The Backbone Method for Ultra-High Dimensional Sparse Machine Learning

Dimitris Bertsimas · Vassilis Digalakis Jr.

First submission: 06/2020. Revised and resubmitted to Machine Learning: 04/2021.

Abstract We present the backbone method, a generic framework that enables sparse and interpretable supervised machine learning methods to scale to ultra-high dimensional problems. We solve sparse regression problems with 10^7 features in minutes and 10^8 features in hours, as well as decision tree problems with 10^5 features in minutes. The proposed method operates in two phases; we first determine the backbone set, that consists of potentially relevant features, by solving a number of tractable subproblems; then, we solve a reduced problem, considering only the backbone features. For the sparse regression problem, we show that, under certain assumptions and with high probability, the backbone set consists of the true relevant features. Numerical experiments on both synthetic and real-world datasets demonstrate that our method outperforms or competes with state-of-the-art methods in ultra-high dimensional problems, and competes with optimal solutions in problems where exact methods scale, both in terms of recovering the true relevant features and in its out-of-sample predictive performance.

Keywords Ultra-high dimensional machine learning · Sparse machine learning · Mixed integer optimization · Sparse regression · Decision trees

1 Introduction

In the Big Data era, the scalability of machine learning models is constantly challenged. *Ultra-high dimensional* datasets are present in a variety of applications, ranging from physical sciences and engineering to social sciences and medicine. From a practical standpoint, we consider a dataset to be ultra-high dimensional

Dimitris Bertsimas
Operations Research Center and Sloan School of Management, Massachusetts Institute of Technology, Cambridge, MA 02139, USA
dbertsim@mit.edu

Vassilis Digalakis Jr.
Operations Research Center, Massachusetts Institute of Technology, Cambridge, MA 02139, USA
vvdig@mit.edu

when the number of features p exceeds the number of data points n and is at least two orders of magnitude greater than the known scalability limit of the learning task at hand; as we discuss in the sequel, this limit is $p \sim 10^5$ for the sparse regression algorithm that we use, and $p \sim 10^3$ for the decision tree induction algorithm. From a theoretical standpoint, we assume that $\log p = O(n^\xi)$ for some $\xi \in (0, 1)$.

A commonly held assumption to address high dimensional regimes is that the underlying model is *sparse*, that is, among all input features, only $k < n \ll p$ (e.g., $k \leq 100$) are relevant for the task at hand (Hastie et al., 2015). For instance, in the context of regression, the sparsity assumption implies that only a few regressors are set to nonzero level. In the context of decision trees, it means that most features do not appear in any split node. The sparsity requirement can either be explicitly imposed via a cardinality constraint or penalty, as in sparse regression (also known as best subset selection), or can implicitly be imposed by the structure of the model, as in depth-constrained decision trees, where the number of features that we split on is upper bounded as function of the tree depth. In general, sparsity is a desirable property for machine learning models, as it makes them more interpretable and often improves their generalization ability (Ng, 1998).

Learning a *sparse machine learning model* can naturally be formulated as a *mixed integer optimization* (MIO) problem by associating each feature with an auxiliary binary decision variable that is set to 1 if and only if the feature is identified as relevant (Bertsimas et al., 2016). Despite the -theoretical- computational hardness of solving MIO problems, the remarkable progress in the field has motivated their use in a variety of machine learning problems, ranging from sparse regression and decision trees to principal component analysis and matrix completion (Bertsimas and Dunn, 2019). The rich modeling framework of MIO enables us to directly model the problem at hand and often compute provably optimal or near-optimal solutions using either exact methods (e.g., branch and bound) or tailored heuristics (e.g., local search). Moreover, the computational efficiency and scalability of MIO-based methods is far beyond what one would imagine a decade ago. For example, we are able to solve sparse regression problems with $p \sim 10^5$ features (Bertsimas and Van Parys, 2020) and induce decision trees for problems with $p \sim 10^3$ features (Bertsimas and Dunn, 2017). Nevertheless, MIO-based formulations are still *challenged in ultra-high dimensional regimes* as the ones we examine.

The *standard way of addressing ultra-high dimensional problems* is to perform either a screening (i.e., filtering) step (Fan and Lv, 2008), that eliminates a large portion of the features, or more sophisticated dimensionality reduction methods: **(a)** feature selection methods, that aim to select a subset of relevant features, or **(b)** feature extraction methods, that project the original features to a new, lower-dimensional feature space; see Guyon and Elisseeff (2003); Li et al. (2017) for detailed reviews. After the dimensionality reduction step, we can solve the MIO formulation for the reduced problem, provided that sufficiently many features have been discarded. However, commonly used screening (or, more generally, feature selection) approaches are heuristic and often lead to suboptimal solutions, whereas feature extraction approaches are uninterpretable in terms of the original problem's features.

In this paper, we introduce the *backbone method for sparse supervised learning*, a two-phase framework that, as we empirically show, scales to ultra-high dimensions (as defined earlier) and provides significantly higher quality solutions than screening

approaches, without sacrificing interpretability (as opposed to feature extraction methods). In the first phase of our proposed method, we aim to identify the "backbone" of the problem, which consists of features that are likely to be part of an optimal solution. We do so by collecting the solutions to a series of subproblems that are more tractable than the original problem and whose solutions are likely to contain relevant features. These subproblems also need to be "selective," in the sense that only the fittest features survive after solving each of them. In the second phase, we solve the MIO formulation considering only the features that have been included in the backbone. Although our framework is generic and can be applied to a large variety of problems, we focus on two particular problems, namely sparse regression and decision trees, and illustrate how the backbone method can be applied to them. We accurately solve sparse regression problems with 10^7 features in minutes and 10^8 features in hours, as well as decision tree problems with 10^5 features in minutes, that is, two orders of magnitude larger than the known scalability limit of methods that solve to optimality or near-optimality the actual MIO formulation for each problem.

Our proposed backbone method was inspired by a large-scale vehicle routing application; using a backbone algorithm, Bertsimas et al. (2019) solve, in seconds, problems with thousands of taxis serving tens of thousands of customers per hour. In this work, we develop the backbone method for sparse supervised learning in full generality. Moreover, we remark that the method can be applied to a variety of machine learning problems that exhibit sparse structure, even beyond supervised learning. For example, in the clustering problem, a big portion of pairs of data points will never be clustered together in any near optimal cluster assignment and hence we need not consider assigning them to the same cluster. Finally, we note that, within the sparse supervised learning framework that we examine in this paper, the backbone method can also be used as a feature selection technique in combination with *any sparsity-imposing but not necessarily MIO-based method.*

1.1 Advances in Sparse Regression

In this section, we briefly review the landscape of the sparse regression problem, which is one of the two problems that we tackle using the backbone method. Formally, a regression problem is considered *high dimensional* when the rank of the design matrix $\boldsymbol{X}$ is smaller than the number of features p, i.e., $\text{rank}(\boldsymbol{X}) < p$; this is, for example, the case when n, the number of data points in the dataset, satisfies $n < p$. In such regimes, the regression model is challenged, as the underlying linear system is underdetermined, and further assumptions are required. The application of sparse or sparsity-inducing methods provides a way around this limitation and, in addition, hopefully leads to more interpretable models, where only a few features actually affect the prediction.

Exact Sparse Regression Formulation. The sparse regression problem with an explicit sparsity constraint, also known as the best subset selection problem, is the most natural way of performing simultaneous feature selection and model fitting for regression. The sparsity constraint is imposed by requiring that the ℓ_0 (pseudo)norm of the vector of regressors $\boldsymbol{w} \in \mathbb{R}^p$ is less than a predetermined degree of sparsity $k \ll p$, namely, $\|\boldsymbol{w}\|_0 \leq k$, where $\|\boldsymbol{w}\|_0 = \sum_{j=1}^{p} \mathbf{1}_{(w_j \neq 0)}$ and

$\mathbf{1}_{(\cdot)}$ denotes the indicator function. However, the ℓ_0 constraint is very far from being convex and the resulting combinatorial optimization problem is NP-hard (Natarajan, 1995).

Motivated by the advances in MIO and despite the diagnosed hardness of the problem, a recent line of work solves the problem exactly to optimality or near-optimality (i.e., within a user-specified optimality gap that can be set to an arbitrarily small value up to machine precision). Bertsimas et al. (2016) cast best subset selection as a mixed integer quadratic program and solve it with a commercial solver for problems with $p \sim 10^3$. Bertsimas and Van Parys (2020) reformulate the sparse regression problem as a pure binary optimization problem and use a cutting planes-type algorithm that, based on the outer approximation method of Duran and Grossmann (1986), iteratively tightens a piece-wise linear lower approximation of the objective function. By doing so, they solve to optimality sparse regression problems with $p \sim 10^5$. Bertsimas et al. (2021) consider a variant of the sparse regression problem, namely, slowly varying sparse regression, and develop a highly optimized version of the outer approximation method by utilizing a novel convex relaxation of the objective function; their technique can be directly applied to the standard sparse regression problem and scale it to larger instances. Hazimeh et al. (2020) develop a specialized, nonlinear branch and bound framework that exploits the structure of the sparse regression problem (e.g., they use tailored heuristics to solve node relaxations) and are able to solve problems with $p \sim 8 \cdot 10^6$. In our work, we use the solution method by Bertsimas and Van Parys (2020), but we remark that the method by Hazimeh et al. (2020) can also be used in combination with our proposed framework.

From a practical point of view, one argument commonly used against best subset selection is that, in real-world problems, the actual support size k is not known and needs to be thoroughly cross-validated hence resulting in a dramatic increase in the required computational effort. Although, in many cases, k is determined by the application, Kenney et al. (2018) address such concerns by proposing efficient cross validation strategies. In this paper, we assume that k is known and given; the combination of efficient cross validation procedures with our proposed method is straightforward.

Heuristic Solution Methods and Surrogate Formulations. Traditionally, the exact sparse regression formulation has been addressed via heuristic and greedy procedures (dating back to Beale et al. (1967); Efroymson (1966); Hocking and Leslie (1967)). Recently, numerous more sophisticated methods have been proposed, including the boolean relaxation of Pilanci et al. (2015), the first-order method of Bertsimas et al. (2016), the subgradient method of Bertsimas and Van Parys (2020) and Bertsimas et al. (2020), the method of Hazimeh and Mazumder (2020) that combines coordinate descent with local search, and the approach of (Boyd et al., 2011) that is based on the alternating direction method of multipliers.

A lot of effort has been dedicated to developing and solving surrogate problems; the most studied of them all is the lasso formulation (Tibshirani, 1996), whereby the non-convex ℓ_0 norm is replaced by the convex ℓ_1 norm. Namely, denoting by $\|\boldsymbol{w}\|_1 = \sum_{j=1}^{p} |w_j|$, we require that $\|\boldsymbol{w}\|_1 \leq t$ or add a penalty term $\lambda\|\boldsymbol{w}\|_1$ in the objective. Due to convexity, the constrained and the penalized versions can be shown to be equivalent for properly selected values of t and λ; this is not the case if the ℓ_0 norm is used. Due to the geometry of the unit ℓ_1 ball, the resulting

formulations indeed shrink the coefficients toward zero and produce sparse solutions by setting many coefficients to be exactly zero. There has been a substantial amount of algorithmic work on the lasso formulation (Efron et al., 2004; Beck and Teboulle, 2009; Friedman et al., 2010) and its many variants (e.g., the elastic net formulation of Zou and Hastie (2005)), on replacing the ℓ_1 norm in the Lasso formulation by other sparsity inducing penalties (e.g., Fan and Li (2001); Zhang (2010)), and on developing scalable implementations (Friedman et al., 2020).

Sparse Regression in Ultra-High Dimensions: Screening. Sure independence screening (SIS) is a two-phase learning framework, introduced by Fan and Lv (2008, 2018). In the first phase, SIS ranks the features based on their marginal utilities; e.g., for linear regression, the proposed marginal utility is the correlation between each feature and the response. By keeping only the highest-ranked features, SIS guarantees that, under certain conditions, all relevant features are selected with high probability. The screening procedure can be implemented iteratively, conditioned on the estimated set of features from the previous step. In the second phase, SIS conducts learning and inference in the reduced feature space consisting only of the selected features, using surrogate formulations such as lasso. Among the various other extensions of SIS, we emphasize the works of Fan et al. (2009) and Fan and Song (2010) that extend SIS to generalized linear models, as well as Fan et al. (2011) and Ni and Fang (2016) that extend SIS beyond the linear model. Atamturk and Gomez (2020) propose safe screening rules for the exact sparse regression formulation, derived from a convex relaxation solution; such rules eliminate features based on guarantees that a feature may or may not be selected in an optimal solution. In this paper, we aim to address substantially bigger problems, so our proposed framework does not require solving a convex relaxation of the entire problem.

Sparse Regression in Ultra-High Dimensions: Distributed Approaches. An alternative path to address large scale sparse regression problems is using distributed methods. The ADMM framework (Boyd et al., 2011) can be used to fit, in a distributed fashion, a regression model on vertically partitioned data, provided that the regularizer is separable at the level of the blocks of features; this is, e.g., the case in lasso. The DECO framework of Wang et al. (2016), after arbitrarily partitioning the features, performs a decorrelation step via the singular value decomposition of the design matrix, fits a lasso model in each feature subset, and combines the estimated coefficients are centrally. Several hybrid methods that combine screening with ideas from the distributed literature have also been developed (Yang et al., 2016; Zhou et al., 2014; Song and Liang, 2015).

1.2 Advances in Decision Trees

In this section, we present recent advances in the decision tree problem.

Decision trees are one of the most popular and interpretable methods in machine learning. At a high level, decision trees recursively partition the feature space into disjoint regions and assign to each resulting partition either a label, in the context of classification, or a constant or linear prediction, in the context of regression. The leading work for decision tree methods in classification is the classification and

regression trees framework (CART), proposed by Breiman et al. (1984), which takes a top-down approach to determining the partitions. Briefly, the CART method operates in two phases:

- A top-down induction phase, where, starting from a root node, a split is determined by minimizing an "impurity measure" (e.g., entropy), and then recursively applying the process to each of the resulting child nodes until no more splits are possible.
- A pruning phase, where the learned tree is pruned to penalize more complex structures that might not generalize well.

The tree induction process of CART has several limitations. First, the process is one-step optimal and not overall optimal. Second, it does not directly optimize over the actual objective (e.g., misclassification error). Third, it only considers univariate splits, i.e., splits that involve a single feature. Despite of the aforementioned limitations, CART have been widely used as building blocks in ensemble learning methods based on bagging and boosting; examples include the random subspace method (Ho, 1998), random forest (Breiman, 2001), and, more recently, xgboost (Chen and Guestrin, 2016). Indeed, such methods enhance the performance of the resulting classifier, sacrificing, however, the interpretability of the model.

Bertsimas and Dunn (2017) formulate the decision tree problem using MIO and propose a tailored coordinate descent-based solution method. The resulting optimal trees framework (OT) overcomes many of the limitations of CART (optimization is over the actual objective, multivariate splits are possible), while still leading to highly interpretable models. The key observation that led to OT is that, when learning a decision tree, there is a number of discrete decisions and outcomes we need to consider:

- Whether to split at any node and which feature to split on.
- Which leaf node a point falls into and (for classification problems) whether this point is correctly classified based on its label.

At each branch node, a split of the form $\boldsymbol{a}^T\boldsymbol{x} < b$ is applied. Points that satisfy this constraint follow the left branch of the tree, whereas those that violate the constraint follow the right branch. Each leaf node is assigned a label, and each point is assigned the label of the leaf node into which the point falls. The resulting MIO formulation aims to make the aforementioned decisions in such a way that the linear combination of an error metric (e.g., misclassification error) and a tree complexity measure is minimized. The formulation also includes a number of constraints ensuring that the resulting tree is indeed valid and consistent. The resulting MIO formulation is solved using a tailored coordinate descent approach; the problem is nonconvex, so the solution process is repeated from a variety of starting trees that are generated randomly and, in the end, the one with the lowest overall objective function is selected. A long stream of literature has followed the original work by Bertsimas and Dunn (2017) in trying to optimally or near-optimally solve the decision tree problem, including, e.g., the work by Aghaei et al. (2020), who develop a flow-based MIO formulation.

Decision Trees in Ultra High Dimensions. Decision trees are known to suffer from the curse of dimensionality and to not perform well in the presence of many irrelevant features (Almuallim and Dietterich, 1994). Among the most notable

attempts to scale decision trees to ultra-high dimensional problems is the recent work by Liu and Tsang (2017), who develop a sparse version of the perceptron decision trees framework (Bennett et al., 2000) and solve problems with $p \sim 10^6$ features.

1.3 Backbones in Optimization

Schneider et al. (1996), Walsh and Slaney (2001), and the many references therein use the term backbone of a discrete optimization problem to refer to a set of frozen decision variables that *must be set to one in any optimal or near-optimal solution*. Thus, the backbone can be obtained as the intersection of all optimal solutions to the problem. For example, in the satisfiability decision problem, the backbone of a formula is the set of literals which are true in every model. This definition is fundamentally different from our notion of a backbone and, from a practical perspective, identifying such backbone can be as hard as solving the actual problem. In our approach, the term backbone refers to all variables that *are set to one in at least one near-optimal solution*. Thus, the backbone can be obtained as the union of all near-optimal solutions. To the best of our knowledge, the notion of backbone that we examine in this paper first appeared in Bertsimas et al. (2019) in the context of online vehicle routing.

1.4 Outline & Contributions

Our key contributions can be summarized as follows:

- We develop the backbone method, a novel, two-phase framework that, as we empirically show, performs highly effective feature selection and enables sparse machine learning models to scale to ultra-high dimensions. As mixed integer optimization offers a natural way to model sparsity, we focus on MIO-based machine learning methods. Nonetheless, our framework is developed in full generality and can be applied to any sparsity-inducing machine learning method.
- We apply the backbone method to the sparse regression problem. We show that, under certain assumptions and with high probability, the backbone set consists of the true relevant features. Our computational results on both synthetic and real-world data indicate that the backbone method outperforms or competes with state-of-the-art methods for ultra-high dimensional problems, accurately scales to problems with $p \sim 10^7$ in minutes and problems with $p \sim 10^8$ in hours. In problems with $p \sim 10^5$, where exact methods apply and hence we can compare with near-optimal solutions, the backbone method, by drastically reducing the problem size, achieves, faster (by $15 - 20\%$) and with fewer data points, the same levels of accuracy as exact methods, while providing optimality guarantees, albeit for the reduced problem.
- We apply the backbone method to the decision tree problem. Our computational results indicate that the backbone method scales to problems with $p \sim 10^5$ in minutes, outperforms CART, and competes with random forest, while still outputting a single, interpretable tree. In problems with $p \sim 10^3$, the backbone method can accurately filter the feature set and compute, substantially faster (by $10 - 100$ times), decision trees with comparable out-of-sample performance

to that of the decision tree obtained by applying the optimal trees framework to the entire problem.

The structure of the paper is as follows. In Section 2, we develop the generic framework for the backbone method in a general supervised learning setting; we unfold the components of the method and discuss topics that include hyperparameter selection, termination, complexity, and parallel implementation. In Section 3, we apply the backbone method to the sparse regression problem, discuss several implementation aspects and challenges, and present a theoretical result on the method's accuracy, which can provide guidance in tuning the method's hyperparameters. In Section 4, we apply the backbone method to the decision tree problem and again discuss several implementation aspects and challenges. Section 5 investigates, via experiments on synthetic datasets, the backbone method's scalability, performance in various regimes, and sensitivity to the method's hyperparameters and components. In Section 6, we evaluate the method through computational experiments on real-world datasets. Finally, Section 7 concludes the paper.

2 The Backbone Method

In this section, we describe the backbone method in a general supervised learning context. The main idea is that many real-world ultra-high dimensional machine learning problems are indeed sparse, which means that most features are uninformative for performing regression or classification and hence need not be considered when solving the actual problem's formulation. Therefore, the core of the two-phase backbone method is the construction of the *backbone set*, which consists of features identified as potentially relevant. The two phases of the backbone method are:

1. **Backbone Construction Phase:** Form a series of tractable subproblems that can be solved efficiently while still admitting sparse solutions, by reducing the dimension and/or relaxing the original problem. Construct the backbone set by collecting all decision variables that participate in the solution to at least one subproblem.
2. **Reduced Problem Solution Phase:** Solve the reduced problem to near-optimality, considering only the decision variables that were included in the backbone set.

The backbone method differs from existing feature selection methods in that we propose to filter uninformative features via a procedure closely related to the actual problem. For example, to perform feature selection for the decision tree problem, we train more tractable decision trees in the subproblems and select those features that are identified as relevant within the subproblems' trees. A major difference between the backbone method and heuristic solution methods is that we aim to reduce the problem dimension as much as possible so that the resulting reduced problem can actually be solved to near-optimality. From an optimization perspective, the backbone method can loosely be viewed as a heuristic branch and price approach.

2.1 A Generic Hierarchical Backbone Algorithm

We proceed by applying the backbone method in a general sparse supervised learning setting (Algorithm 1). Let $\boldsymbol{X} \in \mathbb{R}^{n \times p}$ be the input data (n and p denote

the number of data points and features respectively) and $\boldsymbol{y}$ the vector of responses (for regression problems) or class labels (for classification problems). At the end of the backbone construction phase, the backbone set, which we denote by $\mathcal{B}$, must be small enough so that we can actually solve the resulting reduced problem. Therefore, we implement the backbone construction phase in Algorithm 1 hierarchically, in a bottom-up divide and conquer fashion, until the size of the backbone set is sufficiently small.

The notation used in Algorithm 1 also includes the following. The set $\mathcal{U}_t$ consists of all features that are candidates to enter the backbone during iteration t. Initially, all features are considered, so $\mathcal{U}_0 = [p] = \{1, ..., p\}$ (unless the `screen` function, which will be explained shortly, is used). When the data matrix $\boldsymbol{X}$ is indexed by a set $\mathcal{C} \subseteq [p]$, denoted by $\boldsymbol{X}_{\mathcal{C}}$, it is implied that only the features contained in $\mathcal{C}$ have been selected.

As part of the input to Algorithm 1, we need to specify the application-specific functions that will be used within our framework. In particular, these include:

- Function `screen`: The function used to eliminate features that are highly likely to be irrelevant. This function is optionally called before the backbone construction phase. The output of this function is the set of features that have not been eliminated, along with their marginal utilities.
- Function `construct_subproblems`: The function used to construct more tractable subproblems. Since p is assumed to be very large, the `construct_subproblems` function acts by selecting a subset of features $\mathcal{P}$ for each subproblem. Thus, the output of this function is a collection of features for every subproblem.
- Function `fit_subproblem`: The model fitting function to be used within each subproblem. Fits a sparse model of choice (regression, trees, etc.) to the data that are given as input to the subproblem. This function has to be highly efficient for the backbone construction to be fast, so we propose that a relaxed/surrogate formulation or heuristic solution method is used. For simplicity in notation, we assume that all input hyperparameters (e.g., for lasso, the regularization weight/sequence of weights) are "hidden" within the definition of the function. The output of this function is the learned model.
- Function `extract_relevant`: The function that takes a sparse model as input and extracts all features that the model identifies as relevant.
- Function `fit`: The target fitting function that fits a sparse model of choice to the data given as input. As this function is to be applied to the (small) backbone set, it needs not be extremely efficient, so we propose that a method that comes with optimality guarantees is used. For simplicity in notation, we assume that all input hyperparameters are "hidden" within the definition of the function. The output of this function is the learned model.

Algorithm 1 has a number of hyperparameters; in the sequel, we discuss how the hyperparameters can be tuned in practice (Section 2.4) and in theory (Section 3.2), and examine how sensitive the backbone method is with respect to each of them (Section 5.7). The hyperparameters are the following:

- $M \in \mathbb{N}$: The number of subproblems to solve in each iteration (or hierarchy).
- $\beta \in (0, 1]$: The fraction of features to include in each subproblem's feature set.
- $\alpha \in (0, 1]$: The fraction of features to keep after applying the `screen` function.

Algorithm 1 Generic Hierarchical Backbone Algorithm

Input: Data $(\boldsymbol{X}, \boldsymbol{y})$, hyperparameters $(M, \beta, \alpha, B_{\max})$, functions (`screen`, `construct_subproblems`, `fit_subproblem`, `extract_relevant`, `fit`).
Output: Learned model.

```
t ← 0
U_0, s ← screen(X, y, α)
repeat
    B ← ∅
    (P_m)_{m∈[M]} ← construct_subproblems(U_t, s, ⌈M/2^t⌉, β)
    for m ∈ [M] do
        model_m ← fit_subproblem(X_{P_m}, y)
        B ← B ∪ extract_relevant(model_m)
    end for
    t ← t + 1
    U_t ← B
until |B| ≤ B_max (or other termination criterion is met)
model ← fit(X_B, y)
return model
```

- $B_{\max} \in \mathbb{N}$: The maximum allowable backbone size.

The backbone method provides a generic framework that can be used to address a large number of different problems. As illustrated by the applications which we investigate in the following sections, depending on the problem at hand, there are several design choices that affect the performance of the method. For example:

- How to form the subproblems, so that they are tractable and, at the same time, useful for the original problem? We address this question in Section 2.3 by developing a generic framework for regression and classification problems.
- How to solve the subproblems and, in particular, what is the impact of applying heuristics within each subproblem for the quality of the backbone set? We examine these questions for the sparse regression and the decision tree problems in Sections 3 and 4, respectively.

2.2 The `screen` Function: Sure Independence Screening at a Glance

In this section, we describe the `screen` component of the backbone method. We first briefly present the sure independence screening (SIS) framework (Fan and Lv, 2008), which plays an important role in our approach.

SIS is a two-phase sparse learning framework. In the first phase, we rank the features based on their marginal utilities s_j and retain only the top d features, i.e.,

$$\mathcal{M} = \{1 \leq j \leq p : \ s_j \text{ is among the top } d \text{ largest ones}\}.$$

Given a convex loss function ℓ (e.g., ordinary least squares or logistic loss), we measure the marginal utility s_j of each feature $j \in [p]$ using the empirical maximum marginal likelihood estimator, i.e.,

$$(\hat{w}_{0,j}, \hat{w}_j) = \text{argmin}_{w_j^0, w_j} \frac{1}{n} \sum_{i=1}^{n} \ell(y_i, w_j^0 + w_j X_{i,j}). \tag{1}$$

We then estimate the marginal utility of feature j as either the magnitude of the maximum marginal likelihood estimator $s_j := |\hat{w}_j|$ or the minimum of the loss function $s_j := \frac{1}{n}\sum_{i=1}^{n} \ell(y_i, \hat{w}_{0,j} + \hat{w}_j X_{i,j})$. In practice, we usually choose d using cross validation. In the second phase, we conduct learning and inference in the reduced feature space consisting only of features in $\mathcal{M}$. Under certain conditions, SIS possesses the sure screening property, which requires that all relevant features are contained in the set $\mathcal{M}$ with probability tending to one (Fan and Lv, 2008; Fan et al., 2009; Fan and Song, 2010).

Algorithm 2 Function `screen`

Input: Data $(\boldsymbol{X}, \boldsymbol{y})$, hyperparameter α.
Output: Screened features $\mathcal{M}$, marginal utilities $\boldsymbol{s}$.

```
s ← 0_p
for j ∈ [p] do
    (ŵ_{0,j}, ŵ_j) ← argmin_{w_j^0, w_j} (1/n) Σ_{i=1}^n ℓ(y_i, w_j^0 + w_j X_{i,j})
    s_j ← |ŵ_j|
end for
r ← arg_sort(s)          ▷ Return a permutation of [p] that sorts s in decreasing order.
M ← {r_i}_{i=1}^{⌈αp⌉}                                          ▷ Keep top ⌈αp⌉ features.
return M, s
```

We present the `screen` function that we use within our framework in Algorithm 2. Our proposed `screen` function computes the marginal utility of each feature and eliminates the $(1-\alpha)p$ lowest ranked features. Concerning the choice of the loss function ℓ in Equation (1) or, more generally, the scoring function we use, we make the following remarks:

- In linear regression problems, we use the squared loss function (Fan and Lv, 2008). By doing so, the marginal utility measure given by $s_j := |\hat{w}_j|$ simplifies, under standard assumptions such as standardizing the data, to the absolute marginal empirical correlation between feature j and the response, namely, $s_j := |\widehat{\mathrm{cor}}(\boldsymbol{X}_j, \boldsymbol{y})|$ (note that $\widehat{\mathrm{cor}}$ represents the empirical correlation).
- In binary classification problems, we use the logistic loss function, whereas in multi-class classification problems, the multi-category SVM loss function (Fan et al., 2009).
- In highly nonlinear problems, we use the nonparametric screening approach by Fan et al. (2011) or the entropy-based approach by Ni and Fang (2016), which is particularly suited for decision tree problems.

This approach provides a unified scoring framework for both regression and classification problems. Since we only use `screen` as a preprocessing step, we do not perform cross validation and, instead, select a slightly larger value for α; e.g., we pick α such that $\alpha p \sim 10n$, which is consistent with the theoretical and empirical analysis of Fan and Lv (2008) and, as we empirically show, is a good choice in practice. As we discussed in the Introduction of this paper, by using feature marginal utility measures that satisfy the sure screening property, we obtain strong theoretical guarantees that we will not eliminate any relevant feature during this step.

2.3 The `construct_subproblems` Function

In this section, we describe the `construct_subproblems` component of the backbone method, which aims to construct more tractable subproblems.

Having computed each feature's marginal utility as per Section 2.2, we construct the feature set of subproblem $m \in M$ by sampling $\lceil \beta\alpha p \rceil$ features among the $\lceil \alpha p \rceil$ features that survived the screening step of Algorithm 2. Within each subproblem, we sample the features without replacement and with probability that increases exponentially according to each feature's marginal utility. Algorithm 3 provides pseudocode for the proposed approach.

Algorithm 3 Function `construct_subproblems`

Input: Candidate features $\mathcal{U}$, feature marginal utilities $\boldsymbol{s}$, hyperparameters M, β.
Output: Subproblem feature sets $(\mathcal{P}_m)_{m\in[M]}$.

$\boldsymbol{\pi} \leftarrow \mathbf{0}_{|\mathcal{U}|}$
for $j \in \mathcal{U}$ **do**
  $s_j \leftarrow \frac{s_j}{\max_{i\in\mathcal{U}} s_i}$ ▷ Normalize utilities.
  $\pi_j \leftarrow \exp(s_j + 1)$
end for
$(\mathcal{P}_m)_{m\in[M]} \leftarrow (\emptyset)_{m\in[M]}$
for $m \in [M]$ **do**
  $\mathcal{P}_m \leftarrow$ `sample`$(\mathcal{U}, \beta|\mathcal{U}|, \boldsymbol{\pi})$ ▷ Sample $\beta|\mathcal{U}|$ features from $\mathcal{U}$, each w/ prob $\propto \pi_j$.
end for
return $(\mathcal{P}_m)_{m\in[M]}$

The main benefits of breaking the problem into subproblems, as per Algorithm 3, after -possibly- applying the screening step of Algorithm 2, are as follows:

- *Tractability*: Since p is assumed to be very large and $\lceil \alpha p \rceil$ can also be large, by selecting a subset of features $\mathcal{P}$ for each subproblem, we perform an additional dimensionality reduction step. In particular, assume that all k relevant features survived the screening step and that we randomly sample each subproblem's $\lceil \beta\alpha p \rceil$ features. Then, in expectation, we will have βk relevant features within each subproblem. Thus, the subproblems clearly become more tractable, since, even with an exhaustive search procedure, we end up having to check $\binom{\beta\alpha p}{\beta k}$ subsets of features, instead of the initial $\binom{\alpha p}{k}$ or $\binom{p}{k}$, which can be astronomically larger.
- *Ensemble learning*: Our proposed approach, of sampling the features that form each subproblem's feature set with probability that increases exponentially according to each feature's marginal utility, is partly inspired by the exponential mechanism of McSherry and Talwar (2007). We aim to bridge the gap between the popular random subspace method (Ho, 1998) and its variants, whereby, within each subproblem, a feature subset is selected uniformly at random, and the deterministic subspace method by Koziarski et al. (2017), which ranks features according to a predefined function and selects the top ones.
  Using our proposed approach, we still enjoy the benefits of ensemble learning, whereby each feature gets examined multiple times and as part of different feature sets, while, at the same time, we avoid having subproblems with few or no relevant features. The absence of relevant features from a subproblem's

feature set makes the subproblem harder to solve, since, any relevant feature that is not sampled and hence not contained within the subproblem's feature set, is essentially viewed as noise within this subproblem. In Section 5.7, we empirically show that our proposed approach has an edge over sampling features uniformly at random or with probabilities proportional to their screening scores.

2.4 Discussion: Hyperparameters, Termination, Complexity, and Parallel Implementation

In this section, we discuss several other aspects of the backbone method: how we choose the hyperparameter values in practice, how we ensure the algorithm's termination, what is the computational complexity of the method, and how the algorithm can be implemented in parallel.

Hyperparameter selection. The hyperparameters $M, \beta, \alpha, B_{\max}$ of the backbone method can in practice be tuned via cross validation using a multi dimensional grid search. However, we remark that, in practical applications, β and $B_{\max}$ are partly determined by the available computational resources (e.g., available memory) and, more specifically, by the size of the problems that we can solve using the `fit_subproblem` function and the `fit` function, respectively. The hyperparameter α, which determines how many features the `screen` function will eliminate, is also partly dependent on the available resources, although we do have more freedom in tuning it; Fan and Lv (2008) provide both theoretical and empirical insights on how to choose α. Finally, the number of subproblems M can be selected dynamically: we pick a large value for M; we keep solving subproblems and add their solutions to the backbone set; if the backbone set does not change after two consecutive subproblems, we stop.

In Section 3.2, we provide theoretical guidance on how to choose M and α as function of the remaining problem and algorithm parameters. In Section 5.7, we present a detailed sensitivity analysis of the backbone method with respect to each of the hyperparameters.

Termination. To ensure finite termination of Algorithm 1, we require that the `fit_subproblem` function returns a model with at most $k_{\max} \leq k$ relevant features. In the context of sparse regression, we can directly control $k_{\max}$. In the context of decision trees, we indirectly control $k_{\max}$ via the tree's depth. Further, we reduce the number of subproblems to be solved in each iteration by a factor of 2. Therefore, the number of iterations that Algorithm 1 will perform is at most

$$H = \left\lceil \log_2 \frac{M k_{\max}}{B_{\max}} + 1 \right\rceil.$$

Computational Complexity. We denote by $f_{\texttt{screen}}(n, p)$, $f_{\texttt{construct}}(p, \beta)$, and $f_{\texttt{solve}}(n, p)$ the complexity of the `screen` function, the `construct_subproblems`

function, and the `solve_subproblem` function, respectively. Then, the overall complexity of the backbone construction phase of Algorithm 1 is

$$\begin{aligned}&\mathcal{O}\left[f_{\texttt{screen}}(n,p)+\sum_{t=1}^{H}\frac{M}{2^{t-1}}f_{\texttt{construct}}(\alpha p,\beta)+\sum_{t=1}^{H}\frac{M}{2^{t-1}}f_{\texttt{solve}}(n,\alpha\beta p)\right]\\&=\mathcal{O}\left[f_{\texttt{screen}}(n,p)+Mf_{\texttt{construct}}(\alpha p,\beta)+Mf_{\texttt{solve}}(n,\alpha\beta p)\right].\end{aligned}$$

As an example, consider a sparse regression problem where we use the empirical correlation between each feature and the response as the scoring function for `screen`, use a simple heap-sampling implementation for `construct_subproblems`, and solve subproblems using the LARS algorithm for the lasso formulation (Efron et al., 2004). Then, the resulting complexity is $\mathcal{O}\left[np+M\alpha\beta p\log(\alpha p)+M(\alpha\beta p)^2 n\right]$.

Parallel & Distributed Implementation. An important feature of the backbone method is that it can be naturally executed in parallel by assigning different subproblems to different computing nodes and then centrally collecting all solutions and solving the reduced problem. This is particularly important in distributed applications, where a massive dataset is "vertically partitioned" among different nodes, i.e., each node has access to a subset of the features in the data. This regime, despite being common in practice, has not been considered much in the literature.

3 The Backbone of Sparse Regression

Given data $\{(\boldsymbol{x}_i, y_i)\}_{i\in[n]}$, where, $\forall i \in [n]$, $\boldsymbol{x}_i \in \mathbb{R}^p$ (with $n \ll p$), and $y_i \in \mathbb{R}$ (for regression) or $y_i \in \{-1, 1\}$ (for binary classification), and a convex loss function ℓ, consider the sparse empirical risk minimization problem with Tikhonov regularization and an explicit sparsity constraint, outlined as

$$\min_{\boldsymbol{w}\in\mathbb{R}^p} \quad \sum_{i=1}^{n}\ell(y_i,\boldsymbol{w}^T\boldsymbol{x}_i)+\frac{1}{2\gamma}\|\boldsymbol{w}\|_2^2 \qquad \text{s.t.} \quad \|\boldsymbol{w}\|_0\le k. \tag{2}$$

Problem (2) can be reformulated as a MIO problem, by introducing a binary vector encoding the support of the regressor $\boldsymbol{w}$. The optimal cost for the equivalent formulation is then given by

$$\begin{aligned}\min_{\boldsymbol{z}\in\{0,1\}^p:\sum_{j=1}^{p}z_j\le k} \quad \min_{\boldsymbol{w}\in\mathbb{R}^p} \quad & \sum_{i=1}^{n}\ell(y_i,\boldsymbol{w}^T\boldsymbol{x}_i)+\frac{1}{2\gamma}\|\boldsymbol{w}\|_2^2\\ \text{s.t.} \quad & z_j=0\Rightarrow w_j=0 \quad \forall j\in[p].\end{aligned} \tag{3}$$

As we already noted in the introduction, exact MIO methods that solve Problem (3) scale up to $p \sim 10^5$. We next apply the backbone method to the sparse regression Problem (2).

During the backbone construction phase, we form the M subproblems, whose feature sets are denoted by $\mathcal{P}_m$, $m \in [M]$, using the `construct_subproblems` function (Algorithm 3). Within the m-th subproblem, $m \in [M]$, we may use a

subset of data points $\mathcal{N}_m \subseteq [n]$ and different hyperparameter values k_m and γ_m. We form the backbone set as follows:

$$\mathcal{B} = \bigcup_{m=1}^{M} \Bigg\{ j : \quad w_j^{(m)} \neq 0, \\ \boldsymbol{w}^{(m)} = \underset{\substack{\boldsymbol{w}\in\mathbb{R}^p: \\ \|\boldsymbol{w}\|_0 \leqslant k_m, \\ w_j = 0 \ \forall j \notin \mathcal{P}_m}}{\operatorname{argmin}} \sum_{i\in\mathcal{N}_m} \ell(y_i, \boldsymbol{w}^T \boldsymbol{x}_i) + \frac{1}{2\gamma_m}\|\boldsymbol{w}\|_2^2 \Bigg\}. \tag{4}$$

We note that, instead of solving the sparse regression formulation in each subproblem, as shown in Equation (4), there is a possibility of solving a relaxation or a closely-related surrogate problem, such as the elastic net formulation (Zou and Hastie, 2005). We discuss this possibility in more detail in Section 3.1.

Finally, once the backbone construction phase is completed, we solve the original problem, considering only the features that are contained in the backbone set (i.e., exact solution computation in subset of backbone features), namely,

$$\begin{aligned} \min_{\boldsymbol{w}\in\mathbb{R}^p} \quad & \sum_{i=1}^{n} \ell(y_i, \boldsymbol{w}^T \boldsymbol{x}_i) + \frac{1}{2\gamma}\|\boldsymbol{w}\|_2^2 \\ \text{s.t.} \quad & \|\boldsymbol{w}\|_0 \leqslant k, \\ & w_j = 0 \quad \forall j \notin \mathcal{B}. \end{aligned} \tag{5}$$

3.1 Implementation Details

In this section, we discuss some additional implementation aspects of the backbone method for the sparse regression problem.

Solving Subproblems via the Sparse Regression Formulation. Our first proposed approach to solve the subproblems relies on the actual sparse regression Formulation (2). In our implementation, we use the subgradient method of Bertsimas et al. (2020), which, besides fast, is especially strong in recovering the true support; it should be clear that any of the exact or heuristic solution methods discussed in Section 1.1 can be used. We empirically found this approach to be the most effective in terms of feature selection, namely, it achieves the lowest false detection rate within the backbone set.

A critical design choice in this approach concerns the *number of relevant features that should be extracted from each subproblem*. Unless our sampling procedure is perfectly accurate, it is possible that some subproblems will end up containing $k_{\text{s}} < k$ relevant features. If this is the case, we empirically observed that solution methods for Problem (2) quickly recover the k_{s} truly relevant features but struggle to select the remaining $k - k_{\text{s}}$, i.e., they spend a lot of time trying to decide which irrelevant features to pick. It is therefore crucial to either use cross validation within each subproblem (like the one we describe shortly), or apply an incremental selection procedure (e.g., forward stepwise selection). In our implementation, we use the following *incremental cross validation scheme for the subproblem support size*, which relies on progressively fitting less sparse models. We start at a small number of relevant features k_0, increment to $k_1 = k_0 + k_{\text{step}}$, $k_2 = k_0 + 2k_{\text{step}}$,

and so forth. We stop after i steps if the improvement in terms of validation error of a model with $k_i + k_{\text{step}}$ and $k_i + 2k_{\text{step}}$ features is negligible compared to the error of a model with k_i features. Crucially, when training a model with $k_i > k_0$ features during cross validation, we use the best model with $k_i - k_{\text{step}}$ features as a warm-start.

Concerning the regularization parameter γ in Formulation (2), we apply a simple grid search (with grid length l) cross-validation scheme, between a small value, e.g., $\gamma_0 = \frac{p}{k \; n \; \max_i \|\boldsymbol{x}_i\|^2}$, and $\gamma_l = \frac{1}{\sqrt{n}}$, which is considered a good choice for most regression instances (Chen et al., 2012).

Solving Subproblems via Randomized Rounding. Instead of solving the MIO sparse regression Formulation (3), one can consider its boolean relaxation, whereby the integrality constraints $z_i \in \{0, 1\}$ are relaxed to $z_i \in [0, 1]$ (Xie and Deng, 2020). The resulting solution $\hat{z}^{\text{rel}}$ will, in general, not be integral, so we propose to randomly round it by drawing $\hat{z}_i^{\text{bin}} \sim \text{Bernoulli}(\hat{z}_i^{\text{rel}})$. We form the backbone set by repeating the rounding procedure multiple times for each subproblem and collecting all features that had their associated binary decision variable set to 1 in at least one realization of the Bernoulli random vector. By doing so, we reduce the number of subproblems we solve and, instead, perform multiple random roundings per subproblem. As a result, the overall method is sped up, sacrificing, however, its effectiveness in terms of feature selection. Moreover, in this approach, we cannot directly control the number of relevant features extracted from each subproblem, so we heuristically keep the features that correspond to the $k_{\max}$ largest regressors. For these reasons, we did not use this approach in the computational results we present.

Solving Subproblems via Surrogate Formulations. In our second proposed approach to solve the subproblems, we formulate each subproblem using surrogate formulations, such as the lasso estimator (Tibshirani, 1996) and its extensions. In particular, we utilize the elastic net formulation (Zou and Hastie, 2005) to solve the m-th subproblem, $m \in [M]$, namely,

$$\boldsymbol{w}^{(m)} = \underset{\substack{\boldsymbol{w} \in \mathbb{R}^p: \\ w_j = 0 \;\; \forall j \notin \mathcal{P}_m}}{\text{argmin}} \quad \sum_{i \in \mathcal{N}_m \subseteq [n]} \ell(y_i, \boldsymbol{w}^T \boldsymbol{x}_i) + \lambda_m \left[\mu_m \|\boldsymbol{w}\|_1 + \tfrac{1-\mu_m}{2} \|\boldsymbol{w}\|_2^2 \right]. \quad (6)$$

The popularity of ℓ_1-regularization is justified by the fact that it enjoys a number of properties that are particularly useful in practical applications. First and foremost, its primary mission is to robustify solutions against noise in the data and, specifically, against feature-wise perturbations. Second, the convexity of the ℓ_1-norm makes the task of optimizing over it significantly easier. Third, ℓ_1-regularization provides sparser solutions than ℓ_2-regularization. We refer the interested reader to Bertsimas and Copenhaver (2018) and Xu et al. (2009) for a detailed discussion on the equivalence of regularization and robustification.

We pick the hyperparameters of the elastic net formulation via cross validation. We select μ using grid search in the $[0, 1]$ interval. For each fixed μ, we perform grid search for $\lambda \in [\lambda_{\min}, \lambda_{\max}]$, where $\lambda_{\max}$ is the value of λ that leads to an empty model and $\lambda_{\min}$ is the value of λ that leads to a model consisting of $k_{\max}$ nonzero coefficients. Once the best elastic net hyperparameters are found, we refit and add to the backbone set those features whose associated coefficients are above a user-specified threshold.

Solving the Reduced Problem. After having formed the backbone set, we solve Problem (5) using the cutting planes method by Bertsimas and Van Parys (2020). As Bertsimas et al. (2020) observed, the computational time required for the MIO Formulation (3) to solve to provable optimality is highly dependent on γ; for smaller values of γ, problems with $p \sim 10^5$ can be solved in minutes, whereas, for larger values, it might take a huge amount of time to solve to provable optimality (although the optimal solution is usually attained fast). To address this issue, we impose a time limit on the cutting planes method for each value of γ during the cross validation process. In practice, we observe that the cutting planes method applied to the backbone set recovers the correct support in seconds (provided that the backbone set is sufficiently small).

3.2 Theoretical Justification

In this section, we present a theoretical result, which guarantees that, under certain conditions and with high probability, the true relevant features are selected in the backbone set. Our result, given in Theorem 1, theoretically justifies the proposed approach and, most importantly, as explained above, provides guidance for the selection of the hyperparameters of the method.

Model and Assumptions. The model that we consider is as follows. We have a random design matrix $\boldsymbol{X} \in \mathbb{R}^{n \times p}$ such that each row $\boldsymbol{x}_i, i \in [n]$, is an iid copy of the random vector $X = (X_1, ..., X_p) \sim \mathcal{N}(0, \boldsymbol{I}_p)$. We have fixed but unknown regressors $\boldsymbol{\beta} \in \{-1, 0, 1\}^p$ that satisfy $\|\boldsymbol{\beta}\|_0 \leq k$. Let $\mathcal{S}^{\text{true}}$ denote the set of indices that correspond to the support of the true regressor $\boldsymbol{\beta}$. The response vector is $\boldsymbol{y} = \boldsymbol{X}\boldsymbol{\beta} + \boldsymbol{\varepsilon}$ where the noise term $\boldsymbol{\varepsilon}$ consists of iid entries $\varepsilon_i \sim \mathcal{N}(0, \sigma^2), i \in [n]$. Thus, each entry in $\boldsymbol{Y}$ is distributed as $Y = X^\top \boldsymbol{\beta} + \varepsilon \sim \mathcal{N}(0, k + \sigma^2)$ (assuming that $\|\boldsymbol{\beta}\|_0 = k$). We further assume that $p > n \geq k$ and $\log p = O(n^\xi)$ for some $\xi \in (0, 1)$. When we make asymptotic arguments, we take $p \to \infty$. We analyze a simplified version of the backbone method, which we outline in Appendix A. Our goal is to show that, with high probability, $\mathcal{S}^{\text{true}} \subseteq \mathcal{B}$.

The model we examine is indeed very simple and rather unrealistic; nevertheless, such assumptions are standard for analyzing exact sparse regression methods (see, e.g., Gamarnik and Zadik (2017), Pilanci et al. (2015), and Bertsimas and Van Parys (2020)). Moreover, by the following result, we primarily aim to provide guidance for the selection of the hyperparameters of the method. We believe that the main contribution of the paper is its practical relevance.

Main Result. We denote by ε_2 an upper bound on the probability that the `solve_subproblem` function fails to recover all relevant features that are included in the feature set of an arbitrary subproblem. We denote by $\mathcal{E}$ the event that the (simplified version of) backbone method fails, namely, $\exists j \in [p]$ such that $j \in \mathcal{S}^{\text{true}}$ and $j \notin \mathcal{B}$. We prove the following result:

Theorem 1. *Assume that the data is generated as discussed above, $p > n \geq k$, and $\log p = O(n^\xi)$ for some $\xi \in (0, 1)$. Further, suppose that the sample size n satisfies $n \geq \theta(\sigma^2 + 2k)\log(\beta\alpha p)$, for all $\theta \geq 1$, so that $\varepsilon_2 = O(e^{-\theta}) \to 0$ as $p \to \infty$. Then, when $\alpha = O\left(\frac{n^{1-\phi}}{p}\right)$, for some $\phi < 1$, and when $M = O\left(\frac{\log(\alpha p k)}{\log\left(\frac{1}{1-\beta+\beta\varepsilon_2}\right)}\right)$, it holds*

that $P(\mathcal{E}) \to 0$ as $p \to \infty$, that is, the (simplified version of the) backbone method recovers with high probability the true relevant features in the backbone set.

The proof is included in Appendix A. This result asserts that the number of subproblems increases logarithmically with the product of the number of features that we sample from in each subproblem and the number of relevant features. Moreover, since β controls the subproblem size, the larger the subproblems are, the fewer subproblems we need to solve. Finally, as ε_2 decreases and we therefore solve subproblems more accurately, we again need to solve fewer subproblems.

4 The Backbone of Decision Trees

Given data $\{(\boldsymbol{x}_i, y_i)\}_{i\in[n]}$, where, $\forall i \in [n]$, $\boldsymbol{x}_i \in \mathbb{R}^p$ (with $n \ll p$) and $y_i \in [K]$, decision trees recursively partition the feature space and assign a class label from $[K]$ to each partition. Formally, let T be a decision tree, $\mathcal{T}_B$ the set of branch nodes and $\mathcal{T}_L$ the set of leaf nodes. At each branch node $t \in \mathcal{T}_B$, a split of the form $\boldsymbol{a}_t^T \boldsymbol{x} < b_t$ is applied. Each leaf node $l \in \mathcal{T}_L$ is assigned a class label, typically via a majority vote among the class labels y_i of all data points that fall into leaf l after traversing the tree. The function $N(l; \boldsymbol{X})$ counts the number of data points in leaf l and $N_{\min}$ is a "minbucket" parameter that controls the minimum number of data points that are allowed to fall into any leaf. Finally, $g(T; \boldsymbol{X}, \boldsymbol{y}, \lambda) = \text{error}(T; \boldsymbol{X}, \boldsymbol{y}) + \lambda|T|$ is the objective and consists of two components. The first, $\text{error}(T; \boldsymbol{X}, \boldsymbol{y})$, is typically the misclassification error of the tree T on the training data $(\boldsymbol{X}, \boldsymbol{y})$. The second, $|T|$, is typically the number of branch nodes in the tree T. Then, at a high level, the decision tree problem can be stated as

$$T^\star = \text{argmin}_{T:\text{depth}(T)\leq D} \quad g(T; \boldsymbol{X}, \boldsymbol{y}, \lambda) \quad \text{s.t. } N(l; \boldsymbol{X}) \geq N_{\min} \; \forall l \in \mathcal{T}_L. \tag{7}$$

Ideally, we would like to exactly solve Problem (7) and therefore obtain a tree that achieves global optimality; in practice, however, this is still beyond the reach of MIO solvers. The optimal classification trees (OCT) framework provides a MIO formulation along with a set of heuristics that enable us to approximately solve Problem (7) in significantly larger dimensions than what was known. Nevertheless, despite the success of OCTs, the number of features p remains their primary bottleneck; currently, OCTs scale up to p in the 1,000s.

As we pointed in the introduction, the decision tree induction process has traditionally been addressed via scalable heuristic methods, such as CART. Nonetheless, CART's training process has little to do with the actual misclassification objective and, as a result, the algorithm often settles with trees that are far from optimal for the original problem. The popularity of decision tree classifiers led to their wide use in ensemble models, such as bagging and boosting. Random forest, for example, combines multiple independently trained decision trees and typically boosts their performance, enjoying many of the benefits of ensemble learning, at the cost, however, of sacrificing interpretability.

The aforementioned limitations of decision tree-based methods, along with the fact that decision trees are indeed sparse models, are our main motivations in developing a backbone method for OCTs.

Relevant Features. Intuitively, each split in a decision tree is associated with a feature $j \in [p]$, so there are at most $2^D - 1$ relevant features in tree of max depth D. We define that feature j relevant if at least one split performed on it. During the backbone construction phase, we again form M distinct and tractable subproblems.

Constructing Subproblems. Similarly to regression, in the m-th subproblem, $m \in [M]$, we only consider the features included in the subproblem's feature set $\mathcal{P}_m$ (constructed using the `construct_subproblems` function of Section 2.3) and, possibly, randomly sample a subset of data points $\mathcal{N}_m \subseteq [n]$. Note that, in formulating each subproblem, we may use different hyperparameter values $\lambda_m, N_{\min,m}$ and D_m.

Forming the Backbone Set. Let us denote by $\mathcal{T}(\boldsymbol{X}, \boldsymbol{y}, D, N_{\min})$ the set of all feasible trees on input data $(\boldsymbol{X}, \boldsymbol{y})$ and by $\boldsymbol{a_t}$ the split performed at branch node t. Then, the backbone set can be written as the union of the solutions to all subproblems, namely,

$$\mathcal{B} = \bigcup_{m=1}^{M} \Bigg\{ j : \sum_{t \in \mathcal{T}_B^{(m)}} a_{tj}^{(m)} \geq 1, \quad T^{(m)} = \underset{\substack{T \in \mathcal{T}(\boldsymbol{X}_{\mathcal{N}_m, \mathcal{P}_m}, \\ \boldsymbol{y}_{\mathcal{N}_m}, D_m, N_{\min,m})}}{\operatorname{argmin}} g(T; \boldsymbol{X}_{\mathcal{N}_m, \mathcal{P}_m}, \boldsymbol{y}_{\mathcal{N}_m}, \lambda_m) \Bigg\}. \tag{8}$$

Importantly, in solving each of the subproblems in (8), we need not solve the OCT formulation, shown in (7); instead, we propose solving each subproblem using scalable heuristic methods, such as CART. In fact, we empirically found that applying the OCT framework to subproblems does not significantly improve the backbone accuracy (i.e., fraction of relevant features included in the backbone set). We use cross validation within each subproblem $m \in [M]$ to tune the hyperparameters $D_m, N_{\min,m}, \lambda_m$.

Solving the Reduced Problem. Once the backbone set $\mathcal{B}$ is constructed, we solve the OCT Formulation (7), considering only the features in $\mathcal{B}$, i.e.,

$$T^\star = \operatorname{argmin}_{T:\text{depth}(T) \leq D} \quad g(T; \boldsymbol{X}_\mathcal{B}, \boldsymbol{y}, \lambda) \quad \text{s.t. } N(l; \boldsymbol{X}_\mathcal{B}) \geq N_{\min} \ \forall l \in \mathcal{T}_L. \tag{9}$$

Extension to Optimal Classification Trees with Hyperplane Splits. Instead of limiting the tree learning method to univariate splits, where, $\boldsymbol{a_t} \in \{0, 1\}^p$ for any branch node t, multivariate splits can also be used. If this is the case, it is important that the number of features that participate in each split is artificially constrained (which translates to a sparsity constraint on $\boldsymbol{a_t}$), otherwise the backbone set will generally not be small enough to substantially reduce the problem dimension.

Connections with the Random Subspace Method and Random Forest. Feature bagging methods have had significant success when applied to ensembles of decision trees. The random subspace method (Ho, 1998) relies on training each decision tree in the ensemble on a random subset of the features instead of the entire feature set. In random forest (Breiman, 2001), a subset of the features is considered in each split of each decision tree. Our backbone method for OCTs relies on the `construct_subproblems` function, so each tree induced during the backbone construction phase is also trained on a subset of features. Thus, our approach enjoys many of the benefits of feature bagging (e.g., parallelizeability), while, at the same time, its output is a single, interpretable tree.

5 Computational Results on Synthetic Data

In this section, we investigate the performance of the backbone method on synthetic datasets generated according to ground truth models that are known to be sparse. We start by describing the data generating methodology, the metrics, and the algorithms that we use throughout this (as well as the following) section. Then, we explore the scalability and performance of the method, as well as the sensitivity to the method's hyperparameters and components. Our primary goal in this section is to shed light on the behavior of the proposed backbone method and not to benchmark the backbone method compared to state-of-the-art alternatives; this is, in fact, the focus of Section 6.

5.1 Data Generating Methodology

In all our synthetic experiments throughout this section, we generate data according to the following methodology.

Design Matrix. We assume that the *input data* $\boldsymbol{X} = (\boldsymbol{x}_1, ..., \boldsymbol{x}_n)$ are i.i.d. realizations from a p-dimensional zero-mean normal distribution with covariance matrix $\boldsymbol{\Sigma}$, i.e., $\boldsymbol{x}_i \sim \mathcal{N}(\boldsymbol{0}_p, \boldsymbol{\Sigma}), i \in [n]$. The covariance matrix $\boldsymbol{\Sigma}$ is parameterized by the correlation coefficient $\rho \in [0, 1)$ as $\Sigma_{ij} = \rho^{|i-j|}, \forall i, j \in [p]$. As $\rho \to 1$, the columns of the data matrix $\boldsymbol{X}$, i.e., the features, become more alike which should impede the discovery of nonzero components of the true regressor $\boldsymbol{w}_{\text{true}}$ by obfuscating them with highly correlated look-alikes. In our experiments, we focus on high correlation regimes (e.g., $\rho = 0.6$ or even $\rho = 0.9$).

Sparse Linear Regression Data. For linear regression, the unobserved true regressor $\boldsymbol{w}_{\text{true}}$ is constructed at the beginning of the process and has exactly k-nonzero components at indices selected uniformly without replacement from $[p]$. Likewise, the nonzero coefficients $\boldsymbol{w}_{\text{true}}$ are drawn uniformly at random from the set $\{-1, +1\}$. We next generate the *response vector* $\boldsymbol{y}$, which satisfies the linear relationship $\boldsymbol{y} = \boldsymbol{X}\boldsymbol{w}_{\text{true}} + \boldsymbol{\varepsilon}$, where $\varepsilon_i, i \in [n]$, are i.i.d. noise components from a normal distribution, scaled according to a chosen signal-to-noise ratio $\sqrt{\text{SNR}} = \|\boldsymbol{X}\boldsymbol{w}_{\text{true}}\|_2 / \|\boldsymbol{\varepsilon}\|_2$. Evidently as the SNR increases, recovery of the unobserved true regressor $\boldsymbol{w}_{\text{true}}$ from the noisy observations can be done with higher precision.

Sparse Logistic Regression Data. For logistic regression, the true regressor is constructed in the exact same manner. The signal $\boldsymbol{y}$ is computed according to

$$\boldsymbol{y} = \text{sign}(\boldsymbol{X}\boldsymbol{w}_{\text{true}} + \boldsymbol{\varepsilon}).$$

Classification Tree Data. For classification trees, we first create a full binary tree T_{true} (*ground truth tree*) of given depth D. The structure of T_{true} is determined as follows.

- *Relevant features:* We randomly pick k relevant features among the entire feature set. At each split node, we select a relevant feature to split on. To ensure that all relevant features are actually informative, we require that each of them appears in at least r split nodes in the tree. Thus, the number of relevant features k must satisfy $k \leq \frac{2^D-1}{r}$.
- *Split thresholds:* Within each split node, we randomly pick a split threshold, taking care to maintain consistency of feature ranges across paths in the tree. Moreover, let $x \in [x_{\min}, x_{\max}]$ be the feature on which we split at node t. To ensure that splits are reasonably balanced, we require that the split threshold $b_t \in [x_{\min} + \frac{x_{\max}-x_{\min}}{2} \cdot f,\ x_{\max} - \frac{x_{\max}-x_{\min}}{2} \cdot f]$, where $f \in [0, 1]$ is the parameter that determines how balanced the splits are. In our experiments, we use $f = \frac{1}{2}$.
- *Labels:* We assign class labels to leaf nodes in such a way that no sibling leaves correspond to the same class. We denote by $\mathcal{C}$ the set of all class labels and by $c_l \in \mathcal{C}$ the class label of leaf l. Furthermore, let $K = |\mathcal{C}|$ the total number of classes. In our experiments, we use $K = 2$ and hence examine binary classification problems.

Next, we generate data from T_{true} by setting, for each $i \in [n]$, $y_i = c_l$, where l is the leaf where data point $\boldsymbol{x}_i$ falls after traversing the tree.

5.2 Metrics

In our computational study in this section and in Section 6, we evaluate the quality of each method based on the following metrics:

- *Support recovery accuracy* (SR-ACC): Measures the fraction of the k relevant features that were actually selected by the estimator. For example, in the context of regression, we have

$$\frac{|\{j : w_j \neq 0,\ w_{\text{true},j} \neq 0\}|}{k}.$$

- *Support recovery false alarm rate* (SR-FA): Measures the ratio of number irrelevant features selected over total number of features selected.
- *Fraction of features used that are relevant:* Measures the ratio of the number of relevant features used over the total number of features used in the learned model. This metric simultaneously captures support recovery accuracy and model simplicity, and is particularly useful in decision tree models, whereby one feature might be part of the ground truth tree and yet have little impact on the classification task.

- *Prediction accuracy* (R^2 or AUC): Evaluates the out-of-sample performance of the estimator. We use the R^2 statistic for linear regression and the area under the curve (AUC) for logistic regression and for classification trees (since we deal with binary classification problems).
- *Optimality gap* (OG): Measures the gap between the lower and upper objective bound during the solution process of an MIO problem and, specifically, of Problem (3).
- *Learned tree depth:* Reports the depth of the learned decision tree model.
- *Computational time* (T): Total amount of time used (in seconds).

Additionally, to assess the quality of each component of the backbone method, we also record the following statistics:

- *Backbone accuracy*: Measures the fraction of the k relevant features that were selected in the backbone set.
- *Backbone size*: Number of features included in the backbone set.
- *Subproblem feature sets accuracy*: Measures the fraction of the k relevant features that were sampled in at least one subproblem's feature set.

All experimental results were obtained over 10 independently generated datasets. In each experiment, we report both the mean and standard deviation of each metric. All out-of-sample metrics were obtained from independently generated test sets of size $n_{\text{test}} = 2,000$.

5.3 Algorithms & Software

In this section, we summarize the algorithms and software that we use in our experiments. For the sparse regression problem, we use the following algorithms:

- `SR`: Implementation of the cutting planes method by Bertsimas and Van Parys (2020) in Julia using the commercial MIO solver Gurobi (Gurobi Optimization Inc., 2016). Uses the subgradient method Bertsimas et al. (2020) to compute a warm start. Solves the sparse regression formulation to optimality or near-optimality.
- `SR-REL`: Implementation of the subgradient method by Bertsimas et al. (2020) in Julia using the commercial Interpretable AI software package (Interpretable AI, 2020). Solves the sparse regression heuristically by considering its boolean relaxation and iteratively alternating between a sub-gradient ascent step and a projection step.
- `ENET`: `glmnet` Fortran implementation (Friedman et al., 2020), using the Julia wrapper. Solves the elastic net formulation using cyclic coordinate descent.

For the decision tree problem, we use the following algorithms:

- `OCT`: Implementation of the OCT-learning method from the OT framework (Bertsimas and Dunn, 2017) in Julia using the commercial Interpretable AI software package (Interpretable AI, 2020). Solves the decision tree formulation via a tailored local search procedure.
- `CART`: Implementation of the CART heuristic (Breiman et al., 1984) using the Julia wrapper for the scikit-learn package (Pedregosa et al., 2011).
- `RF`: Implementation of the random forest classifier (Breiman, 2001) using the Julia wrapper for the scikit-learn package (Pedregosa et al., 2011).

To address high-dimensional regimes, we consider the following methods that incorporate a feature selection component:

- `SIS-ENET`: Implementation of the sure independence screening (Fan and Lv, 2008) feature selection heuristic in Julia, as per Section 2.2, followed by `ENET` on the reduced feature set.
- `RFE`: Implementation of the popular recursive feature elimination algorithm (Guyon et al., 2002) using a Julia wrapper for the scikit-learn package (Pedregosa et al., 2011). For sparse regression, we use linear regression to eliminate features, with a step of 100 features; then, we apply `SR` on the selected features (tuned using cross-validation). For classification trees, we use `CART` to eliminate features, with a step of 100 features; then, we apply `CART` on the selected features (tuned using cross-validation).
- `DECO`: Implementation of the DECO framework by Wang et al. (2016) in Julia. We partition the feature space into 5 subsets and, after the de-correlation step, we perform feature selection in each subset using lasso. Then, we apply `ENET` on the selected features (tuned using cross-validation).

Finally, our proposed backbone method is implemented as outlined below:

- `BB`: Implementation of the backbone method in Julia. The components and parameterization of the method are discussed in each experiment separately. We remark that, in the results that we present, we did not make an effort to fine-tune the method's parameters; instead, we selected them based on Theorem 1 and on empirical evidence.

All experiments were performed on a standard Intel(R) Xeon(R) CPU E5-2690 @ 2.90GHz running CentOS release 7. Moreover, all methods' hyperparameters are tuned using the holdout method for cross validation, whereby we split the training set into actual training and validation data at a 0.7 ratio.

5.4 Scalability with the Number of Features

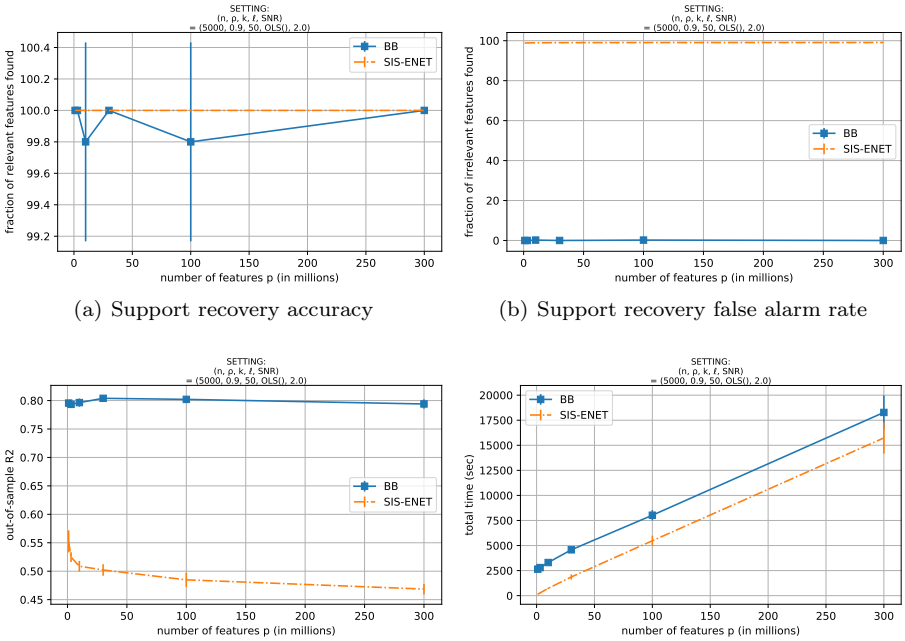
In this experiment, we examine the scalability of `BB` as the number of features p increases. We show that `BB` accurately scales to ultra-high dimensional problems and notably outperforms baseline heuristics.

Sparse Linear Regression. We consider a sparse linear regression problem with $n = 5,000$ data points, $k = 50$ relevant features, SNR $= 2$, and correlation $\rho = 0.9$. We vary the number of features $p \in \{10^6, 3 \cdot 10^6, \ldots, 3 \cdot 10^8\}$.

We tune `BB` as follows. We select the `screen` function's parameter α such that all but $4 \cdot n = 20,000$ features are eliminated. We set $\beta = 0.5$ and solve $M = 10$ subproblems. We set $B_{\max} = 250$. We solve the subproblems using `SR-REL`; in the m-th subproblem, we cross-validate 3 values for $k_m \in \{\lceil \frac{k}{3} \rceil, \lceil \frac{2k}{3} \rceil, k\}$ and set $\gamma_m = \frac{1}{\sqrt{n_m}}$. We solve the reduced problem using `SR` with a time limit of 5 minutes; we cross-validate 5 values for the hyperparameter γ. As a baseline, we compare `BB` with `SIS-ENET`, whereby we select αp features using `SIS` and then apply `ENET`. We tune `ENET` as described in Section 3.1 and, specifically, we cross-validate 5 values for the hyperparameter $\mu \in [0, 1]$ (the pure lasso model is included in the

cross-validation procedure). We discard from the final model any feature whose corresponding regressor has magnitude $\leq 10^{-6}$.

Figure 1 presents the results for this experiment. `BB` achieves near-perfect accuracy for problems with up to 300 million features and substantially outperforms `SIS-ENET`, in terms of both support recovery accuracy (in that `BB` recovers almost the entire true support with near zero false positives, whereas `SIS-ENET` recovers the true support at the cost of a large number of false positives) and out-of-sample predictive performance. As far as the computational time is concerned, we observe that the overhead of `BB` over `SIS-ENET` is by no means prohibitive; we are able to solve problems with 10 million features in less than an hour and problems with 300 million features in less than 10 hours.



(a) Support recovery accuracy

(b) Support recovery false alarm rate

(c) Out-of-sample R^2

(d) Computational Time (sec)

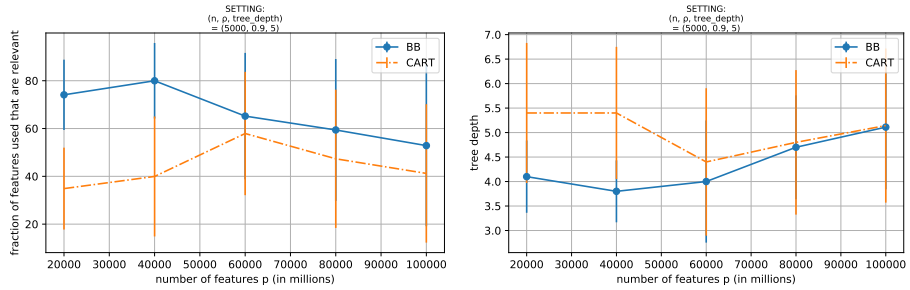
Fig. 1: Scalability with the number of features for sparse linear regression.

Classification Trees. We consider a classification tree problem with $n = 5,000$ data points, tree depth of $D = 5$ and $k = 31$ relevant features, and correlation $\rho = 0.9$. We vary the number of features $p \in \{2 \cdot 10^4, 4 \cdot 10^4, \ldots, 10^5\}$.

We tune `BB` as follows. We select the `screen` function's parameter $\alpha = 1$ such that no features are eliminated. We set $\beta = 0.1$ and solve $M = 50$ subproblems. We set $B_{\max} = 250$. We solve the subproblems using `CART`; in the m-th subproblem, we cross-validate $D_m \in \{2, \ldots, D-2\}$ (we also cross-validate the minbucket and complexity parameter of `CART`). We solve the reduced problem using `OCT`; we cross-validate $D \in \{3, \ldots, D+2\}$ (we also cross-validate the minbucket and complexity

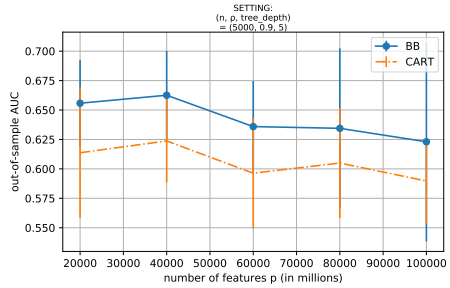
parameter of OCT). As a baseline, we compare BB with CART, tuned in the exact same way as OCT in solving the reduced problem for BB.

As can be observed in Figure 2, BB outperforms CART in terms of out-of-sample predictive performance for problems with up to 100,000 features. Moreover, among the features that are used in the split nodes of the learned classification tree, BB selects a substantially higher fraction of relevant ones and, at the same time, the learned tree is simpler (i.e., of smaller depth). This particular configuration of BB solves problems with 100,000 features in approximately an hour and the computational time scales linearly with the number of features.

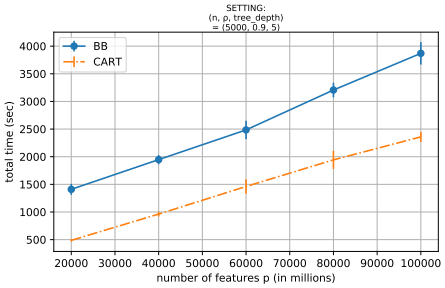


(a) Fraction of features used that are relevant

(b) Learned tree depth



(c) Out-of-sample AUC



(d) Computational Time (sec)

Fig. 2: Scalability with the number of features for classification trees.

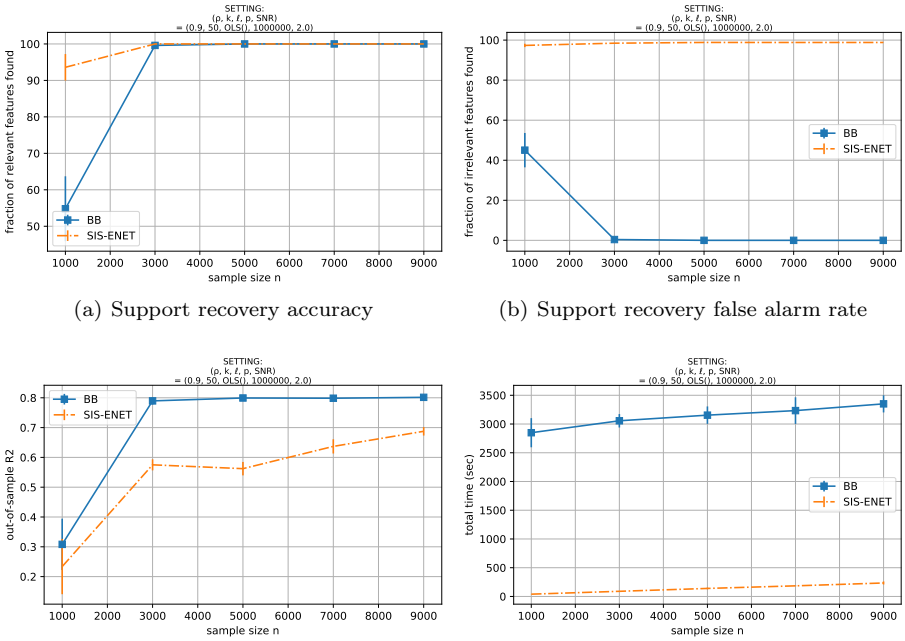
5.5 Scalability with the Number of Samples

In this experiment, we examine the scalability of BB as the number of samples n increases. We show that BB outperforms baseline heuristics in the context of sparse linear regression, sparse logistic regression, and classification trees.

Sparse Linear Regression. We consider a sparse linear regression problem with $p = 10^6$ features, $k = 50$ relevant features, SNR $= 2$, and correlation $\rho = 0.9$. We vary the number of data points $n \in \{1,000, 3,000, \ldots, 9,000\}$.

We tune `BB` and `SIS-ENET` as described in the first experiment in Section 5.4, under the following modifications. For `BB`, we now select the `screen` function's parameter α such that all but $10,000$ features are eliminated, $\beta = 0.5$ and solve $M = 15$ subproblems.

Figure 3 presents the results for this experiment. `BB` outperforms `SIS-ENET` as the number of samples increases, in terms of both support recovery accuracy and out-of-sample predictive performance, and solves problems with 1 million features and $9,000$ samples in less than an hour.



(a) Support recovery accuracy

(b) Support recovery false alarm rate

(c) Out-of-sample R^2

(d) Computational Time (sec)

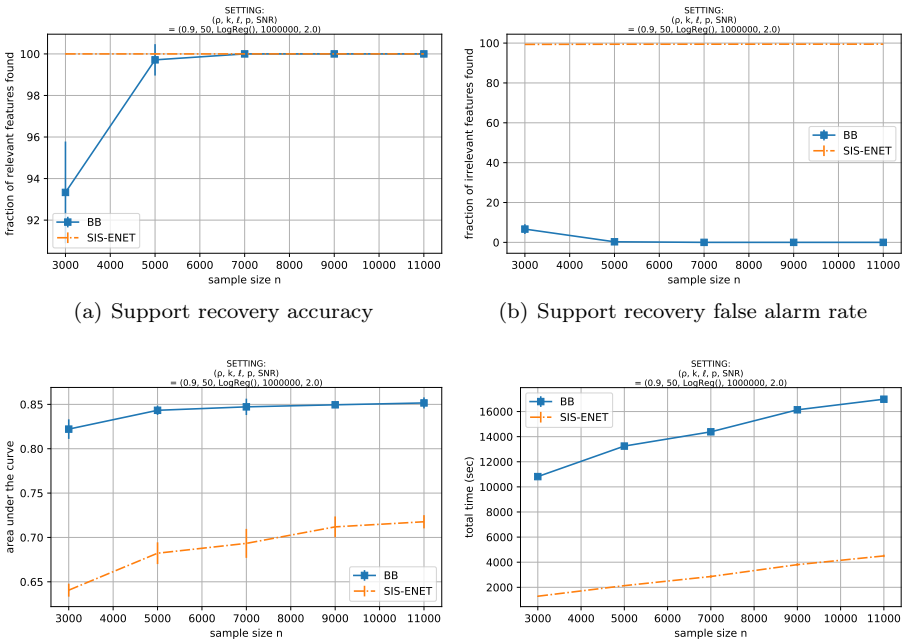
Fig. 3: Scalability with the number of samples for sparse linear regression.

Sparse Logistic Regression. We consider a sparse logistic regression problem with $p = 10^6$ features, $k = 50$ relevant features, SNR $= 2$, and correlation $\rho = 0.9$. We vary the number of data points $n \in \{3,000, 5,000, \ldots, 11,000\}$.

We tune `BB` and `SIS-ENET` as described in the sparse linear regression experiment preceding this one (Section 5.5), under the following modifications. For `BB`, we now cross-validate the hyperparameter γ_m within each subproblem and increase the time limit for the reduced problem to 15 minutes.

In Figure 4, we show that `BB` outperforms `SIS-ENET` as the number of samples increases, in terms of both support recovery accuracy and out-of-sample predictive performance, and solves sparse logistic regression problems with 1 million features and $11,000$ samples in less than five hours. This experiment illustrates that `BB`

performs equally well in classification problems, whereby the logistic loss is used instead of the least squares loss.



(a) Support recovery accuracy

(b) Support recovery false alarm rate

(c) Out-of-sample AUC

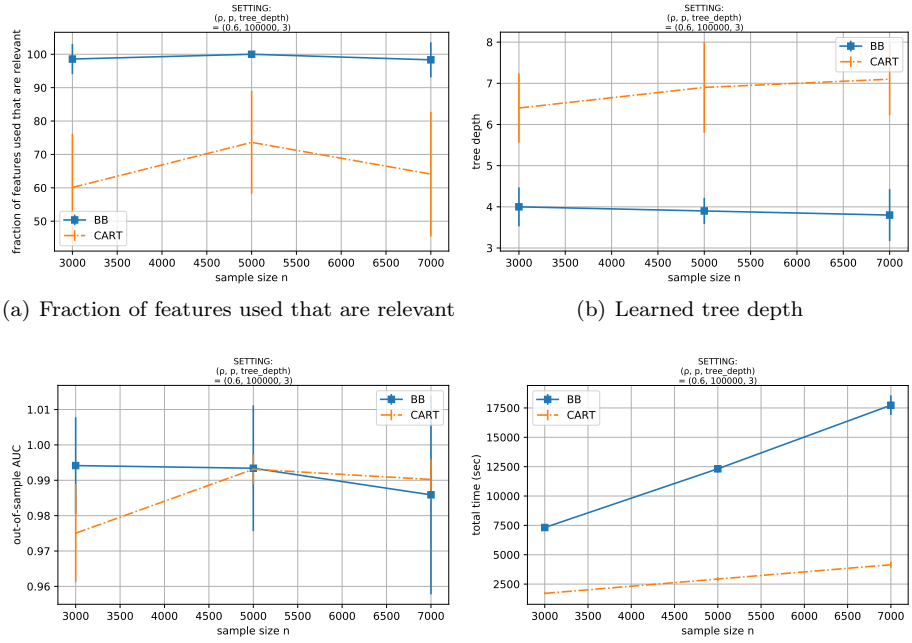
(d) Computational Time (sec)

Fig. 4: Scalability with the number of samples for sparse logistic regression.

Classification Trees. We consider a classification tree problem with $p = 100,000$ features, tree depth of $D = 3$ and $k = 7$ relevant features, and correlation $\rho = 0.6$. By considering a much simpler ground truth tree than in Section 5.4, the methods under investigation will hopefully be able to learn a tree that is closer to the truth. We vary the number of data points $n \in \{3,000, 5,000, 7,000\}$.

We tune `BB` and `CART` as described in the classification tree experiment in Section 5.4, under the following modifications. For `BB`, we now select $\beta = 0.5$ and solve $M = 10$ subproblems (i.e., we solve fewer, larger subproblems compared to Section 5.4).

The results are presented in Figure 5. Both methods achieve near perfect out-of-sample AUC and `BB` is computationally more intensive; this is likely due to the fact that `OCT`, which is used to solve the reduced problem, is more sensitive to the number of samples in the data. Nevertheless, `BB` results in trees that are much simpler and much closer to the ground truth, in that the fraction of features used that are relevant is close to 100% and the learned tree's depth is, on average, within 1 of the ground truth tree's depth.



(a) Fraction of features used that are relevant

(b) Learned tree depth

(c) Out-of-sample AUC

(d) Computational Time (sec)

Fig. 5: Scalability with the number of samples for classification trees.

5.6 Comparison with Exact Methods Applied to the Entire Feature Set

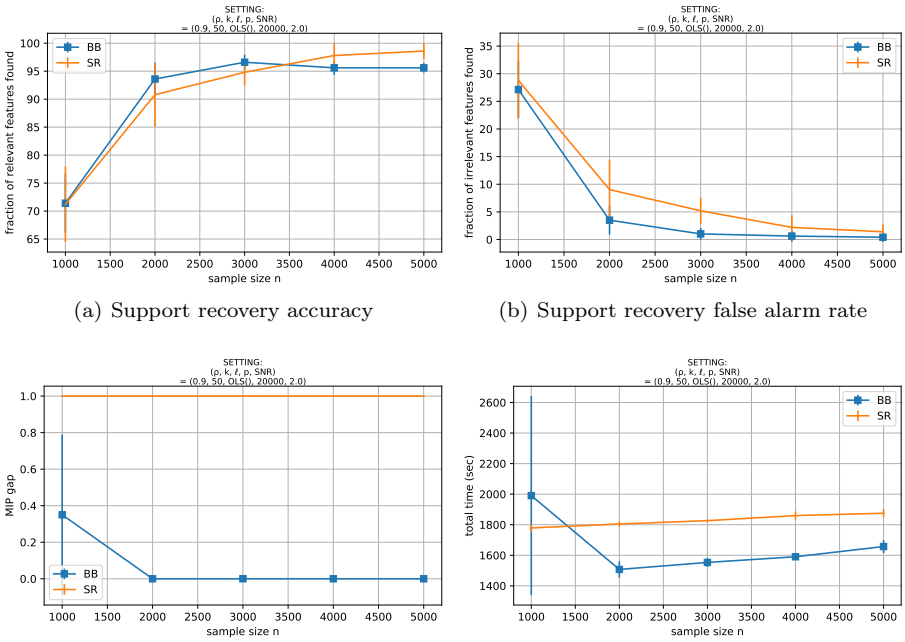
In this experiment, we compare `BB` with `SR` or `OCT` applied to the entire feature set in problems that are sufficiently small and `SR` or `OCT` scale. We show that, in such regimes, `BB` competes with optimal or near-optimal solutions, while substantially reducing the computational time and/or MIO optimality gap.

Sparse Linear Regression. We consider a sparse linear regression problem with $p = 20,000$ features, $k = 50$ relevant features, SNR $= 2$, and correlation $\rho = 0.9$. We vary the number of data points $n \in \{1,000, 2,000, \ldots, 5,000\}$.

We tune `BB` as described in the first experiment in Section 5.4, under the following modifications. For `BB`, we now select the `screen` function's parameter α such that all but $5,000$ features are eliminated, we set $\beta = 0.4$, and solve $M = 10$ subproblems. We compare `BB` with `SR` applied to the entire feature set and tuned exactly as when solving the reduced problem in `BB`.

Figure 6 reports the support recovery accuracy, MIO optimality gap, and computational time of each method as function of the sample size n. Both methods achieve near perfect support recovery accuracy at similar rates; however, `BB` seems to have an edge when it comes to support recovery false alarm rate. One possible explanation for this is that the first phase in `BB` eliminates irrelevant features that `SR` ends up selecting. Additionally, while the solution returned by `SR` comes with no

optimality guarantee, the optimality gap for BB quickly drops to 0 as the number of samples increases, albeit for the reduced problem. In terms of computational time, BB indeed performs effective feature selection and enables us to solve the reduced problem's sparse regression MIO formulation to near-optimality faster.



(a) Support recovery accuracy

(b) Support recovery false alarm rate

(c) Optimality Gap

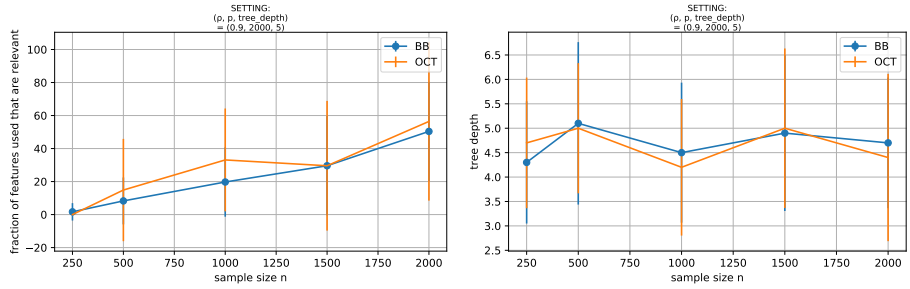
(d) Computational Time (sec)

Fig. 6: Comparison with sparse linear regression applied to the entire feature set.

Classification Trees. We consider a classification tree problem with $p = 2,000$ features, tree depth of $D = 5$ and $k = 31$ relevant features, and correlation $\rho = 0.9$. We vary the number of data points $n \in \{250, 500, 1,000, 1,500, 2,000\}$.

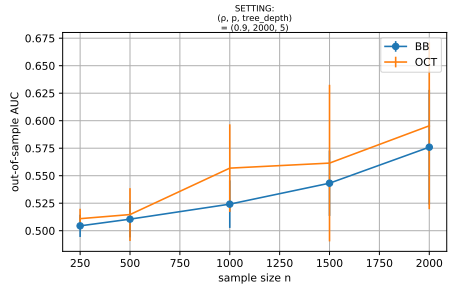
We tune BB as described in the classification tree experiment in Section 5.4, under the following modifications. Since our focus now is solely on feature selection, we incorporate the screening step into BB. Thus, we select $\alpha = 0.5$, we screen features and construct subproblems using the logistic loss, we set $\beta = 0.5$, and solve $M = 10$ subproblems.

We present the results in Figure 7. In terms of support recovery and structure of the learned tree, the two methods performs similarly. Although OCT has a slight edge in terms of predictive power, the gains of BB in terms of computational time are tremendous.

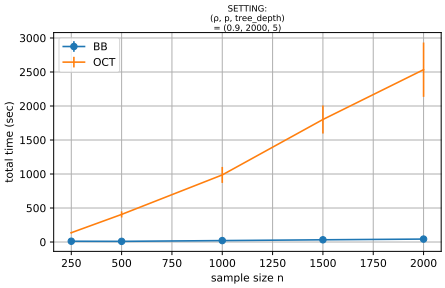


(a) Fraction of features used that are relevant

(b) Learned tree depth



(c) Out-of-sample AUC



(d) Computational Time (sec)

Fig. 7: Comparison with optimal classification trees applied to the entire feature set.

5.7 Sensitivity to the Backbone Method's Hyperparameters and Components

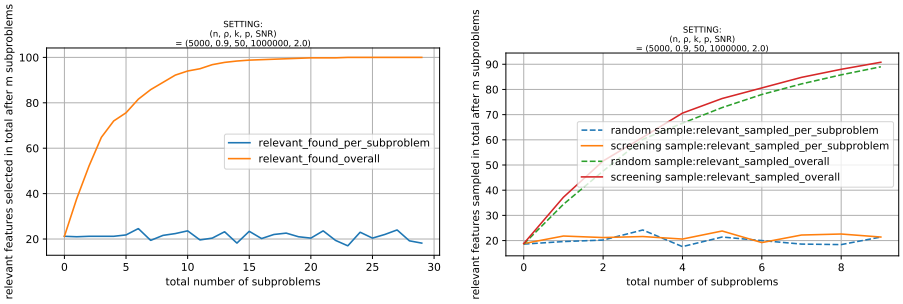
In the remainder of this section, we present a detailed analysis on the sensitivity of `BB` to its components and hyperparameters in the context of regression. The sensitivity analysis for the decision tree problem leads to near-identical conclusions, so we do not include it in the paper.

We consider a sparse linear regression problem with $n = 5,000$ data points, $p = 10^6$ features, $k = 50$ relevant features, SNR $= 2$, and correlation $\rho = 0.9$. Unless stated otherwise, we set the parameters of `BB` as follows: we use the `screen` function described in Algorithm 2 and set $\alpha = 0.01$ so that all but 10^4 features are eliminated; we construct subproblems using the `construct_subproblems` function (Algorithm 3) with $M = 10$ and $\beta = 0.2$; we impose a maximum allowable backbone size of $B_{\max} = 1,000$ features; we solve the subproblems using `SR-REL` and the reduced problem using `SR` with a time limit of 5 minutes.

Number of Subproblems. In this experiment, we study the backbone accuracy (i.e., fraction of relevant features that are included in the backbone set) as function of the subproblem number $m \in [M] = [30]$, that is, we report the accuracy for the same run of `BB`, after solving the first subproblem, after solving the second

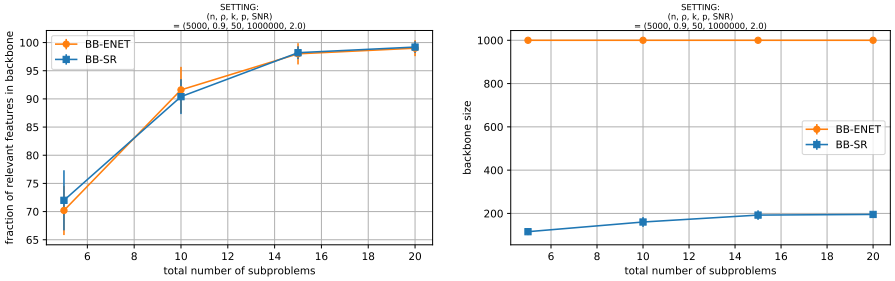
subproblem, and so forth. The results are presented in Figure 8(a). We make the following remarks:

- The changecolor line corresponds to the fraction of relevant features that are selected (i.e., added to the backbone set) in the m-th subproblem, as function of the subproblem number m. Since the m-th subproblem's feature set consists of a fraction $\beta = 0.2$ of the total number $\alpha p = 10^4$ of features, we would expect to select a fraction β of the relevant features in the m-th subproblem (assuming that we solve subproblems perfectly). We observe that, on average, the fraction of relevant features selected in the m-th subproblem is slightly higher; this is due to the more informed sampling scheme used in the `construct_subproblems` function (Algorithm 3).
- The orange line corresponds to the fraction of relevant features that have been selected overall (i.e., across all m subproblems) after solving the m-th subproblem, as function of the subproblem number m. With all other backbone parameters being fixed, as the number of subproblems increases, the backbone accuracy increases and stabilizes after few subproblems, at the expense of an increased computational time. In this case, the backbone set achieves perfect accuracy after $M = 15$ subproblems.



(a) Number of Subproblems: Fraction of relevant features found in each subproblem and in total

(b) Function `construct_subproblems`: Fraction of relevant features sampled in each subproblem and in total



(c) Function `solve_subproblem`: Backbone accuracy

(d) Function `solve_subproblem`: Backbone size

Fig. 8: Hyperparameter M and components of the backbone method.

Function *construct_subproblems.* In this experiment, we examine the impact of the `construct_subproblems` function on `BB`. We compare two approaches for the `construct_subproblems` function. The first one is based on constructing subproblems' feature sets by sampling features uniformly at random, as per the theoretical analysis in Section 3.2 and in Appendix A; we call this approach `random_sample`. The second one is the non-uniformly random sampling approach described in Section 2.3; we call this approach `screening_sample`. The results are presented in Figure 8(b). The dashed lines correspond to the fraction of relevant features that are sampled in the m-th subproblem's feature set and overall after m subproblems (across all m subproblems), as function of the subproblem number m, under the `random_sample` approach; the solid lines correspond to the `screening_sample` approach. With all other backbone parameters being fixed, we observe the following:

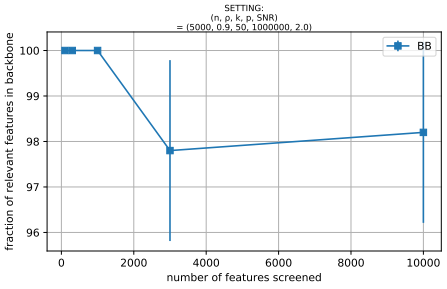
- The `screening_sample` approach samples a slightly larger number of relevant features in the feature set of each subproblem.
- The `screening_sample` approach samples a notably larger number of relevant features across all subproblems. This is due to the fact that, when the `screening_sample` approach does not sample a relevant feature in a subproblem's feature set, it is more likely to sample another relevant feature in its place; this is not the case for the `random_sample` approach.

Function *solve_subproblem.* In this experiment, we explore the impact of the `solve_subproblem` function on `BB`. We compare two approaches for the `solve_subproblem` function: in the first approach, we use `SR-REL` to solve subproblems, whereas in the second, we use `ENET` (we set $\mu = 1$ so that the pure lasso, which generally leads to sparser models compared to the elastic net, is used). With all other backbone parameters being fixed, we observe the following:

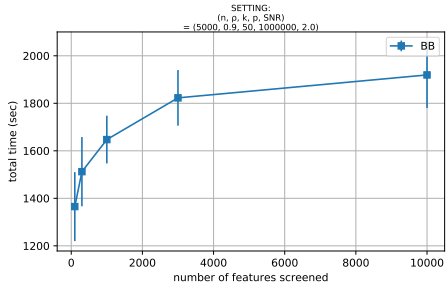
- The two approaches perform comparably in terms of the backbone accuracy (Figure 8(c)).
- The `ENET`-based approach results in a large backbone set, which, in fact, exceeds the limit of $B_{\max} = 1,000$ features. We conclude that this approach does not produce sufficiently sparse solutions to the subproblems (Figure 8(d)).

Number of Features Eliminated by *screen.* In this experiment, we test the impact on the backbone set of the hyperparameter α, which determines what fraction of the features will be eliminated by the `screen` function. Figures 9(a) and 9(b) indicate that a smaller value of α (which results in fewer features surviving the screening step) is beneficial, provided that no relevant features are missed during this step. With all other backbone parameters being fixed, as α increases, it becomes more likely to miss a relevant feature from the backbone set and it takes more to solve the subproblems (as subproblems are both bigger and harder).

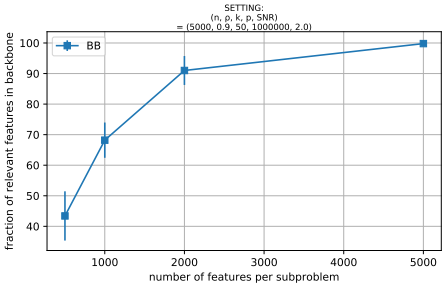
Number of Features per Subproblem. In this experiment, we test the impact on the backbone set of the subproblem size, which we control through the hyperparameter β. Figures 9(c) and 9(d) indicate that a value $\beta \approx 0.5$ is ideal for the problems that we consider. With all other backbone parameters being fixed, as β increases, we observe that the computational time slightly increases, except when the subproblem size is too small, and the backbone accuracy also increases. Intuitively, we want β to be large enough, so that enough signal is contained in the subproblems, and small enough, so that the subproblems can be solved fast.



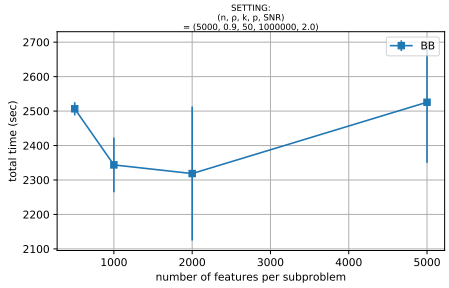
(a) Number of Features Eliminated by `screen`: Backbone accuracy



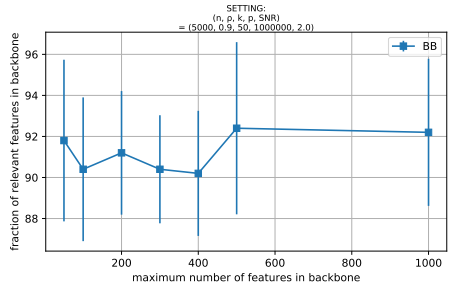
(b) Number of Features Eliminated by `screen`: Computational time (sec)



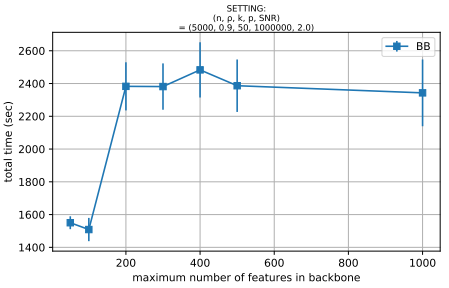
(c) Number of Features per Subproblem: Backbone accuracy



(d) Number of Features per Subproblem: Computational time (sec)



(e) Maximum Backbone Size: Backbone accuracy



(f) Maximum Backbone Size: Computational time (sec)

Fig. 9: Hyperparameters α, β, $B_{\max}$.

Maximum Backbone Size. In this experiment, we explore the impact on the backbone set of the maximum backbone size, controlled by the hyperparameter $B_{\max}$. Figures 9(e) and 9(f) indicate that, with all other backbone parameters being fixed, as $B_{\max}$ increases, the backbone accuracy slightly increases; the computational time is not affected, except when the maximum backbone size is too small. When this is the case, the reduced problem can be solved very fast and therefore the overall computational time drops; this, however, comes with a higher risk of not including relevant features in the backbone set. We also remark that

the maximum backbone size and number of iterations of the hierarchical backbone algorithm (Algorithm 1) are connected, since decreasing the maximum backbone size will likely result in more iterations.

6 Computational Results on Real-World Data

In this section, we empirically evaluate the backbone method on real-world datasets and compare its performance with various baselines and state-of-the-art alternatives. The metrics and algorithms/software that we use are the same as those outlined in Sections 5.2 and 5.3, respectively.

6.1 Datasets

We experiment on 2 regression and 2 classification datasets from the UCI machine learning repository (Dua and Graff, 2017):

- *Communities and Crime:* This is a regression problem; the goal is to predict the crime rate in various communities in the US (Redmond and Baveja, 2002). We remove all features whose values are missing in more than 10 data points; then, we remove all data points that still have any missing value. The resulting dataset consists of $n = 1,993$ data points and $p = 100$ features. The original dataset is available at `https://archive.ics.uci.edu/ml/datasets/Communities+and+Crime`.
- *Housing:* This is a regression problem; the goal is to predict housing prices in Boston. The original dataset has no missing values; we expand the dataset by adding squared for all features (except for the binary ones) and interaction terms for all pairs of features. The resulting dataset consists of $n = 506$ data points and $p = 103$ features. The original dataset is available at `https://www.csie.ntu.edu.tw/~cjlin/libsvmtools/datasets/regression.html#housing`.
- *Breast Cancer:* This is a binary classification problem; the goal is to predict whether a breast cancer instance is benign or malignant (Wolberg and Mangasarian, 1990; Zhang, 1992). We perform no preprocessing to the dataset, which consists of $n = 599$ data points and $p = 9$ features. The original dataset is available at `https://archive.ics.uci.edu/ml/datasets/Breast+Cancer+Wisconsin+%28Original%29`.
- *Ionoshphere:* This is a binary classification problem; the goal is to classify radar returns from the ionosphere (Sigillito et al., 1989). We perform no preprocessing to the dataset, which consists of $n = 351$ data points and $p = 33$ features. The original dataset is available at `https://archive.ics.uci.edu/ml/datasets/Ionosphere`.

In the lines of Hazimeh and Mazumder (2020), we append to the data matrix 1,000 random permutations of each (original) feature (column) for both regression and classification instances. By doing so, we have a way to assess the support recovery performance of each method by measuring the fraction of features in the solution that are original features, that is, they were not generated according to the process described above. Moreover, the expanded datasets become truly ultra-high dimensional, which is the regime that we are interested in.

We conduct our experiments as follows. We randomly split, 5 times independently, each original dataset (which we have not expanded yet) into training and testing, at a ratio of $\frac{n_{\text{train}}}{n_{\text{test}}} = 4$. For each split, we expand, 5 times independently, the resulting training and testing sets, using the process outlined above. Therefore, for each split, we obtain 5 different realizations of the noisy features. In total, we conduct 25 experiments per problem. We report both the mean and the standard deviation of each metric.

6.2 Sparse Linear Regression

In this section, we present the results for the regression datasets, on which we apply sparse linear regression methods.

We tune `BB` as follows. We select the `screen` function's parameter α such that all but $10,000$ features are eliminated. We set $\beta = 0.5$ and solve $M = 10$ subproblems. We set $B_{\max} = 500$. We solve the subproblems using `SR-REL`; in the m-th subproblem, we cross-validate 3 values for $k_m \in \{10, 20, 30\}$ and set $\gamma_m = \frac{1}{\sqrt{n_m}}$. We solve the reduced problem using `SR` with a time limit of 5 minutes; we cross-validate 5 values for $k \in \{10, 20, 30, 40, 50\}$ and 5 values for the hyperparameter γ.

We benchmark `BB` against some of the methods outlined in Section 5.3, tuned as explained below. For `SIS-ENET`, we cross-validate the number of features that are eliminated; then, we apply `ENET` to the selected features, tuned as described in Section 3.1; more specifically, we cross-validate 5 values for the hyperparameter $\mu \in [0, 1]$ (the pure lasso model and the ridge regression model are included in the cross-validation procedure); we discard from the final model any feature whose corresponding regressor has magnitude $\leq 10^{-6}$. For `RFE`, we follow the process outlined in Section 5.3 to eliminate features; then, we apply `SR` on the selected features, tuned in the exact same way that we tune `SR` for the reduced problem in `BB`. For `DECO`, we follow the process outlined in Section 5.3; we tune `ENET` applied to the selected features exactly as in `SIS-ENET`. Finally, we also compare against an oracle model, whereby we apply `SR` on the original features (i.e., we exclude all noisy features from the model); therefore, this approach can serve as an upper bound for the performance of the remaining methods.

In Table 1, we present the results for the communities case study, whereas Table 2 reports the results for the housing case study. We make the following observations:

- Out-of-sample R^2: In both case studies, `BB` achieves an increased out-of-sample R^2 compared to `RFE` and `DECO`. In the communities case study, `BB` notably outperforms `SIS-ENET` and approaches the performance of the oracle model. In the housing case study, `SIS-ENET` is particularly effective, in that the screening step manages to eliminate all noisy features, and hence almost matches the performance of `SR-ORACLE`.
- Support recovery accuracy and sparsity: In both case studies, `BB` uses a fraction of 15-20% of original features and results in models with sparsity close to that of `SR-ORACLE`. In terms of the fraction of features used that are noisy, `BB` achieves the lowest rate in the communities case study, but is outperformed by `SIS-ENET` and `RFE` in the housing case study.

– Optimality gap: In both case studies, BB achieves an optimality gap that is comparable to that of SR-ORACLE and hence returns a solution with optimality guarantees, albeit for the reduced problem.
– Computational time: In both case studies, the solution obtained via BB is computed in less than 2 hours, and in time comparable to that of SR-ORACLE. RFE is substantially slower, taking several hours to run, whereas SIS-ENET and DECO, which do not involve applying SR, are computed in less than 2 minutes.

	R^2	SR-ACC	SR-FA	Sparsity	OG	time (sec)
SR-ORACLE	0.649 (0.017)	32.4 (13.626)	0.0 (0.0)	32.4 (13.626)	0.34 (0.375)	3565.03 (387.337)
SIS-ENET	0.556 (0.035)	43.84 (10.907)	76.773 (8.591)	299.28 (350.675)	-	31.061 (9.529)
RFE	0.57 (0.018)	11.4 (2.082)	71.474 (5.188)	39.96 (0.2)	-	54388.664 (11261.57)
DECO	0.573 (0.031)	29.12 (3.528)	67.178 (12.332)	103.32 (46.992)	-	108.263 (29.504)
BB	0.615 (0.024)	18.36 (4.734)	53.647 (11.718)	40.36 (7.353)	0.147 (0.305)	2505.225 (910.012)

Table 1: Results for the communities case study.

	R^2	SR-ACC	SR-FA	Sparsity	OG	T
SR-ORACLE	0.864 (0.05)	41.553 (10.305)	0.0 (0.0)	42.8 (10.614)	0.475 (0.425)	3928.42 (272.247)
SIS-ENET	0.863 (0.038)	50.563 (11.062)	0.0 (0.0)	52.08 (11.394)	-	6.144 (2.066)
RFE	0.712 (0.064)	15.961 (3.761)	58.453 (10.988)	40.4 (4.546)	-	13462.06 (2958.84)
DECO	0.757 (0.055)	27.379 (2.831)	73.791 (14.516)	140.48 (71.93)	-	28.402 (8.599)
BB	0.765 (0.046)	16.66 (3.715)	64.464 (9.278)	48.76 (3.431)	0.531 (0.445)	4597.401 (1318.081)

Table 2: Results for the housing case study.

6.3 Classification Trees

In this section, we present the results for the classification datasets, on which we apply classification tree methods.

We tune BB as follows. We select the screen function's parameter α such that all but $1,000$ features are eliminated. We set $\beta = 0.5$ and solve $M = 15$ subproblems. We set $B_{\max} = 100$. We solve the subproblems using CART; in the m-th subproblem, we cross-validate $D_m \in \{2, 3, 4, 5\}$ (we also cross-validate the minbucket and complexity parameter of CART). We solve the reduced problem using OCT; we cross-validate $D \in \{3, 4, 5, 6, 7, 8\}$ (we also cross-validate the minbucket and complexity parameter of OCT).

We benchmark BB against some of the methods outlined in Section 5.3, tuned as explained below. The first two baselines are CART, tuned in the exact same way as OCT in solving the reduced problem for BB, and RF, in which we use 100 trees, we impose no depth limit for the trees (which is common in practice), and we cross-validate the number of features that are considered in each split in each tree between 5 values. We also benchmark against RFE, which is tuned as explained in Section 5.3. Finally, we also compare against an oracle model, whereby we apply OCT to the original features (i.e., we exclude all noisy features from the model); therefore, this approach can serve as an upper bound for the performance of the remaining methods.

In Table 3, we present the results for the breast cancer case study, whereas Table 4 reports the results for the ionosphere case study. We make the following observations:

- Out-of-sample AUC: In the breast cancer case study, `BB` outperforms `CART` and `RFE`, matches the performance of `OCT-ORACLE`, and achieves a 0.03 lower AUC compared to `RF`, while still outputting a single, interpretable tree. In the ionosphere case study, the feature selection based methods, namely, `BB` and `RFE`, outperform `OCT-ORACLE` and `CART`, and compete with `RF`, having a 0.02 lower AUC.
- Support recovery accuracy and sparsity: In both case studies, `BB` uses more original features compared to `CART` and `RFE`, and results in models that are sparser than those obtained via `OCT-ORACLE`. In terms of the fraction of features used that are noisy, `BB` outperforms `CART` and is outperformed by `RFE`, which generally results in very shallow trees that use the smallest number of features.
- Computational time: In both case studies, `BB`'s computational time is comparable with that of `CART` and `OCT-ORACLE`. Compared to `RF` and `RFE`, `BB` is approximately 5 times faster in the breast cancer case study and more than 30 times faster in the ionosphere case study.

	AUC	**SR-ACC**	**SR-FA**	**Sparsity**	**T**
`OCT-ORACLE`	0.943 (0.023)	73.778 (11.055)	0.0 (0.0)	6.64 (0.995)	1.364 (0.091)
`CART`	0.917 (0.031)	28.444 (5.629)	55.353 (18.613)	6.4 (2.102)	6.167 (0.478)
`RF`	0.972 (0.014)	-	-	-	68.593 (6.154)
`RFE`	0.933 (0.014)	28.444 (5.629)	20.711 (26.468)	3.84 (2.055)	42.121 (5.391)
`BB`	0.941 (0.022)	34.222 (9.58)	46.252 (13.927)	5.92 (1.498)	9.815 (0.489)

Table 3: Results for the breast cancer case study.

	AUC	**SR-ACC**	**SR-FA**	**Sparsity**	**T**
`OCT-ORACLE`	0.864 (0.033)	25.939 (6.724)	0.0 (0.0)	8.56 (2.219)	2.639 (0.161)
`CART`	0.844 (0.021)	6.061 (0.0)	61.825 (11.906)	5.8 (1.893)	31.731 (3.009)
`RF`	0.891 (0.023)	-	-	-	314.683 (42.199)
`RFE`	0.877 (0.017)	6.303 (0.839)	11.2 (20.478)	2.56 (1.044)	801.467 (152.225)
`BB`	0.871 (0.042)	12.606 (1.134)	14.667 (10.887)	4.92 (0.493)	11.399 (0.25)

Table 4: Results for the ionosphere case study.

7 Concluding Remarks

In this paper, we developed the backbone method, a novel framework that can be used to train a variety of sparse machine learning models. As we showed, the backbone method can accurately and effectively sparsify the set of possible solutions and, as a result, the MIO formulation that exactly models the learning problem can be solved fast for ultra-high dimensional problems. We gave concrete examples of problems where the backbone method can be applied and discussed

in detail the implementation details for the sparse regression problem and the decision tree problem. For the sparse regression problem, we showed that, under certain assumptions and with high probability, the backbone set consists of the true relevant features.

As far as the sparse regression problem is concerned, our computational study illustrated that the backbone method outperforms or competes with state-of-the-art methods for ultra-high dimensional problems, accurately scales to problems with $p \sim 10^7$ features in minutes and $p \sim 10^8$ features in hours, and drastically reduces the problem size in problems with $p \sim 10^5$ features hence making the work of exact methods much easier. Regarding the decision tree problem, the backbone method scales to problems with $p \sim 10^5$ features in minutes and, assuming that the underlying problem is indeed sparse (in that only few features are involved in splits in the decision tree), outperforms CART, and competes with random forest, while still outputting a single, interpretable tree. In problems with $p \sim 10^3$, the backbone method can accurately filter the feature set and compute decision trees that match those obtained by applying the state-of-the-art optimal trees framework to the entire problem.

Finally, as we discussed throughout the paper, the backbone method is generic and can be directly applied to any sparse supervised learning model; examples include sparse support vector machines and sparse principal component analysis. In addition, our proposed framework can be extended to non-supervised sparse machine learning problems, such as the clustering problem. Furthermore, the backbone construction phase can naturally be implemented in a parallel/distributed fashion.

A Proof of Theorem 1

A.1 Model & Assumptions.

We consider the following model:

- Random design matrix $\boldsymbol{X} \in \mathbb{R}^{n \times p}$ such that each row $\boldsymbol{x}_i, i \in [n]$, is an iid copy of the random vector $X = (X_1, ..., X_p) \sim \mathcal{N}(0, \boldsymbol{I}_p)$.
- Fixed but unknown regressors $\boldsymbol{\beta} \in \{-1, 0, 1\}^p$ that satisfy $\|\boldsymbol{\beta}\|_0 \leq k$. Let $\mathcal{S}^{\text{true}}$ denote the set of indices that correspond to the support of the true regressor $\boldsymbol{\beta}$.
- Response vector $\boldsymbol{y} = \boldsymbol{X}\boldsymbol{\beta} + \boldsymbol{\varepsilon}$ where the noise term $\boldsymbol{\varepsilon}$ consists of iid entries $\varepsilon_i \sim \mathcal{N}(0, \sigma^2), i \in [n]$. Thus, each entry in $\boldsymbol{Y}$ is distributed as $Y = X^\top \boldsymbol{\beta} + \varepsilon \sim \mathcal{N}(0, k + \sigma^2)$ (further assuming $\|\boldsymbol{\beta}\|_0 = k$).

We further assume that $p > n \geq k$ and $\log p = O(n^\xi)$ for some $\xi \in (0, 1)$. When we make asymptotic arguments, we take $p \to \infty$. Given $\mathcal{S} \subseteq [p]$, we denote by $X_\mathcal{S}$ the random vector constructed by selecting from X the entries that are in $\mathcal{S}$. We let $\mathcal{S}_k^p = \{\boldsymbol{s} \in \{0, 1\}^p : \sum_{j \in [p]} s_j \leq k\}$. We analyze a simplified version of the backbone method, which we outline below:

1. Screening step: select top αp features based on their empirical marginal correlation with the response $s_j = \boldsymbol{X}_j^\top \boldsymbol{y}$. Parameter α satisfies $0 < \alpha \leq 1$.
2. Construct M subproblems. In each subproblem, sample uniformly at random $\beta\alpha p$ features among those that survived in Step 1. Parameter β satisfies $0 < \beta \leq 1$. Note that, within each subproblem, features are sampled without replacement, i.e., each feature can appear at most once in a subproblem's feature set.
3. For $m = 1, \ldots, M$, approximately solve sparse regression via its boolean relaxation on the m-th selected subset of features. Let $\mathcal{S}^m$ be the solution to the m-th subproblem.
4. Define the backbone set $\mathcal{B} = \cup_{m=1}^M \mathcal{S}^m$.

Our goal is to show that, with high probability, $\mathcal{S}^{\text{true}} \subseteq \mathcal{B}$.

A.2 High-Level Approach

We consider the following events:

- $\mathcal{E}$: the simplified algorithm fails, namely, $\exists j \in [p]$ such that $j \in \mathcal{S}^{\text{true}}$ and $j \notin \mathcal{B}$.
- $\mathcal{E}_1$: any relevant feature is missed in Step 1.
- $\mathcal{E}_{2,m}$: sparse regression's boolean relaxation fails to recover the relevant features in subproblem $m \in [M]$. Let ε_2 be an upper bound on the probability of this event across all subproblems, i.e., $P(\mathcal{E}_{2,m}) \leq \varepsilon_2,\ \forall m \in [M]$.
- $\mathcal{E}_{3,j}$: relevant feature $j \in \mathcal{S}^{\text{true}}$ is selected in Step 1 but is not selected after Steps 2 and 3.

We fix an arbitrary feature $j \in \mathcal{S}^{\text{true}}$. The number of subproblems in which feature j is sampled follows a binomial distribution with parameters M (number of subproblems) and β (probability of sampling feature j in any subproblem); we denote $Z_j \sim \text{Bino}(M, \beta)$. Then, by the union bound, the probability of failure of the simplified algorithm is at most:

$$\begin{aligned} P(\mathcal{E}) &= P\left[\mathcal{E}_1 \cup \left(\cup_{j \in S^{\text{true}}} \mathcal{E}_{3,j}\right)\right] \\ &\leq P\left(\mathcal{E}_1\right) + \sum_{j \in S^{\text{true}}} P\left(\mathcal{E}_{3,j}\right) \\ &\leq P\left(\mathcal{E}_1\right) + \sum_{j \in S^{\text{true}}} \sum_{m=0}^{M} \varepsilon_2^m P\left(Z_j = m\right) \\ &= P\left(\mathcal{E}_1\right) + k \sum_{m=0}^{M} \varepsilon_2^m P\left(Z = m\right), \end{aligned} \tag{10}$$

where $Z \sim \text{Bino}(M, \beta)$.

A.3 Analysis of Event $\mathcal{E}_1$

Fan and Lv (2008) show that, under five conditions and provided that α is sufficiently large, Step 1 succeeds with high probability. The model described in Section A.1 trivially satisfies four out of these five conditions (e.g., on the eigenvalues of the data covariance matrix, on the magnitude of the nonzero regressors, etc.), whereas the fifth condition requires that $p > n$, $\log p = O(n^{\xi})$ for some $\xi \in (0, 1)$. Then, the following theorem holds:

Theorem 2. *Assume that the data is generated according to the model described in Section A.1. Then, $\exists \phi < 1$ such that, when $\alpha = O\left(\frac{n^{1-\phi}}{p}\right)$, we have $P(\mathcal{E}_1) = O\left[\exp\left(-\frac{n}{\log n}\right)\right]$.*

A.4 Analysis of Events $\mathcal{E}_{2,m}$

We momentarily turn to the original sparse regression problem. Let $\mathcal{S}$ denote the set of features that sparse regression's boolean relaxation selects when applied to the entire problem (where all p features are considered). Bertsimas and Van Parys (2020) prove the following theorem:

Theorem 3. *Let $\gamma = \frac{1}{n}$, $p - k > k$, and suppose the data are generated as discussed in Section A.1. Then, for all $\theta \geq 1$, for samples $n \geq \theta(\sigma^2 + 2k)\log(p-k)$, we have $\mathbb{P}(\mathcal{S} \neq \mathcal{S}^{true}) = O\left(e^{-\theta}\right)$.*

In our setting, we cannot directly apply Theorem 3 since, within each subproblem, we do not necessarily sample all relevant features. We consider an arbitrary subproblem $m \in [M]$ and denote by $\mathcal{P}^m$ the set of features sampled in the m-th subproblem's feature set. When we solve sparse regression's boolean relaxation for the m-th subproblem, we only observe the relevant features in $\mathcal{S}^{\text{true}} \cap \mathcal{P}^m$. Any relevant feature j such that $j \in \mathcal{S}^{\text{true}}$ and $j \notin \mathcal{P}^m$ is viewed as noise in the m-th subproblem. Therefore, we consider the new noise term

$$\varepsilon' = X^{\top}_{\mathcal{S}^{\text{true}} \cap (\mathcal{P}^m)^c} \boldsymbol{\beta}_{\mathcal{S}^{\text{true}} \cap (\mathcal{P}^m)^c} + \varepsilon,$$

which is the sum of at most $k - k_0$ (with $k_0 = |\mathcal{S}^{\text{true}} \cap \mathcal{P}^m|$) independent $\mathcal{N}(0,1)$ random variables and one $\mathcal{N}(0,\sigma^2)$ random variable. Thus, $\varepsilon' \sim \mathcal{N}(0, k - k_0 + \sigma^2)$.

Let $\mathcal{S}^m$ denote the set of features that sparse regression's boolean relaxation selects when applied to subproblem m. Then, by directly applying the result stated in Theorem 3, we obtain the following lemma:

Lemma 1. *Let $\gamma = \frac{1}{n}$, $\beta\alpha p > 2k_0$, and suppose the data are generated as discussed in Section A.1. Then, for all $\theta \geq 1$, for samples $n \geq \theta(\sigma^2 + 2k)\log(\beta\alpha p) \geq \theta(\sigma^2 + k + k_0)\log(\beta\alpha p - k_0)$, we have*

$$\mathbb{P}(\mathcal{E}_{2,m}) = \mathbb{P}(\mathcal{S}^{true} \cap \mathcal{P}^m \setminus \mathcal{S}^m \neq \emptyset) \leq \mathbb{P}(\mathcal{S}^m \neq \mathcal{S}^{true} \cap \mathcal{P}^m) = O\left(e^{-\theta}\right).$$

Lemma 1 asserts that the probability of not exactly recovering the true support -which is, in fact, more restrictive than our requirement to not miss any relevant feature- decreases exponentially with the parameter θ; as the sample size n increases, we are able to select larger θ and hence obtain tighter guarantees. In the case $k_0 = 0$, we are dealing with a problem with no relevant features, so the upper bound on the probability of error $\mathbb{P}(\mathcal{E}_{2,m})$ is trivially satisfied. Moreover, we note that, since all subproblems are constructed in the same way, the analysis is the same for all $m \in [M]$. This gives the upper bound $\mathbb{P}(\mathcal{E}_{2,m}) \leq \varepsilon_2 = O\left(e^{-\theta}\right)$. Finally, similar to Fan and Lv (2008), to avoid the selection bias in the screening step, we can split the sample in two halves and use the first for Step 1 and the second for Step 3 of the simplified algorithm.

A.5 Analysis of Event $\mathcal{E}_{3,j}$ and Final Result

We now turn back to the last equation in (10). By Theorem 2, the first summand converges to 0. We focus on the second summand and apply the binomial theorem to obtain:

$$\begin{aligned} k\sum_{m=0}^{M} \varepsilon_2^m P\left(Z = m\right) &= k\sum_{m=0}^{M} \varepsilon_2^m \binom{M}{m} \beta^m (1-\beta)^{M-m} \\ &= k(1-\beta)^M \sum_{m=0}^{M} \binom{M}{m} \left(\frac{\varepsilon_2\beta}{1-\beta}\right)^m \\ &= k(1-\beta)^M \left(1 + \frac{\varepsilon_2\beta}{1-\beta}\right)^M \\ &= k\left(1 - \beta + \varepsilon_2\beta\right)^M. \end{aligned} \tag{11}$$

Let us briefly review the quantities that appear in equation (11): k is the number of relevant features; ε_2 is an upper bound on the probability of failure for any subproblem and converges to 0 at rate given by Lemma 1; M denotes the number of subproblems that we solve; β denotes the fraction of the αp screened features that we sample in each subproblem.

M and β are parameters of the algorithm; among the two, we believe it is realistic to only control M, since β depends on the available computational resources (e.g., memory). Therefore, we next determine how M has to be selected as function of k, ε_2, β, so as to guarantee that $k\left(1 - \beta + \varepsilon_2\beta\right)^M \to 0$. To guarantee a $\frac{1}{\alpha p}$ rate of convergence, we pick

$$k\left(1 - \beta + \varepsilon_2\beta\right)^M = O\left(\frac{1}{\alpha p}\right) \tag{12}$$

and, therefore,

$$M = O\left(\frac{\log\left(\alpha p k\right)}{\log\left(\frac{1}{1-\beta+\beta\varepsilon_2}\right)}\right). \tag{13}$$

This completes the proof of Theorem 1.

Acknowledgements We would like to thank Ryan Cory-Wright, Korina Digalaki, Michael Li, Theodore Papalexopoulos, Jean Pauphilet, and Ilias Zadik for fruitful discussions. We are grateful to the referees for their constructive comments.

References

Aghaei S., Gomez A., Vayanos P. (2020). Learning optimal classification trees: Strong max-flow formulations. *arXiv preprint arXiv:200209142*.

Almuallim H., Dietterich T. (1994). Learning boolean concepts in the presence of many irrelevant features. *Artificial Intelligence*, 69(1-2), 279–305.

Atamturk A., Gomez A. (2020). Safe screening rules for l0-regression from perspective relaxations. In: III H. D., Singh A. (eds) *Proceedings of the 37th International Conference on Machine Learning*, PMLR, Proceedings of Machine Learning Research, vol 119, pp. 421–430, `http://proceedings.mlr.press/v119/atamturk20a.html`.

Beale E., Kendall M., Mann D. (1967). The discarding of variables in multivariate analysis. *Biometrika*, 54(3-4), 357–366.

Beck A., Teboulle M. (2009). A fast iterative shrinkage-thresholding algorithm for linear inverse problems. *SIAM Journal on Imaging Sciences*, 2(1), 183–202.

Bennett K., Cristianini N., Shawe-Taylor J., Wu D. (2000). Enlarging the margins in perceptron decision trees. *Machine Learning*, 41(3), 295–313.

Bertsimas D., Copenhaver M. (2018). Characterization of the equivalence of robustification and regularization in linear and matrix regression. *European Journal of Operational Research*, 270(3), 931–942.

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

Bertsimas D., Dunn J. (2019). *Machine learning under a modern optimization lens*. Dynamic Ideas LLC.

Bertsimas D., Van Parys B. (2020). Sparse high-dimensional regression: Exact scalable algorithms and phase transitions. *The Annals of Statistics*, 48(1), 300–323.

Bertsimas D., King A., Mazumder R. (2016). Best subset selection via a modern optimization lens. *The Annals of Statistics*, 44(2), 813–852.

Bertsimas D., Jaillet P., Martin S. (2019). Online vehicle routing: The edge of optimization in large-scale applications. *Operations Research*, 67(1), 143–162.

Bertsimas D., Pauphilet J., Van Parys B., et al. (2020). Sparse regression: Scalable algorithms and empirical performance. *Statistical Science*, 35(4), 555–578.

Bertsimas D., Digalakis Jr. V., Li M., Skali Lami O. (2021). Slowly varying regression under sparsity. `2102.10773`.

Boyd S., Parikh N., Chu E., Peleato B., Eckstein J. (2011). Distributed optimization and statistical learning via the alternating direction method of multipliers. *Foundations and Trends® in Machine Learning*, 3(1), 1–122.

Breiman L. (2001). Random forests. *Machine learning*, 45(1), 5–32.

Breiman L., Friedman J., Olshen R., Stone C. (1984). *Classification and regression trees*. Monterey, CA: Wadsworth and Brooks.

Chen P., Tsai C., Chen Y., Chou K., et al. (2012). A linear ensemble of individual and blended models for music rating prediction. In: *Proceedings of KDD-Cup 2011*, pp. 21–60.

Chen T., Guestrin C. (2016). Xgboost: A scalable tree boosting system. In: *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pp. 785–794.

Dua D., Graff C. (2017). UCI machine learning repository. `http://archive.ics.uci.edu/ml`.

Duran M., Grossmann I. (1986). An outer-approximation algorithm for a class of mixed-integer nonlinear programs. *Mathematical Programming*, 36(3), 307–339.

Efron B., Hastie T., Johnstone I., Tibshirani R. (2004). Least angle regression. *The Annals of Statistics*, 32(2), 407–499.

Efroymson M. (1966). Stepwise regression–a backward and forward look. *Eastern Regional Meetings of the Institute of Mathematical Statistics*.

Fan J., Li R. (2001). Variable selection via nonconcave penalized likelihood and its oracle properties. *Journal of the American Statistical Association*, 96(456), 1348–1360.

Fan J., Lv J. (2008). Sure independence screening for ultrahigh dimensional feature space. *Journal of the Royal Statistical Society: Series B (Statistical Methodology)*, 70(5), 849–911.

Fan J., Lv J. (2018). Sure independence screening. *Wiley StatsRef: Statistics Reference Online*.

Fan J., Song R. (2010). Sure independence screening in generalized linear models with np-dimensionality. *The Annals of Statistics*, 38(6), 3567–3604.

Fan J., Samworth R., Wu Y. (2009). Ultrahigh dimensional feature selection: beyond the linear model. *Journal of Machine Learning Research*, 10, 2013–2038.

Fan J., Feng Y., Song R. (2011). Nonparametric independence screening in sparse ultra-high-dimensional additive models. *Journal of the American Statistical Association*, 106(494), 544–557.

Friedman J., Hastie T., Tibshirani R. (2010). Regularization paths for generalized linear models via coordinate descent. *Journal of Statistical Software*, 33(1), 1–22.

Friedman J., Hastie T., Tibshirani R. (2020). glmnet: Lasso and elastic-net regularized generalized linear models. *R package version 4*.

Gamarnik D., Zadik I. (2017). High-dimensional regression with binary coefficients. estimating squared error and a phase transition. *arXiv preprint arXiv:170104455*.

Gurobi Optimization Inc. (2016). *Gurobi Optimizer Reference Manual*, `http://www.gurobi.com`.

Guyon I., Elisseeff A. (2003). An introduction to variable and feature selection. *Journal of Machine Learning Research*, 3, 1157–1182.

Guyon I., Weston J., Barnhill S., Vapnik V. (2002). Gene selection for cancer classification using support vector machines. *Machine learning*, 46(1), 389–422.

Hastie T., Tibshirani R., Wainwright M. (2015). *Statistical learning with sparsity: the lasso and generalizations*. CRC press.

Hazimeh H., Mazumder R. (2020). Fast best subset selection: Coordinate descent and local combinatorial optimization algorithms. *Operations Research*, 68(5), 1517–1537.

Hazimeh H., Mazumder R., Saab A. (2020). Sparse regression at scale: Branch-and-bound rooted in first-order optimization. *arXiv preprint arXiv:200406152*.

Ho T. (1998). The random subspace method for constructing decision forests. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 20(8), 832–844.

Hocking R., Leslie R. (1967). Selection of the best subset in regression analysis. *Technometrics*, 9(4), 531–540.

Interpretable AI (2020). *Interpretable AI Documentation*, `https://www.interpretable.ai`.

Kenney A., Chiaromonte F., Felici G. (2018). Efficient and effective l_0 feature selection. *arXiv preprint arXiv:180802526*.

Koziarski M., Krawczyk B., Woźniak M. (2017). The deterministic subspace method for constructing classifier ensembles. *Pattern Analysis and Applications*, 20(4), 981–990.

Li J., Cheng K., Wang S., Morstatter F., et al. (2017). Feature selection: A data perspective. *ACM Computing Surveys (CSUR)*, 50(6), 1–45.

Liu W., Tsang I. (2017). Making decision trees feasible in ultrahigh feature and label dimensions. *The Journal of Machine Learning Research*, 18(81), 1–36.

McSherry F., Talwar K. (2007). Mechanism design via differential privacy. In: *48th Annual IEEE Symposium on Foundations of Computer Science (FOCS'07)*, pp. 94–103, DOI 10.1109/FOCS.2007.66.

Natarajan B. (1995). Sparse approximate solutions to linear systems. *SIAM Journal on Computing*, 24(2), 227–234.

Ng A. (1998). On feature selection: Learning with exponentially many irrelevant features as training examples. In: *Proceedings of the Fifteenth International Conference on Machine Learning*, Morgan Kaufmann Publishers Inc., pp. 404–412.

Ni L., Fang F. (2016). Entropy-based model-free feature screening for ultrahigh-dimensional multiclass classification. *Journal of Nonparametric Statistics*, 28(3), 515–530.

Pedregosa F., Varoquaux G., Gramfort A., Michel V., et al. (2011). Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12, 2825–2830.

Pilanci M., Wainwright M. J., El Ghaoui L. (2015). Sparse learning via boolean relaxations. *Mathematical Programming*, 151(1), 63–87.

Redmond M., Baveja A. (2002). A data-driven software tool for enabling cooperative information sharing among police departments. *European Journal of Operational Research*, 141(3), 660–678.

Schneider J., Froschhammer C., Morgenstern I., Husslein T., Singer J. (1996). Searching for backbones—an efficient parallel algorithm for the traveling salesman problem. *Computer Physics Communications*, 96(2-3), 173–188.

Sigillito V. G., Wing S. P., Hutton L. V., Baker K. B. (1989). Classification of radar returns from the ionosphere using neural networks. *Johns Hopkins APL Technical Digest*, 10(3), 262–266.

Song Q., Liang F. (2015). A split-and-merge bayesian variable selection approach for ultra-high dimensional regression. *Journal of the Royal Statistical Society: Series B (Statistical Methodology)*, 77(5), 947–972.

Tibshirani R. (1996). Regression shrinkage and selection via the lasso. *Journal of the Royal Statistical Society: Series B (Methodological)*, 58(1), 267–288.

Walsh T., Slaney J. (2001). Backbones in optimization and approximation. In: *Proceedings of the Seventeenth International Joint Conference on Artificial Intelligence*, pp. 254–259.

Wang X., Dunson D., Leng C. (2016). Decorrelated feature space partitioning for distributed sparse regression. In: *Advances in Neural Information Processing Systems*, pp. 802–810.

Wolberg W. H., Mangasarian O. L. (1990). Multisurface method of pattern separation for medical diagnosis applied to breast cytology. *Proceedings of the national academy of sciences*, 87(23), 9193–9196.

Xie W., Deng X. (2020). Scalable algorithms for the sparse ridge regression. *SIAM Journal on Optimization*, 30(4), 3359–3386.

Xu H., Caramanis C., Mannor S. (2009). Robust regression and lasso. In: *Advances in Neural Information Processing Systems*, pp. 1801–1808.

Yang J., Mahoney M., Saunders M., Sun Y. (2016). Feature-distributed sparse regression: a screen-and-clean approach. In: *Advances in Neural Information Processing Systems*, pp. 2712–2720.

Zhang C. (2010). Nearly unbiased variable selection under minimax concave penalty. *The Annals of Statistics*, 38(2), 894–942.

Zhang J. (1992). Selecting typical instances in instance-based learning. In: *Machine Learning Proceedings 1992*, Elsevier, pp. 470–479.

Zhou Y., Porwal U., Zhang C., Ngo H., et al. (2014). Parallel feature selection inspired by group testing. In: *Advances in Neural Information Processing Systems*, pp. 3554–3562.

Zou H., Hastie T. (2005). Regularization and variable selection via the elastic net. *Journal of the Royal Statistical Society: Series B (Statistical Methodology)*, 67(2), 301–320.