Machine Learning for K-Adaptability in Two-Stage Robust Optimization

Julien, Esther; Postek, Krzysztof; Birbil, Illker

DOI
10.1287/ijoc.2022.0314

Publication date
2025

Document Version
Accepted author manuscript

Published in
INFORMS Journal on Computing

MACHINE LEARNING FOR K-ADAPTABILITY IN TWO-STAGE ROBUST OPTIMIZATION

Esther Julien
TU Delft
e.a.t.julien@tudelft.nl

Krzysztof Postek
Independent
Researcher

Ş. İlker Birbil
University of Amsterdam
s.i.birbil@uva.nl

October 16, 2024

ABSTRACT

Two-stage robust optimization problems constitute one of the hardest optimization problem classes. One of the solution approaches to this class of problems is K-adaptability. This approach simultaneously seeks the best partitioning of the uncertainty set of scenarios into K subsets, and optimizes decisions corresponding to each of these subsets. In general case, it is solved using the K-adaptability branch-and-bound algorithm, which requires exploration of exponentially-growing solution trees. To accelerate finding high-quality solutions in such trees, we propose a machine learning-based node selection strategy. In particular, we construct a feature engineering scheme based on general two-stage robust optimization insights that allows us to train our machine learning tool on a database of resolved B&B trees, and to apply it as-is to problems of different sizes and/or types. We experimentally show that using our learned node selection strategy outperforms a vanilla, random node selection strategy when tested on problems of the same type as the training problems, also in case the K-value or the problem size differs from the training ones.

Keywords node selection; clustering; two-stage robust optimization; K-adaptability; machine learning; tree search

1 Introduction

Many optimization problems are affected by data uncertainty caused by errors in the forecast, implementation, or measurement. Robust optimization (RO) is one of the key paradigms to solve such problems, where the goal is to find an optimal solution among the ones that remain feasible for all data realizations within an *uncertainty set* [Ben-Tal et al., 2009]. This set includes all *reasonable* data outcomes.

A specific class of RO problems comprises two-stage robust optimization (2SRO) problems in which some decisions are implemented *before* the uncertain data is known (here-and-now decisions), and other decisions are implemented *after* the data is revealed (wait-and-see decisions). Such a problem can be formulated as

$$\min_{\boldsymbol{x}\in\mathcal{X}} \max_{\boldsymbol{z}\in\mathcal{Z}} \min_{\boldsymbol{y}\in\mathcal{Y}} \left\{\boldsymbol{c}(\boldsymbol{z})^\intercal\boldsymbol{x} + \boldsymbol{d}(\boldsymbol{z})^\intercal\boldsymbol{y} : \boldsymbol{T}(\boldsymbol{z})\boldsymbol{x} + \boldsymbol{W}(\boldsymbol{z})\boldsymbol{y} \leq \boldsymbol{h}(\boldsymbol{z}),\ \forall \boldsymbol{z}\in\mathcal{Z}\right\}, \tag{1}$$

where $\boldsymbol{x} \in \mathcal{X} \subseteq \mathbb{R}^{N_x}$ and $\boldsymbol{y} \in \mathcal{Y} \subseteq \mathbb{R}^{N_y}$ are the here-and-now and wait-and-see decisions, respectively, and $\boldsymbol{z}$ is the vector of initially unknown data belonging to the uncertainty set $\mathcal{Z} \subseteq \mathbb{R}^{N_z}$. Solving problem (1) is difficult in general, since $\mathcal{Z}$ might include an infinite number of scenarios, and hence different values of $\boldsymbol{y}$ might be optimal for different realizations of $\boldsymbol{z}$. In fact, finding optimal $\boldsymbol{x}$ is an NP-hard problem [Guslitzer, 2002]. To address this difficulty, several approaches have been proposed. The first one is to use so-called decision rules which explicitly formulate the second-stage decision $\boldsymbol{y}$ as a function of $\boldsymbol{z}$, and hence the function parameters become first-stage decisions next to $\boldsymbol{x}$; see Ben-Tal et al. [2004]. Another approach is to partition $\mathcal{Z}$ into subsets and to assign a separate copy of $\boldsymbol{y}$ to each of the subsets. The partitioning is then iteratively refined, and the decisions become increasingly *customized* to the outcomes of $\boldsymbol{z}$.

In this paper, we consider a third approach to (1) known as K-adaptability. There, at most K possible wait-and-see decisions $\boldsymbol{y}_1, \ldots, \boldsymbol{y}_K$ are allowed to be constructed, and the decision maker must select one of those. The values of the possible $\boldsymbol{y}_k$'s become the first-stage variables, and the problem boils down to

$$\min_{\boldsymbol{x}\in\mathcal{X}, \boldsymbol{y}\in\mathcal{Y}^K} \max_{\boldsymbol{z}\in\mathcal{Z}} \min_{k\in\mathcal{K}} \left\{\boldsymbol{c}(\boldsymbol{z})^\intercal \boldsymbol{x} + \boldsymbol{d}(\boldsymbol{z})^\intercal \boldsymbol{y}_k : \boldsymbol{T}(\boldsymbol{z})\boldsymbol{x} + \boldsymbol{W}(\boldsymbol{z})\boldsymbol{y}_k \leq \boldsymbol{h}(\boldsymbol{z})\right\}, \tag{2}$$

where $\mathcal{K} = \{1, \ldots, K\}$ and $\mathcal{Y}^K = \bigtimes_{k=1}^K \mathcal{Y}$. Although the solution space of (2) is finite-dimensional, it remains an NP-hard problem. For certain cases, (2) can be equivalently rewritten as a mixed integer linear programming (MILP) model [Hanasusanto et al., 2015].

The above formulation requires that for given $\boldsymbol{x} \in \mathcal{X}$ and $\boldsymbol{z} \in \mathcal{Z}$, there is at least one decision $\boldsymbol{y}_k$, $k \in \mathcal{K}$ satisfying $\boldsymbol{T}(\boldsymbol{z})\boldsymbol{x} + \boldsymbol{W}(\boldsymbol{z})\boldsymbol{y}_k \leq \boldsymbol{h}(\boldsymbol{z})$, and among those one (or more) minimizing the objective. Looking at (2) from the point of view $\boldsymbol{y}_k$, we can say that for each $\boldsymbol{y}_k$, we can identify a subset $\mathcal{Z}_k$ of $\mathcal{Z}$ for which a given $\boldsymbol{y}_k$ is optimal among K selected recourses. The union of sets $\mathcal{Z}_k$, $k \in \mathcal{K}$ is equal to $\mathcal{Z}$ although they need not be mutually disjoint (but a mutually disjoint partition of $\mathcal{Z}$ can be constructed). Consequently, solving (2) involves implicitly (i) clustering $\mathcal{Z}$, and (ii) optimizing the per-cluster decision so that the objective function corresponding to *the most difficult cluster* is minimized. Such simultaneous clustering and per-cluster optimization also occur, for example in retail. A line of K products is to be designed to attract the largest possible group of customers. The customers are clustered into K groups, and the nature of the products is guided by the cluster characteristics.

In this manuscript, we focus on the general MILP K-adaptability case for which the only existing solution approach is the K-adaptability branch-and-bound (K-B&B) algorithm of Subramanyam et al. [2020]. Other methods to solve the K-adaptability problem have been proposed by Hanasusanto et al. [2015] that deals with binary decisions, and Ghahtarani et al. [2023] that assumes integer first-stage decisions. The K-B&B algorithm, as opposed to the top-down partitioning of $\mathcal{Z}$ of Bertsimas and Dunning [2016] or Postek and den Hertog [2016], proceeds by gradually building up discrete subsets $\bar{\mathcal{Z}}_k$ of scenarios. In most practical cases, a solution to (2), where $\boldsymbol{y}_1, \ldots, \boldsymbol{y}_K$ are feasible for large $\bar{\mathcal{Z}}_1, \ldots, \bar{\mathcal{Z}}_K$, is also feasible to the original problem. The problem, however, lies in knowing which scenarios should be grouped together. In other words, a decision needs to be made on which scenarios of $\mathcal{Z}$ should be responded to with the same decision. How well this question is answered, determines the (sub)optimality of $\boldsymbol{y}_1, \ldots, \boldsymbol{y}_K$. In Subramanyam et al. [2020], a search tree is used to determine the best collection (see Section 2 for details). However, this approach suffers from exponential growth.

We introduce a method for learning the best strategy to explore this tree. In particular, we learn which nodes to evaluate next in depth-first search *dives* to obtain good solutions faster. These predictions are made using a supervised machine learning (ML) model. As the input does not range for different instance sizes, or values of K, as will be explained in future sections, the ML model does not require difficult architectures. Standard ML models, such as feed-forward neural networks, random forests, or support vector machines can be used for this work.

Due to the supervised nature, some *oracle* is required to be imitated. In the design of this oracle, we are partly inspired by Monte Carlo tree search (MCTS) [Browne et al., 2012], which is often used for exploring large trees. Namely, the training data is obtained by exploring K-B&B trees via an adaptation of MCTS (see Section 3.4). The scores given to the nodes in the MCTS-like exploration are stored and used as labels in our training data.

In the field of solving MILPs, learning node selection to speed up exploring the B&B tree has been done, *e.g.*, by He et al. [2014]. Here, a node selection policy is designed by imitating an oracle. This oracle is constructed using the optimal solutions of various MILP data sets. More recently, Khalil et al. [2022a] used a graph neural network to learn node selection. Our method distinguishes itself from these approaches as we specifically use the nature of our problem. Namely, in the design of the node selection strategy, we use the actual meaning of selecting a node; adding a scenario to a subset. Therefore, we try to learn what characteristics (or features) scenarios that should belong to the same subset have.

For an overview on ML for learning branching policies in B&B, see Bengio et al. [2020]. There has also been done a vast amount of research on applying MCTS directly to solving combinatorial problems. In Sabharwal et al. [2012] a special case of MCTS called Upper Confidence bounds for Trees (UCT), is used for designing a node selection strategy to explore B&B trees (for MIPs). In Khalil et al. [2022b] MCTS is used to find the best backdoor (*i.e.*, a subset of variables used for branching) for solving MIPs. Loth et al. [2013] have used MCTS for enhancing constraint programming solvers, which naturally use a search tree for solving combinatorial problems. For an elaborate overview on modifications and applications of MCTS, we refer to Świechowski et al. [2022].

The remainder of the paper is structured as follows. In Section 2 we describe the inner workings of the K-adaptability branch-and-bound to set the stage for our contribution. In Section 3 we outline our ML methodology along with the data generation procedure. Section 4 discusses the results of a numerical study, and Section 5 concludes with some remarks on future works.

2 Preliminaries

It is instructive to conceptualize a solution to (2) as a solution to a nested clustering and optimization-for-clusters methodology. As already mentioned in Section 1, a feasible solution to (2) can be used to construct a partition of the uncertainty set into subsets $\mathcal{Z}_1, \ldots, \mathcal{Z}_K$ such that $\bigcup_{k=1}^{K} \mathcal{Z}_k = \mathcal{Z}$. Here, decision $\boldsymbol{y}_k$ is applied in the second stage if $\boldsymbol{z} \in \mathcal{Z}_k$. The decision framework associated with a given solution is illustrated in Figure 1.

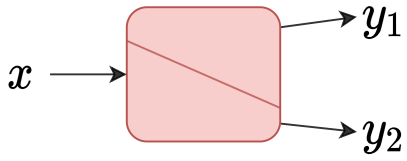


Figure 1: A framework of the K-adaptability problem, where we split the uncertainty set (red box) in $K = 2$ parts. Here, $\boldsymbol{x}$ represents the first-stage decisions, and $\boldsymbol{y}_1$ with $\boldsymbol{y}_2$ those of the second-stage.

For such a *fixed partition* the corresponding optimization problem becomes

$$\min_{\boldsymbol{x}\in\mathcal{X},\boldsymbol{y}\in\mathcal{Y}^K} \max_{k\in\mathcal{K}} \max_{\boldsymbol{z}\in\mathcal{Z}_k} \left\{\boldsymbol{c}(\boldsymbol{z})^\intercal \boldsymbol{x} + \boldsymbol{d}(\boldsymbol{z})^\intercal \boldsymbol{y}_k\right\} \tag{3}$$
$$\text{s.t.} \quad \boldsymbol{T}(\boldsymbol{z})\boldsymbol{x} + \boldsymbol{W}(\boldsymbol{z})\boldsymbol{y}_k \leq \boldsymbol{h}(\boldsymbol{z}).$$

The optimal solution to (2) also corresponds to an optimal partitioning of $\mathcal{Z}$, and the optimal decisions of (3) with that partitioning. Finding an optimal partition and the corresponding decisions has been shown to be NP-hard by Bertsimas and Caramanis [2010]. For that reason, Subramanyam et al. [2020] have proposed the K-B&B algorithm. There, the idea is to gradually build up a collection of finite subsets $\bar{\mathcal{Z}}_1, \ldots, \bar{\mathcal{Z}}_K$, such that for each $k \in \mathcal{K}$ an optimal solution to (3) with $\mathcal{Z}_k = \bar{\mathcal{Z}}_k$ is also an optimal solution to (2).

The algorithm follows a master-subproblem approach. The master problem solves (2) with K finite subsets of scenarios. The subproblem finds the scenario for which the current master solution is not robust. The number K of possible assignments of this new scenario to one of the existing subsets gives rise to using a search tree. Each tree node corresponds to a partition of all scenarios found so far into K subsets. The goal is to find the node with the best partition. An illustration of the search tree is given in Figure 2. The tree grows exponentially and thus only (very) small-scale problems can be solved in reasonable time. The method we propose in the next section learns a good node selection strategy with the goal of converging to the optimal solution much faster than K-B&B.

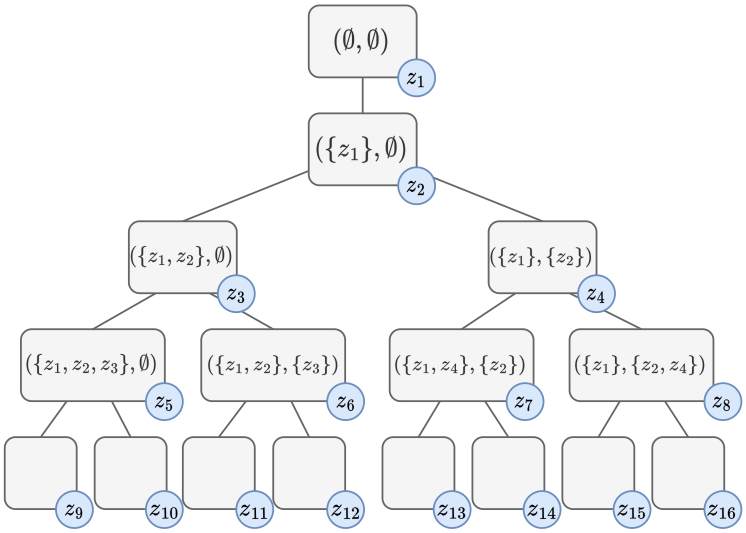


Figure 2: Search tree for K-adaptability branch-and-bound ($K = 2$).

Master problem. This problem solves the K-adaptability problem (2) with respect to the currently found scenarios grouped into $\bar{\mathcal{Z}}_k \subset \mathcal{Z}$ for all $k \in \mathcal{K}$. For a collection $\bar{\mathcal{Z}}_1, \ldots, \bar{\mathcal{Z}}_K$, the problem formulation is defined as follows:

$$\begin{aligned} \min_{\theta \in \mathbb{R}, \boldsymbol{x} \in \mathcal{X}, \boldsymbol{y} \in \mathcal{Y}^K} \quad & \theta & & \\ \text{s.t.} \quad & \boldsymbol{c}(\boldsymbol{z})^\intercal \boldsymbol{x} + \boldsymbol{d}(\boldsymbol{z})^\intercal \boldsymbol{y}_k \leq \theta, & & \forall \boldsymbol{z} \in \bar{\mathcal{Z}}_k, \forall k \in \mathcal{K}, \\ & \boldsymbol{T}(\boldsymbol{z})\boldsymbol{x} + \boldsymbol{W}(\boldsymbol{z})\boldsymbol{y}_k \leq \boldsymbol{h}(\boldsymbol{z}), & & \forall \boldsymbol{z} \in \bar{\mathcal{Z}}_k, \forall k \in \mathcal{K}, \end{aligned} \tag{4}$$

where θ is the current estimate of the objective function value. We denote the optimal solution of (4) by the triplet $(\theta^*, \boldsymbol{x}^*, \boldsymbol{y}^*)$.

Subproblem. The subproblem aims to find a scenario $\boldsymbol{z}$ for which the current master solution is infeasible. That is, a scenario is found such that for each k, at least one of the following is true:

- the current estimate of θ^* is too low, *i.e.*, $\boldsymbol{c}(\boldsymbol{z})^\intercal \boldsymbol{x}^* + \boldsymbol{d}(\boldsymbol{z})^\intercal \boldsymbol{y}_k^* > \theta^*$;
- at least one of the original constraints is violated, *i.e.*, $\boldsymbol{T}(\boldsymbol{z})\boldsymbol{x}^* + \boldsymbol{W}(\boldsymbol{z})\boldsymbol{y}_k^* > \boldsymbol{h}(\boldsymbol{z})$.

If no such scenario exists, we define the solution $(\theta^*, \boldsymbol{x}^*, \boldsymbol{y}^*)$ as a robust solution. When such a scenario $\boldsymbol{z}^*$ does exist, the solution is not robust and the newly-found scenario is assigned to one of the sets $\bar{\mathcal{Z}}_1, \ldots, \bar{\mathcal{Z}}_K$.

Definition 1. *A solution $(\theta^*, \boldsymbol{x}^*, \boldsymbol{y}_1^*, \ldots, \boldsymbol{y}_K^*)$ to (4) is robust if*

$$\forall \boldsymbol{z} \in \mathcal{Z}, \exists k \in \mathcal{K} : \ \boldsymbol{T}(\boldsymbol{z})\boldsymbol{x}^* + \boldsymbol{W}(\boldsymbol{z})\boldsymbol{y}_k^* \leq \boldsymbol{h}(\boldsymbol{z}), \ \boldsymbol{c}(\boldsymbol{z})^\intercal \boldsymbol{x}^* + \boldsymbol{d}(\boldsymbol{z})^\intercal \boldsymbol{y}_k^* \leq \theta^*.$$

Example 1. *Consider the following master problem*

$$\begin{aligned} \min_{\theta, \boldsymbol{x}, \boldsymbol{y}} \quad & \theta & & \\ \textit{s.t.} \quad & \theta \in \mathbb{R}, \boldsymbol{x} \in \{0,1\}^2, \boldsymbol{y} \in \{0,1\}^2, & & \\ & \boldsymbol{z}^\intercal \boldsymbol{x} + [2z_1, 2z_2]\boldsymbol{y}_k \leq \theta, & & \forall \boldsymbol{z} \in \bar{\mathcal{Z}}_k, \forall k \in \mathcal{K}, \\ & \boldsymbol{y}_k \geq \mathbf{1} - \boldsymbol{z}, & & \forall \boldsymbol{z} \in \bar{\mathcal{Z}}_k, \forall k \in \mathcal{K}, \\ & \boldsymbol{x} + \boldsymbol{y}_k \geq \mathbf{1}, & & \forall k \in \mathcal{K}, \end{aligned}$$

where $\boldsymbol{z} \in \{-1, 0, 1\}^2$ and $\mathcal{K} = \{1, 2\}$. We look at three solutions for different groupings of $\boldsymbol{z}_1 = [1,1]^\intercal$, $\boldsymbol{z}_2 = [0,1]^\intercal$, and $\boldsymbol{z}_3 = [0,0]^\intercal$. One of them is not robust, and the other ones are but have a different objective value due to the partition. The first partition we consider is $\bar{\mathcal{Z}}_1 = \{\boldsymbol{z}_1\}$, $\bar{\mathcal{Z}}_2 = \{\boldsymbol{z}_2\}$. The corresponding solution is $\theta^* = 2$, $\boldsymbol{x}^* = [1,1]^\intercal$, $\boldsymbol{y}_1^* = [0,0]^\intercal$, and $\boldsymbol{y}_2^* = [1,0]^\intercal$. This solution is not robust, since for $\boldsymbol{z}_3 = [0,0]^\intercal$ the following constraint is violated for all $k \in \mathcal{K}$:

$$\boldsymbol{y}_1 \not\geq \mathbf{1} - \boldsymbol{z}_3 \iff \begin{bmatrix} 0 \\ 0 \end{bmatrix} \not\geq \begin{bmatrix} 1 \\ 1 \end{bmatrix} - \begin{bmatrix} 0 \\ 0 \end{bmatrix}, \qquad \boldsymbol{y}_2 \not\geq \mathbf{1} - \boldsymbol{z}_3 \iff \begin{bmatrix} 1 \\ 0 \end{bmatrix} \not\geq \begin{bmatrix} 1 \\ 1 \end{bmatrix} - \begin{bmatrix} 0 \\ 0 \end{bmatrix}.$$

Next, consider $\bar{\mathcal{Z}}_1 = \{\boldsymbol{z}_1, \boldsymbol{z}_2\}$, $\bar{\mathcal{Z}}_2 = \{\boldsymbol{z}_3\}$, with solution $\theta^* = 3$, $\boldsymbol{x}^* = [0,1]^\intercal$, $\boldsymbol{y}_1^* = [1,0]^\intercal$, $\boldsymbol{y}_2^* = [1,1]^\intercal$. There is no $\boldsymbol{z}$ that violates this solution, which makes it robust. The third partition is $\bar{\mathcal{Z}}_1 = \{\boldsymbol{z}_1, \boldsymbol{z}_3\}$, $\bar{\mathcal{Z}}_2 = \{\boldsymbol{z}_2\}$, with solution $\theta^* = 4$, $\boldsymbol{x}^* = [0,0]^\intercal$, $\boldsymbol{y}_1^* = [1,1]^\intercal$, and $\boldsymbol{y}_2^* = [1,1]^\intercal$. This solution is also robust. Their objective values are 3 and 4, which shows that different partitions can significantly influence solution quality.

Mathematically, we formulate the subproblem using the big-M reformulation:

$$\begin{aligned} \max_{\zeta, \boldsymbol{z}, \gamma} \quad & \zeta & & \\ \text{s.t.} \quad & \zeta \in \mathbb{R}, \quad \boldsymbol{z} \in \mathcal{Z}, \quad \gamma_{kl} \in \{0,1\}, \quad (k,l) \in \mathcal{K} \times \mathcal{L}, & & \\ & \sum_{l \in \mathcal{L}} \gamma_{kl} = 1, & & \forall k \in \mathcal{K}, \\ & \zeta + M(\gamma_{k0} - 1) \leq \boldsymbol{c}(\boldsymbol{z})^\intercal \boldsymbol{x}^* + \boldsymbol{d}(\boldsymbol{z})^\intercal \boldsymbol{y}_k^* - \theta^*, & & \forall k \in \mathcal{K}, \\ & \zeta + M(\gamma_{kl} - 1) \leq \boldsymbol{t}_l(\boldsymbol{z})^\intercal \boldsymbol{x}^* + \boldsymbol{w}_l(\boldsymbol{z})^\intercal \boldsymbol{y}_k^* - h_l(\boldsymbol{z}), & & \forall l \in \mathcal{L}, \forall k \in \mathcal{K}, \end{aligned} \tag{5}$$

where M is some big scalar and $\mathcal{L}$ is the index set of constraints. When $\zeta \leq 0$, we have not found any violating scenario, and the master solution is robust. Otherwise, the found $\boldsymbol{z}^*$ is added to one of $\bar{\mathcal{Z}}_1, \ldots, \bar{\mathcal{Z}}_K$. Then, the master problem is re-solved, so that a new solution $(\theta^*, \boldsymbol{x}^*, \boldsymbol{y}^*)$, which is guaranteed to be feasible for $\boldsymbol{z}^*$ as well, is found.

Throughout the process, the key issue is to which of $\bar{\mathcal{Z}}_1, \ldots, \bar{\mathcal{Z}}_K$ to assign $\boldsymbol{z}^*$. As this cannot be determined in advance, all options have to be considered. Figure 2 illustrates that from each tree node we create K child nodes, each corresponding to adding $\boldsymbol{z}^*$ to the k-th subset:

$$\tau^k = \{\bar{\mathcal{Z}}_1, \ldots, \bar{\mathcal{Z}}_k \cup \{\boldsymbol{z}^*\}, \ldots, \bar{\mathcal{Z}}_K\}, \quad \forall k \in \mathcal{K},$$

where τ^k is the partition corresponding to the k-th child node of node τ. If a node is found such that θ^* is greater than or equal to the value of another robust solution, then this branch can be pruned as the objective value of the master problem cannot improve if more scenarios are added.

The pseudocode of K-B&B is given in Algorithm 2 in the Appendix. The tree becomes very deep for 2SRO problems where many scenarios are needed for a robust solution. Moreover, the tree becomes wider when K increases. Due to these issues, solving the problem to optimality becomes computationally intractable in general. Our goal is to investigate if ML can be used to make informed decisions regarding the assignment of newly found $\boldsymbol{z}^*$ to the discrete subsets, so that a smaller search tree has to be explored in a shorter time before a high-quality robust solution is found.

3 ML methodology

We propose a method that enhances the node selection strategy of the K-B&B algorithm. It relies on four steps:

1. ***Decision on what and how we want to predict:*** Section 3.1.
2. ***Feature engineering:*** Section 3.2.
3. ***Label construction:*** Section 3.3.
4. ***Using partial K-B&B trees for training data generation:*** Section 3.4.

All these steps are combined into the K-B&B-NODESELECTION algorithm (Section 3.5).

3.1 Learning setup

As there is no clearly well-performing node selection strategy for K-B&B, we cannot simply try to imitate one. Instead, we investigate what choices a good strategy would make. We will focus on learning how to make informed decisions about the order of inspecting the children of a given node. The scope of our approach is illustrated with rounded-square boxes in Figure 3.

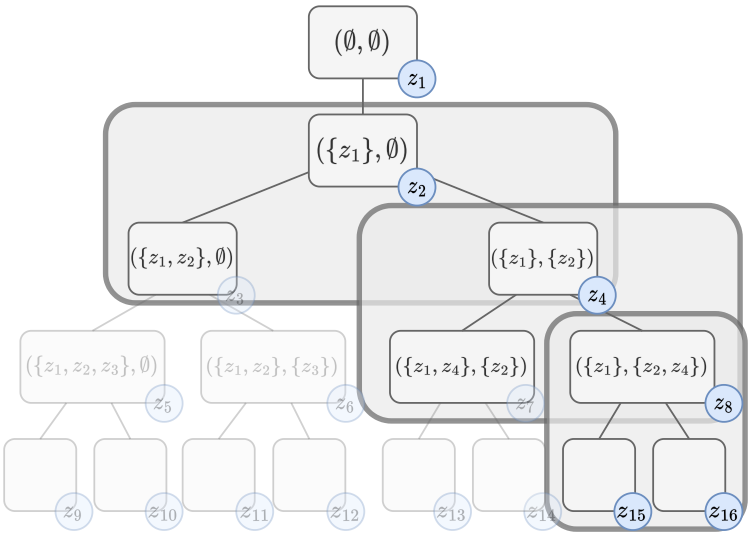


Figure 3: The scope of node selection

To rank the child nodes in the order they ideally be explored, one of the ways is to have a certain form of child node information whether selecting this node is *good* or *bad*. In other words, how likely is a node to guide us towards a high-quality robust solution fast. Indeed, this shall be exactly the quantity that we predict with our model. To train such a model, we will construct a dataset consisting of the following input-output pairs:

- **Input.** feature vector F of the decision to insert a scenario to a subset (Section 3.2)
- **Output.** $[0, 1]$ label that informs how good a given insertion was, based on an ex-post constructed strategy; 'what would have been the best node selection strategy, had we known the entire tree?' (Section 3.3).

We gather this data by creating a proxy for an oracle (see Section 3.4). Once the predictive model is trained, we can apply it to the search tree where in each iteration we predict for all K child nodes a score μ, in the interval $[0, 1]$. The child node with the highest score is explored first (see Figure 4). Formally, the working order of the trained method

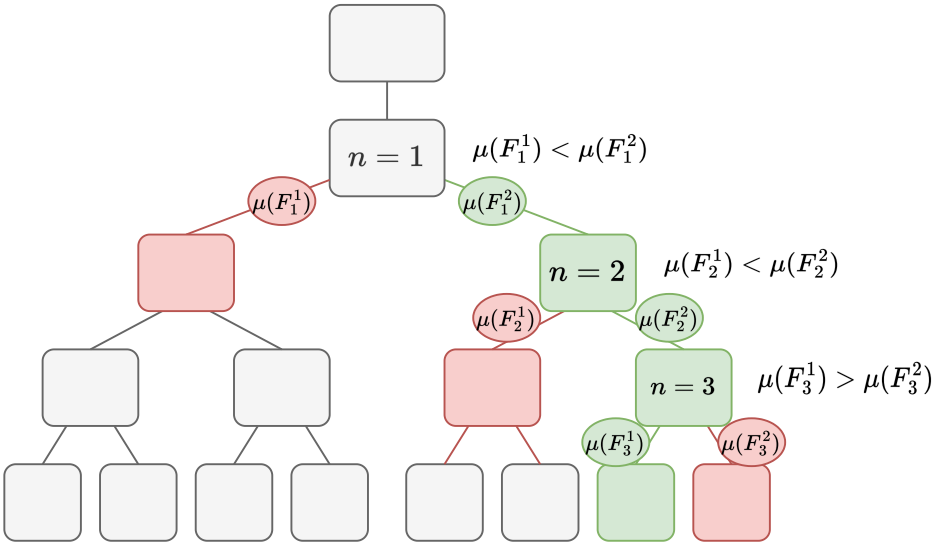


Figure 4: Node selection with ML predictions. Node selections are decided by the prediction of the function μ with input features F_n^k, where n is the node from which a selection is made and k relates to its k-th child node.

shall be as follows:

1. **Process node.** Solve master and subproblem in current node n. If prune conditions hold, then select a new node. Otherwise, continue to Step 2.
2. **Compute features.** For each one of the K child nodes, generate a vector of features $F_n^1, \ldots, F_n^K$ (see Section 3.2 for details).
3. **Predict.** For each child node k, predict the goodness score $\mu(F_n^k)$.
4. **Node selection.** Select the child node with the highest score.

It is desirable for an ML tool to be applicable to data that is different from the one which it is trained on. In the context of an ML model constructed to solve optimization problems, this means the potential to use a given trained method on various optimization problems of various sizes, with different values of K. Indeed, we shall demonstrate the generality of our method. First, we note that each parent-child node pair corresponds to one data point, and hence, training on a problem with a certain K does not prevent us from using it for different K. Next, we construct our training dataset so that the model becomes independent of (i) the instance size, and (ii) the type of objective function and constraints. This will be explained in Section 3.2.

3.2 Feature engineering

If we take a look at a single rounded box in Figure 3, the ML model we are about to train is going to give a goodness score which will depend on the parent node and the scenario. Therefore, we need to design features which we group into (i) state features that describe the master problem and the subproblem solved in the parent node, (ii) scenario features that describe the assignment of a newly-found scenarios to one of the subsets $\bar{\mathcal{Z}}_k$. In what follows, we present the feature list:

1. **State features.** This input describes the parent node n, *i.e.*, the current state of the algorithm. Different states might benefit from different strategies. Hence, information on the current node might increase the prediction

performance. This also means that all the child nodes have the same state features: $\boldsymbol{s}_n^k = \boldsymbol{s}_n$ for all $k \in \mathcal{K}$. To scale the features, we always initialize a tree search with a so-called *initial run*. This is a dive with random child-node selections, where we stop until robustness is reached. The following values are taken from this initial run:

- θ^0: the objective function value of the robust solution found in the initial run.
- ζ^0: the violation of the root node.
- κ^0: depth reached in the initial run.

In the experiments, multiple initial runs are done. The averages of θ^0, ζ^0, and κ^0 over the dives are then used for scaling. Additional meta-features for K and the dimensions of $\mathcal{Z}$, $\mathcal{X}$, and $\mathcal{Y}$ could be added. We omitted these as in our experimental setup we do not mix training data for different combinations of these values.

2. **Scenario features.** Intuitively, scenarios contained in the same group should have similar characteristics. Therefore, the features for each node are constructed in the following way: each newly found scenario $\boldsymbol{z}^*$ is assigned a set of characteristics, or attributes. Based on the attributes of the new scenario, and the attributes of the scenarios already grouped into the K subsets, we formulate the input of one data point. Some of the scenario attributes can be directly determined from the master problem. Others are extracted from easily solvable optimization problems: the *deterministic problem* and the *static problem*. The deterministic problem is a version of the problem where $\boldsymbol{z}^*$ is the only scenario. The solution to this problem gives information on the optimal objective and decisions found, in the most naive sense. One would expect that for good-performing subset formations, the optimal decisions of its scenarios would have similarities. For the static problem, we first solve for a single $\boldsymbol{y}$ (no adaptability) for all $\boldsymbol{z} \in \mathcal{Z}$. Then, for the obtained $\boldsymbol{x}^*$ and for a given $\boldsymbol{z}^*$, we solve for the best $\boldsymbol{y}$. The solution to this problem gives additional information as it has a sense of embedded robustness.

As our goal is to use the model also on different problems, we engineer the features to keep them independent from the problem size or type. In Tables 1-2, we give an outline of the two feature types that we construct. For a detailed description of how they are computed, we refer the reader to Appendix A.

Table 1: State features. θ^0, ζ^0, and κ^0 are as defined above. θ^p is the objective value of the parent node, ζ^p is the violation of the parent node, and κ is the depth of the current node.

Num.	State feature name	Description	Calculation
1	Objective	Relative objective value of this node to the first robust solution	θ/θ^0
2	Objective difference	Ratio of objective to that of the parent node	θ/θ^p
3	Violation	Relative violation with respect to the first violation found	ζ/ζ^0
4	Violation difference	Ratio of violation to that of the parent node	ζ/ζ^p
5	Depth	Relative depth of this node to the depth of the fist robust solution	κ/κ^0

The state features in Table 1 are readily problem-independent. However, this is not the case for the scenario attributes in Table 2. They are, for example, dependent on the instance size. To deal with the size dependency, and to adopt the actual meaning of node selection (*i.e.*, placing a scenario to a group of other scenarios), we introduce the *attribute distance* as a feature to our model. This is the distance between scenario-subset pairs in terms of the attributes attached to the scenarios, as a proxy to the unknown implications the addition of the scenario would have on the solution of the problem. For each attribute, the scenario-subset distance is taken by the Euclidean distance from the attribute of the new scenario to the average of the attribute of the scenarios already in the subset. This feature is described as:

$$\delta_f^k = \frac{\|\boldsymbol{a}_f^{k,\boldsymbol{z}^*} - \frac{1}{|\bar{\mathcal{Z}}_k|}\sum_{\boldsymbol{z}\in\bar{\mathcal{Z}}_k} \boldsymbol{a}_f^{k,\boldsymbol{z}}\|_2}{\text{length}(\boldsymbol{a}_f^{k,\boldsymbol{z}^*})}, \qquad \forall k \in \mathcal{K}, \forall f \in \{1,\dots,9\}, \tag{6}$$

where δ_f^k is the attribute distance of the new scenario $\boldsymbol{z}^*$ to subset $\bar{\mathcal{Z}}_k$ and $\boldsymbol{a}_f^{k,\boldsymbol{z}}$ is the data vector related to the f-th attribute (of Table 2) for the k-th child node. The attribute distance is scaled by the length of the attribute vector, denoted by 'length' in the denominator of (6), to control for varying attribute vectors in the feature value. Larger attribute

Table 2: Attributes assigned to scenario $\boldsymbol{z}^*$.

	Num.	Attribute name	Description
	1	Scenario values	Vector of scenario values $\boldsymbol{z}^*$
Master problem	2	Constraint distance	A measure for the change of the feasibility region when z^* is added to a subset. We look at the distance between the constraints already in the master problem, and the one to be added.
	3	Scenario distance	With this attribute we measure how far away $\boldsymbol{z}^*$ is from *not* being a violating scenario, for each of the k subsets. This is done by looking at the constraints in the space of $\mathcal{Z}$, given the current solutions $\boldsymbol{x}$ and $\boldsymbol{y}^k$.
	4	Constraint slacks	The slack values of the uncertain constraint per subset decisions.
Deterministic problem	5	Objective value	The objective value of the deterministic problem
	6	First-stage decisions	First-stage decisions of deterministic problem
	7	Second-stage decisions	Second-stage decisions of deterministic problem
Static problem	8	Objective value	The objective value of the static problem
	9	second-stage decisions	Second-stage decisions of static problem

vectors would otherwise result in higher feature values. Then, the *scenario feature* vector of the k-th child node is defined as $\boldsymbol{d}_n^k = [\delta_1^k, \ldots, \delta_9^k]^\intercal$.

Then, the input of the ML model for the k-th child of node n is given by $F_n^k = [\boldsymbol{s}_n \quad \boldsymbol{d}_n^k]^\intercal$.

In Figure 5 we outline the feature generation procedure. Steps 2 and 5 indicate how new attributes need to be generated for every child node. In practice many attributes are the same for all subsets, and the attributes that do differ are the master problem-based ones, which are easily computed.

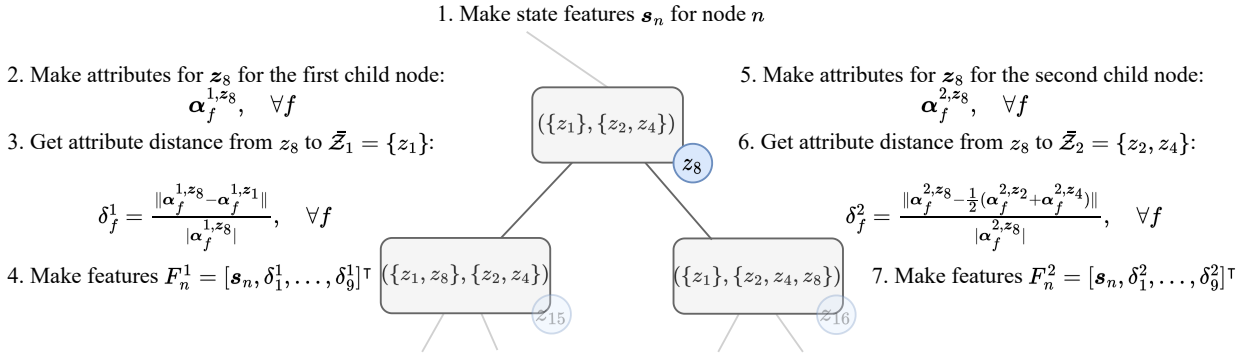


Figure 5: Example of a feature generation procedure for node n and its two child nodes.

3.3 Label construction

To learn how to assign a new scenario to one of the subsets $\bar{\mathcal{Z}}_k$, we need another piece of the input-output pairs in our database – labels that would indicate how good, ex post, it was to perform a given assignment, *i.e.*, how likely a given assignment is to lead the search strategy towards a *good solution*. We shall assume that given a K-B&B tree, a good solution is a robust solution with objective that belongs to the best $\alpha\%$ of the found robust solutions. We will now introduce a notion of scenario-to-set assignments and illustrate a method of constructing labels q.

We define p_ν – the probability of node ν leading to a good robust solution. If this node is selected, one of a finite, possibly large, number of *leaves* (*i.e.*, terminal nodes) is reached with depth-first search. Then, taking g_ν to be the number of good solutions and M_ν as the number of leaves under node ν, we define $p_\nu = g_\nu / M_\nu$ as the fraction of leaves with a good robust solution. We define a node ν as good if the probability p_ν of it leading to a good robust solution is higher than some threshold ϵ. This is computed by the quality value $q_\nu = \mathbf{1}_{\{p_\nu \geq \epsilon\}}$.

Example 2. *Consider the tree in Figure 6. For three nodes in the search tree, its subtree (consisting of all its successors) and leaves (coloured nodes) are shown. Red coloured nodes represent bad and green nodes represent good robust solutions. Then, if we pick $\epsilon = \frac{1}{5}$ as threshold, the corresponding success probabilities p_ν and the quality values of the three nodes are:*

$$p_1 = \frac{1}{4},\ p_2 = 0,\ p_3 = \frac{2}{5},\ q_1 = \mathbf{1}_{\{p_1 \geq \frac{1}{5}\}} = 1,\ q_2 = \mathbf{1}_{\{p_2 \geq \frac{1}{5}\}} = 0,\ q_3 = \mathbf{1}_{\{p_3 \geq \frac{1}{5}\}} = 1.$$

The first and third nodes would have been good node selections, whereas the second selection would have been a bad one.

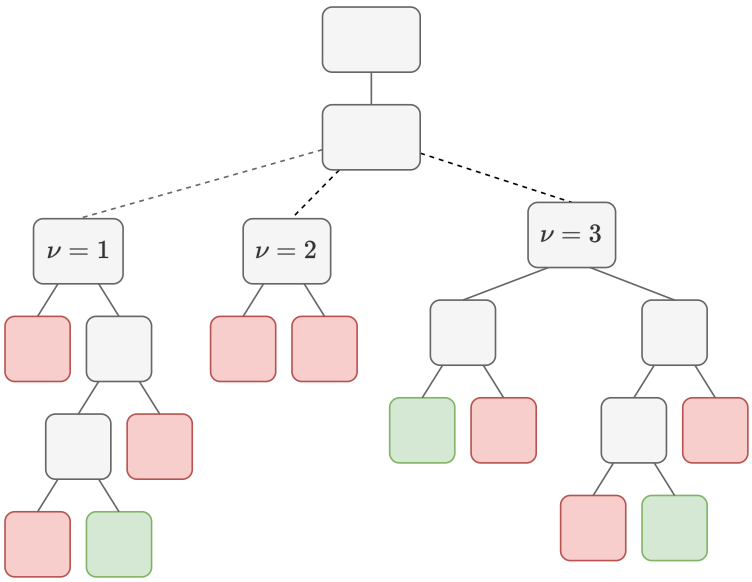


Figure 6: Example of a tree where three nodes are considered. The coloured nodes are leaves; green and red nodes are good and bad robust solutions, respectively.

In practice, we do not know for a node ν how many underlying leaves are good and bad. This is the reason that in this method we will predict q_ν with a model μ. We will also call this model the strategy model (or function) since it guides us in making node selections.

3.4 Training data generation

Our goal is to learn a supervised ML model $\mu(F_\nu) = \hat{q}_\nu$, where μ is the strategy function, F_ν the input features, and $\hat{q}_\nu$ the prediction of the quality of moving to node ν. To train this model, we first need to generate training data. Given a single tree, the difficulty of generating data does not lie in the input, but in the output: an expert is needed to determine the correct values of $\hat{q}_\nu$ for its nodes. Consider the following; if nodes of the entire tree are *processed* (*i.e.*, solving the master problem and the subproblem), we could easily find the paths from the root to good solutions. Then, all the nodes in these paths would get the values $\hat{q}_\nu = 1$, and all others $\hat{q}_\nu = 0$. Or equivalently, the success probabilities would be set to $p_\nu > \epsilon$. Since the trees grow exponentially, this approach is not practical.

As already mentioned in the introduction, a popular method for exploring intractable decision trees is Monte Carlo tree search (MCTS) [Browne et al., 2012]. By randomly running deep in the tree, we can gather information on the search space. In our method, we use the idea of random runs to mimic an expert, and hence, label the data points. Generating training data is done as follows (see Figure 7):

1. **Get instance.** Generate an instance of a 2SRO problem.
2. **Initial run.** One (or multiple) dives are executed to gather feature information.
3. **Initialize search tree.** Process all nodes up to a predetermined level L of the tree. Generate features for each of the explored nodes.
4. **Downward pass.** Per node $\nu \in \{1, \ldots, N_L\}$ of the L-th layer (N_l is the number of nodes of the l-th layer), perform dives for a total of R times.

5. **Probability of bottom nodes.** Set probability $p_{L,\nu}$ of each node $\nu \in \{1, \ldots, N_L\}$ in layer L as $p_{L,\nu} = \frac{g_\nu}{R}$ where $g_\nu \in \{0, \ldots, R\}$ is the number of good solutions from the samples of the ν-th node in layer L.
6. **Upward pass.** Propagate the probabilities $p_{l,\nu}$ upwards through the tree, for all nodes $\nu \in \{1, \ldots, N_l\}$, for all levels $l \in \{L-1, \ldots, 2\}$, as follows:

$$\begin{aligned} p_{l,\nu} &= \mathbb{P}(\text{at least one child node is successful}) \\ &= 1 - \mathbb{P}(\text{no successful child nodes}) \\ &= 1 - \prod_{k \in \mathcal{K}} (1 - p^k_{l,\nu}), \end{aligned}$$

   where $p^k_{l,\nu}$ is the k-th child of node ν of layer l (while this child node is in the $(l+1)$-th level). Note that $p^k_{l,\nu} = p_{l+1,\nu'}$ for some $\nu \in \{1, \ldots, N_l\}$ and $\nu' \in \{1, \ldots, N_{l+1}\}$.
7. **Label nodes.** Determine the label $\hat{q}_{l,\nu}$ for nodes $\nu \in \{1, \ldots, N_l\}$ for levels $l \in \{2, \ldots, L\}$.

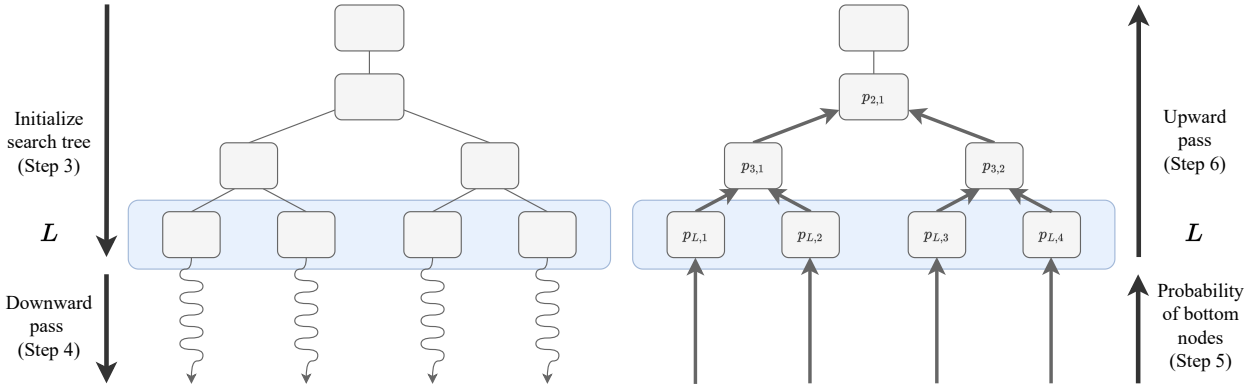


Figure 7: Downward and upward pass for generating training data. The blue layer represents the L-th layer in the tree from which random runs are made.

The advantage of this structure is that both bad and good decisions are well represented in the dataset. However, it only consists of input-output pairs of the top L levels. This is not necessarily a disadvantage as good decisions at start can be expected to be more important. The above generation method is applied per instance, thus it can be parallelized, like Step 4.

3.5 Complete node selection algorithm

We now combine all the steps above into one algorithm which is, essentially, a variant of K-B&B enhanced with: (i) node selection, (ii) feature engineering, and (iii) training data generation (see Algorithm 1). Our K-B&B-NODESELECTION algorithm has two preprocessing steps:

1. STRATEGYMODEL (Procedure 3 in Appendix B): Generate the data applying 1-4, and train the ML model.
2. INITIALRUN (Procedure 4 in Appendix B): Start with a random dive through the tree to obtain θ^0, ζ^0, and κ^0 used to scale the features (see Table 1).

Algorithm 1: K-B&B-NODESELECTION

Input : Test instance $\mathcal{P}^{test}(N^{test})$, train instances $\mathcal{P}_1^{train}(N^{train}), \ldots, \mathcal{P}_I^{train}(N^{train})$ number of partitions for training K^{train} testing K^{test}, level for training L^{train} and testing L^{test}, quality threshold ϵ, number of random dives per node R

Output : Objective value θ, first-stage decisions $\boldsymbol{x}$, second-stage decisions $\boldsymbol{y} = \{\boldsymbol{y}_1, \ldots, \boldsymbol{y}_K\}$, subsets with scenarios $\bar{\mathcal{Z}}_k$ for all $k \in \{1, \ldots, K\}$

Initialization : Incumbent partition: $\tau^i := \{\bar{\mathcal{Z}}_1, \ldots, \bar{\mathcal{Z}}_K\}$, where $\bar{\mathcal{Z}}_k = \emptyset$ for all $k \in \mathcal{K}$, set containing all node partitions yet to explore: $\mathcal{N} := \{\tau^i\}$, initial incumbent solution: $(\theta^i, \boldsymbol{x}^i, \boldsymbol{y}^i) := (\infty, \emptyset, \emptyset)$

`// Preprocessing`

model $\leftarrow$ STRATEGYMODEL$(\mathcal{P}_1^{train}(N^{train}), \ldots, \mathcal{P}_I^{train}(N^{train}), K^{train}, L^{train}, \epsilon, R)$ `// Proc. 3`

scaling info $\leftarrow$ INITIALRUN$(\mathcal{P}^{test}(N^{test}), K^{test})$ `// Proc. 4`

`// Tree search`

while $\mathcal{N}$ *not empty* **do**
- **if** *no solution yet or previous node pruned* **then**
  - Select a random node with partition $\tau = \{\bar{\mathcal{Z}}_1, \ldots, \bar{\mathcal{Z}}_K\}$ from $\mathcal{N}$, then $\mathcal{N} \leftarrow \mathcal{N} \setminus \{\tau\}$
- **else**
  - $\tau := \{\bar{\mathcal{Z}}_1, \ldots, \bar{\mathcal{Z}}_{k^*} \cup \{\boldsymbol{z}^*\}, \ldots, \bar{\mathcal{Z}}_K\}$
- **end**
- $(\theta^*, \boldsymbol{x}^*, \boldsymbol{y}^*) \leftarrow$ *master problem*(τ)
- **if** $\theta^* > \theta^i$ **then**
  - Prune tree since current objective is worse than best solution found, continue to line 4.
- **end**
- $(\boldsymbol{z}^*, \zeta^*) \leftarrow$ *subproblem*$(\theta^*, \boldsymbol{x}^*, \boldsymbol{y}^*)$
- **if** $\zeta^* > 0$ **then**
  - Solution not robust. Create K new branches.
  - **if** *current level is more than* L^{test} **then**
    - $k^* \leftarrow$ *random uniform sample*$([1, K])$
  - **else**
    - Create feature vectors for K child nodes.
    - $(D_1, \ldots, D_K) \leftarrow$ *generate features*$($*scaling info*$, \theta, \zeta)$ `// steps of Section 3.2`
    - $k^* \leftarrow$ *predict node qualities*$(D_1, \ldots, D_K,$ *model*$)$
  - **end**
  - Make K new branches, of which the k^*-th is selected.
  - **for** $k \in \{1, \ldots, K\} \setminus \{k^*\}$ **do**
    - $\tau^k := \{\bar{\mathcal{Z}}_1, \ldots, \bar{\mathcal{Z}}_k \cup \{\boldsymbol{z}^*\}, \ldots, \bar{\mathcal{Z}}_K\}$
    - $\mathcal{N} \leftarrow \mathcal{N} \cup \{\tau^k\}$
  - **end**
- **else**
  - Current solution robust, prune tree.
  - $(\theta^i, \boldsymbol{x}^i, \boldsymbol{y}^i, \tau^i) \leftarrow (\theta^*, \boldsymbol{x}^*, \boldsymbol{y}^*, \tau)$
- **end**

end

return $(\theta^i, \boldsymbol{x}^i, \boldsymbol{y}^i, \tau^i)$

4 Experiments

We now investigate if *it is possible to learn a node selection strategy that generalizes (i) to other problem sizes, (ii) to different values of K, and (iii) to various problems.* We answer this question by a detailed study of two problems: capital budgeting (with loans) and shortest path [Subramanyam et al., 2020], whose formulations are given in Appendix C. This section is set up as follows: First, the effectiveness of the original K-B&B is tested on the problems. Then, we compare K-B&B to K-B&B-NODESELECTION. We shall observe that the results obtained with our approach are very promising.

For solving the MILPs, Gurobi 9.1.1 [Gurobi Optimization, 2020] is used. All computations of generating training data, K-B&B, and K-B&B-NODESELECTION are performed on an Intel Xeon Gold 6130 CPU @ 2.1 GHz with 96 GB RAM. Training of the ML model is executed on an Intel Core i7-10610U CPU @ 1.8 GHz with 16 GB RAM. Our implementation along with the scripts to reproduce our results are available online[1].

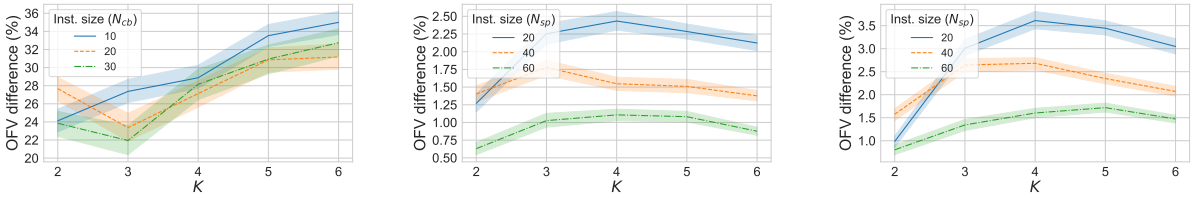
4.1 Performance of K-B&B

In our experiments, we investigate the potential of improving the node selection strategy with ML. For that reason, it is important to identify problems on which such an improvement matters, *i.e.*, where different partitions give varying outcomes and are nontrivial. To identify such problems, we run K-B&B on several problems. Compared to the algorithm in Subramanyam et al. [2020], we made some minor changes in K-B&B:

- Instead of breadth-first search, depth-first dives are performed with random node selection. Depth-first search returns an incumbent solution early on, which is used for pruning nodes. This is increasingly vital when K grows, as each node branches on K other nodes. Likely a combination of breadth- and depth-first search would be preferred. This would require additional bookkeeping of the features for K-B&B-NODESELECTION.
- In the starting node selection step (Step 2, Algorithm 2 in Appendix B), a random node is taken from $\mathcal{N}$ instead of the first one. This step is in line with random node selection.

The quantity we are interested in is the relative change in the objective function value (OFV) – the OFV of the first robust solution divided by the best one after 30 minutes. The higher the value, the more potential for a smart node selection strategy.

First, we consider the capital budgeting and the shortest path problems, only mentioned now and formally described later, for which the results are in Figure 8. We observed that the objective function values of the capital budgeting instances are changing more than those of shortest path (in which nodes of a graph are located on a 2D plane). This is why we implemented another instance type for shortest path: graph with nodes located on a 3D sphere (see the Appendix). The OFV differences for these instances are still smaller than those of capital budgeting, but more than for the 'normal' type. Therefore, further experiments on shortest path were conducted with the sphere instances alone.



(a) Capital budgeting (b) Shortest path (normal) (c) Shortest path (sphere)

Figure 8: Objective function value (OFV) difference (in %) for capital budgeting and shortest path with 'normal' and 'sphere' instances, within 30 minutes using K-B&B. For each problem type, experiments were done for 100 instances per $K \in \{2, \ldots, 6\}$ and instance size. The 75% confidence interval (CI) is also given as shaded strips along the curves.

Another problem on which we ran K-B&B was the knapsack problem parameterized by the combination of the capacity (c), the number of items (N_{ks}), and the maximum deviation of the profit of items (*i.e.*, the uncertainty parameter γ). For this problem, we considered instances of a similar size as the earlier problems, fixing $K = 4$ and $N_{ks} = 100$, and tested K-B&B on 16 instances for all combinations of $c \in \{0.05, 0.15, 0.35, 0.5\}$ and $\gamma \in \{0.05, 0.15, 0.35, 0.5\}$. The OFV differences (in percentages) are given in Table 3, where it is visible that different partitions barely play a role. Any

[1] `https://github.com/estherjulien/KAdaptNS`

random robust solution seems to be performing well. Thus, node selection will most likely not enhance K-B&B for knapsack and will therefore not be tested.

Table 3: The OFV difference (in %) of the knapsack problem, within 30 minutes, for different values of the capacity (c) and uncertainty (γ) parameters. $K = 4$ and $N = 100$ are fixed.

c	γ 0.05	0.15	0.35	0.5
0.05	0.002	0.086	0.000	0.000
0.15	0.002	0.011	0.040	0.000
0.35	0.001	0.001	0.019	0.038
0.5	0.001	0.003	0.012	0.027

Solving problem instances with K-B&B takes a long time, where we have to think in the range of multiple hours until optimality is proven. This also holds for small-size instances. After having studied the convergence over runtime, we have fixed the time limit to 30 minutes for all the problem instances. For capital budgeting with $(N, K) = (10, 2)$ all instances could be solved within 30 minutes. For the same instance size and $K = 3$, 25 out of 100 are solved, only one for $K = 4$, and none for larger K. For the 3D shortest path problem with the smallest instance size and $K = 2$, only six instances are solved.

4.2 Experimental setup K-B&B-NODESELECTION

In our experiments, we also investigate what would be a good node selection strategy in K-B&B that would perform well after it is trained on a selection of problems and then applied to other problems. Naturally, we are interested in generalization of trained tools to different instance sizes, K, and problem types. To this end, we have performed an ablation-study described in Table 4, where each row informs about the similarity of the testing problem instances, compared to the training ones.

Table 4: Different types of experiments (EXP), where the instance size N, K, and the problem itself can be different for training and testing.

Type	**Instance size** N	K	**Problem**
EXP1	Same	Same	Same
EXP2	Same	Different	Same
EXP3	Different	Same	Same
EXP4	Different	Different	Same
EXP5	Different	Same	Different
EXP6	Different	Different	Different

The problems we study are the capital budgeting problem with loans and the shortest path problem on a sphere (see the Appendix). We now describe the design choices we made regarding the ML model and data generation. As our focus lies in how ML is used for optimization and not in differences between ML models, we select a frequently-used model: random forest of `scikit-learn` [Pedregosa et al., 2011] with default settings. We note that the training times do not exceed a couple of minutes for different data sets.

As for the data generation process, it is governed by STRATEGYMODEL (Procedure 3), driven by the following parameters: I (number of training instances), L^{train} (depth level used in training data), and R (number of dives per node). In these experiments, we instead made them depend on T – the total duration in hours, and ι – the time per training instance in minutes. First, we set $I = 60T/\iota$. Selection of the right L^{train} value is more challenging since for some problem instances the master problem takes a lot more time or deep trees are needed. Therefore, to get sufficiently many random dives for each node in L^{train} within the time limit ι, we make L^{train} depend negatively on the total duration of the initial run (INITIALRUN, Procedure 4). This means that if a random dive takes a long time, the number of starting nodes should be lower, and therefore, the value of L^{train} should also decrease. Finally, the number of dives per node (R) depends on how many nodes we have in level L^{train} and the time we have for the instance (ι).

First, the experiments of EXP1-EXP4 are run on the capital budgeting and shortest path problem. Then, the experiments where the training and testing problems are mixed, are conducted (EXP5 and EXP6). We only discuss a representative selection of the results in the main body, referring the reader to Appendix E for a complete overview. Finally, we discuss the feature importance scores we obtained with our random forests.

4.3 Capital budgeting

The capital budgeting problem is a 2SRO problem where investments in a total of N_{cb} projects can be made in two time periods. In the first period, the cost and revenue of these projects are uncertain. In the second period, these values are known but an extra penalty needs to be paid for postponement. The MILP formulation of capital budgeting has uncertain objective function, fixed number of uncertain constraints, and the dimension of $\mathcal{Z}$ is fixed to 4 for all instance sizes. For the full description, see Appendix C.1. Recall that K-B&B-NODESELECTION takes more parameters than the ones being tested in the ablation study: L^{test} (level up to where node selection is performed), ϵ (node quality threshold), and T and ι for generating training data. These parameters will be tuned in the first part of the experiments.

Parameter tuning (with EXP1)

For this type of experiment, we use the same K and N for testing and training, applied to the smallest instance size: $N_{cb} = 10$, but for all $K \in \{2, 3, 4, 5, 6\}$. For different values of K, we noticed that different hyperparameters for generating training data were performing well. In Appendix D.1, the tuning is performed of parameter ι (minutes spent per training instance) based on the number of data points obtained in total, together with some other information. The testing accuracy scores for different data sets we trained on, range between 0.92-0.99.

We next tune the values of L^{test} (level up to which node selection is performed) and ϵ (quality threshold) using the sets $L^{test} \in \{5, 10, 20, 30, 40, 50\}$ and $\epsilon \in \{0.05, 0.1, 0.2, 0.3, 0.4\}$. Since there are 30 combinations per K value, we only considered $K = 6$. The results are shown in Figure 9, where using a low value for ϵ and high L^{test} gives the best results for both values of K. When a node n has success probability p_n larger than zero, this is already considered a good quality node. For further experiments of capital budgeting, we fix $\epsilon = 0.05$. Since high values of L^{test} outperform lower ones, we also consider the possibility of applying the strategy always ($L^{test} = \infty$), with fixed $\epsilon = 0.05$. This option is analyzed in the Appendix. We noticed that choosing $L^{test} = \infty$ performs well for $K = 6$ but for lower values of K, choosing $L^{test} = 40$ gives even better solutions. Therefore, we continue with $L^{test} = 40$.

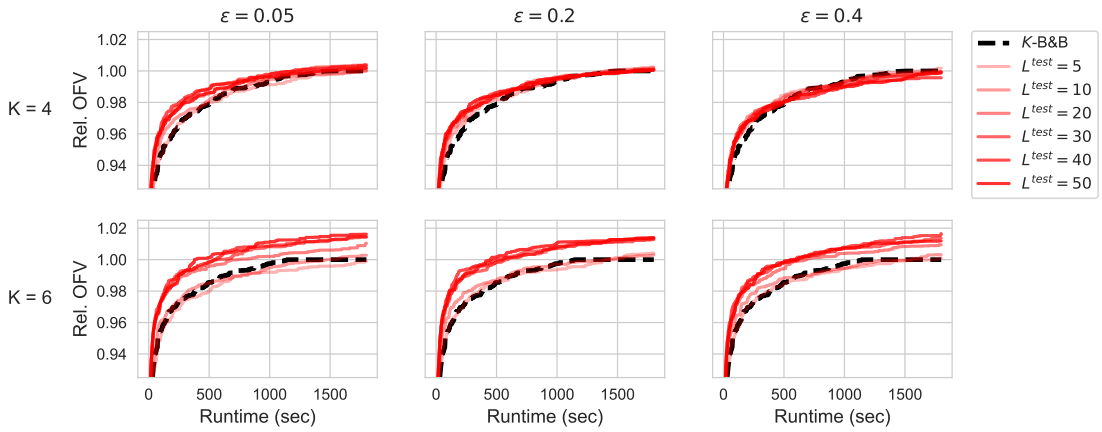


Figure 9: Results of K-B&B with random dives and K-B&B-NODESELECTION with combinations of K, L^{test} and ϵ. The plots show the average relative objective value over the runtime of 100 instances for the capital budgeting problem, for $N = 10$.

Another important parameter is the number of hours spent on generating training data (T). In Figure 10, results are shown for $T \in \{1, 2, 5, 10\}$ and $K \in \{3, 4, 5, 6\}$. Note that only for $K = 3$ higher values of T result in a better performance of K-B&B-NODESELECTION. For the other values of K the performance is similar, or even worse, for higher values of T compared to lower ones. For further experiments, we shall use $T = 2$.

Alternative scaling

The step taken to obtain scales for the objective (θ^0), violation (ζ^0), and depth (κ^0) in INITIALRUN seems to be avoidable if we decide to take other values for these parameters. In this section, we will describe a small experimental study of different scaling methods: (i) INITIALRUN, (ii) Alternative, and (iii) No scaling, where the latter corresponds to setting $\theta^0 = \zeta^0 = \kappa^0 = 1$. The Alternative setting takes as scale for θ^0 the objective of the root node, and $\kappa^0 = K^{\log(\dim(\mathcal{Z}))}$. Next to these 'internal' scaling approaches, one can also use min/max scaling of all feature values. The scales are taken from the training data set and then used in the testing phase. These two scaling approaches can be

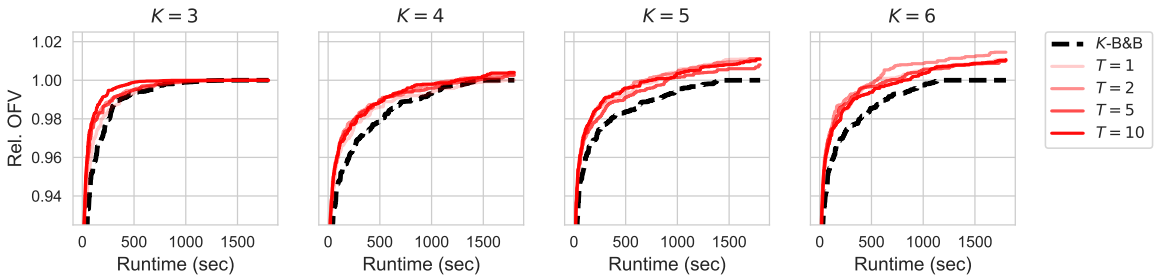


Figure 10: Results of K-B&B with random dives and K-B&B-NODESELECTION with combinations of K and T. The plots show the average objective over THE runtime of 100 instances for the capital budgeting problem, for $N = 10$.

used in combination. The results obtained for the capital budgeting problem, with $N = 10$ and $K^{test} = K^{train} = 6$, are given in Table 5. Here, for the first time, we use performance statistics to measure the effectiveness of the method. From the results presented in the table, we can conclude that, on average, solutions found after 30 minutes are best when INITIALRUN is used. Additional min/max scaling on top of the different scaling approaches gives mixed results.

Table 5: The results for a variety of scaling methods tested on capital budgeting problem, with $N = 10$, $K^{test} = K^{train} = 6$, on 16 instances. Different 'internal' scaling methods: 'INITIALRUN', 'Alternative', and 'No scaling'. These internal scaling methods can be combined with min/max scaling. The results for all combinations is given. Four performance statistics are given: (i) 'OFV 30m': Difference (in %) in relative OFV found after 30 minutes from K-B&B to K-B&B-NODESELECTION (higher is better). (ii) 'OFV 1m': is the relative OFV from the two methods after one minute. (iii): 'to OFV=1': Runtime speedup (in %) to reach a relative OFV of 1 for K-B&B-NODESELECTION compared to K-B&B (higher is better). (iv) 'NS to OFV=1': Number of instances of K-B&B-NODESELECTION that reached a relative OFV of 1 (higher is better).

Statistic	**min/max**	**Initial dive**		**Alternative**		**No scaling**	
		NO	YES	NO	YES	NO	YES
OFV 30m		2.10	1.98	1.28	1.11	1.33	1.50
OFV 1m		2.14	2.39	1.49	1.93	2.03	1.39
to OFV=1		44.10	42.53	52.58	38.08	51.72	44.29
NS to OFV=1		16	16	13	14	13	14

Results

We compare K-B&B to K-B&B-NODESELECTION for all combinations of $K^{test}, K^{train} \in \{2, 3, 4, 5, 6\}$, and $N_{test} \in \{10, 20, 30\}$. The results of the full set of combinations are displayed in Appendix E.1. Table 6 lists the results using four performance statistics of EXP1 and EXP2 (described in the caption). The table shows that K-B&B-NODESELECTION outperforms K-B&B, with speedups ranging from on average 2 to 50%, except for $K^{train} = 2$ and the instances of $K^{test} = 2$. Higher values of K^{train} generally perform better both in terms of relative OFV and speedup.
In the next sections, we look at specific problem instances in more detail, where we also focus on the stability of the algorithm. Due to the many combinations for EXP3-EXP4, a table would be too convoluted.

EXP1 and EXP2 results. For EXP1, the results for $K^{train} = K^{test} \in \{3, 4, 5\}$ are shown in Figure 11. As in Figure 8, the shaded strips around the solid curves show the confidence interval. We can see that K-B&B-NODESELECTION outperforms K-B&B for all K. Moreover, the convergence is steeper: good solutions are found earlier. For $K \in \{4, 5\}$ the final solution is also better when node selection is guided by ML predictions. Then for EXP2, where we also apply ML models that are trained with $K^{train} \neq K^{test}$, we see the performance of K-B&B-NODESELECTION improving when K^{train} increases. This indicates that data obtained with higher values of K^{train} are more informative than those with lower values. See Figure 12 for an illustration with $K^{test} = 5$ and $K^{train} \in \{4, 5, 6\}$.

EXP3 and EXP4 results. In Figure 13, we depict the results for experiments with same K but different instance sizes. On average, K-B&B-NODESELECTION performs better both in terms of speedup and final relative OFV found: for $N^{test} = 20$, the speedups are between 20-50% and relative OFV increase around 1.5%. For $N^{test} = 30$, we see speedups ranging from 11-47% and relative OFV increases of 1.6-2.1%. However, the confidence interval of $N^{test} = 30$ is larger than that of $N^{test} = 10$. This suggests that testing on higher values of N gives rise to a higher

Table 6: Combined results of EXP1 (along diagonal) and EXP2 for the capital budgeting problem. The four statistics described in the caption of Table 5 are given: (i) 'OFV 30m' (higher is better), (ii) 'OFV 1m', (iii) 'to OFV=1' (higher is better), and (iv) 'NS to OFV=1' (higher is better).

K^{test}	**Statistic**	K^{train} 2	3	4	5	6
2	OFV 30m	-0.00	-0.00	-0.00	-0.00	-0.00
	OFV 1m	-0.13	-0.12	-0.09	0.02	-0.09
	to OFV=1	-101.47	-78.31	-60.72	-54.80	-33.46
	NS to OFV=1	47	46	56	50	55
3	OFV 30m	-0.08	-0.13	0.05	-0.06	0.12
	OFV 1m	0.56	2.07	2.38	2.31	2.74
	to OFV=1	3.13	18.67	26.95	12.00	24.38
	NS to OFV=1	47	44	49	56	64
4	OFV 30m	-0.47	-0.02	0.34	0.66	0.95
	OFV 1m	-0.19	1.58	1.98	2.22	2.33
	to OFV=1	21.19	35.19	43.33	45.80	49.88
	NS to OFV=1	33	45	62	74	78
5	OFV 30m	-0.08	0.43	0.66	1.15	1.55
	OFV 1m	-0.70	1.36	1.34	1.38	1.79
	to OFV=1	-0.57	35.43	40.61	48.15	52.80
	NS to OFV=1	53	66	69	79	88
6	OFV 30m	-0.02	0.74	0.88	1.18	1.74
	OFV 1m	0.07	1.49	1.68	1.68	2.33
	to OFV=1	13.50	30.99	24.11	27.75	31.75
	NS to OFV=1	49	69	76	78	89

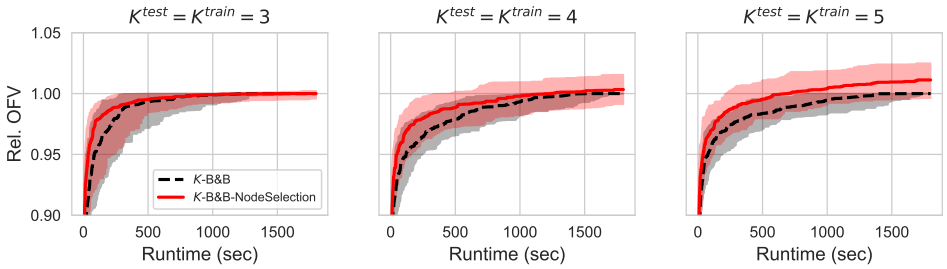


Figure 11: EXP1 results for $K^{train} = K^{test} \in \{3, 4, 5\}$ on 100 instances. The black line gives the average relative objective function value (Rel. OFV) over the runtime (in seconds) of K-B&B, with a 30 minute time limit. The red line is the Rel. OFV trajectory of K-B&B-NODESELECTION. The shaded area around the lines is their respective 75% confidence interval.

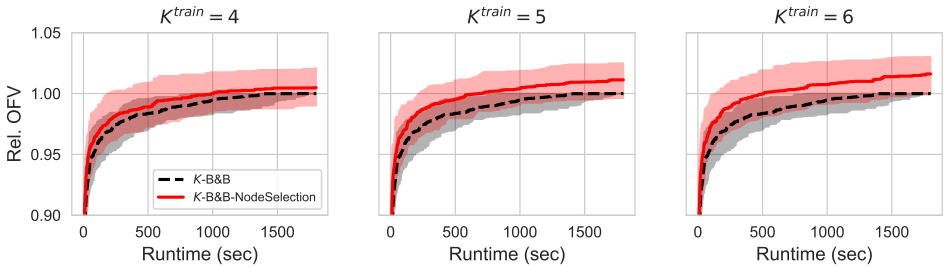


Figure 12: EXP2 results for $K^{test} = 5$ and $K^{train} \in \{4, 5, 6\}$ on 100 instances. The black line gives the average Rel. OFV of K-B&B and the red line that of K-B&B-NODESELECTION.

risk. Testing on higher instance sizes for different values of K (Figure 14) has a similar effect as before: higher values of K^{train} result in better performance (Rel. OFV ranging from 1.4-1.9%, and speedup 5-27%), although marginally for some parameter combinations.

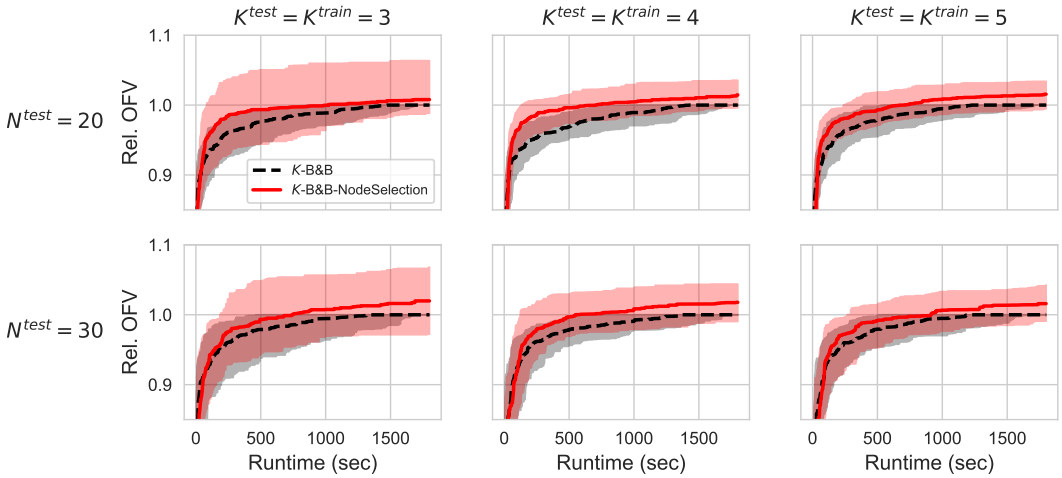


Figure 13: EXP3 results for $K^{train} = K^{test} \in \{3, 4, 5\}$ and $N^{test} \in \{20, 30\}$ on 100 instances. The black line gives the average Rel. OFV of K-B&B and the red line that of K-B&B-NODESELECTION.

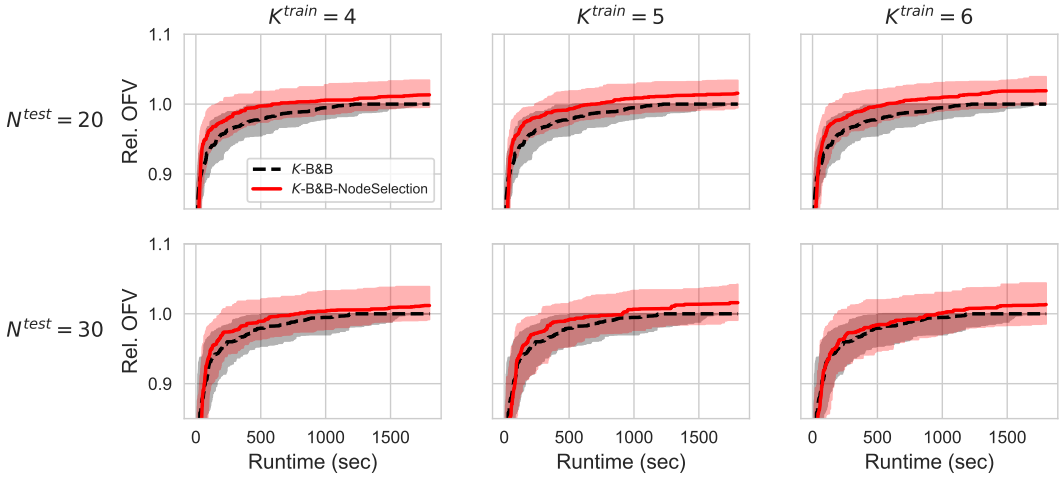


Figure 14: EXP4 results for $K^{test} = 5$, $K^{train} \in \{4, 5, 6\}$, and $N^{test} \in \{20, 30\}$ on 100 instances. The black line gives the average Rel. OFV of K-B&B and the red line that of K-B&B-NODESELECTION.

4.4 Shortest path on a sphere

The shortest path problem can be described as a 2SRO problem where we make the planning of a route from source to target, for which the lengths of the N_{sp} arcs are uncertain. This problem only has second-stage decisions and an uncertain objective. The dimension of the uncertainty set grows with the number of arcs in the graph. For the full description, see Appendix C.2. Since there is only a second stage, some attributes disappear for this problem: 'Deterministic first-stage' (attribute 6), and the static objective problem-related attributes (8 and 9). Hence, we are left with six attributes for this problem.

Parameter tuning (with EXP1)

As we did for the capital budgeting problem, we first tune the parameter ι. The details of this parameter tuning are given in Appendix D.2. The testing accuracy scores for different data sets we trained on is lower than for capital budgeting;

between 0.88-0.97. Since the distribution of the probability success values p_n is very similar to the capital budgeting problem, the quality threshold is set to $\epsilon = 0.05$. In Figure 15, the level is tested again with $L^{test} \in \{5, 10, 20, 40, \infty\}$. Here we see that especially for smaller values of K, a higher level of L^{test} outperforms others. Therefore, we select $L^{test} = \infty$ for the remainder of the experiments.

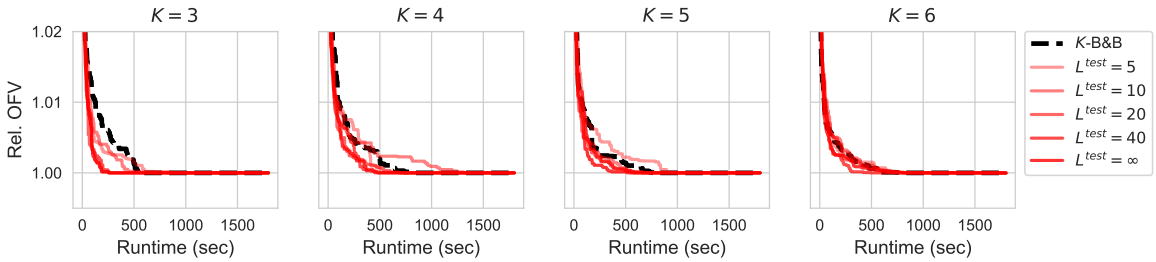


Figure 15: Results of K-B&B with random dives and K-B&B-NODESELECTION with combinations of K and L^{test}. The plots show the average objective over runtime of 100 instances for the shortest path problem, for $N = 20$.

Also note that when K grows, the performances of K-B&B and K-B&B-NODESELECTION are very similar. For the shortest path problem, we noticed that more training data points led to a substantial performance gain. See Figure 16 for these results. Therefore, for EXP1-EXP4, we select T for each K separately. For an overview of which parameters are chosen per K; see Appendix D.2.

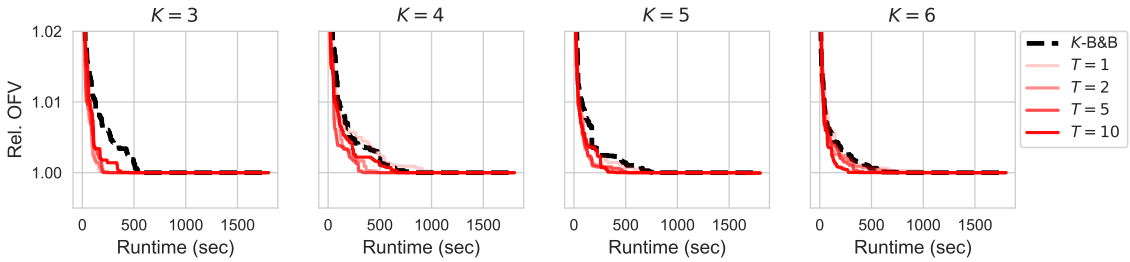


Figure 16: Results of K-B&B with random dives and K-B&B-NODESELECTION with combinations of K and T. The plots show the average objective over a runtime of 100 instances for the shortest path problem, for $N = 20$.

Results

The entirety of the results for all combinations of K^{test}, K^{train}, and N^{test} for K-B&B versus K-B&B-NODESELECTION can be found in Appendix E.2. Again, some performance statistics for EXP1-EXP2 are given, see Table 7. This table shows that the relative OFV found by K-B&B-NODESELECTION is marginally better than those of K-B&B (0.01-0.12% when excluding $K^{train} = 2$), but that the speedups can be very significant: for $K = 6$, the average speedups range from 53-94%. For this problem class, the instances of $K = 2$ seem to benefit from ML enhancements.

EXP1 and EXP2 results. As is visible from Figure 17, even though no better solutions are found (probably because the optimal solution is found early on), it is still noticeable that K-B&B-NODESELECTION converges faster than K-B&B. This phenomenon is thus consistent over the two problems. The convergence of K-B&B over different values of K^{test} differs significantly (*e.g.*, compare K^{test} equal to 3 and 5). The convergence of K-B&B-NODESELECTION is however quite stable across different values of K^{test}. Then, for EXP2, in Figure 18, we see that for $K^{test} = 4$, $K^{train} = 6$ performs best. For other values of K^{test}, $K^{train} = 6$ behaves well too, just as it did for the capital budgeting problem.

EXP3 and EXP4 results. We see in Figure 19 that the two algorithms behave very similarly. The average relative OFV ranges from 0.14-0.55%. Interestingly, as not easily visible from the figure, the average speedup values are quite high: ranging from 21-98%. The speedup of 98% is achieved for $K = 6$ and $N = 40$, where the line of K-B&B consistently lies slightly above the one of K-B&B-NODESELECTION. In terms of stability, the confidence interval of K-B&B-NODESELECTION grows with N^{test}. However, this does not necessarily mean that testing and training on different sizes is not stable: the CI of K-B&B is also bigger. Moreover, the CI is mostly in the region below one,

Table 7: Combined results of EXP1 (along diagonal) and EXP2 for the shortest path problem. The four statistics described in the caption of Table 5 are given: (i) 'OFV 30m' (higher is better), (ii) 'OFV 1m', (iii) 'to OFV=1' (higher is better), and (iv) 'NS to OFV=1' (higher is better).

K^{test}	**Statistic**	K^{train} 2	3	4	5	6
2	OFV 30m	0.05	0.07	0.03	0.03	0.08
	OFV 1m	0.08	1.10	1.08	1.04	0.10
	to OFV=1	36.63	15.83	24.77	22.31	12.31
	NS to OFV=1	98	100	99	100	97
3	OFV 30m	0.01	0.14	0.13	0.11	0.12
	OFV 1m	0.17	0.45	0.51	0.19	0.52
	to OFV=1	60.44	46.93	34.30	49.02	29.06
	NS to OFV=1	99	100	99	100	100
4	OFV 30m	-0.12	0.10	0.12	0.11	0.12
	OFV 1m	0.89	0.41	0.41	0.47	0.65
	to OFV=1	55.90	63.34	75.11	45.12	45.79
	NS to OFV=1	99	100	100	100	100
5	OFV 30m	-0.07	0.12	0.09	0.15	0.10
	OFV 1m	-0.33	0.21	0.16	0.23	0.37
	to OFV=1	62.87	81.84	77.14	64.61	52.25
	NS to OFV=1	100	100	100.00	100	100
6	OFV 30m	-0.07	0.04	0.01	0.10	0.07
	OFV 1m	-0.07	0.11	0.17	0.20	0.23
	to OFV=1	52.26	67.37	67.55	88.17	93.68
	NS to OFV=1	100	100	100	100	100

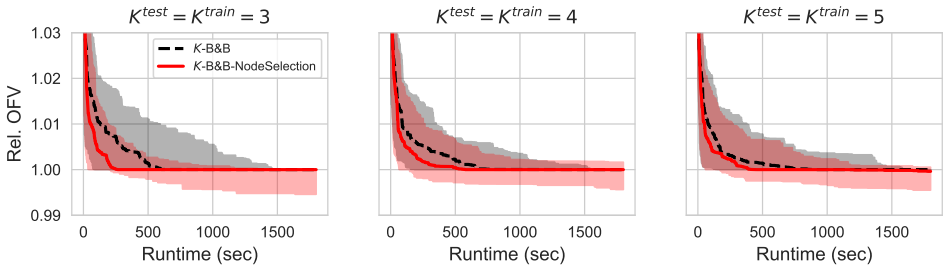


Figure 17: EXP1 results for $K^{train} = K^{test} \in \{3, 4, 5\}$ on 100 instances. The black line gives the average relative objective function value (Rel. OFV) over the runtime (in seconds) of K-B&B, with a 30 minute time limit. The red line is the Rel. OFV trajectory of K-B&B-NODESELECTION. The shaded area around the lines is their respective 80% confidence interval.

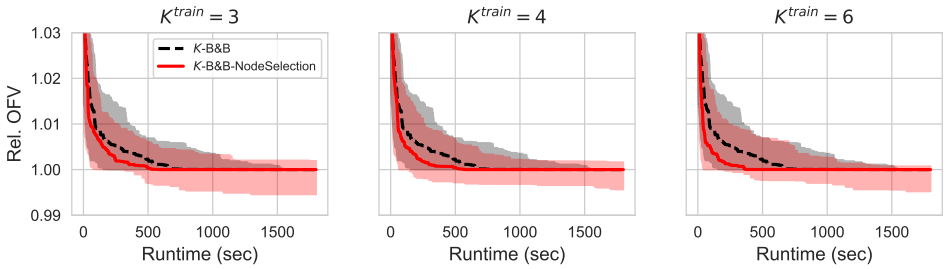


Figure 18: EXP2 results for $K^{test} = 4$ and $K^{train} \in \{3, 4, 6\}$ on 100 instances. The black line gives the average Rel. OFV of K-B&B and the red line that of K-B&B-NODESELECTION.

which indicates that we mainly have well-performing outliers. In Figure 20, the results of EXP4 are shown, where the average relative OFV improvement ranges from 0.29-0.51% and the speedups ($K = 4$ excluded) are between 32-77%. We see that for $N^{test} = 40$, $K^{train} = 6$ performs best, but not necessarily for the biggest instance size.

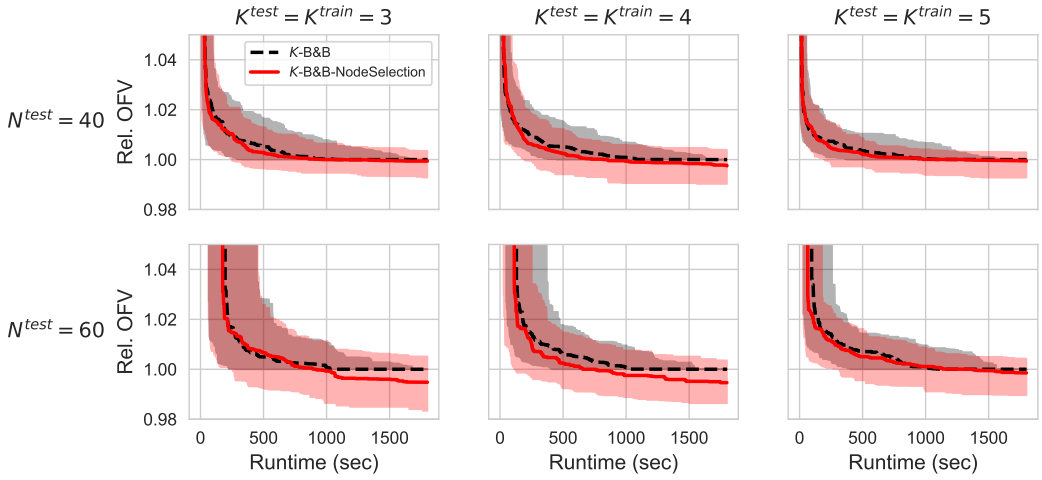


Figure 19: EXP3 results for $K^{train} = K^{test} \in \{3, 4, 5\}$ and $N^{test} \in \{40, 60\}$ on 100 instances. The black line gives the average Rel. OFV of K-B&B and the red line that of K-B&B-NODESELECTION.

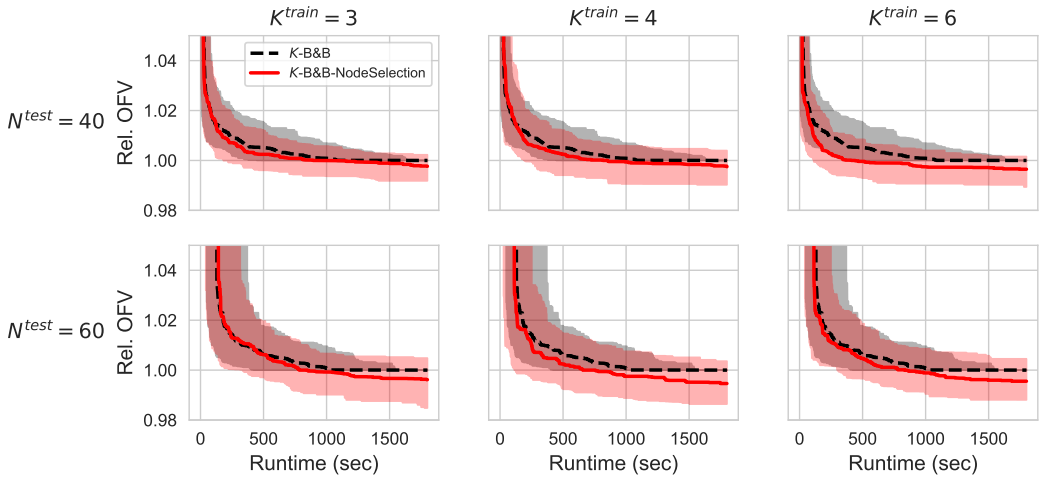
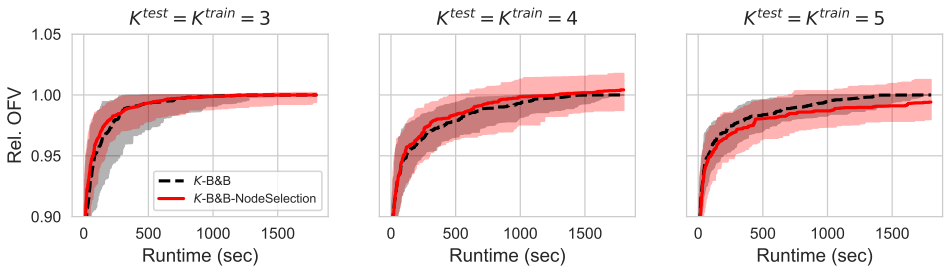


Figure 20: EXP4 results for $K^{test} = 4$, $K^{train} \in \{3, 4, 6\}$, and $N^{test} \in \{40, 60\}$ on 100 instances. The black line gives the average Rel. OFV of K-B&B and the red line that of K-B&B-NODESELECTION.

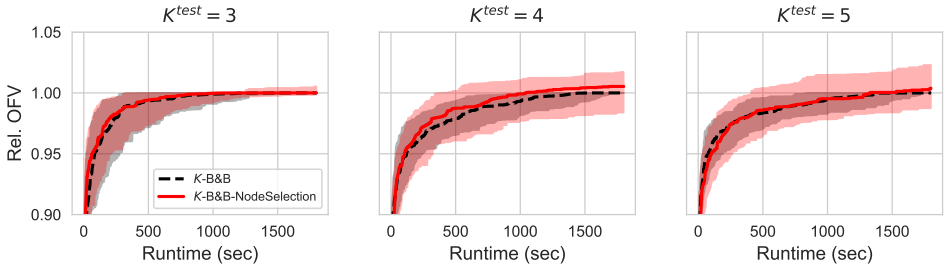
4.5 Training and testing on different problems

In this section, we show the results of EXP5 and EXP6, where we apply the node selection strategy to a different problem than it has been trained on. Note that the shortest path problem does not have first-stage decisions. This results in the features being slightly different than for the capital budgeting problem. Therefore, to create a model that can be trained by shortest path data, and used for the capital budgeting problem, the first-stage-related attributes are not constructed while running K-B&B-NODESELECTION. Full results of these experiments are given in Appendix E.3.

Figure 21a shows the results of EXP5 for the capital budgeting problem. We can see that the performances of the two algorithms are very close. Recall that $K^{train} = 6$ previously resulted in (one of) the best solutions. Then, when we look at some of the solutions of EXP6 with $K^{train} = 6$ (see Figure 21b), we see this is not the case here.



(a) EXP5



(b) EXP6

Figure 21: EXP5 results of capital budgeting for $K^{test} = K^{train} \in \{3, 4, 5\}$ and EXP6 results for $K^{train} = 6$ and $K^{test} \in \{3, 4, 5\}$ on 100 instances. The black line gives the average Rel. OFV of K-B&B and the red line that of K-B&B-NODESELECTION.

Now we show the results of the shortest path problem that uses a ML model trained on capital budgeting data. Due to the mismatch of features, we delete the three first-stage-related features not used by shortest path from the capital budgeting data. We can still use the generated capital budgeting data. For an illustration of EXP5, see Figure 22a. These plots illustrate that for two out of three values of K^{test}, K-B&B-NODESELECTION outperforms K-B&B, even though it is trained on data of another problem. More interesting is the following: the performance of different values of K^{test} with $K^{train} = 6$ gives very good results. They are as good as the ones trained on shortest path data. For an illustration of this, see Figure 22b.

4.6 Feature importance

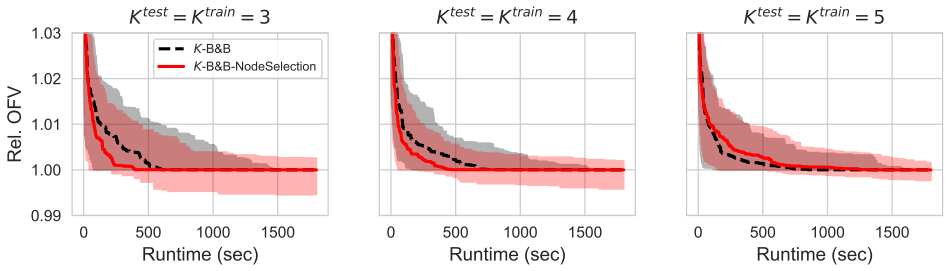
A trained random forest model allows us to compute feature importance scores. For both problems tested on, these scores are given in Table 8 for $K^{train} \in \{2, 3, 4, 5, 6\}$. We observe that the relatively highest importance corresponds to the 'Objective' feature (a state feature), with the importance of the other features (both state and scenario ones), being of similar magnitude. Because of that, we cannot draw significant conclusions about the relative importance of the remaining features.

5 Conclusion and future work

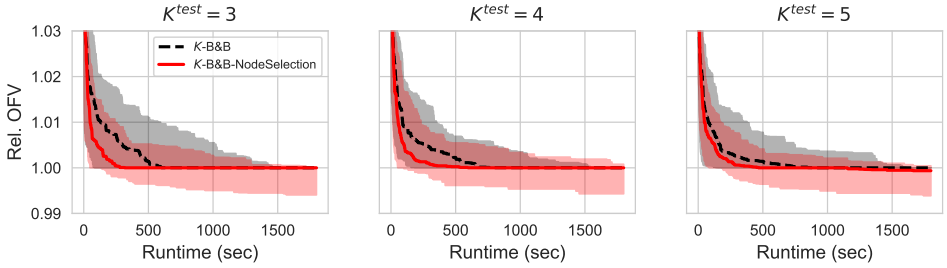
We introduce an ML-based method to improve the K-B&B algorithm [Subramanyam et al., 2020] for solving 2SRO optimization problems. K-B&B uses a search tree to optimally partition the uncertainty set into K parts and we propose to use a supervised ML model that learns the best node selection strategy to explore such trees faster.

For this, we designed a procedure for generating training data and formulated the ML features based on our knowledge of 2SRO so that they are independent of the size, the value of K, and the type of problems on which the ML tool is trained. We experimentally show that our method outperforms K-B&B on the problems we test on. We see that when a problem is trained on a smaller instance size, and then applied to the same problem type with bigger instances, our method still outperforms K-B&B, although being less stable. Training and testing on entirely different problem types resulted in mixed results.

As K-B&B has a tree search structure, we believe that our work can be used to tackle other problems solved by a similar tree search structure, wherever expert knowledge can be used to construct meaningful problem size-independent features.



(a) EXP5



(b) EXP6

Figure 22: EXP5 results of shortest path for $K^{test} = K^{train} \in \{3, 4, 5\}$ and EXP6 results for $K^{train} = 6$ and $K^{test} \in \{3, 4, 5\}$ on 100 instances. The black line gives the average Rel. OFV of K-B&B and the red line that of K-B&B-NODESELECTION.

Table 8: Feature importance scores of the selected random forest models for the two problems capital budgeting and shortest path, for each K^{train}. The bold scores are given to the scores that belong to the two highest ones of that model. The dashes correspond to the omitted features for the shortest path problem.

Feature name		**Capital budgeting**					**Shortest path**				
	K^{train}	2	3	4	5	6	2	3	4	5	6
Objective		**0.25**	**0.25**	**0.24**	**0.28**	**0.25**	**0.10**	**0.11**	**0.24**	**0.13**	**0.15**
Objective difference		0.05	0.04	0.04	0.03	0.04	0.06	0.05	0.04	0.05	0.05
Violation		**0.12**	**0.10**	0.07	0.07	0.06	0.09	0.09	0.10	0.10	0.10
Violation difference		0.05	0.06	0.05	0.05	0.05	0.09	0.09	0.08	0.10	0.09
Depth		0.05	0.08	0.07	0.04	0.03	**0.12**	**0.13**	**0.12**	**0.11**	**0.10**
Scenario values		0.06	0.06	0.05	0.05	0.06	0.09	**0.11**	0.08	0.10	0.09
Constraint distance		0.08	0.06	0.05	0.05	0.05	0.10	0.10	0.08	0.10	**0.10**
Scenario distance		0.06	0.06	0.07	0.07	0.05	0.09	0.10	0.07	0.09	0.09
Constraint slacks		0.07	0.06	0.05	0.04	0.04	0.09	0.06	0.05	0.07	0.07
Det. Objective value		0.05	0.06	**0.11**	**0.12**	**0.12**	0.09	0.09	0.08	0.09	0.09
Det. First-stage decisions		0.05	0.06	0.05	0.06	0.07	-	-	-	-	-
Det. Second-stage decisions		0.00	0.00	0.00	0.00	0.00	0.08	0.07	0.06	0.06	0.06
Stat. Objective value		0.06	0.06	0.08	0.10	**0.12**	-	-	-	-	-
Stat. Second-stage decisions		0.06	0.05	0.05	0.05	0.05	-	-	-	-	-

References

Aharon Ben-Tal, Alexander Goryashko, Elana Guslitzer, and Arkadi Nemirovski. Adjustable robust solutions of uncertain linear programs. *Mathematical Programming*, 99(2):351–376, 2004.

Aharon Ben-Tal, Laurent El Ghaoui, and Arkadi Nemirovski. *Robust Optimization*, volume 28. Princeton University Press, 2009.

Yoshua Bengio, Andrea Lodi, and Antoine Prouvost. Machine learning for combinatorial optimization: a methodological tour d'horizon. *European Journal of Operational Research*, 2020.

Dimitris Bertsimas and Constantine Caramanis. Finite adaptability in multistage linear optimization. *IEEE Transactions on Automatic Control*, 55(12):2751–2766, 2010.

Dimitris Bertsimas and Iain Dunning. Multistage robust mixed-integer optimization with adaptive partitions. *Operations Research*, 64(4):980–998, 2016.

Cameron B Browne, Edward Powley, Daniel Whitehouse, Simon M Lucas, Peter I Cowling, Philipp Rohlfshagen, Stephen Tavener, Diego Perez, Spyridon Samothrakis, and Simon Colton. A survey of monte carlo tree search methods. *IEEE Transactions on Computational Intelligence and AI in games*, 4(1):1–43, 2012.

Christoph Buchheim and Jannis Kurtz. Min–max–min robust combinatorial optimization. *Mathematical Programming*, 163(1):1–23, 2017.

Alireza Ghahtarani, Ahmed Saif, Alireza Ghasemi, and Erick Delage. A double-oracle, logic-based benders decomposition approach to solve the k-adaptability problem. *Computers & Operations Research*, 155:106243, 2023.

LLC Gurobi Optimization. Gurobi optimizer reference manual, 2020. URL `http://www.gurobi.com`.

Elana Guslitzer. Uncertainty-immunized solutions in linear programming. Master's thesis, Technion, Israeli Institute of Technology, 2002.

Grani A Hanasusanto, Daniel Kuhn, and Wolfram Wiesemann. K-adaptability in two-stage robust binary programming. *Operations Research*, 63(4):877–891, 2015.

He He, Hal Daume III, and Jason M Eisner. Learning to search in branch and bound algorithms. *Advances in neural information processing systems*, 27:3293–3301, 2014.

Elias B Khalil, Christopher Morris, and Andrea Lodi. Mip-gnn: A data-driven framework for guiding combinatorial solvers. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 36, pages 10219–10227, 2022a.

Elias B Khalil, Pashootan Vaezipoor, and Bistra Dilkina. Finding backdoors to integer programs: A monte carlo tree search framework. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 36, pages 3786–3795, 2022b.

Manuel Loth, Michele Sebag, Youssef Hamadi, and Marc Schoenauer. Bandit-based search for constraint programming. In *International Conference on Principles and Practice of Constraint Programming*, pages 464–480. Springer, 2013.

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

Krzysztof Postek and Dick den Hertog. Multistage adjustable robust mixed-integer optimization via iterative splitting of the uncertainty set. *INFORMS Journal on Computing*, 28(3):553–574, 2016.

Ashish Sabharwal, Horst Samulowitz, and Chandra Reddy. Guiding combinatorial optimization with uct. In *International conference on integration of artificial intelligence (AI) and operations research (OR) techniques in constraint programming*, pages 356–361. Springer, 2012.

Anirudh Subramanyam, Chrysanthos E Gounaris, and Wolfram Wiesemann. K-adaptability in two-stage mixed-integer robust optimization. *Mathematical Programming Computation*, 12(2):193–224, 2020.

Maciej Świechowski, Konrad Godlewski, Bartosz Sawicki, and Jacek Mańdziuk. Monte carlo tree search: A review of recent modifications and applications. *Artificial Intelligence Review*, pages 1–66, 2022.

A Attribute descriptions

Each scenario gets its own set of attributes. In total there are nine types: one is the scenario vector $\boldsymbol{z}$ itself ($\boldsymbol{a}_1^{\boldsymbol{z}} = \boldsymbol{z}$), three are determined by the solutions of the main problem, three are extracted from solving the *deterministic problem*, and two are taken from solving the *static problem*.

***Main problem-based attributes*.** For these attributes, only little additional computation is needed. This is due to the fact that the values we use can be taken from the current solution of the main problem. Attributes 2-4 in Table 2 of the paper share this property:

2. **Constraint distance:** For the first attribute of the 'main-problem' type, we look at the constraints generated by the new scenario $\boldsymbol{z}^*$. We call these constraints the '$\boldsymbol{z}^*$-constraints'. When a constraint is added to a problem, the resulting feasible region will always be as large as, or smaller than, the feasible region we had before. When the feasible region is large, the objective value we find is often better than for smaller feasible regions (the optimal value found in the large region may be cut off in the small region). Each subset $\bar{\mathcal{Z}}_1, \dots, \bar{\mathcal{Z}}_K$ consists of its own set of scenarios, which translates to its own set of constraints in the main problem. These are the '$\bar{\mathcal{Z}}_k$-constraints', for all $k \in \mathcal{K}$. These $\bar{\mathcal{Z}}_k$-constraints form a feasibility region for the decision pair $(\boldsymbol{x}, \boldsymbol{y}_k)$. Ideally, we would calculate the volume of the feasible region whenever the $\boldsymbol{z}^*$-constraints are added to the existing feasible regions. However, obtaining this result is computationally intractable. Therefore we only look at the distance between the constraints. We generate the cosine similarity between the $\boldsymbol{z}^*$-constraints and the $\bar{\mathcal{Z}}_k$-constraints, for each subset separately. When the cosine similarity is high, the distance between the constraints is low. Then, the attribute of the k-th child node $\boldsymbol{a}_2^{k,\boldsymbol{z}^*}$ is the cosine similarity of the $\boldsymbol{z}^*$-constraints and the $\bar{\mathcal{Z}}_k$-constraints:
$$a_{2,c}^{k,\boldsymbol{z}^*} = \max_{\boldsymbol{z} \in \bar{\mathcal{Z}}_k} \quad \frac{\gamma_c(\boldsymbol{z}^*) \cdot \gamma_c(\boldsymbol{z})}{\|\gamma_c(\boldsymbol{z}^*)\| \|\gamma_c(\boldsymbol{z})\|}, \qquad \forall c \in \{1, \dots, C\}, \quad \forall k \in \mathcal{K},$$
where $a_{2,c}^{k,\boldsymbol{z}^*} \in [-1, 1]$ and $\gamma_c : \mathcal{Z} \to \mathcal{X} \times \mathcal{Y}$ is a function of the left-hand-side of constraint $c \in \{1, \dots, C\}$ with input scenario $\boldsymbol{z}$. The first- and second-stage decisions are variable. Finally, the attribute of the k-th child node is formulated as $\boldsymbol{a}_2^{k,\boldsymbol{z}^*} = [a_{2,1}^{k,\boldsymbol{z}^*}, \dots, a_{2,C}^{k,\boldsymbol{z}^*}]$. Thus, the length of $\boldsymbol{a}_2^{k,\boldsymbol{z}^*}$ is equal to the number of uncertain constraints of the MILP formulation of the problem.
3. **Scenario distance:** Per subset, we wish to know how far away the new scenario $\boldsymbol{z}^*$ is from *not* being a violating scenario. We suspect that if the distance is small rather than large, the current solution will not change much. Thus, will not become much worse. This attribute first takes the current solutions of the main problem. Then, for the k-th subset it determines the distance between each of the following planes (boundaries of constraints of Eq. (4) in the paper) and the new scenario $\boldsymbol{z}^* \in \mathcal{Z}$:
$$\begin{aligned} &c(\boldsymbol{z})^\intercal \boldsymbol{x}^* + d(\boldsymbol{z})^\intercal \boldsymbol{y}_k^* - \theta^* = 0, \\ &T_c(\boldsymbol{z})\boldsymbol{x}^* + W_c(\boldsymbol{z})\boldsymbol{y}_k^* - h_c(\boldsymbol{z}) = 0, \quad \forall c \in \{2, \dots, C\}. \end{aligned} \tag{7}$$
Hence, we need to determine the distance between a plane and a point, for each constraint $c \in \{1, \dots, C\}$, where $c = 1$ corresponds to the objective function. The point-to-plane distance of the c-th plane for subset k is calculated by projecting $\boldsymbol{z}^*$ on the normal vector of the plane as follows:
$$\chi_c^k = \frac{|\boldsymbol{\rho}_c(\boldsymbol{x}^*, \boldsymbol{y}^k)^\intercal \boldsymbol{z}^*|}{\|\boldsymbol{\rho}_c(\boldsymbol{x}^*, \boldsymbol{y}^k)\|},$$
where $\boldsymbol{\rho}_c : \mathcal{X} \times \mathcal{Y} \to \mathcal{Z}$ is a vector of coefficients of constraint c of Eq. (7). To compare the point-to-plane distance of the K subsets, we scale over the sum of the distance of the subsets. Then, the attribute is given as:
$$a_{3,c}^{k,\boldsymbol{z}^*} = \frac{\chi_c^k}{\sum_{k' \in \mathcal{K}} \chi_c^{k'}}, \quad \forall c \in \{1, \dots, C\}, \quad \forall k \in \mathcal{K}.$$
4. **Constraint slacks:** This attribute takes the slack values of the uncertain constraints of the main problem for the new scenario $\boldsymbol{z}^*$ with fixed first- and second-stage decisions $\boldsymbol{x}^*$ and $\boldsymbol{y}^*$. For the k-th subset and the constraints $c \in \{1, \dots, C\}$, with $c = 1$ the objective constraint, we get:
$$\begin{aligned} s_1^k &= |c(\boldsymbol{z}^*)^\intercal \boldsymbol{x}^* + d(\boldsymbol{z}^*)^\intercal \boldsymbol{y}_k^* - \theta^*|, \\ s_c^k &= |T_c(\boldsymbol{z}^*)\boldsymbol{x}^* + W_c(\boldsymbol{z}^*)\boldsymbol{y}_k^* - h_c(\boldsymbol{z}^*)|, \qquad \forall c \in \{2, \dots, C\}. \end{aligned}$$
Similarly as with the previous attribute, we compare the slack values of the subsets by scaling over the sum of the slacks of all subsets:
$$a_{4,c}^{k,\boldsymbol{z}^*} = \frac{s_c^k}{\sum_{k' \in \mathcal{K}} s_c^{k'}}, \quad \forall c \in \{1, \dots, C\}, \quad \forall k \in \mathcal{K}.$$

Deterministic problem-based attributes. This problem solves the 2SRO problem for where the newly found scenario $\boldsymbol{z}^*$ is the only scenario considered. We call this a deterministic problem, since we no longer deal with uncertainty. The problem is formulated as follows:

$$\begin{aligned} \min_{\theta^n, \boldsymbol{x}^n, \boldsymbol{y}^n} \quad & \theta^n \\ \text{s.t.} \quad & \theta^n \in \mathbb{R}, \boldsymbol{x}^n \in \mathcal{X}, \boldsymbol{y}^n \in \mathcal{Y}, \\ & \boldsymbol{c}(\boldsymbol{z}^*)^\intercal \boldsymbol{x}^n + \boldsymbol{d}(\boldsymbol{z}^*)^\intercal \boldsymbol{y}^n \leq \theta^n, \\ & \boldsymbol{T}(\boldsymbol{z}^*)\boldsymbol{x}^n + \boldsymbol{W}(\boldsymbol{z}^*)\boldsymbol{y}^n \leq \boldsymbol{h}(\boldsymbol{z}^*). \end{aligned} \tag{8}$$

By solving this problem we obtain Attributes 5-7:

5. **Deterministic objective** function value θ^n,
6. **Deterministic first-stage** decisions $\boldsymbol{x}^n$,
7. **Deterministic second-stage** decisions $\boldsymbol{y}^n$.

Static problem-based attributes. A very simple method for approximately solving the 2SRO problem is to first solve the first-stage decision for all scenarios in the uncertainty set. Then, after a realization of uncertainty, we combine this first-stage decision with the scenario to determine the second-stage decision. This is an naive way of solving 2SRO, thus not optimal for all scenarios. But, it could give us some information on the approximate solutions to scenarios in the problem. Solving the static problem consists of two steps: First, we obtain the static robust first-stage decisions $\bar{\boldsymbol{x}}$ by solving

$$\begin{aligned} \min_{\theta, \boldsymbol{x}, \boldsymbol{y}} \quad & \theta \\ \text{s.t.} \quad & \theta \in \mathbb{R}, \boldsymbol{x} \in \mathcal{X}, \boldsymbol{y} \in \mathcal{Y}, \\ & \boldsymbol{c}(\boldsymbol{z})^\intercal \boldsymbol{x} + \boldsymbol{d}(\boldsymbol{z})^\intercal \boldsymbol{y} \leq \theta, \qquad \forall \boldsymbol{z} \in \mathcal{Z}, \\ & \boldsymbol{T}(\boldsymbol{z})\boldsymbol{x} + \boldsymbol{W}(\boldsymbol{z})\boldsymbol{y} \leq \boldsymbol{h}(\boldsymbol{z}), \qquad \forall \boldsymbol{z} \in \mathcal{Z}. \end{aligned} \tag{9}$$

This problem can be reformulated via the mathematically tractable formulation presented in Ben-Tal et al. [2009]. Secondly, by fixing $\boldsymbol{x}$ to $\bar{\boldsymbol{x}}$, we obtain the objective value θ^s and second-stage decisions $\boldsymbol{y}^s$ by solving

$$\begin{aligned} \min_{\theta^s, \boldsymbol{y}^s} \quad & \theta^s \\ \text{s.t.} \quad & \theta^s \in \mathbb{R}, \boldsymbol{y}^s \in \mathcal{Y}, \\ & \boldsymbol{c}(\boldsymbol{z}^*)^\intercal \bar{\boldsymbol{x}} + \boldsymbol{d}(\boldsymbol{z}^*)^\intercal \boldsymbol{y}^s \leq \theta^s, \\ & \boldsymbol{T}(\boldsymbol{z}^*)\bar{\boldsymbol{x}} + \boldsymbol{W}(\boldsymbol{z}^*)\boldsymbol{y}^s \leq \boldsymbol{h}(\boldsymbol{z}^*). \end{aligned} \tag{10}$$

Note that problem (9) is solved only once in the algorithm, while problem (10) needs to be solved for each scenario. By solving this problem we obtain Attributes 8 and 9:

8. **Static objective** function value θ^s,
9. **Static second-stage** decisions $\boldsymbol{y}^s$

B Omitted pseudocodes

The K-adaptability branch-and-bound (K-B&B) algorithm, as presented in Subramanyam et al. [2020], is given in Algorithm 2. In our implementation of this algorithm, we apply a depth-first search strategy and select a random node of $\mathcal{N}$ instead of the first one.

Algorithm 2: K-B&B

Input : Problem instance $\mathcal{P}(N)$ with size N,
number of partitions K

Output : Objective value θ, first-stage decisions $\boldsymbol{x}$, second-stage decisions $\boldsymbol{y} = \{\boldsymbol{y}_1, \ldots, \boldsymbol{y}_K\}$,
subsets with scenarios $\bar{\mathcal{Z}}_k$ for all $k \in \{1, \ldots, K\}$

Initialization : Incumbent partition: $\tau^i := \{\bar{\mathcal{Z}}_1, \ldots, \bar{\mathcal{Z}}_K\}$, where $\bar{\mathcal{Z}}_k = \emptyset$ for all $k \in \mathcal{K}$,
set containing all node partitions yet to explore: $\mathcal{N} := \{\tau^i\}$,
incumbent solutions: $(\theta^i, \boldsymbol{x}^i, \boldsymbol{y}^i) := (\infty, \emptyset, \emptyset)$

1 **while** $\mathcal{N}$ *not empty* **do**
2 Select the first node with partition $\tau = \{\bar{\mathcal{Z}}_1, \ldots, \bar{\mathcal{Z}}_K\}$ from $\mathcal{N}$, then $\mathcal{N} \leftarrow \mathcal{N} \setminus \{\tau\}$
3 $(\theta^*, \boldsymbol{x}^*, \boldsymbol{y}^*) \leftarrow$ *main problem*(τ)
4 **if** $\theta^* > \theta^i$ **then**
5 Prune tree since current objective is worse than best solution found.
6 Continue to line 2.
7 **end**
8 $(\boldsymbol{z}^*, \zeta^*) \leftarrow$ *subproblem*$(\theta^*, \boldsymbol{x}^*, \boldsymbol{y}^*)$
9 **if** $\zeta^* > 0$ **then**
10 Solution not robust, create K new branches.
11 **for** $k \in \{1, \ldots, K\}$ **do**
12 $\tau^k := \{\bar{\mathcal{Z}}_1, \ldots, \bar{\mathcal{Z}}_k \cup \{\boldsymbol{z}^*\}, \ldots, \bar{\mathcal{Z}}_K\}$
13 $\mathcal{N} \leftarrow \mathcal{N} \cup \{\tau^k\}$
14 **end**
15 **else**
16 Current solution robust, prune tree.
17 $(\theta^i, \boldsymbol{x}^i, \boldsymbol{y}^i, \tau^i) \leftarrow (\theta^*, \boldsymbol{x}^*, \boldsymbol{y}^*, \tau)$
18 **end**
19 **end**
20 **return** $(\theta^i, \boldsymbol{x}^i, \boldsymbol{y}^i, \tau^i)$

The steps for obtaining the ML model used for node selection consists of two parts: (i) making training data and (ii) training the ML model. More details on these steps are given in Procedure 3.

Procedure 3: StrategyModel

Input : Train instances $\mathcal{P}_1^{train}(N^{train}), \ldots, \mathcal{P}_I^{train}(N^{train})$
number of partitions for training K^{train},
level for training L^{train},
quality threshold ϵ,
R for random dives per node

Output : Trained node selection strategy *model*.

`// Get training data for` I `instances`
for $i \in \{1, \ldots, I\}$ **do**
  $(D, \boldsymbol{p})_i \leftarrow$ *generate train data*$(\mathcal{P}_i^{train}(N^{train}), K^{train}, L^{train}, R)$ $\boldsymbol{q}_i \leftarrow$ *quality*$(\boldsymbol{p}_i, \epsilon)$
end
`// Train node selection strategy model`
Set *model* $\leftarrow$ *train ML model*$(\{(D, \boldsymbol{q})_1, \ldots, (D, \boldsymbol{q})_I\})$
return *model*

For scaling some of the features (see Section 3.2 of the paper), information needs to be gathered by performing several initial dives in the tree. The steps of these dives are given in Procedure 4.

Procedure 4: INITIALRUN

Input : Test instance $\mathcal{P}^{test}(N^{test})$,
number of partitions for testing K^{test}

Output : Objective value θ, first-stage decisions $\boldsymbol{x}$, second-stage decisions $\boldsymbol{y} = \{\boldsymbol{y}_1, \ldots, \boldsymbol{y}_K\}$,
Subsets with scenarios $\bar{\mathcal{Z}}_k$ for all $k \in \{1, \ldots, K\}$

Initialization : Incumbent partition: $\tau^i := \{\bar{\mathcal{Z}}_1, \ldots, \bar{\mathcal{Z}}_K\}$, where $\bar{\mathcal{Z}}_k = \emptyset$ for all $k \in \mathcal{K}$,

Set $\tau = \tau^i$
while *solution not robust* **do**
  $(\theta^*, \boldsymbol{x}^*, \boldsymbol{y}^*) \leftarrow$ *main problem*(τ)
  $(\boldsymbol{z}^*, \zeta^*) \leftarrow$ *subproblem*$(\theta^*, \boldsymbol{x}^*, \boldsymbol{y}^*)$
  if $\zeta^* > 0$ **then**
    Solution not robust, random node selection.
    $k^{'} \leftarrow$ *random uniform sample*$([1, K])$
    $\tau := \{\bar{\mathcal{Z}}_1, \ldots, \bar{\mathcal{Z}}_{k'} \cup \{\boldsymbol{z}^*\}, \ldots, \bar{\mathcal{Z}}_K\}$
  else
    Robust solution found
    scaling info $\leftarrow (\theta^0, \zeta^0, \kappa^0)$
  end
end
return *scaling info*

C Problem formulations

In the experiments, our method has been tested for several problems. The descriptions of these problems and their MILP formulations are given in this section.

C.1 Capital budgeting with loans

We consider the capital budgeting with loans problem as defined in Subramanyam et al. [2020], where a company wishes to invest in a subset of N projects. Each project i has an uncertain cost $c_i(\boldsymbol{z})$ and an uncertain profit $r_i(\boldsymbol{z})$, defined as

$$c_i(\boldsymbol{z}) = \big(1 + \boldsymbol{\Phi}_i^\intercal \boldsymbol{z}/2\big)c_i^0 \quad \text{and} \quad r_i(\boldsymbol{z}) = \big(1 + \boldsymbol{\Psi}_i^\intercal \boldsymbol{z}/2\big)r_i^0, \quad \forall i \in \{1, \ldots, N\},$$

where c_i^0 and r_i^0 represent the nominal cost and the nominal profit of project i, respectively. $\boldsymbol{\Phi}_i^\intercal$ and $\boldsymbol{\Psi}_i^\intercal$ represent the i-th row vectors of the sensitivity matrices $\boldsymbol{\Phi}, \boldsymbol{\Psi} \in \mathbb{R}^{N \times N_z}$. The realizations of the uncertain vector $\boldsymbol{z}$ belong to the uncertainty set $\mathcal{Z} = [-1, 1]^{N_z}$, where N_z is the dimension of the uncertainty set.

The company can invest in a project either before or after observing the risk factor $\boldsymbol{z}$. In the latter case, the company generates only a fraction η of the profit, which reflects a penalty of postponement. However, the cost remains the same as in the case of an early investment. The company has a given budget B, which the company can increase by loaning from the bank at a unit cost of $\lambda > 0$, before the risk factors $\boldsymbol{z}$ are observed. A loan after the observation occurs, has a unit cost of $\mu\lambda$, with $\mu > 1$. The objective of the capital budgeting problem is to maximize the total revenue subject to the budget. This problem can be formulated as an instance of the K-adaptability problem as follows:

$$\begin{aligned}
\max_{(x_0, \boldsymbol{x}) \in \mathcal{X}, (y_0, \boldsymbol{y}) \in \mathcal{Y}^K} \min_{\boldsymbol{z} \in \mathcal{Z}} \max_{k \in \mathcal{K}} \quad & \theta \\
\text{s.t.} \quad & r(\boldsymbol{z})^\intercal(\boldsymbol{x} + \eta \boldsymbol{y}_k) - \lambda(x_0 + \mu y_0^k) \geq \theta, \\
& \boldsymbol{x} + \boldsymbol{y}_k \leq \boldsymbol{e}, \\
& \boldsymbol{c}(\boldsymbol{z})^\intercal \boldsymbol{x} \leq B + x_0, \\
& \boldsymbol{c}(\boldsymbol{z})^\intercal(\boldsymbol{x} + \boldsymbol{y}_k) \leq B + x_0 + y_0^k,
\end{aligned}$$

where $\mathcal{X} = \mathcal{Y} = \mathbb{R}_+ \times \{0, 1\}^N$, $y_0 = \{y_0^1, \ldots, y_0^K\}$, $\boldsymbol{y} = \{\boldsymbol{y}_1, \ldots, \boldsymbol{y}_K\}$, x_0 and y_0 are the amounts of taken loan in the first and second stage, respectively. Moreover, x_i and y_i are the binary variables that indicate whether we invest in the i-th project in the first- and second-stage, respectively. The constraints $\boldsymbol{c}(\boldsymbol{z})^\intercal \boldsymbol{x} \leq B + x_0$ ensure that for the first stage, the expenditures are not more than the budget plus the loan taken before the realization of uncertainty.

Test case. Similarly as in Subramanyam et al. [2020], the uncertainty set dimension N_z is set to $N_z = 4$. The nominal cost vector $\boldsymbol{c}^0$ is chosen uniformly at random from the set $[0, 10]^N$. Let $\boldsymbol{r}^0 = \boldsymbol{c}^0/5$, $B = \boldsymbol{e}^\intercal \boldsymbol{c}^0/2$, and $\eta = 0.8$. The

rows of the sensitivity matrices $\boldsymbol{\Phi}$ and $\boldsymbol{\Psi}$ are sampled uniformly from the i-th row vector, which is sampled from $[0,1]^{N_z}$, such that $\boldsymbol{\Phi}_i^\intercal \boldsymbol{e} = \boldsymbol{\Psi}_i^\intercal \boldsymbol{e} = 1$ for all $i \in \{1,\dots,N\}$. This is also known as the unit simplex in $\mathbb{R}^{N_p}$. For determining the cost of the loans, we set $\lambda = 0.12$ and $\mu = 1.2$.

C.2 Shortest path

We consider the shortest path problem with uncertain arc weights as defined in Subramanyam et al. [2020]. Let $G = (V, A)$ be a directed graph with nodes $V = \{1, ..., N\}$, arcs $A \subseteq V \times V$, and arc weights $d_{ij}(\boldsymbol{z}) = (1 + z_{ij}/2)d_{ij}^0$, $(i,j) \in A$. Where $d_{ij}^0 \in \mathbb{R}_+$ represents the nominal weight of the arc $(i,j) \in A$ and z_{ij} denotes the uncertain deviation from the nominal weight. The uncertainty set is defined as

$$\mathcal{Z} = \Big\{ \boldsymbol{z} \in [0,1]^{|A|} : \sum_{(i,j)\in A} z_{ij} \leq \Gamma \Big\}.$$

This uncertainty set imposes that at most Γ arc weights may maximally deviate from their nominal values. We need to find the shortest path from the source node s, to the sink node t before observing the realized arc weights. This shortest problem can be formulated as an instance of the K-adaptability problem

$$\begin{aligned}
\min_{\boldsymbol{y}\in\mathcal{Y}^K} \max_{\boldsymbol{z}\in\mathcal{Z}} \min_{k\in\mathcal{K}} \quad & \theta \\
\text{s.t.} \quad & \boldsymbol{d}(\boldsymbol{z})^\intercal \boldsymbol{y}_k \leq \theta, \\
& \sum_{(j,l)\in A} y_{jl}^k - \sum_{(i,j)\in A} y_{ij}^k \geq \mathbf{1}_{\{j=s\}} - \mathbf{1}_{\{j=t\}}, \qquad \forall j \in V, \\
& \mathcal{Y} \subseteq \{0,1\}^{|A|}.
\end{aligned}$$

Note that this problem contains only binary second-stage decisions and uncertainty in the objective function. The K-B&B algorithm, will find K shortest paths from s to t. After $\boldsymbol{z}$ is observed, the path $\boldsymbol{y}_k$ will be chosen if $\boldsymbol{z} \in \mathcal{Z}_k$.

Normal test case. The coordinates in $\mathbb{R}^2$ for each vertex $i \in V$ are uniformly chosen at random from the region $[0,10]^2$. The nominal weight of the arc $(i,j) \in A$ is the Euclidean distance between node i and j. The source node s and the sink node t are defined to be the nodes with the maximum nominal distance between them. The $\lfloor 0.9(N^2 - N) \rfloor$ arcs with the highest nominal weight will be deleted to define the arc set A. The budget of the uncertainty set Γ is set to seven.

Sphere test case. The instances of this type have nodes that are spread over a sphere. This is done as follows. First, each node in the three-dimensional graph is sampled from the standard normal distribution and then normalized. The distance between node i and j is then derived by its spherical distance. To obtain this, first the Euclidean distance d_{ij} between node i and j is computed. Then, the arc sine of $d_{ij}/2$ is computed to get the spherical distance. The $\lfloor 0.7(N^2 - N) \rfloor$ arcs with the highest nominal weight will be deleted to define the arc set A. The budget of the uncertainty Γ is set to seven.

C.3 Knapsack

We consider the two-stage version of the knapsack problem where the profit per item is uncertain. This formulation is based on that of Buchheim and Kurtz [2017]. Let N be the number of items, $p_i(\boldsymbol{z}) = (1 - z_i/2)p_i^0$ the profit for item $i \in \{1,\dots,N\}$, where $p_i^0 \in \mathbb{R}$ is the nominal profit value, z_i is the deviation, $\boldsymbol{w} \in \mathbb{R}^N$ is the weight vector, and $C = c\sum_{i=1}^N w_i$ is the total capacity of the knapsack with $c \in (0,1)$. The uncertainty set is defined as

$$\mathcal{Z} = \Big\{ \boldsymbol{z} \in [0,1]^N : \sum_{i=1}^N z_i \leq \Gamma \Big\},$$

where $\Gamma = \gamma N$ and $\gamma \in (0,1)$. This problem can be formulated as an instance of the K-adaptability problem

$$\begin{aligned}
\max_{\boldsymbol{y}\in\mathcal{Y}^K} \min_{\boldsymbol{z}\in\mathcal{Z}} \max_{k\in\mathcal{K}} \quad & \theta \\
\text{s.t.} \quad & \boldsymbol{p}(\boldsymbol{z})^\intercal \boldsymbol{y}_k \geq \theta, \\
& \boldsymbol{w}^\intercal \boldsymbol{y}_k \leq C, \\
& \mathcal{Y} \subseteq \{0,1\}^N,
\end{aligned}$$

where y_i is the decision of putting item i in the knapsack.

Test case. The weight w_i of each item $i \in \{1, \ldots, N\}$ is uniformly chosen at random from $[1, 15]$ and the cost c_i from $[100, 150]$. The values of c and γ are selected in the experiments section of the paper.

D Parameter tuning

For both training the ML model (a random forest) and applying it to a problem, we have defined multiple parameters in STRATEGYMODEL and K-B&B-NODESELECTION. The tuning of these parameters is explained in this section.

D.1 Capital budgeting with loans

For each $K \in \{2, \ldots, 6\}$ we have trained five random forests: each for $\epsilon \in \{0.05, 0.1, 0.2, 0.3, 0.4\}$. We have first decided on the values of ι for generating training data. Problems become more complex when K grows, which results in more time needed per dive. For tuning these parameters, we have generated training data by running the algorithm for two hours. Thus, we have fixed the parameter T to two. In Table 9, for each combination of K and $\iota \in \{2, 5, 10, 15\}$ the following information is shown: number of data points, number of searched instances I, and the average of L^{train} reached per instance.

Table 9: Generated training data info for combinations of K and ι. (num. data points, I, average L^{train}).

K	ι (in minutes)			
	2	5	10	15
2	(17284, 60, 9)	(7192, 24, 10)	-	-
3	(37227, 60, 7)	(36510, 24, 8)	(34536, 12, 9)	(30510, 8, 9)
4	(21572, 60, 5)	(21860, 24, 6)	(24312, 12, 7)	(17008, 8, 7)
5	(22425, 60, 5)	(23830, 24, 5)	(22510, 12, 6)	(20040, 8, 6)
6	(17820, 60, 4)	(17544, 24, 5)	(18834, 12, 5)	(17382, 8, 5)

We have then selected per K a value of ι that has high values of the number of data points, I and L^{train}. Then, for each ϵ and K, the dataset related to this value of ι has been trained. The accuracy of these models are given in Table 10.

Table 10: Test accuracy of random forest for combinations of K (with best ι) and the threshold ϵ.

K (ι)	ϵ				
	0.05	0.1	0.2	0.3	0.4
2 (2)	0.971	0.988	0.971	0.988	0.983
3 (5)	0.929	0.959	0.959	0.967	0.981
4 (5)	0.922	0.950	0.977	0.982	0.995
5 (5)	0.958	0.937	0.967	0.975	0.992
6 (10)	0.937	0.952	0.968	0.974	0.984

The table above shows that ϵ does not influence the accuracy of the model. However, Figure 9 shows that the algorithm performs better when ϵ is very small. If we look at the density of success probabilities p in Figure 23, we notice that the vast majority of data points have $p_n \approx 0$. These two observations indicate that any value of p_n slightly higher than zero is special, and the corresponding node is considered as a good node to visit.

In the experiments section of the paper, we noticed that high values of L^{test} outperformed lower ones in the K-B&B-NODESELECTION algorithm. In Figure 24, we show the results for higher values of L^{train} than the ones shown in Figure 9 in the paper for a fixed $\epsilon = 0.05$.

D.2 Shortest path on a sphere

For the shortest path problem we also want to decide on the parameter ι per $K \in \{2, 3, 4, 5, 6\}$. We noticed that the total number of scenarios needed until a robust solution is found, is larger for shortest path than for capital budgeting. Therefore, the duration per training instance should increase. The range of the number of minutes is $\iota \in \{5, 10, 15, 20\}$. In Table 11, for each combination of K and ι the number of data points, instances, and training level L^{train} is given. For now, $T = 2$ is fixed.

We have then selected per K a value of ι that has high values of the number of data points, I and L^{train}. Then, for each ϵ and K, the dataset related to this value of ι has been trained. See Table 12 for the accuracy of these models.

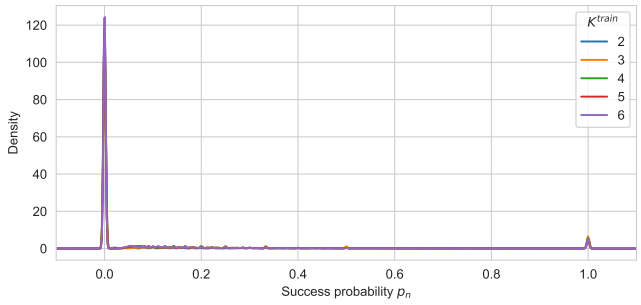


Figure 23: Density of the success probability p_n for each value of K^{train}.

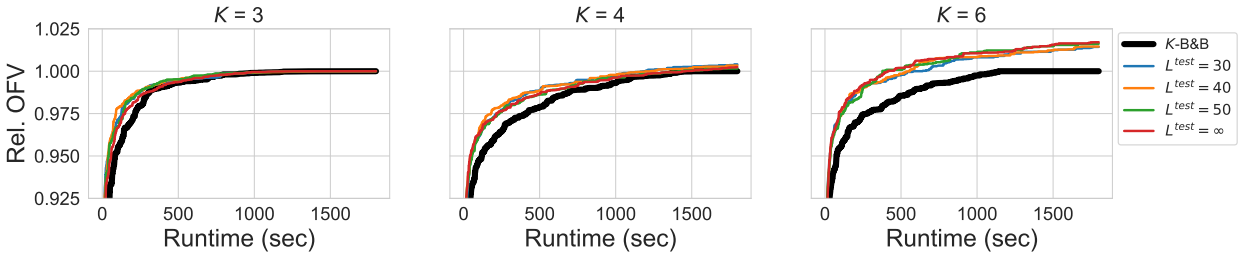


Figure 24: Results of K-B&B with random dives and K-B&B-NODESELECTION with combinations of K and bigger values of L^{test}.

Table 11: Generated training data info for combinations of K and ι. (num. data points, I, average L^{train}).

K	ι (in minutes) 5	10	15	20
2	(10802, 24, 8)	(8290, 12, 9)	(9160, 8, 10)	(8494, 6, 10)
3	(8370, 24, 6)	(6726, 12, 6)	(9858, 8, 6)	(7506, 6, 7)
4	(9884, 24, 5)	(8496, 12, 6)	(7916, 8, 6)	(15872, 6, 6)
5	(13795, 24, 4)	(7415, 12, 5)	(12490, 8, 5)	(21110, 6, 6)
6	(26346, 24, 5)	(10794, 12, 5)	(18948, 8, 5)	(25788, 6, 5)

Table 12: Number of data points used for training and test accuracy of random forest for combinations of K (with best ι) and hours spent for generating training data T, given the threshold $\epsilon = 0.05$. (num. data points, test accuracy).

$K(\iota)$	T 1	2	5	10
2 (15)	(6586, 0.955)	(9160, 0.946)	(16852, 0.923)	(35674, 0.947)
3 (15)	(4170, 0.952)	(9858, 0.939)	(28347, 0.919)	(47346, 0.937)
4 (20)	(2828, 0.931)	(15872, 0.969)	(37652, 0.966)	(69244, 0.932)
5 (20)	(4790, 0.875)	(21110, 0.972)	(47000, 0.93)	(77735, 0.91)
6 (15)	(13500, 0.948)	(18948, 0.916)	(54564, 0.945)	(91254, 0.955)

We noticed for the shortest path problem that more data points significantly increased the performance of the ML model on the algorithm. An overview of the chosen values of ι and T (hours spent for getting training data) per K are given in Table 13.

Table 13: Chosen parameter combination for each K. The values of ϵ and L^{test} are fixed to 0.05 and ∞, respectively.

K	ι	T
2	15	10
3	15	5
4	20	5
5	20	10
6	15	10

The density of the success probabilities for the data points of the shortest path problem is given in Figure 25. This is very similar to the density of capital budgeting (see Figure 23).

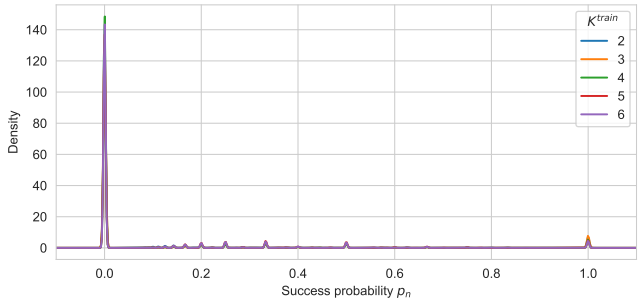


Figure 25: Density of the success probability p_n for each value of K^{train}.

E Full Results

We have applied K-B&B-NODESELECTION to multiple problems, where the training and testing instance specifications also varied. In the main body, only a subset of the experiments are shown. In this section, all of them are given.

E.1 Capital budgeting with loans

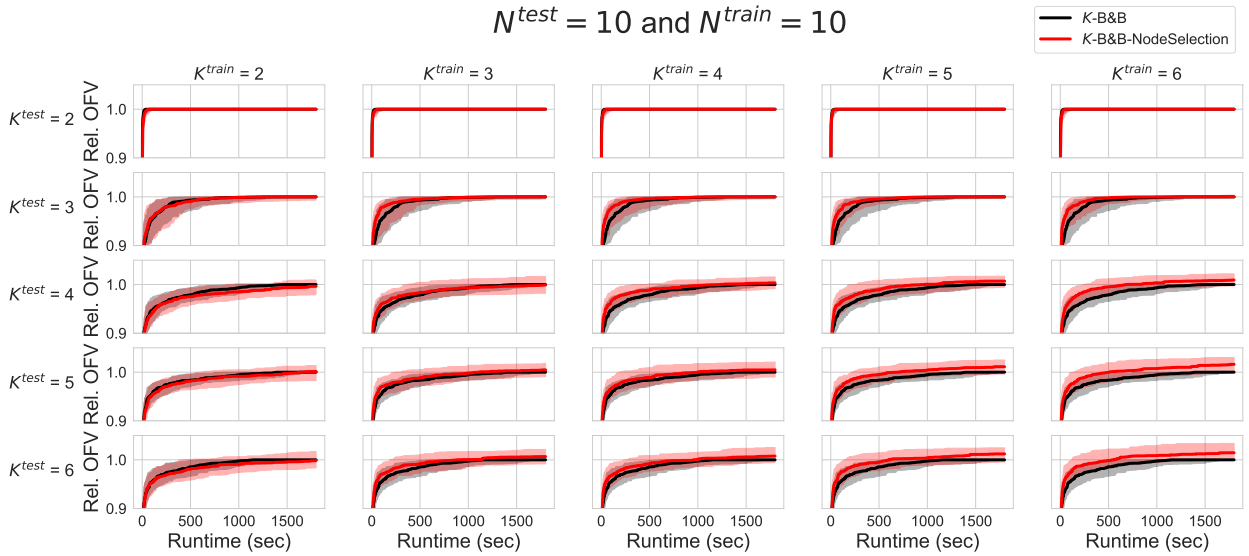


Figure 26: Comparison of results between K-B&B and K-B&B-NODESELECTION for 100 instances of the capital budgeting problem. The results of EXP1 and EXP2 are shown, where $N^{test} = N^{train} = 10$. The regions with shaded color around the curves denote its 75% CI.

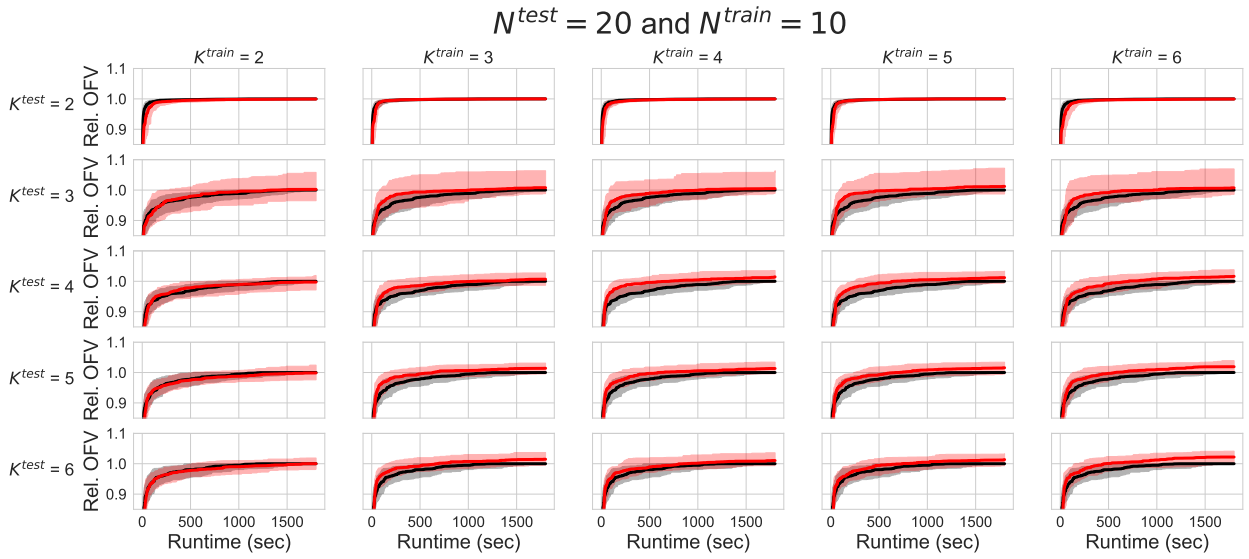


Figure 27: Comparison of results between K-B&B and K-B&B-NODESELECTION for 100 instances of the capital budgeting problem. The results of EXP3 and EXP4 are shown, where $N^{train} = 10$, $N^{test} = 20$. The regions with shaded color around the curves denote its 75% CI.

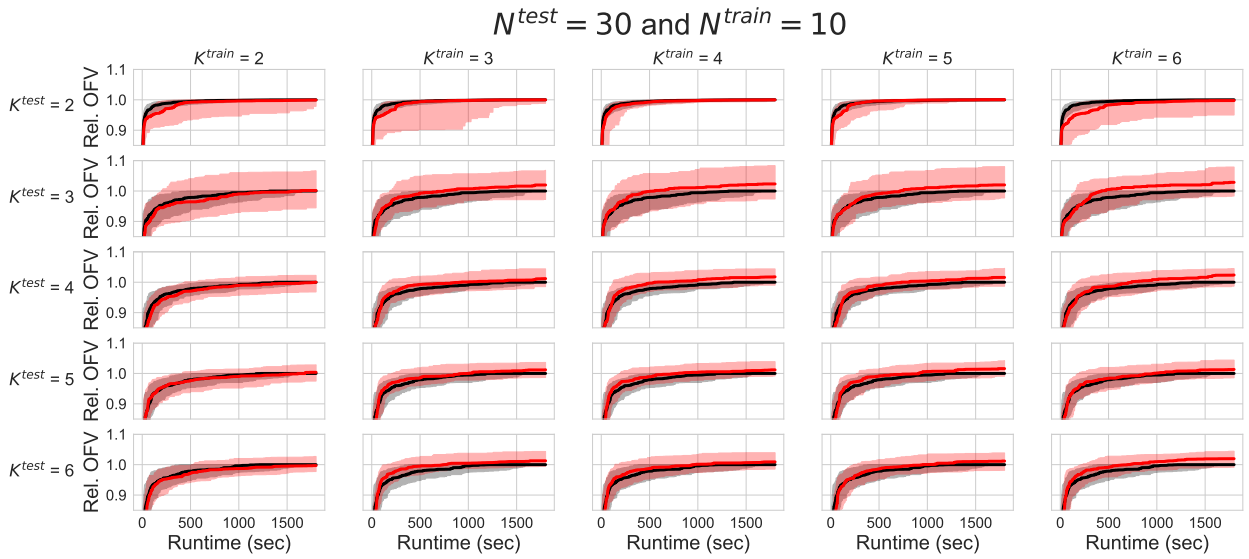


Figure 28: Comparison of results between K-B&B and K-B&B-NODESELECTION for 100 instances of the capital budgeting problem. The results of EXP3 and EXP4 are shown, where $N^{train} = 10$, $N^{test} = 30$. The regions with shaded color around the curves denote its 75% CI.

E.2 Shortest path

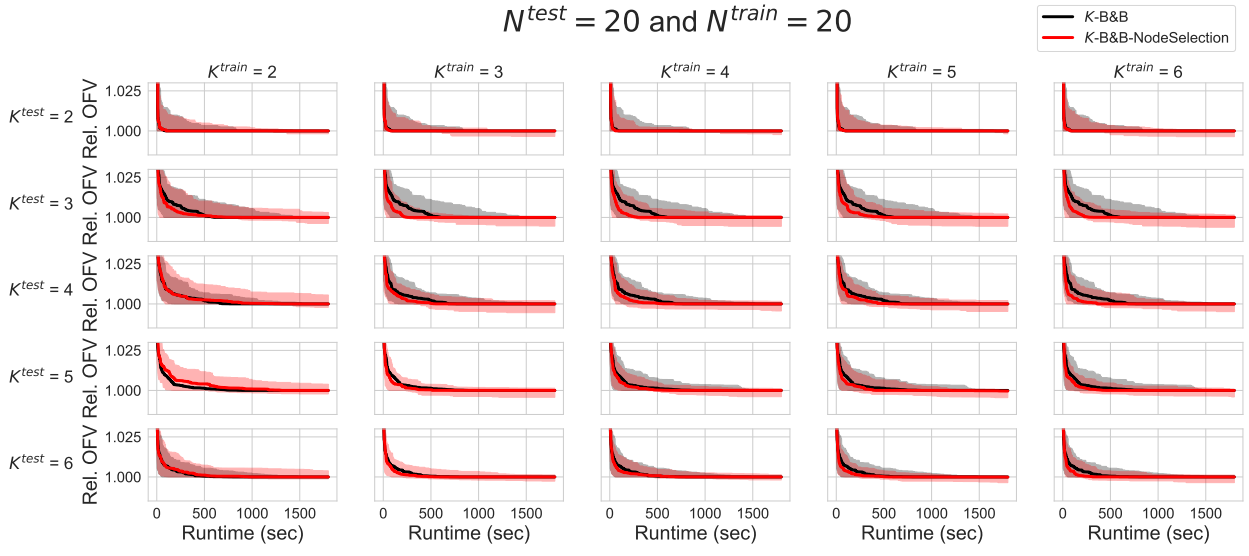


Figure 29: Comparison of results between K-B&B and K-B&B-NODESELECTION for 100 instances of the shortest path problem. The results of EXP1 and EXP2 are shown, where $N^{test} = N^{train} = 20$. The regions with shaded color around the curves denote its 75% CI.

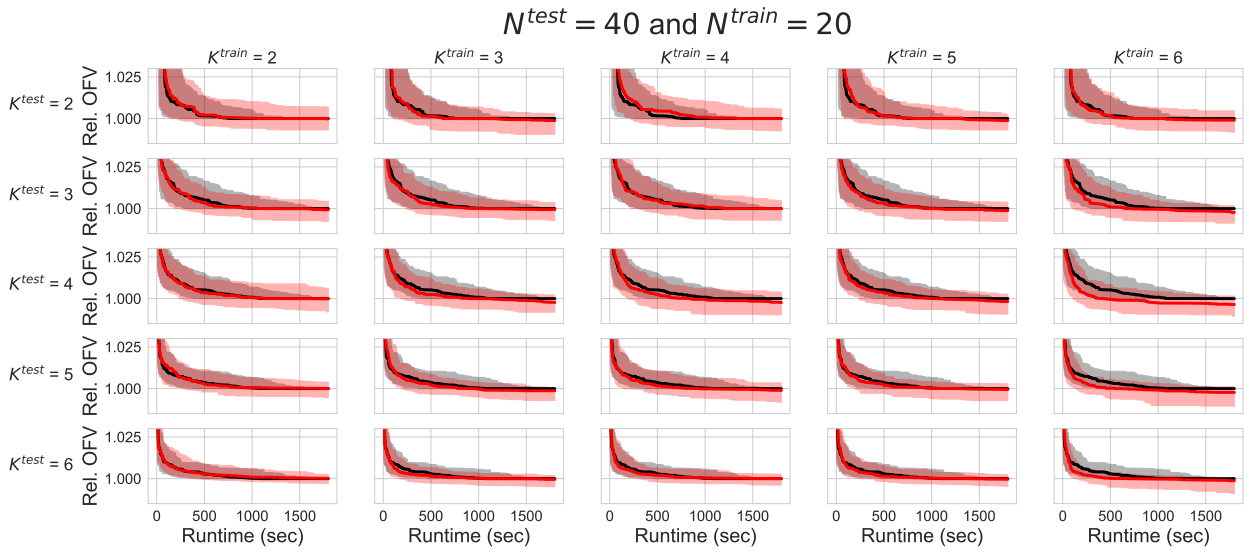


Figure 30: Comparison of results between K-B&B and K-B&B-NODESELECTION for 100 instances of the shortest path problem. The results of EXP3 and EXP4 are shown, where $N^{train} = 20, N^{test} = 40$. The regions with shaded color around the curves denote its 75% CI.

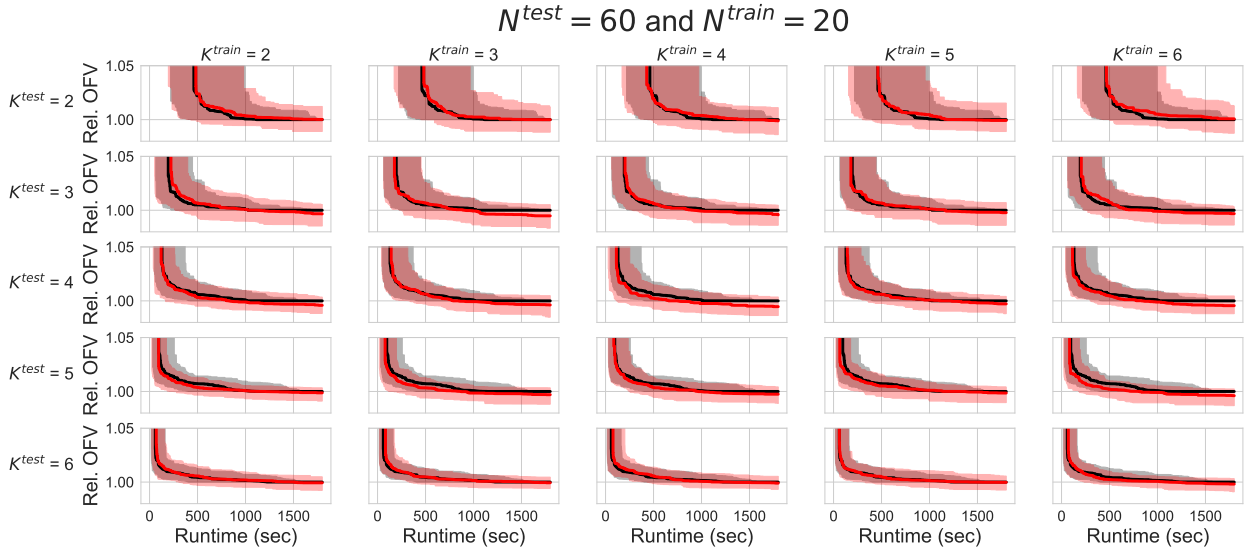


Figure 31: Comparison of results between K-B&B and K-B&B-NODESELECTION for 100 instances of the shortest path problem. The results of EXP3 and EXP4 are shown, where $N^{train} = 20, N^{test} = 60$. The regions with shaded color around the curves denote its 75% CI.

E.3 Mixed problems

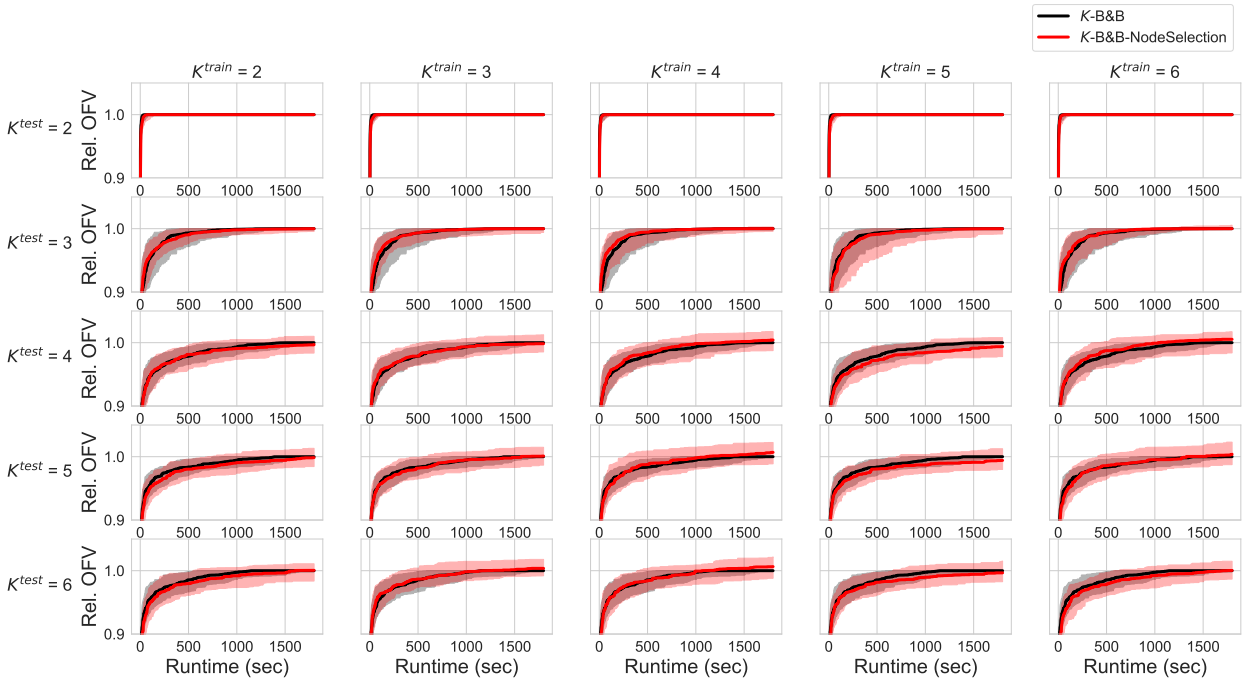


Figure 32: Results of the capital budgeting problem. The ML model that is used is trained on shortest path data.

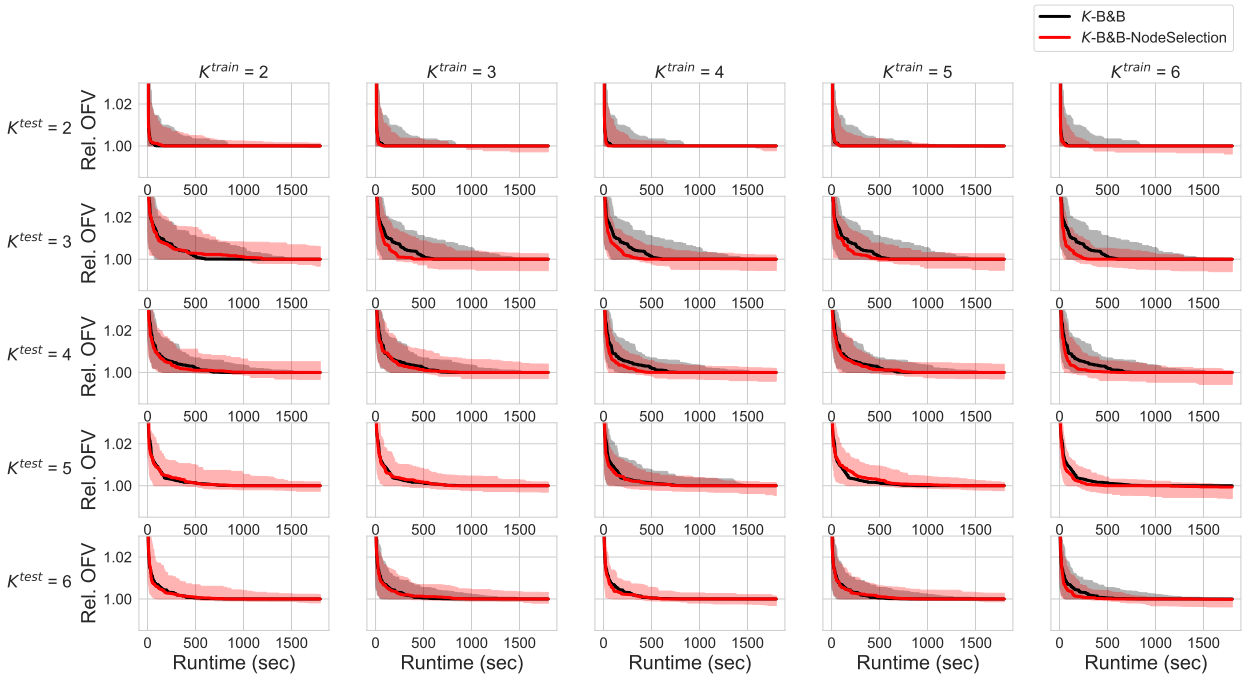


Figure 33: Results of the shortest path problem. The ML model that is used is trained on capital budgeting data.