider optimal methods and improve the scalability of optimal methods by developing three new dynamic programming approaches. First, we improve the performance of a piecewise constant regression tree method using a special algorithm for trees of depth two. Second, we provide the first optimal dynamic programming method for piecewise multiple linear regression. Third, we develop the first optimal method for piecewise simple linear regression, for which we also provide a special algorithm for trees of depth two. The experimental results show that our methods improve scalability by one or more orders of magnitude over the state-of-the-art optimal methods while performing similarly or better in out-of-sample performance.

1. Introduction

Regression trees generalize linear regression by splitting the data before learning a regression model, as seen in Fig. 1. This makes regression trees a powerful tool for regression analysis, with wide applications ranging from ecology analysis (De'ath & Fabricius, 2000) to clinical psychology (King & Resick, 2014). Because of their rule-based nature, they satisfy the need for complex, nonlinear models that are human-comprehensible (Loh, 2014; Freitas, 2014; Carrizosa et al., 2021; Rudin, 2019).

Because human comprehensibility of decision trees is directly related to the size of the tree (Piltaver et al., 2016), we are interested in small trees with high performance, e.g., small mean squared error. Traditionally, heuristics, such as CART (Breiman et al., 1984), dominate the decision tree literature due to their scalability. Heuristics can learn small trees but may yield a suboptimal representation of the data. In contrast, optimal decision trees provably maximize performance over training data for a given size limit. On average, optimal trees also perform better in out-of-sample tests (Bertsimas & Dunn, 2017; Bertsimas et al., 2017).

Since computing optimal decision trees is NP-hard (Hyafil & Rivest, 1976), *scalability* (the ability to keep run time low for larger data sets and larger tree sizes) remains a challenge. For example, optimal approaches that use general-purpose solvers such as Mixed-Integer Programming (MIP, Bertsimas & Dunn, 2017), constraint programming (CP, Verhaeghe et al., 2020), or maximum satisfiability (MaxSAT, Hu et al., 2020), often even do not scale beyond small data sets and depth limits as small as depth three.

Recent work shows significant scalability improvements using dynamic programming (DP) approaches (Aglin et al., 2020a; Lin et al., 2020; Demirović et al., 2022). DP directly exploits the recursive tree structure by solving subtrees as independent subproblems, caching results to subproblems, and using branching and bounding to limit the search space.

Most optimal decision tree methods consider only classification. Dunn (2018) and Verwer & Zhang (2017) show how their MIP models can also be used for regression. Dunn (2018) considers both piecewise constant regression trees (with a constant predictor in each leaf node) and piecewise

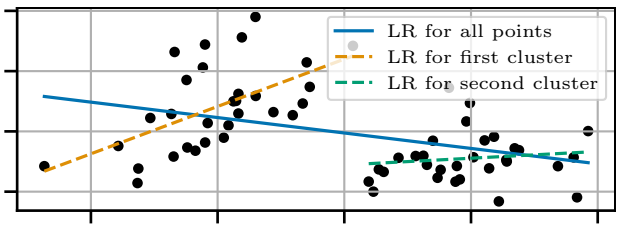

Figure 1. The blue line represents standard linear regression (LR), whereas orange and green show the advantage of adding one split before linear regression.

[*]Equal contribution [1]Department of Software Technology, Delft University of Technology, Delft, The Netherlands. Correspondence to: Jacobus G. M. van der Linden <J.G.M.vanderLinden@tudelft.nl>.

Proceedings of the 41st International Conference on Machine Learning, Vienna, Austria. PMLR 235, 2024. Copyright 2024 by the author(s).

linear regression trees (with a linear predictor in each leaf node, as seen in Fig. 1), whereas Verwer & Zhang (2017) only consider constant predictors. However, Dunn (2018) observes that lack of scalability prevents the use of MIP for practical applications. Zhang et al. (2023) are the first to apply DP, but they only consider piecewise constant regression trees. Moreover, they do not incorporate some of the latest scalability improvements for optimal classification trees.

Therefore, we develop three novel contributions. First, we provide a new DP algorithm with scalability improvements for piecewise constant regression trees that incorporates and adapts an efficient algorithm for classification trees of depth two to regression. Second, we provide the first optimal DP algorithm for piecewise linear regression trees. Third, we consider the special case of piecewise *simple* linear regression, which restricts each linear model to only one independent variable, and provide another depth-two algorithm that also yields large scalability improvements.

As commonly done, we train *binary* decision trees and assume all possible tree predicates to be given, e.g., in the form of binary features. The linear models in the leaf nodes are trained with the original continuous features.

We compare our methods with ten methods from the literature, both heuristics and optimal methods. Our methods surpass previous optimal methods by one or more orders of magnitude in scalability, mainly because of our new depth-two algorithms. Our methods' out-of-sample performance is on par with or better than the state-of-the-art.

2. Related work

Regression tree heuristics Since computation of optimal decision trees is NP-Hard (Hyafil & Rivest, 1976), heuristics are commonly used for computing regression trees. The first regression tree heuristics was AID (Morgan & Sonquist, 1963) which greedily partitions a node such that the sum of squared errors (SSE) is minimized. AID halts when no partition exists which results in an improvement above a certain threshold. CART (Breiman et al., 1984) is one of the most-used heuristics for piecewise constant regression trees. Instead of respecting a minimum improvement threshold, CART grows trees that overfit on the data and then uses cross-validation to prune overfitting branches. GUIDE (Loh, 2002) is a more recent greedy heuristic for both piecewise constant and piecewise (simple) linear trees that uses χ^2-tests to address the bias of preferring to branch on features with many unique values. Other examples of commonly used greedy heuristics for piecewise linear and polynomial regression trees are M5 (Quinlan, 1992; Wang & Witten, 1997), and MARS (Friedman, 1991), respectively.

Apart from greedy heuristics, Hemmateenejad et al. (2011) and Grubinger et al. (2014) propose to use ant colony optimization and evolutionary algorithms. Yang et al. (2016) greedily choose a single continuous splitting feature for the whole tree, and then use MIP on this tree to globally optimize a piecewise linear model for each leaf node. Bertsimas et al. (2021) use Adam (Kingma & Ba, 2015) for gradient optimization of tree splits and then apply polynomial ridge regression in the leaf nodes. Blanquero et al. (2022) use randomized splits to transform the problem into a nonlinear continuous optimization problem with better scalability, but with reduced interpretability. Dunn (2018) uses local search with random restarts to iteratively optimize a single node in the tree. Similarly, Yang et al. (2017) use local search to iteratively optimize a single split decision together with the leaf node regression function using MIP.

Optimal classification trees In contrast to heuristics, optimal decision tree methods guarantee global optimality over the training data for the given tree size limit. Early approaches focused on *classification*. Bertsimas & Dunn (2017) propose a first MIP model, with improved models quickly following after (Verwer & Zhang, 2017; 2019; Zhu et al., 2020; Aghaei et al., 2021); Verhaeghe et al. (2020) propose a constraint programming approach; whereas Narodytska et al. (2018); Janota & Morgado (2020) use SAT to find a perfect tree of minimum size; and Hu et al. (2020) maximize accuracy using MaxSAT. However, despite improvements, these methods based on general-purpose solvers struggle to scale beyond a thousand data points. Therefore, Schidler & Szeider (2021), for example, propose to combine heuristics and optimal methods, such as SAT, to improve scalability.

Better scalability results are obtained with dynamic programming approaches of which DL8 is one of the first (Nijssen & Fromont, 2007). Aglin et al. (2020a;b) extend DL8 into DL8.5 by adding bound-based pruning of the search space. Hu et al. (2019) and Lin et al. (2020) combine ideas from DP with new lower bounds and sparsity-based pruning. Demirović et al. (2022) contribute several improvements, among which a similarity-based lower bound and a special algorithm for trees of at most depth two that significantly improves scalability by applying smart precomputations.

DP has also been successfully applied beyond classification, for example, to classification under a group fairness constraint (Van der Linden et al., 2022) and survival analysis (Huisman et al., 2024). Van der Linden et al. (2023) show that dynamic programming can be used for any optimization task for which optimal solutions to subtrees can be computed independently of the rest of the tree. This indicates that the improvements obtained in DP for optimal classification trees may also be applied to regression.

Optimal regression trees Optimal MIP models for *regression* trees were proposed by Bertsimas et al. (2017) and Verwer & Zhang (2017). Dunn (2018) presents an optimal

MIP method for both piecewise constant and linear trees, but observes the lack of scalability and then proposes to use a local search heuristic instead. The only optimal DP-based approach for piecewise constant regression trees is OSRT (Zhang et al., 2023). They observe that for each subtree with at most k nodes, the error of an optimal k-means clustering provides a lower bound for the current subtree, which can be computed efficiently by using the algorithm by Wang & Song (2011) and Song & Zhong (2020).

Summary Only a few of the many regression tree methods are optimal and among those, scalability remains challenging. Furthermore, no scalable optimal piecewise linear regression method exists yet. We address these gaps by improving the scalability of piecewise constant regression methods and by providing the first optimal DP approaches for piecewise simple and multiple linear regression trees.

3. Preliminaries

This section introduces the notation, provides a formal problem definition, and explains the basics of the DP approach.

Notation Let $\mathcal{F}$ be a set of features, $\mathcal{F}_b$ a corresponding set of binarized features, and $\mathcal{D}$ a data set of instances (x, b, y). The vector $x \in \mathbb{R}^{|\mathcal{F}|}$ represents the continuous feature vector and $b \subseteq \mathcal{F}_b$ represents the binary feature vector, where $f \in b$ means the binary predicate f is satisfied in vector b. Similarly, let $\bar{f}$ represent $f \notin b$. The symbol $y \in \mathbb{R}$ represents the label of an instance. Data sets can be split on binary features f into two data sets $\mathcal{D}_f$ and $\mathcal{D}_{\bar{f}}$ that respectively contain all instances (x, b, y) that satisfy $(f \in b)$ or do not satisfy $(f \notin b)$ feature f. Similarly, we define $\mathcal{D}_{f_i, f_j}$ as the subset of $\mathcal{D}$ that satisfies both f_i and f_j. Let $\bar{y}$ denote the mean of the labels of a data set $\mathcal{D}$.

Problem definition A binary regression tree is a function $\tau : \mathbb{R}^{|\mathcal{F}|} \times \mathcal{P}(\mathcal{F}_b) \to \mathbb{R}$ that maps an instance (x, b) to a predicted output label $\hat{y}$. Here $\mathcal{P}$ denotes the power set. Each branching node performs a binary feature test from $\mathcal{F}_b$ and directs instances to the left or right subtree. Leaf nodes assign output labels to every instance that ends up in it. The quality of a decision tree is determined by computing the error, such as the Sum of Squared Errors (SSE), of the tree output with respect to the true values.

Optimal regression trees provably globally optimize the SSE for a given size limit (number of nodes or depth of the tree). To prevent overfitting, we penalize the size of the tree, given by the number of branching nodes $N(\tau)$, by a regularization parameter λ, which we scale by the total sum of squares of the training data. Therefore, given a maximum depth d and a training data set $\mathcal{D}$, the objective function is:

$$\min_{\tau} \sum_{(x,b,y)\in\mathcal{D}} (y - \tau(x, b))^2 + \lambda N(\tau) \tag{1}$$

Dynamic programming approach The basis of the dynamic programming approach is the recursive formulation of the problem. A simplified version of the recursive formulation for minimizing misclassification score from (Demirović et al., 2022) is as follows:

$$T(\mathcal{D}, d) = \begin{cases} \min_{\hat{k}\in\mathcal{K}} \sum_{(b,k)\in\mathcal{D}} \mathbb{1}(k \neq \hat{k}) & \text{if } d = 0 \\ \min_{f\in\mathcal{F}_b} \{T(\mathcal{D}_f, d-1) + & \\ \qquad T(\mathcal{D}_{\bar{f}}, d-1)\} & \text{if } d > 0 \end{cases} \tag{2}$$

In this equation, the maximum depth of the tree d and the data set $\mathcal{D}$ define the DP state. $\mathcal{K}$ is the set of labels, k the true label and $\hat{k}$ the predicted label. At each branching node ($d > 0$), all possible branching decisions $f \in \mathcal{F}_b$ are considered, resulting in two independent subproblems each. Leaf nodes ($d = 0$) select the label with the lowest misclassification costs. For brevity of notation, this and subsequent DP formulations only consider complete trees.

Demirović et al. (2022) significantly enhance scalability by introducing a specialized algorithm for depth-two trees. Instead of using the default recursion, this algorithm efficiently precomputes class occurrences for the whole data set and all possible depth-two splits, to prevent traversing the whole data set repeatedly for computing the misclassification score for each possible leaf node. In the next section, we describe how a similar idea can be used to significantly improve computation of regression trees up to depth two.

4. Piecewise Constant Regression Trees

This section explains how optimal piecewise constant regression trees can be computed using DP and how a special solver for trees of depth two greatly improves scalability.

Dynamic programming formulation The following equation adapts Eq. (2) for regression:

$$T(\mathcal{D}, d) = \begin{cases} \sum_{(b,y)\in\mathcal{D}} (y - \bar{y})^2 & \text{if } d = 0 \\ \min_{f\in\mathcal{F}_b} \{T(\mathcal{D}_f, d-1) + & \\ \qquad T(\mathcal{D}_{\bar{f}}, d-1) + \lambda\} & \text{if } d > 0 \end{cases} \tag{3}$$

The updated formulation selects the mean $\bar{y}$ as the leaf node's label and returns the SSE as its cost. Adding a branching node is penalized with a regularization parameter λ. See Appendix A for the full pseudocode including non-complete trees, caching, and bound-based pruning.

Precomputing per-instance costs As introduced in the preliminaries, Demirović et al. (2022) obtain a major increase in scalability because of their special solver for trees

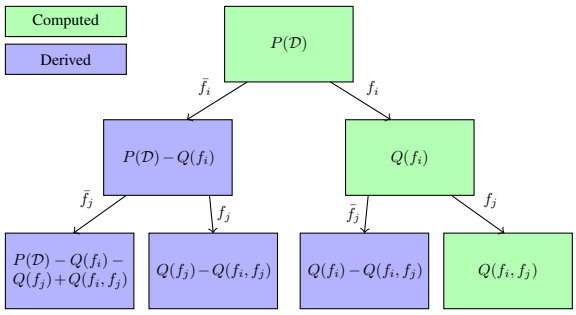

Figure 2. For depth-two trees, only the green values need to be precomputed per feature pair f_i and f_j. This can be done efficiently by looping only over the list of satisfied features per instance in $\mathcal{D}$. The other (blue) values can be derived according to Eq. (7).

of maximum depth two. This is obtained by precomputing class occurrences before looping over every possible combination of feature splits. As a result, instead of traversing the whole data set for every combination of two features, the precomputation only considers the features that are present for every instance. This algorithm is shown to be very effective for classification, and Van der Linden et al. (2023) generalize this special depth-two solver to any optimization task, provided that the costs for a leaf node can be expressed as a function of the per-instance contribution to the costs.

However, for regression, the error of each instance depends on the mean of all instances in a leaf node, and when one instance is added to or removed from a leaf node, the error of every other instance changes. Despite this, we can rewrite the SSE as a function of three sums over the instances: the sum of y, the sum of y^2, and the number of instances $|\mathcal{D}|$:

$$\sum_{(b,y)\in\mathcal{D}} (y-\bar{y})^2 = \sum_{(b,y)\in\mathcal{D}} y^2 - \frac{\left(\sum_{(b,y)\in\mathcal{D}} y\right)^2}{|\mathcal{D}|} \tag{4}$$

For example, with labels 4, 5, and 6, we have $\bar{y} = 5$, and the SSE is 2. Eq. (4) yields $4^2 + 5^2 + 6^2 = 77$, $(4+5+6)^2 = 225$, and $|\mathcal{D}| = 3$, so $77 - 225/3 = 2$. Therefore, Eq. (4) allows us to define the per-instance costs as a three-tuple $(y, y^2, 1)$, which can be summed for multiple instances using element-wise addition. Let P represent the sum of per-instance cost for a data set $\mathcal{D}$:

$$P(\mathcal{D}) = \sum_{(b,y)\in\mathcal{D}} (y, y^2, 1) \tag{5}$$

The resulting sums of $\sum y$, $\sum y^2$ and $n = |\mathcal{D}|$ can be used to obtain the SSE through Eq. (4).

$$C(\sum y, \sum y^2, n) = \sum y^2 - \frac{(\sum y)^2}{n} \tag{6}$$

The optimal label $\hat{y} = \bar{y}$ is obtained through $\sum y/n$.

Algorithm 1: Depth-two tree search for a data set $\mathcal{D}$ and a feature set $\mathcal{F}_b$. $BestL.C$ and $BestR.C$ are the best left and right subtrees respectively.

$Q(f_i) \leftarrow (0,0,0) \quad \forall f_i \in \mathcal{F}_b$
$Q(f_i, f_j) \leftarrow (0,0,0) \quad \forall f_i, f_j, \in \mathcal{F}_b$ s.t. $i < j$
for $(b, y) \in \mathcal{D}$ **do**
  for $f_i \in b$ **do**
    $Q(f_i) \leftarrow Q(f_i) + (y, y^2, 1)$
    for $f_j \in b$, *s.t.* $i < j$ **do**
      $Q(f_i, f_j) \leftarrow Q(f_i, f_j) + (y, y^2, 1)$
for $f_i \in \mathcal{F}_b$ **do**
  for $f_j \in \mathcal{F}_b$ **do**
    $C_L = C(Q(\bar{f}_i, f_j)) + C(Q(\bar{f}_i, \bar{f}_j))$
    $C_R = C(Q(f_i, f_j)) + C(Q(f_i, \bar{f}_j))$
    if $BestL.C(f_i) > C_L$ **then**
      $BestL.C(f_i) \leftarrow C_L$
    if $BestR.C(f_i) > C_R$ **then**
      $BestR.C(f_i) \leftarrow C_R$
return $\min_{f_i \in \mathcal{F}_b} BestL.C(f_i) + BestR.C(f_i)$

This break-down in per-instance costs allows for a similar performance gain as in (Demirović et al., 2022) by precomputing $Q(f_i) = P(\mathcal{D}_{f_i})$ and $Q(f_i, f_j) = P(\mathcal{D}_{f_i, f_j})$. Based on these precomputations, the values for other splits, such as $P(\mathcal{D}_{\bar{f}_i, f_j}) = Q(\bar{f}_i, f_j)$ can be obtained as follows:

$$\begin{aligned} Q(\bar{f}_i) &= P(\mathcal{D}) - Q(f_i) \\ Q(f_i, \bar{f}_j) &= Q(f_i) - Q(f_i, f_j) \\ Q(\bar{f}_i, f_j) &= Q(f_j) - Q(f_i, f_j) \\ Q(\bar{f}_i, \bar{f}_j) &= P(\mathcal{D}) - Q(f_i) - Q(f_j) + Q(f_i, f_j) \end{aligned} \tag{7}$$

Fig. 2 and Algorithm 1 show how the values for $Q(f_i)$ and $Q(f_i, f_j)$ can be efficiently computed by looping only over the features present for every instance. After precomputing the values for $P(\mathcal{D}), Q(f_i)$, and $Q(f_i, f_j)$, the algorithm loops over all possible combinations of two feature splits, computes the costs of the four resulting leaf nodes, and for each first-level split f_i, stores the costs of the best left and right subtree. Finally, it finds the first-level split that minimizes the total cost.

This procedure reduces the run time for finding depth-two trees from $O(|\mathcal{F}|^2|\mathcal{D}|)$ to $O(m^2|\mathcal{D}|)$, with m the maximum number of positive features in any instance. Since binary features are commonly sparse, this results in a significant decrease in run time at the cost of $O(|\mathcal{F}|^2)$ extra memory.

Lower bounds Scalability is further increased by lower bounds that help prune the search. We incorporate the equivalent points, k-means, and look-ahead lower bound from (Zhang et al., 2023), and the similarity lower bound from (Demirović et al., 2022) by using the worst-case contribution of one instance from (Dunn, 2018). We implement the

equivalent points lower bound by grouping instances with equivalent feature vectors b as one instance with a weight, label y, and y^2 equal to the sum of the combined instances.

5. Piecewise Linear Regression Trees

Instead of a constant predictor in each leaf node, this section explains how optimal trees with linear regression models in each leaf node can be computed using DP. We consider both multiple linear and simple linear regression models.

5.1. Multiple Linear Regression

For multiple linear regression, we optimize an elastic net regression model in every leaf node. An elastic net promotes model sparsity by penalizing the L1-norm of the coefficients $\hat{\beta}$ with a factor κ and the L2-norm with a factor γ. We reuse the DP formulation of Eq. (3), but replace the base case ($d = 0$) with the following:

$$\min_{\hat{\beta}_0,\hat{\beta}} \sum_{(x,b,y)\in\mathcal{D}} (y - \hat{\beta}_0 - x^T\hat{\beta})^2 + \kappa\|\hat{\beta}\|_1 + \gamma\|\hat{\beta}\|_2^2 \quad (8)$$

We use the GLMNet coordinate descent algorithm (Friedman et al., 2010) to compute the elastic net model. With GLMNet, standardization of the data takes $O(|\mathcal{D}||\mathcal{F}|)$ steps. Each iteration in the coordinate descent takes $O(m|\mathcal{F}|)$ steps, where m is the number of nonzero coefficients in the model. Adding a new nonzero coefficient takes $O(|\mathcal{D}||\mathcal{F}|)$ steps. We stop the computation after 10,000 iterations or when the change in SSE is less than 0.1%.

5.2. Simple Linear Regression

It remains an open challenge to formulate an efficient per-instance cost breakdown for multiple linear regression, and therefore we do not provide a special algorithm for trees of depth two. Instead, we consider the special case of *simple* linear regression. Simple linear regression considers only a single explanatory variable. This makes it easier to compute and also more human-comprehensible because the model can easily be plotted in 2D. The leaf node function becomes:

$$\min_{j,\hat{\beta}_0,\hat{\beta}_j} \sum_{(x,b,y)\in\mathcal{D}} (y - \hat{\beta}_0 - x_j\hat{\beta}_j)^2 + \gamma\hat{\beta}_j^2 \quad (9)$$

In this case, we only use the L2-norm (ridge) penalization, since the main aim for the L1 penalization is to reduce the number of non-zero coefficients, but simple linear regression already considers only one non-zero coefficient.

Per-instance costs In Appendix B, we derive the formulas for an optimal simple linear regression model, which we here rewrite in terms of sums that we can precompute:

$$\hat{\beta}_0 = \sum y/n - \hat{\beta}_j \sum x_j/n \quad (10)$$

$$\hat{\beta}_j = \frac{n\sum x_j y - \sum y \sum x_j}{n\sum x_j^2 - (\sum x_j)^2 + n\gamma} \quad (11)$$

By expanding Eq. (9), and rewriting it in terms of computable sums, it can be seen that the SSE of a model with intercept $\hat{\beta}_0$ and slope $\hat{\beta}_j$ is:

$$\sum y^2 - 2\hat{\beta}_j \sum yx_j - 2\hat{\beta}_0 \sum y + \hat{\beta}_j^2 \sum x_j^2 + 2\hat{\beta}_0\hat{\beta}_j \sum x_j + \gamma\hat{\beta}_j^2 + |\mathcal{D}|\hat{\beta}_0 \quad (12)$$

The sums necessary for computing Eqs. (10)-(12), can be precomputed as in the piecewise constant case:

$$P(\mathcal{D}) = \textstyle\sum_{(x,b,y)\in\mathcal{D}}(y, y^2, 1, x_1, x_1^2, x_1y, ..., x_{|\mathcal{F}|}, x_{|\mathcal{F}|}^2, x_{|\mathcal{F}|}y) \quad (13)$$

The breakdown of the per-instance costs of simple linear regression is more complex than in the constant case. Before, only the sum of y, the sum of y^2, and the number of instances were computed, but now also for every continuous feature f the sum of x_f, x_f^2, and $x_f y$ is computed, resulting in a tuple of size $3+3|\mathcal{F}|$. With this new definition of $P(\mathcal{D})$, we again apply Eq. (7) to indirectly compute the sums for all possible leaf nodes in a depth-two tree. With all these sums precomputed as before using the technique from Algorithm 1, we can compute the SSE $C_j(.)$ for when feature j is used as the explanatory variable using Eqs. (10)-(12). We do this for each feature and select the linear model with the smallest SSE: $C(\sum y, ...) = \min_j C_j(\sum y, ...)$. Algorithm 1 can now be applied in the same way as before and again yields a large scalability improvement.

6. Experiments

In our experiments, we first provide a scalability analysis and measure the effect of the depth-two algorithms. Second, we compare out-of-sample performance with the state-of-the-art. The results show that our methods outperform the state-of-the-art optimal methods in scalability by one or more orders of magnitude and that the depth-two algorithm improves scalability by one order of magnitude on average. The out-of-sample analysis shows that our methods have similar out-of-sample performance as the state-of-the-art.

6.1. Experiment Setup

Data Table 2 lists the benchmarking data sets. All data sets were obtained from the UCI Repository (Dua & Graff, 2017) and split into five folds. Categorical variables are binarized using one-hot encoding. Numerical features are

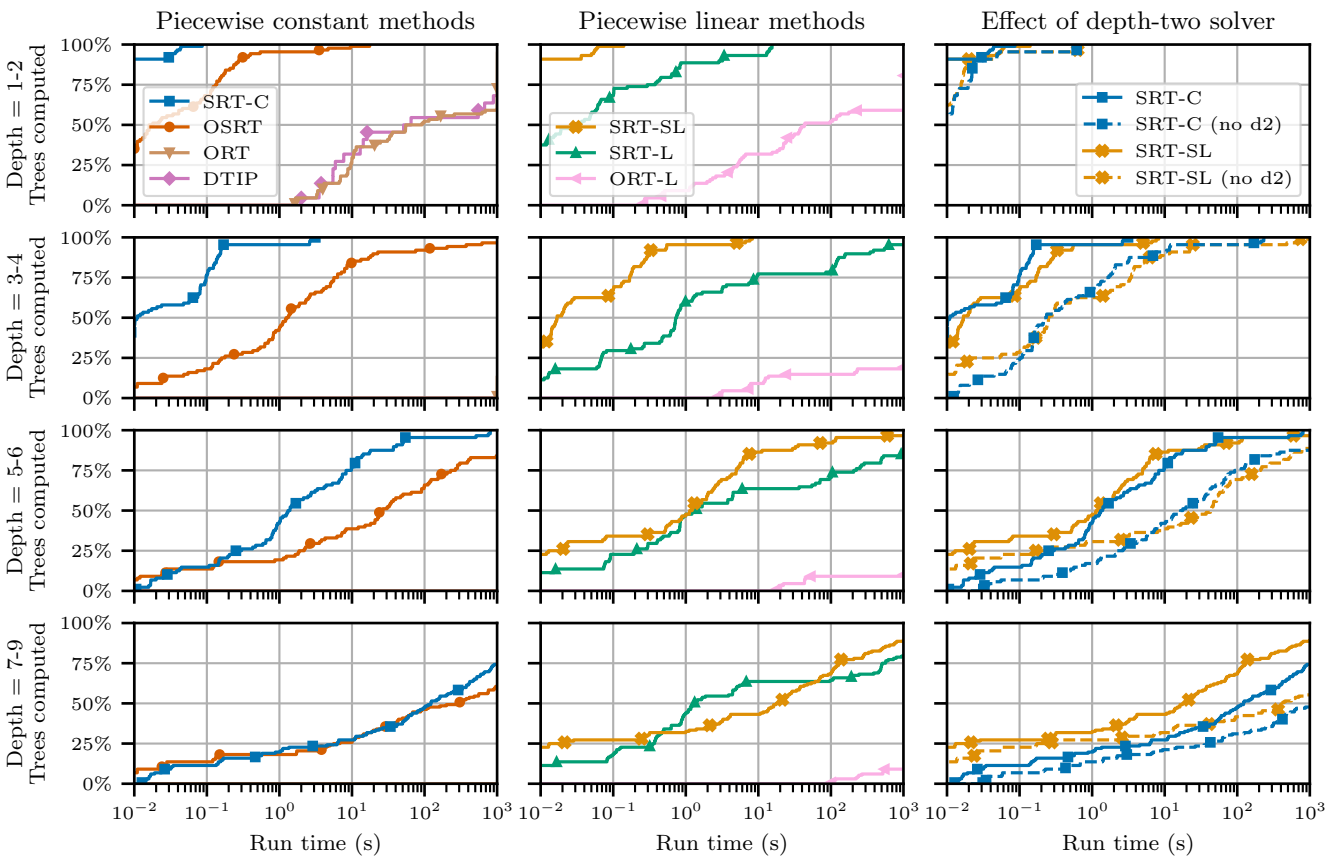

Figure 3. Run time comparison for optimal methods for $d = 1...9$. Each line shows for what percentage of problem instances the optimal tree could be computed within the given run time (higher is better). The right-most plot shows the effect of our depth-two algorithms. Our methods outperform previous optimal methods by one or more orders of magnitude. Note the logarithmic scale on the horizontal axis.

binarized by training a regression tree for only that feature with at most 10 branching nodes. The splits on each branching node are used as binary predicates. The DP methods use the binarized features as predicates for the tree splits. The other methods use the original numerical data. All piecewise linear regression methods compute their linear models on the numerical data.

Hardware Experiments were run with one thread on an Intel Xeon E5-6248R 3.0GHz with 100GB RAM. All experiments were run with a time-out of 15 minutes.

Methods We have implemented our methods in C++ using the STreeD framework (Van der Linden et al., 2023) and provide it as a Python package.[1] We name our STreeD regression tree methods SRT-C, SRT-SL, and SRT-L, referring to our piecewise constant, simple linear, and multiple linear regression methods respectively.[2]

[1] https://github.com/AlgTUDelft/pystreed

[2] See https://github.com/mimvdb/regression-murtree for our experiment setup.

Table 1 lists all methods included in our experiments. From (Dunn, 2018; Bertsimas & Dunn, 2017; Bertsimas et al., 2017), we include the univariate optimal MIP methods for both piecewise constant and linear regression (ORT and ORT-L). In the same work, Dunn (2018) also proposes a local search method for both cases, which confusingly are also named ORT and ORT-L. Instead, we refer to these methods as IAI and IAI-L after their implementation by Interpretable AI (2023). For the piecewise linear methods, we adhere to the rule of thumb that the number of training samples for a linear model should be at least 10 times the number of independent variables, thus requiring each leaf node to have at least $10|\mathcal{F}|$, or 10 instances in the case of simple linear regression. The MIP methods optimize the mean absolute error and are solved with Gurobi 9.5.

6.2. Scalability

Comparison with optimal methods To compare the scalability of optimal methods, we run each optimal method on each full data set for depth $d = 1...9$ and for normalized $\lambda = 0.1, 0.01, 0.001$, and 0.0001 (except the much

Table 1. Overview of the analyzed methods. The DP methods split on the binarized data. The piecewise linear methods use the numerical data for training the linear models in the leaf nodes.

Method	Type	Reference
Piecewise constant regression trees		
CART	Greedy	Breiman et al. (1984)
GUIDE	Greedy	Loh (2002)
IAI	Local search	Dunn (2018)
ORT	Optimal MIP	Dunn (2018)
DTIP	Optimal MIP	Verwer & Zhang (2017)
OSRT	Optimal DP	Zhang et al. (2023)
SRT-C	Optimal DP	This paper
Piecewise simple linear regression trees		
GUIDE-SL	Greedy	Loh (2002)
SRT-SL	Optimal DP	This paper
Piecewise linear regression trees		
GUIDE-L	Greedy	Loh (2002)
IAI-L	Local search	Dunn (2018)
ORT-L	Optimal MIP	Dunn (2018)
SRT-L	Optimal DP	This paper

larger Household data set, which we use below to measure scalability with respect to data set size).

Fig. 3 shows a cumulative distribution plot of the run time required for finding the optimal solution, divided in rows for different depth limits. It shows how our DP approach outperforms the MIP approaches by several orders of magnitude, both for piecewise constant and linear trees. The MIP methods ORT and DTIP do not scale beyond depth two and ORT-L only scales for larger depth for the smallest data sets. It also shows that SRT-C on average is 18 times faster than OSRT (geometric mean performance ratio). The difference is smaller for larger depth limits because then the effect of the depth-two solver for SRT-C becomes smaller. The greatest difference is observed for computing trees of depth three: SRT-C is on average 131 times faster than OSRT.

Comparison with heuristics The heuristics CART and GUIDE(-L) easily outperform all optimal methods in scalability and are therefore not included in our scalability analysis. IAI was not included in Fig. 3 because it is not an optimal method and its license requirement prevented us from running it on our distributed experiment server setup. Instead, we compared IAI with SRT-C separately on a local machine with an Intel i7-6600U CPU with 8GB RAM.

On average, SRT-C is faster than IAI for $d = 3$ and $d = 4$ by 26 and 5 times, respectively. For $d = 6$, IAI is 9 times faster. This is expected, as IAI is a local search method and SRT-C an optimal method. SRT-L and IAI-L show a similar relationship, but not as pronounced. SRT-L is faster than IAI-L up to $d = 3$ and has similar performance for $d = 4$ and $d = 5$. At $d = 6$, IAI-L is approximately 2 times faster than SRT-SL. Though SRT-C and SRT-L scale exponentially with the number of features and maximum depth and IAI and IAI-L scale linearly for the number of features and depth, both SRT-C and SRT-L remain competitive in run time with these local search algorithms for the cases we considered, while guaranteeing optimal solutions.

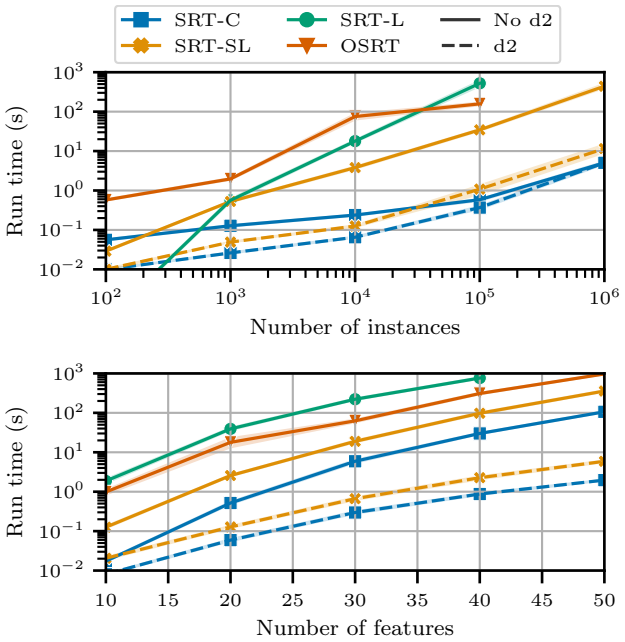

Figure 4. Run time increase for the number of instances (Household data set) and features (Seoul Bike data set) for $d = 4$. OSRT exceeds the memory limit for $|\mathcal{D}| \geq 10^6$.

Scalability break down Fig. 4 shows that –as expected– all DP methods scale linearly for the number of instances and exponentially for the number of features.

Surprisingly, the performance of SRT-SL is very close to SRT-C's performance and much better than OSRT's, which means we can now fit a simple linear regression model in each leaf node even faster than previous methods could fit constant predictors in each leaf node.

Even SRT-L, which fits a full linear regression model in its leaves, scales similarly to OSRT (with constant predictors) for an increasing number of features. However, because of the coordinate descent algorithm, it scales slightly worse for an increasing number of instances. OSRT exceeds the memory limit of 100GB for $|\mathcal{D}| \geq 10^6$.

Effect of the depth-two algorithms Fig. 3 and Fig. 4 also show the effect of our depth-two algorithm. For SRT-C and SRT-SL, on average, it improves the run time by a factor 8 and 10 respectively. For trees of depth three, it

Table 2. Out-of-sample R^2 results for the piecewise constant methods with maximum depth $d = 5$. Time-outs indicated by '-'. Best results are marked bold. $|\mathcal{D}|$ is the number of instances, $|\mathcal{F}|$ the number of original features and $|\mathcal{F}_b|$ is the number of binary features.

				Heuristic			Optimal DP	
Data set	$\lvert\mathcal{D}\rvert$	$\lvert\mathcal{F}\rvert$	$\lvert\mathcal{F}_b\rvert$	CART	GUIDE	IAI	OSRT	SRT-C
Airfoil	1503	5	28	0.61	0.59	**0.69**	-	0.67
Auction	2043	7	26	0.95	0.89	**0.97**	**0.97**	**0.97**
Auto MPG	392	7	38	0.77	0.80	0.80	-	**0.81**
Energy (C)	768	8	37	0.96	0.95	0.96	**0.97**	**0.97**
Energy (H)	768	8	37	0.99	0.97	0.99	**1.00**	**1.00**
Household	2049280	3	26	**1.00**	-	-	-	0.98
Optical Net.	640	7	42	**0.96**	0.91	0.93	0.95	0.95
Real Estate	414	6	46	**0.66**	0.62	0.63	-	**0.66**
Seoul Bike	8760	9	57	0.72	0.62	**0.74**	-	-
Servo	167	2	17	0.78	-0.02	0.78	**0.89**	**0.89**
Synch.	557	4	34	**1.00**	**1.00**	**1.00**	0.99	0.99
Yacht	308	6	43	**0.99**	**0.99**	**0.99**	-	**0.99**
Best				5	2	5	4	**7**

improves the run time of SRT-C and SRT-SL on average by a factor 20 and 12 respectively. At larger depth, the data set splits are much smaller and thus the effect of the depth-two solver is less significant. For SRT-SL, the performance improvement remains significant even at larger depth limits. The benefit of the depth-two algorithm increases with more features. SRT-C sees a diminishing impact of the depth-two algorithm for an increasing number of instances, but for SRT-SL its impact is even greater.

6.3. Out-of-Sample Performance

Setup For out-of-sample analysis, each method is tested on each of the five folds of the data sets with the other folds as training data. For CART, OSRT, SRT-C, SRT-SL, and SRT-L we hyper-tune the regularization parameter λ using five-fold cross-validation. For GUIDE, GUIDE-SL, GUIDE-L, IAI, and IAI-L, we use the default hyper-tuning as included in these methods. Furthermore, SRT-L and IAI-L tune the Lasso and Ridge penalization (if applicable) in a second hyper-tuning phase, as explained in (Dunn, 2018). The piecewise constant and linear methods are trained with a maximum depth of five and four respectively. The MIP models ORT, ORT-L, and DTIP resulted in time-outs for almost every scenario even without hyper-tuning, and are therefore left out of the results. We show training scores at time-out (without hyper-tuning) for the MIP methods and all other methods in Appendix C. Every method, except LR, CART, SRT-C, and SRT-SL resulted in time-outs or out-of-memory errors for the Household data set.

Results Table 2 shows that the out-of-sample coefficients of determination (R^2) for all piecewise constant methods are close, with each method scoring best for at least one data set. SRT-C most often performs best. With a 5% significance level, Wilcoxon signed rank tests show that SRT-C performs better than GUIDE and equal to IAI, ORST, and CART. OSRT results in time-outs for several data sets.

Table 3 shows the out-of-sample R^2-scores for all piecewise linear methods. For comparison, the results of linear regression are also included. Since IAI-L uses lasso penalization, we compare it with both SRT-L using only lasso regularization and also elastic net regularization.

All methods perform significantly better than standard linear regression. SRT-SL, with its simple linear regression models, performs best for most data sets. The runner-up is SRT-L (with either regularization technique). Both SRT-SL and SRT-L perform significantly better than GUIDE-SL and GUIDE-L. The other differences are not significant.

Binarization and optimality To show the effect of preprocessing the feature data for possible binary tree splits and the difference between optimal and non-optimal methods, we also compare the training R^2 score of SRT-C with that of IAI and CART when the latter are trained with the original numerical and with the binarized data. Fig. 5 shows that SRT-C, despite losing information in the binarization, obtains approximately the same training R^2 score as IAI and outperforms IAI and CART clearly when those methods are also trained with binary features.

Discussion Our analysis shows that our optimal methods have better out-of-sample generalization than previous heuristics, although the differences are small. The optimal methods guarantee optimality on the training data for

Table 3. Out-of-sample R^2 results for the piecewise linear methods with $d = 4$. Time-outs indicated by '-'. The best results are marked in bold. Optimal methods are marked with an *.

		Simple Regression		Multiple Regression			
Data set	LR	GUIDE-SL	SRT-SL*	GUIDE-L	IAI-L	SRT-L* (Lasso)	SRT-L* (Elastic Net)
Airfoil	0.51	0.65	0.72	0.85	0.88	**0.89**	-
Auction	0.38	0.89	**0.96**	0.94	0.94	0.94	0.94
Auto MPG	0.82	0.83	**0.84**	**0.84**	**0.84**	**0.84**	**0.84**
Energy (C)	0.89	0.95	**0.97**	**0.97**	**0.97**	**0.97**	**0.97**
Energy (H)	0.92	0.98	**1.00**	0.99	0.99	0.99	0.99
Household	**1.00**	-	**1.00**	-	-	-	-
Optical Net.	0.29	0.15	**0.96**	0.20	0.59	0.73	0.77
Real Estate	0.58	0.59	0.58	0.62	**0.63**	**0.63**	**0.63**
Seoul Bike	0.56	0.69	-	**0.79**	-	-	-
Servo	0.45	-0.05	**0.64**	-0.03	0.50	0.49	0.53
Synch.	**1.00**	0.96	**1.00**	0.95	**1.00**	**1.00**	**1.00**
Yacht	0.63	**0.99**	**0.99**	0.98	0.98	**0.99**	**0.99**
Best per category		3	**10**	5	7	8	**9**
Best overall	2	1	**9**	3	4	6	5

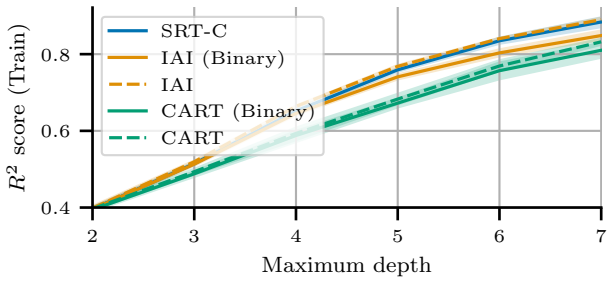

Figure 5. Training R^2 score for Airfoil. SRT-C (with binary features) performs similarly to IAI with original features, better than IAI with binary features, and better than CART in both cases

a given size limit and binarization. However, the choice of binarization and tree-size hypertuning techniques have not been studied well enough yet in the literature, which prevents full utilization of the strength of optimal methods.

Though our method SRT-C obtains the same optimal solutions as OSRT by Zhang et al. (2023), our conclusion is more cautious than theirs, when they conclude that OSRT outperforms CART, GUIDE, and IAI in out-of-sample performance. Differences that may explain this are: (1) they compare all methods on the binarized data, whereas we train methods with the original numerical data if possible; (2) for most data sets, they binarize numerical features into four binary features, instead of at most 10, as we do; (3) their setup for GUIDE mistakenly reuses the train predictions to assess the test performance, which explains why they report a poor performance for GUIDE; and (4) they do not test the statistical significance of their results.

7. Conclusion

We present three new dynamic programming (DP) approaches for regression trees: a new algorithm for computing piecewise constant regression trees that improves scalability using a special algorithm for depth-two trees; and the first optimal DP methods for piecewise multiple and simple linear regression trees. For piecewise simple linear regression trees, we also provide a special algorithm for depth-two trees. The results show that our methods improve scalability by one or more orders of magnitude in comparison to the state-of-the-art, and mainly so because of our depth-two algorithms. The out-of-sample performance of our methods is on par with or better than the state-of-the-art.

Complexity-tuning techniques should be further researched to fully exploit the power of optimal regression trees. Other extensions include dealing with non-binary features for the branching decisions, as done for example in (Mazumder et al., 2022), and further exploiting the subtree independence through parallelization of the algorithm.

Impact Statement

This paper advances the use of regression trees. There are many potential societal consequences of our work, none of which we feel must be specifically highlighted here.

References

Aghaei, S., Gómez, A., and Vayanos, P. Strong Optimal Classification Trees. *arXiv preprint arXiv:2103.15965*, 2021.

Aglin, G., Nijssen, S., and Schaus, P. Learning Optimal Decision Trees Using Caching Branch-and-Bound Search. In *Proceedings of AAAI-20*, pp. 3146–3153, 2020a.

Aglin, G., Nijssen, S., and Schaus, P. PyDL8.5: a Library for Learning Optimal Decision Trees. In *Proceedings of IJCAI-20*, pp. 5222–5224, 2020b.

Akay, M. Optical Interconnection Network . UCI Machine Learning Repository, 2018.

Bertsimas, D. and Dunn, J. Optimal classification trees. *Machine Learning*, 106(7):1039–1082, 2017.

Bertsimas, D., Dunn, J., and Paschalidis, A. Regression and Classification using Optimal Decision Trees. In *2017 IEEE MIT undergraduate research technology conference*, pp. 1–4, 2017.

Bertsimas, D., Dunn, J., and Wang, Y. Near-optimal Nonlinear Regression Trees. *Operations Research Letters*, 49 (2):201–206, 2021.

Blanquero, R., Carrizosa, E., Molero-Río, C., and Morales, D. R. On Sparse Optimal Regression Trees. *European Journal of Operational Research*, 299(3):1045–1054, 2022.

Breiman, L., Friedman, J. H., Olshen, R. A., and Stone, C. J. *Classification and Regression Trees*. Wadsworth and Brooks, Monterey, CA, 1984.

Brooks, T., Pope, D., and Marcolini, M. Airfoil Self-Noise. UCI Machine Learning Repository, 2014.

Carrizosa, E., Molero-Río, C., and Morales, D. R. Mathematical optimization in classification and regression trees. *TOP: An Official Journal of the Spanish Society of Statistics and Operations Research*, 29(1):5–33, 2021.

De'ath, G. and Fabricius, K. E. Classification and Regression Trees: A Powerful yet Simple Technique for Ecological Data Analysis. *Ecology*, 81(11):3178–3192, 2000.

Demirović, E., Lukina, A., Hebrard, E., Chan, J., Bailey, J., Leckie, C., Ramamohanarao, K., and Stuckey, P. J. MurTree: Optimal Classification Trees via Dynamic Programming and Search. *Journal of Machine Learning Research*, 23(26):1–47, 2022.

Dua, D. and Graff, C. UCI Machine Learning Repository, 2017. URL http://archive.ics.uci.edu/ml.

Dunn, J. W. *Optimal Trees for Prediction and Prescription*. PhD thesis, Massachusetts Institute of Technology, 2018.

Freitas, A. A. Comprehensible Classification Models – a position paper. *ACM SIGKDD Explorations Newsletter*, 15(1):1–10, 2014.

Friedman, J., Hastie, T., and Tibshirani, R. Regularization Paths for Generalized Linear Models via Coordinate Descent. *Journal of Statistical Software*, 33(1):1–22, 2010.

Friedman, J. H. Multivariate adaptive regression splines. *The Annals of Statistics*, 19(1):1–67, 1991.

Gerritsma, J., Onnink, R., and Versluis, A. Yacht Hydrodynamics. UCI Machine Learning Repository, 2013.

Grubinger, T., Zeileis, A., and Pfeiffer, K.-P. evtree: Evolutionary Learning of Globally Optimal Classification and Regression Trees in R. *Journal of Statistical Software*, 61 (1):1–29, 2014.

Gurobi Optimization LLC. Gurobi Optimizer Reference Manual, 2022. URL https://www.gurobi.com.

Hebrail, G. and Berard, A. Individual household electric power consumption. UCI Machine Learning Repository, 2012.

Hemmateenejad, B., Shamsipur, M., Zare-Shahabadi, V., and Akhond, M. Building optimal regression tree by ant colony system–genetic algorithm: Application to modeling of melting points. *Analytica Chimica Acta*, 704: 57–62, 2011.

Hu, H., Siala, M., Hebrard, E., and Huguet, M.-J. Learning Optimal Decision Trees with MaxSAT and its Integration in AdaBoost. In *IJCAI-PRICAI 2020*, pp. 1170–1176, 2020.

Hu, X., Rudin, C., and Seltzer, M. Optimal Sparse Decision Trees. In *Advances in NeurIPS-19*, pp. 7267–7275, 2019.

Huisman, T., Van der Linden, J. G. M., and Demirović, E. Optimal Survival Trees: A Dynamic Programming Approach. In *Proceedings of AAAI-24*, 2024.

Hyafil, L. and Rivest, R. L. Constructing optimal binary decision trees is NP-complete. *Information processing letters*, 5(1):15–17, 1976.

Interpretable AI. Interpretable AI Documentation, 2023. URL https://www.interpretable.ai/.

Janota, M. and Morgado, A. SAT-Based Encodings for Optimal Decision Trees with Explicit Paths. In *Proceedings of the International Conference on Theory and Applications of Satisfiability Testing (SAT 2020)*, pp. 501–518, 2020.

King, M. W. and Resick, P. A. Data Mining in Psychological Treatment Research: A Primer on Classification and Regression Trees. *Journal of Consulting and Clinical Psychology*, 82(5):895–905, 2014.

Kingma, D. P. and Ba, J. Adam: A Method for Stochastic Optimization. In *International Conference on Learning Representations*, 2015.

Lin, J., Zhong, C., Hu, D., Rudin, C., and Seltzer, M. Generalized and Scalable Optimal Sparse Decision Trees. In *Proceedings of ICML-20*, pp. 6150–6160, 2020.

Van der Linden, J. G. M., De Weerdt, M. M., and Demirović, E. Fair and Optimal Decision Trees: A Dynamic Programming Approach. In *Advances in NeurIPS-22*, pp. 38899–38911, 2022.

Van der Linden, J. G. M., De Weerdt, M. M., and Demirović, E. Necessary and Sufficient Conditions for Optimal Decision Trees using Dynamic Programming. In *Advances in NeurIPS-23*, 2023.

Loh, W.-Y. Regression Trees with Unbiased Variable Selection and Interaction Detection. *Statistica Sinica*, 12(2): 361–386, 2002.

Loh, W.-Y. Fifty Years of Classification and Regression Trees. *International Statistical Review*, 82(3):329–348, 2014.

Mazumder, R., Meng, X., and Wang, H. Quant-BnB: A Scalable Branch-and-Bound Method for Optimal Decision Trees with Continuous Features. In *ICML-22*, pp. 15255–15277, 2022.

Morgan, J. N. and Sonquist, J. A. Problems in the Analysis of Survey Data, and a Proposal. *Journal of the American Statistical Association*, 58(302):415–434, 1963.

Narodytska, N., Ignatiev, A., Pereira, F., and Marques-Silva, J. Learning Optimal Decision Trees with SAT. In *Proceedings of IJCAI-18*, pp. 1362–1368, 2018.

Nijssen, S. and Fromont, E. Mining Optimal Decision Trees from Itemset Lattices. In *Proceedings of SIGKDD-07*, pp. 530–539, 2007.

Ordoni, E., Bach, J., and Fleck, A.-K. Auction Verification. UCI Machine Learning Repository, 2022.

Piltaver, R., Luštrek, M., Gams, M., and Martinčić-Ipšić, S. What makes classification trees comprehensible? *Expert Systems with Applications*, 62:333–346, 2016.

Quinlan, J. R. Learning with continuous classes. In *Proceedings of the 5th Australian Joint Conference on Artificial Intelligence*, pp. 343–348, 1992.

Quinlan, R. Auto MPG. UCI Machine Learning Repository, 1993.

Rudin, C. Stop explaining black box machine learning models for high stakes decisions and use interpretable models instead. *Nature Machine Intelligence*, 1(5):206–215, 2019.

Schidler, A. and Szeider, S. SAT-based Decision Tree Learning for Large Data Sets. In *Proceedings AAAI-21*, pp. 3904–3912, 2021.

Song, M. and Zhong, H. Efficient weighted univariate clustering maps outstanding dysregulated genomic zones in human cancers. *Bioinformatics*, 36(20):5027–5036, 2020.

Tsanas, A. and Xifara, A. Energy efficiency. UCI Machine Learning Repository, 2012.

Ulrich, K. Servo. UCI Machine Learning Repository, 1993.

Verhaeghe, H., Nijssen, S., Pesant, G., Quimper, C.-G., and Schaus, P. Learning Optimal Decision Trees using Constraint Programming. *Constraints*, 25(3):226–250, 2020.

Verwer, S. and Zhang, Y. Learning decision trees with flexible constraints and objectives using integer optimization. In *Proceedings of CPAIOR-17*, pp. 94–103, 2017.

Verwer, S. and Zhang, Y. Learning Optimal Classification Trees Using a Binary Linear Program Formulation. In *Proceedings of AAAI-19*, pp. 1625–1632, 2019.

Wang, H. and Song, M. Ckmeans.1d.dp: optimal k-means clustering in one dimension by dynamic programming. *The R Journal*, 3(2):29–33, 2011.

Wang, Y. and Witten, I. H. Induction of Model Trees for Predicting Continuous Classes. In *Proceedings of the European Conference on Machine Learning*, 1997.

Yang, L., Liu, S., Tsoka, S., and Papageorgiou, L. G. Mathematical Programming for Piecewise Linear Regression Analysis. *Expert Systems with Applications*, 44:156–167, 2016.

Yang, L., Liu, S., Tsoka, S., and Papageorgiou, L. G. A regression tree approach using mathematical programming. *Expert Systems with Applications*, 78:347–357, 2017.

Zhang, R., Xin, R., Seltzer, M., and Rudin, C. Optimal Sparse Regression Trees. In *Proceedings of AAAI-23*, pp. 11270–11279, 2023.

Zhu, H., Murali, P., Phan, D. T., Nguyen, L. M., and Kalagnanam, J. R. A Scalable MIP-based Method for Learning Optimal Multivariate Decision Trees. In *Advances in NeurIPS-20*, pp. 1771–1781, 2020.

A. Pseudocode

Algorithm 2 shows the full pseudocode for SRT. It first checks if the data set size is at least the minimum leaf node size. If the depth budget d or node budget n is zero, it returns the SSE of a leaf node using either Eq. (4), (8), or (9), for SRT-C, SRT-SL, and SRT-L respectively. If the depth or node budget exceeds what is possible according to the other budget, the budgets are updated and SRT is called again. If the regularization costs λ multiplied by the node budget exceeds the upper bound, the node budget is updated. SRT then checks the cache to see if the subproblem has been encountered before, and if so, returns it. If the depth budget is at most two, SRT uses the special depth-two algorithm as explained in Algorithm 1 (not for SRT-L).

In all other cases, SRT will consider all possible branching decisions and node budget divisions. If a split does not respect the minimum leaf node size, it is skipped. Otherwise, lower bounds are obtained (from cache or using the lower bounds specified in the main paper). If the lower bound for a split exceeds the upper bound, the split is skipped. Otherwise, a left and right subtree are generated, while updating the upper bound by subtracting the lower bound and the regularization parameter from the current upper bound. If the combination of the two solutions yields a better solution, the current best solution is replaced. Finally, if a solution below the upper bound has been found, it is stored in the cache. Otherwise, the upper bound is stored as a lower bound in the cache.

Algorithm 2: Pseudo-code for SRT given a data set $\mathcal{D}$, feature sets $\mathcal{F}$ and $\mathcal{F}_b$, a depth budget d, a node budget n, an upper bound ub, a complexity cost λ and a minimum leaf node size M.

```
SRT(D, d, n, ub)
    if |D| < M then return ∞
    if d = 0 ∨ n = 0 then
        s ← solveLeaf(D)
        if s ≥ ub then return ∞
        return s
    if n > 2^d − 1 then return SRT(D, d, 2^d − 1, ub)
    if d > n then return SRT(D, d, d, ub)
    if nλ > ub then return SRT(D, d, ⌊ub/λ⌋)
    ⟨s, lb, stat⟩ ← cache[D, d, n]
    if lb ≥ ub then return ∞
    if stat = optimal then return s
    if d ≤ 2 then s ← SolveD2(D, d, n)
    else
        s ← solveLeaf(D)
        if s ≥ ub then s ← ∞
        for f ∈ F_b, n_L ∈ [0, n − 1] do
            if |D_f| < M ∨ |D_f̄| < M then continue
            n_R ← n − n_L − 1
            lb_L ← LB(D_f̄, d − 1, n_L)
            lb_R ← LB(D_f, d − 1, n_R)
            lb = lb_L + lb_R + λ
            if lb ≥ ub then continue
            ub_L ← ub − lb_R − λ
            s_L ← SRT(D_f̄, d − 1, n_L)
            if s_L = ∞ then continue
            ub_R ← ub − s_L − λ
            s_R ← SRT(D_f, d − 1, n_R)
            if s_R = ∞ then continue
            if s_L + s_R + λ < s then s ← s_L + s_R + λ
    if s ≠ ∞ then cache[D, d, n] ← ⟨s, s, optimal⟩
    else cache[D, d, n] ← ⟨∞, ub, lowerbound⟩
    return s
```

B. Simple Linear Regression Derivation

The error for simple linear regression, including the Ridge L2-norm penalization, is as follows:

$$\text{SSE} = \sum_{i=1}^{n}(y_i - \hat{\beta}_0 - \hat{\beta}x_i)^2 + \gamma\hat{\beta}^2 \tag{14}$$

The optimal value for the intercept $\hat{\beta}_0$ is found by setting the derivative to zero:

$$\frac{\partial\,\text{SSE}}{\partial\hat{\beta}_0} = \sum_{i=1}^{n}(-2y_i + 2\hat{\beta}_0 + 2\hat{\beta}x_i) \tag{15}$$

$$0 = \sum_{i=1}^{n}(-y_i + \hat{\beta}_0 + \hat{\beta}x_i) \tag{16}$$

$$\hat{\beta}_0 = \frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{\beta}x_i) \tag{17}$$

$$\hat{\beta}_0 = \bar{y} - \hat{\beta}\bar{x} \tag{18}$$

The optimal value for the slope $\hat{\beta}$ is also found by setting the derivative to zero, and by filling in Eq. (18) for $\hat{\beta}_0$:

$$\frac{\partial\,\text{SSE}}{\partial\hat{\beta}} = \sum_{i=1}^{n}(-2x_iy_i + 2\hat{\beta}_0x_i + 2\hat{\beta}x_i^2) + 2\gamma\hat{\beta} \tag{19}$$

$$0 = \sum_{i=1}^{n}(-x_iy_i + \hat{\beta}_0x_i + \hat{\beta}x_i^2) + \gamma\hat{\beta} \tag{20}$$

$$= \sum_{i=1}^{n}\left(-x_iy_i + \left(\bar{y} - \hat{\beta}\bar{x}\right)x_i + \hat{\beta}x_i^2\right) + \gamma\hat{\beta} \tag{21}$$

$$\hat{\beta} = \frac{\sum_{i=1}^{n}(x_iy_i - \bar{y}x_i)}{\sum_{i=1}^{n}(x_i^2 - \bar{x}x_i) + \gamma} \tag{22}$$

By observing that $\bar{y} = \sum_i y_i/n$ and $\bar{x} = \sum_i x_i/n$, we can rewrite to Eq. (10) and (11):

$$\hat{\beta}_0 = \sum_{i=1}^{n} y_i/n + \hat{\beta} \sum_{i=1}^{n} x_i/n \tag{23}$$

$$\hat{\beta} = \frac{n \sum_{i=1}^{n} x_i y_i - \sum_{i=1}^{n} y_i \sum_{i=1}^{n} x_i}{n \sum_{i=1}^{n} x_i^2 - (\sum_{i=1}^{n} x_i)^2 + n\gamma} \tag{24}$$

We set the ridge penalty $\gamma = \sigma^2 \gamma'$, with σ^2 the variance of the feature vector x, and γ' a hyper-parameter. We tune for $\gamma' \in \{0, 0.01, 0.1, 1, 10, 100, 1000\}$.

C. Training Results

Tables 4 and 5 show the training R^2-scores for all methods. We train each method on the five folds of each data set and report the average training score. When at least one run resulted in a time-out ($> 900s$), we mark it with an asterisk and report the training score at time-out.

MIP results The results show that (even without hyper-tuning) DTIP never finishes before the time-out and ORT and ORT-L only twice and four times respectively. Furthermore, they never exceed the training score of SRT-C and SRT-L.

To keep the objective linear, we train DTIP, ORT, and ORT-L by minimizing the mean absolute error. We have also tested minimizing the quadratic mean squared error, but this yielded even larger run times. This difference can explain why ORT, when it obtains the optimal solution, is still worse than SRT-C.

Because the MIP methods time out in almost every case even without hyper-tuning, we did not include these methods in our evaluation in the main text.

GUIDE is a greedy algorithm like CART but differs in a few ways: it creates unbiased splits, it can group multiple values for categorical variables in one branch, and it uses significance tests for each split. These significance tests prevent overfitting and also explain why GUIDE's training accuracy is typically lower than CART's in Table 4. GUIDE-SL is always worse than SRT-SL in Table 5. GUIDE-L is best for one data set, and similar or worse for all others.

IAI IAI and IAI-L do not provide a time limit parameter. Therefore, when these methods exceeded the time limit for the household data set, we reran it with a reduced number of random restarts until the observed run time was only just above the time limit. For the household data set, we ran IAI with 18 random restarts and IAI-L with 15 random restarts. For the other data sets, we use the default 100 random restarts.

The training performance of IAI and SRT-C are close. On four data sets, IAI's R^2 is marginally better because it uses the numerical feature data directly instead of the binarized features. On three data sets, SRT-C is marginally better than IAI because its exhaustive search considers trees that IAI's local search does not find.

Similarly, for multiple linear regression, the performance is close. IAI-L is marginally better for one data set. SRT-L is marginally better for one data set and significantly better for two others.

OSRT The two DP methods OSRT and SRT-C obtain as expected the same training score. The only differences are for the household data set, where OSRT runs out of memory, and the Seoul-bike data set, where OSRT does not find the optimal solution within the time limit but SRT-C does.

Table 4. Training R^2 scores (without hyper-tuning) for trees with constant predictors at $d = 5$. An * indicates this is the best result obtained at the time-out (900s). The best results per category are shown in bold. OoM indicates out of memory.

	Heuristic			Optimal MIP		Optimal DP	
Data set	CART	GUIDE	IAI	DTIP	ORT	OSRT	SRT-C
Airfoil	0.68	0.67	**0.77**	0.40 *	0.56 *	0.76	0.76
Auction	0.96	0.91	**0.97**	0.14 *	-0.49	**0.97**	**0.97**
Auto MPG	0.93	0.91	**0.94**	0.91 *	0.88 *	**0.94**	**0.94**
Energy (C)	0.96	0.96	0.97	0.94 *	0.93 *	**0.98**	**0.98**
Energy (H)	0.99	0.98	0.99	0.97 *	0.96 *	**1.00**	**1.00**
Household	**1.00**	OoM	**1.00** *	OoM	OoM	OoM	0.98
Optical Net.	**1.00**	0.80	**1.00**	**1.00** *	0.96 *	**1.00**	**1.00**
Real Estate	0.86	0.76	**0.89**	0.80 *	0.78 *	0.88	0.88
Seoul Bike	0.73	0.63	0.75	-0.08 *	-1.19	0.24 *	**0.76**
Servo	0.96	0.12	**0.99**	0.98 *	0.93 *	**0.99**	**0.99**
Synch.	**1.00**	**1.00**	**1.00**	0.99 *	0.99 *	0.99	0.99
Yacht	**1.00**	**1.00**	**1.00**	**1.00** *	0.99 *	**1.00**	**1.00**
Best	4	2	9	2	0	7	8

Table 5. Training R^2 scores (without hyper-tuning) for trees with linear predictors at $d = 4$. An * indicates this is the best result obtained at the time-out (900s). The best results per category are shown in bold. OoM indicates out of memory.

	Simple linear regression		Multiple linear regression			
	Heuristic	Optimal DP	Heuristic		Optimal MIP	Optimal DP
Data set	GUIDE-SL	SRT-SL	GUIDE-L	IAI-L	ORT-L	SRT-L
Airfoil	0.69	**0.80**	0.86	**0.92**	0.79 *	**0.92**
Auction	0.91	**0.96**	0.94	**0.95**	-0.50	**0.95**
Auto MPG	0.91	**0.93**	0.87	**0.90**	0.89 *	**0.90**
Energy (C)	0.96	**0.98**	**0.97**	**0.97**	**0.97** *	**0.97**
Energy (H)	0.98	**1.00**	**0.99**	**0.99**	0.98 *	**0.99**
Household	OoM	**1.00**	OoM	**1.00** *	OoM	**1.00** *
Optical Net.	0.68	**0.98**	0.27	0.11	-0.01	**0.90**
Real Estate	0.76	**0.85**	0.68	**0.75**	0.70 *	**0.75**
Seoul Bike	0.69	**0.76**	**0.80**	0.77	-0.97 *	0.76 *
Servo	0.10	**0.87**	0.04	0.75	0.34	**0.76**
Synch.	0.97	**1.00**	0.95	**1.00**	**1.00**	**1.00**
Yacht	0.99	**1.00**	**0.99**	0.91	0.91 *	**0.99**
Best per category	0	12	4	8	2	11