# Finding Optimal Diverse Feature Sets with Alternative Feature Selection

Jakob Bach
Independent researcher*
jakob.bach.ka@gmail.com

**Abstract**

Feature selection is popular for obtaining small, interpretable, yet highly accurate prediction models. Conventional feature-selection methods typically yield one feature set only, which might not suffice in some scenarios. For example, users might be interested in finding alternative feature sets with similar prediction quality, offering different explanations of the data. In this article, we introduce alternative feature selection and formalize it as an optimization problem. In particular, we define alternatives via constraints and enable users to control the number and dissimilarity of alternatives. We consider sequential as well as simultaneous search for alternatives. Next, we discuss how to integrate conventional feature-selection methods as objectives. In particular, we describe solver-based search methods to tackle the optimization problem. Further, we analyze the complexity of this optimization problem and prove $\mathcal{NP}$-hardness. Additionally, we show that a constant-factor approximation exists under certain conditions and propose corresponding heuristic search methods. Finally, we evaluate alternative feature selection in comprehensive experiments with 30 binary-classification datasets. We observe that alternative feature sets may indeed have high prediction quality, and we analyze factors influencing this outcome.

**Keywords:** feature selection, alternatives, constraints, mixed-integer programming, explainability, interpretability, XAI

## 1 Introduction

**Motivation** Feature-selection methods are ubiquitous for a variety of reasons. By reducing dataset dimensionality, they lower the computational cost and memory requirements of prediction models. Next, prediction models may generalize better without irrelevant and spurious predictors. While some model

*Most of the research for this article was carried out while the author was affiliated with the Karlsruhe Institute of Technology (KIT), Karlsruhe, Germany.

types can implicitly select relevant features, others cannot. Finally, prediction models may become simpler [68], improving interpretability.

Most conventional feature-selection methods only return one feature set [14]. These methods optimize a criterion of feature-set quality, e.g., prediction performance. However, besides the optimal feature set, there might be other, differently composed feature sets with similar quality. Such alternative feature sets are interesting for users, e.g., to obtain several diverse explanations. Alternative explanations can provide additional insights into predictions, enable users to develop and test different hypotheses, appeal to different kinds of users, and foster trust in the predictions [56, 118].

For example, in a dataset describing physical experiments, feature selection may help to discover relationships between physical quantities. In particular, highly predictive feature sets indicate which input quantities are strongly related to the output quantity. Domain experts may use these feature sets to formulate hypotheses on physical laws. However, if multiple alternative sets of similar quality exist, further analyses and experiments may be necessary to reveal the true underlying physical mechanism. Only knowing one predictive feature set and using it as the only explanation is misleading in such a situation.

**Problem statement** This article addresses the problem of alternative feature selection, which we informally define as follows: Find multiple, sufficiently different feature sets that optimize feature-set quality. We provide formal definitions in Section 3.2. This problem entails an interesting trade-off: Depending on how many alternatives are desired and how different the alternatives should be, one may have to compromise on quality. In particular, a higher number of alternatives or a stronger dissimilarity requirement may necessitate selecting more low-quality features in the alternatives.

Two points are essential for alternative feature selection, which we both address in this article. First, one needs to formalize and quantify what an alternative feature set is. In particular, users should be able to control the number and dissimilarity of alternatives and hence the quality trade-off. Second, one needs an approach to find alternative feature sets efficiently. Ideally, the approach should be general, i.e., cover a broad range of conventional feature-selection methods, given the variety of the latter [18, 68].

**Related work** While finding alternative solutions has already been addressed extensively in the field of clustering [11], there is a lack of such approaches for feature selection. Only a few feature-selection methods target at obtaining multiple, diverse feature sets [14]. In particular, techniques for ensemble feature selection [100, 104] and statistically equivalent feature subsets [63] produce multiple feature sets but not optimal alternatives. These approaches do not guarantee the diversity of the feature sets, nor do they let users control diversity. In fields related to feature selection, the goal of obtaining multiple, diverse solutions has been studied as well, e.g., for subspace clustering [48, 79], subgroup discovery [66], subspace search [112], or explainable-AI techniques [2,

55, 78, 99] like counterfactuals. These approaches are not directly applicable or easily adaptable to feature selection, and most of them provide limited or no user control over alternatives, as we will elaborate in Section 4.

**Contributions** Our contribution is five-fold.

First, we formalize alternative feature selection as an optimization problem. In particular, we define alternatives via constraints on feature sets. This approach is orthogonal to the feature-selection method so that users can choose the latter according to their needs. This approach also allows integrating other constraints on feature sets, e.g., to capture domain knowledge [8, 37]. Finally, this approach lets users control the search for alternatives with two parameters, i.e., the number of alternatives and a dissimilarity threshold. For multiple alternatives, we consider sequential as well as simultaneous search.

Second, we discuss how to solve this optimization problem. To that end, we describe how to integrate different categories of conventional feature-selection methods in the objective function of the optimization problem. In particular, we outline solver-based search methods for white-box and black-box optimization.

Third, we analyze the time complexity of the optimization problem. We show $\mathcal{NP}$-hardness, even for a simple notion of feature-set quality, i.e., univariate feature qualities, as used in filter feature selection.

Fourth, we propose heuristic search methods for univariate feature qualities. We show that, under certain conditions, the optimization problem resides in the complexity class $\mathcal{APX}$, i.e., a constant-factor approximation exists.

Fifth, we evaluate alternative feature selection with comprehensive experiments. In particular, we use 30 binary-classification datasets from the Penn Machine Learning Benchmarks (PMLB) [90, 98] and five feature-selection methods. We focus our evaluation on the feature-set quality of the alternatives relative to our search methods for alternatives and user parameters. Additionally, we evaluate runtime. We publish all code and data online (cf. Section 5.5).

**Experimental results** We observe that several factors influence the quality of alternatives, i.e., the dataset, feature-selection method, metric for feature-set quality, search method, and user parameters for searching alternatives. As expected, feature-set quality tends to decrease with an increasing number of alternatives and an increasing dissimilarity threshold for alternatives. Thus, these parameters allow users to control the trade-off between dissimilarity and quality of alternatives. Also, no valid alternative may exist if the parameter values are too strict. Runtime-wise, a solver-based sequential search for multiple alternatives was significantly faster than a simultaneous one while yielding a similar quality. Additionally, our heuristic search methods for univariate feature qualities achieved a high quality within negligible runtime. Finally, we observe that the prediction performance of feature sets may only weakly correlate with the quality assigned by feature-selection methods. In particular, seemingly bad alternatives regarding the latter might still be good regarding the former.

**Outline** Section 2 introduces notation and fundamentals. Section 3 describes and analyzes alternative feature selection. Section 4 reviews related work. Section 5 outlines our experimental design, while Section 6 presents the experimental results. Section 7 concludes. Appendix A contains supplementary materials.

**Related versions** The dissertation [6] contains a shortened version of this article, which retains all central experimental and theoretical results. In particular, the dissertation evaluates the same run of the experimental pipeline. The search method *Greedy Depth Search* (cf. Appendix A.6) appears exclusively in this arXiv version but is not evaluated anyway.

There is also a journal version of this article [7]. However, it is outdated in some places since we derived it from the first arXiv version. For example, the journal version lacks the heuristic search methods for alternatives (cf. Section 3.5), uses an older version of the *Greedy Wrapper* approach (cf. Algorithm 1), and partly differs in notation and formalization, e.g., definitions.

# 2 Fundamentals

In this section, we introduce basic notation (cf. Section 2.1) and review different methods to measure the quality of feature sets (cf. Section 2.2).

## 2.1 Notation

$X \in \mathbb{R}^{m \times n}$ stands for a dataset in the form of a matrix. Each row is a data object, and each column is a feature. $\tilde{F} = \{f_1, \ldots, f_n\}$ is the corresponding set of feature names. We assume that categorical features have already been made numeric, e.g., via one-hot encoding. $X_{\cdot j} \in \mathbb{R}^m$ denotes the vector representation of the $j$-th feature. $y \in Y^m$ represents the prediction target with domain $Y$, e.g., $Y = \{0, 1\}$ for binary classification or $Y = \mathbb{R}$ for regression.

In feature selection, one makes a binary decision $s_j \in \{0, 1\}$ for each feature, i.e., either selects it or not. The vector $s \in \{0, 1\}^n$ combines all these selection decisions and yields the selected feature set $F_s = \{f_j \mid s_j = 1\} \subseteq \tilde{F}$. To simplify notation, we drop the subscript $s$ in definitions where we do not explicitly refer to the value of $s$ but only the set $F$. The function $Q(s, X, y)$ returns the quality of such a feature set. Without loss of generality, we assume that this function should be maximized.

## 2.2 Measuring Feature (Set) Quality

There are different ways to evaluate feature-set quality $Q(s, X, y)$. We only give a short overview here; see [18, 68, 89] for comprehensive studies and surveys of feature selection. Also, note that we assume a supervised feature-selection scenario, i.e., feature-set quality depending on a prediction target $y$. In principle, our definitions of alternatives also apply to an unsupervised scenario.

Since the prediction target only appears in the function $Q(s, X, y)$, one could replace $Q(s, X, y)$ with $Q(s, X)$, i.e., an unsupervised notion of quality.

A conventional categorization of feature-selection methods distinguishes between filter, wrapper, and embedded methods [42].

**Filter methods** Filter methods evaluate feature sets without training a prediction model. Univariate filters assess each feature independently, e.g., using the absolute Pearson correlation or the mutual information between a feature and the prediction target. Multivariate filters additionally consider interactions between features. Such methods often combine a measure of feature relevance with a measure of feature redundancy. Examples include CFS [43, 44], FCBF [125], and mRMR [93]. Some filter methods also consider feature intercooperation, i.e., the joint relevance of two or more features [109].

**Wrapper methods** Wrapper methods [58] evaluate feature sets by training prediction models with them and measuring prediction quality. They employ a generic search strategy to iterate over candidate feature sets, e.g., genetic algorithms. Feature-set quality is a black-box function in this search.

**Embedded methods** Embedded methods train prediction models with built-in feature selection, e.g., decision trees [16] or random forests [15]. Thus, the criterion for feature-set quality is model-specific. For example, tree-based models often use information gain or the Gini index to select features during training.

**Post-hoc feature-importance methods** Apart from conventional feature selection, there are various methods that assess feature importance after training a model. These methods range from local explanation methods like LIME [95] or SHAP [70] to global importance methods like permutation importance [15] or SAGE [23]. In particular, assessing feature importance plays a crucial role in the field of machine-learning interpretability [17, 75].

# 3 Alternative Feature Selection

In this section, we present the problem of and approaches for alternative feature selection. First, we define the overall structure of the optimization problem, i.e., objective and constraints (cf. Section 3.1). Second, we formalize the notion of alternatives via constraints (cf. Section 3.2). Third, we discuss objective functions corresponding to different feature-set quality measures from Section 2.2 and describe how to solve the resulting optimization problem (cf. Section 3.3). Fourth, we analyze the time complexity of the optimization problem (cf. Section 3.4). Fifth, we propose and analyze heuristic search methods for the optimization problem with univariate feature qualities (cf. Section 3.5).

## 3.1 Optimization Problem

Alternative feature selection has two goals. First, the quality of an alternative feature set should be high. Second, an alternative feature set should differ from one or more other feature set(s). There are several ways to combine these two goals in an optimization problem:

First, one can consider both goals as objectives, obtaining an unconstrained multi-objective problem. Second, one can treat feature-set quality as objective and enforce alternatives with constraints. Third, one can consider being alternative as objective and constrain feature-set quality, e.g., with a lower bound. Fourth, one can define constraints for both, feature-set quality and being alternative, searching for feasible solutions instead of optimizing.

We stick to the second formulation, i.e., optimizing feature-set quality subject to being alternative. This formulation has the advantage of keeping the original objective function of feature selection. Thus, users do not need to specify a range or a threshold on feature-set quality but can control how alternative the feature sets must be instead. We obtain the following optimization problem for a single alternative feature set $F_s$:

$$\begin{aligned} \max_{s} \quad & Q(s, X, y) \\ \text{subject to:} \quad & F_s \text{ being alternative} \end{aligned} \tag{1}$$

In the following, we discuss different objective functions $Q(s, X, y)$ and suitable constraints for *being alternative*. Additionally, many feature-selection methods also limit the feature-set size $|F_s|$ to a user-defined value $k \in \mathbb{N}$, which adds a further, simple constraint to the optimization problem.

## 3.2 Constraints – Defining Alternatives

In this section, we formalize alternative feature sets. First, we discuss the base case where an individual feature set is an alternative to another one (cf. Section 3.2.1). Second, we extend this notion to multiple alternatives, considering sequential and simultaneous search as two different search problems (cf. Section 3.2.2).

Our notion of alternatives is independent of the feature-selection method. We provide two parameters, i.e., a dissimilarity threshold $\tau$ and the number of alternatives $a$, allowing users to control the search for alternatives.

### 3.2.1 Single Alternative

We consider a feature set an alternative to another feature set if it differs sufficiently. Mathematically, we express this notion with a set-dissimilarity measure [22, 29]. These measures typically assess how strongly two sets overlap and relate this to their sizes. E.g., a well-known set-dissimilarity measure is the Jaccard distance, which is defined as follows for the feature sets $F'$ and $F''$:

$$d_{\text{Jacc}}(F', F'') = 1 - \frac{|F' \cap F''|}{|F' \cup F''|} = 1 - \frac{|F' \cap F''|}{|F'| + |F''| - |F' \cap F''|} \tag{2}$$

In this article, we use a dissimilarity measure based on the Dice coefficient:

$$d_{\text{Dice}}(F', F'') = 1 - \frac{2 \cdot |F' \cap F''|}{|F'| + |F''|} \tag{3}$$

Generally, we only have mild assumptions on the set-dissimilarity measure $d(\cdot)$. Our subsequent definitions, examples, and propositions assume symmetry, i.e., $d(F', F'') = d(F'', F')$, normalization $d(\cdot) \in [0, 1]$, and that $d(\cdot) = 1$ implies an empty intersection of the two sets. In particular, $d(\cdot)$ does not need to be a metric but can also be a semi-metric [120] like $d_{\text{Dice}}(\cdot)$. In contrast to metrics, semi-metrics may violate the triangle inequality.

We leverage the set-dissimilarity measure for the following definition:

**Definition 1** (Single alternative)**.** Given a symmetric set-dissimilarity measure $d(\cdot) \in [0, 1]$ with $d(\cdot) = 1$ implying no set overlap, and a dissimilarity threshold $\tau \in [0, 1]$, a feature set $F'$ is an alternative to a feature set $F''$ (and vice versa) if $d(F', F'') \geq \tau$.

The threshold $\tau$ controls how alternative the feature sets must be and depends on the dataset as well as user preferences. In particular, requiring strong dissimilarity may cause a significant drop in feature-set quality. Some datasets may contain many features of similar utility, thereby enabling many alternatives of similar quality, while predictions on other datasets may depend on a few key features. Only users can decide which drop in feature-set quality is acceptable as a trade-off for obtaining alternatives. Thus, we leave $\tau$ as a user parameter. In case the set-dissimilarity measure $d(\cdot)$ is normalized to $[0, 1]$, like the Dice dissimilarity (cf. Equation 3) or Jaccard distance (cf. Equation 2), the interpretation of $\tau$ is user-friendly: Setting $\tau = 0$ allows identical alternatives, while $\tau = 1$ implies zero overlap.

If the choice of $\tau$ is unclear a priori, users can try out different values and compare the resulting feature-set quality. One systematic approach is a binary search: Start with the mid-range value of $\tau = 0$, i.e., 0.5 for $\tau \in [0, 1]$. If the quality of the resulting alternative is too low, decrease $\tau$ to 0.25, i.e., allow more similarity. If the quality of the resulting alternative is acceptably high, increase $\tau$ to 0.75, i.e., check a more dissimilar feature set. Continue this procedure till an alternative with an acceptable quality-dissimilarity trade-off is found.

When implementing Definition 1, the following proposition gives way to using a broad range of solvers to tackle the related optimization problem:

**Proposition 1** (Linearity of constraints for alternatives)**.** *Using the Dice dissimilarity (cf. Equation 3), alternative feature sets (cf. Definition 1) can be expressed with 0-1 integer linear constraints.*

*Proof.* We re-arrange terms in the Dice dissimilarity (cf. Equation 3) to eliminate the quotient of feature-set sizes:

$$\begin{aligned} d_{\text{Dice}}(F', F'') = \quad & 1 - \frac{2 \cdot |F' \cap F''|}{|F'| + |F''|} \geq \tau \\ \Leftrightarrow \quad & |F' \cap F''| \leq \frac{1 - \tau}{2} \cdot (|F'| + |F''|) \end{aligned} \tag{4}$$

Next, we express the set sizes in terms of the feature-selection vector $s$:

$$\begin{aligned} |F_s| &= \sum_{j=1}^{n} s_j \\ |F_{s'} \cap F_{s''}| &= \sum_{j=1}^{n} s'_j \cdot s''_j \end{aligned} \tag{5}$$

Finally, we replace each product $s'_j \cdot s''_j$ with an auxiliary variable $t_j$, bound by additional constraints, to linearize it [76]:

$$\begin{aligned} t_j &\leq s'_j \\ t_j &\leq s''_j \\ 1 + t_j &\geq s'_j + s''_j \\ t_j &\in \{0, 1\} \end{aligned} \tag{6}$$

Combining Equations 4, 5, and 6, we obtain a set of constraints that only involve linear expressions of binary decision variables. In particular, there are only sum expressions and multiplications with constants but no products between variables. If one feature set is known, i.e., either $s'$ or $s''$ is fixed, Equation 5 only multiplies variables with constants and is already linear without Equation 6. □

Given a suitable objective function, which we discuss later, linear constraints allow using a broad range of solvers. As an alternative formulation, one could also encode such constraints into propositional logic (SAT) [113].

If the set sizes $|F'|$ and $|F''|$ are constant, e.g., user-defined, Equation 4 implies that the threshold $\tau$ has a linear relationship to the maximum number of overlapping features $|F' \cap F''|$. This correspondence eases the interpretation of $\tau$ and makes us use the Dice dissimilarity in the following. In contrast, the Jaccard distance exhibits a non-linear relationship between $\tau$ and the overlap size, which follows from re-arranging Equation 2 in combination with Definition 1:

$$\begin{aligned} d_{\text{Jacc}}(F', F'') = \quad & 1 - \frac{|F' \cap F''|}{|F'| + |F''| - |F' \cap F''|} \geq \tau \\ \Leftrightarrow \quad & |F' \cap F''| \leq \frac{1 - \tau}{2 - \tau} \cdot (|F'| + |F''|) \end{aligned} \tag{7}$$

Further, if $|F'| = |F''|$, as in our experiments, the Dice dissimilarity (cf. Equation 4) becomes identical to several other set-dissimilarity measures [29]. The parameter $\tau$ then directly expresses which fraction of features in one set needs to differ from the other set and vice versa, which further eases interpretability:

$$d_{\text{Dice}}(F', F'') \geq \tau \Leftrightarrow |F' \cap F''| \leq (1 - \tau) \cdot |F'| = (1 - \tau) \cdot |F''| \tag{8}$$

Thus, if users are uncertain how to choose $\tau$ and $|F'|$ is reasonably small, they can try out all values of $\tau \in \{l/|F'|\}$ with $l \in \{1, \ldots, |F'|\}$. In particular, these $|F'|$ unique values of $\tau$ suffice to produce all distinct solutions that one could obtain with an arbitrary $\tau \in (0, 1]$.

Table 1: Size of the optimization problem, i.e., number of variables and constraints, for $a$ alternatives ($a+1$ feature sets overall) and $n$ features.

| | Sequential search | | Simult. search |
|---|---|---|---|
| | $l$-th Alternative | Summed | |
| Decision variables $s$ | $n$ | $(a+1) \cdot n$ | $(a+1) \cdot n$ |
| Linearization variables $t$ | 0 | 0 | $\frac{a \cdot (a+1) \cdot n}{2}$ |
| Alternative constraints | $l$ | $\frac{a \cdot (a+1)}{2}$ | $\frac{a \cdot (a+1)}{2}$ |
| Linearization constraints | 0 | 0 | $\frac{3 \cdot a \cdot (a+1) \cdot n}{2}$ |

#### 3.2.2 Multiple Alternatives

If users desire multiple alternative feature sets rather than only one, we can determine these alternatives sequentially or simultaneously. The number of alternatives $a \in \mathbb{N}_0$ is a parameter to be set by the user. The overall number of feature sets is $a+1$ since we deem one feature set the 'original' one. Table 1 compares the sizes of the optimization problems for these two search problems.

**Sequential-search problem** In the sequential-search problem, users obtain several alternatives iteratively, with one feature set per iteration. We constrain this new set to be an alternative to all previously found ones, which are given in the set $\mathbb{F}$:

**Definition 2** (Sequential alternative)**.** A feature set $F''$ is an alternative to a set of feature sets $\mathbb{F}$ (and vice versa) if $F''$ is a single alternative (cf. Definition 1) to each $F' \in \mathbb{F}$.

One could also think of less strict constraints, e.g., requiring only the average dissimilarity to all previously found feature sets to pass a threshold $\tau$. However, definitions like the latter may allow some feature sets to overlap heavily or even be identical if other feature sets are very dissimilar. Thus, we require pairwise dissimilarity in Definition 2. Combining Equation 1 with Definition 2, we obtain the following optimization problem for each iteration of the search:

$$\begin{aligned} \max_{s} \quad & Q(s, X, y) \\ \text{subject to:} \quad & \forall F' \in \mathbb{F} : \; d(F_s, F') \geq \tau \end{aligned} \tag{9}$$

The full textual problem definition corresponding to Equation 9 is the following:

**Definition 3** (Sequential-search problem for one alternative feature set)**.** Given

- a dataset $X \in \mathbb{R}^{m \times n}$ with prediction target $y \in Y^m$,
- a set $\mathbb{F}$ of existing feature sets for $X$,

- a symmetric set-dissimilarity measure $d(\cdot) \in [0, 1]$ with $d(\cdot) = 1 \rightarrow$ no set overlap,
- and a dissimilarity threshold $\tau \in [0, 1]$,

sequential search for one alternative feature set is the problem of making feature-selection decisions $s \in \{0, 1\}^n$ that maximize a given notion of feature-set quality $Q(s, X, y)$ while making the corresponding feature set $F_s$ a sequential alternative to $\mathbb{F}$ (cf. Definition 2).

The objective function remains the same as for a single alternative ($|\mathbb{F}| = 1$), i.e., we only optimize the quality of one feature set at once. In particular, with $\mathbb{F} = \emptyset$ in the first iteration, we optimize for the 'original' feature set, which is the same as in conventional feature selection without constraints for alternatives. Thus, the number of variables in the optimization problem is independent of the number of alternatives $a$. Instead, we solve the optimization problem repeatedly; each alternative only adds one constraint to the problem. As we always compare only one variable feature set to existing, constant feature sets, we also do not need to introduce auxiliary variables as in Equation 6. Thus, we expect the runtime of exact, e.g., solver-based, sequential search to scale well with the number of alternatives. Further runtime gains may arise if the solver keeps a state between iterations and can warm-start.

However, as the search space becomes narrower over iterations, feature-set quality can deteriorate with each further alternative. In particular, multiple alternatives from the same sequential search might differ significantly in their quality. As a remedy, users can decide after each iteration if the feature-set quality is already unacceptably low or if another alternative should be found. In particular, users do not need to define the number of alternatives $a$ a priori.

**Simultaneous-search problem** In the simultaneous-search problem, users obtain multiple alternatives at once, so they need to decide on the number of alternatives $a$ beforehand. We use pairwise dissimilarity constraints for alternatives again:

**Definition 4** (Simultaneous alternatives)**.** A set of feature sets $\mathbb{F}$ contains simultaneous alternatives if each feature set $F' \in \mathbb{F}$ is a single alternative (cf. Definition 1) to each other feature set $F'' \in \mathbb{F}$ with $F' \neq F''$.

Combining Equation 1 with Definition 4, we obtain the following optimization problem for $a + 1$ feature sets:

$$\begin{aligned} \max_{s^{(0)},\ldots,s^{(a)}} \quad & \operatorname*{agg}_{l \in \{0,\ldots,a\}} Q(s^{(l)}, X, y) \\ \text{subject to:} \quad & \forall l_1, l_2 \in \{0, \ldots, a\},\ l_1 \neq l_2 :\ d(F_{s^{(l_1)}}, F_{s^{(l_2)}}) \geq \tau \end{aligned} \tag{10}$$

The full textual problem definition corresponding to Equation 10 is the following:

**Definition 5** (Simultaneous-search problem for alternative feature sets)**.** Given

- a dataset $X \in \mathbb{R}^{m \times n}$ with prediction target $y \in Y^m$,
- the number of alternatives $a \in \mathbb{N}_0$,
- an aggregation operator $\text{agg}(\cdot) : \mathbb{R}^{a+1} \to \mathbb{R}$ for feature-set qualities,
- a symmetric set-dissimilarity measure $d(\cdot) \in [0, 1]$ with $d(\cdot) = 1 \to$ no set overlap,
- and a dissimilarity threshold $\tau \in [0, 1]$,

simultaneous search for alternative feature sets is the problem of making feature-selection decisions $s^{(l)} \in \{0, 1\}^n$ for $l \in \{0, \ldots, a\}$ that maximize a given notion of feature-set quality $Q(s, X, y)$ aggregated over the alternatives with $\text{agg}_{l \in \{0,\ldots,a\}} Q(s^{(l)}, X, y)$ while making the corresponding feature sets $\mathbb{F} = \{F_{s^{(0)}}, \ldots, F_{s^{(a)}}\}$ simultaneous alternatives (cf. Definition 4).

In contrast to the sequential case (cf. Definition 3), the problem requires $a + 1$ instead one decision vector $s$, and a modified objective function. The operator $\text{agg}(\cdot)$ defines how to aggregate the feature-set qualities of the alternatives. In our experiments, we consider the sum as well as the minimum to instantiate $\text{agg}(\cdot)$, which we refer to as *sum-aggregation* and *min-aggregation*. The latter explicitly fosters balanced feature-set qualities. Appendix A.1 discusses these two aggregation operators and additional ideas for balancing qualities in detail.

Runtime-wise, we expect exact simultaneous search to scale worse with the number of alternatives than exact sequential search, as it tackles one large optimization problem instead of multiple smaller ones. In particular, the number of decision variables increases linearly with the number of alternatives $a$. Also, for each feature and each pair of alternatives, we need to introduce an auxiliary variable if we want to obtain linear constraints (cf. Equation 6 and Table 1).

In contrast to the greedy definition of sequential search, simultaneous search optimizes alternatives globally. Thus, the simultaneous procedure should yield the same or higher average feature-set quality for the same number of alternatives. Also, the quality can be more evenly distributed over the alternatives, as opposed to the dropping quality over the course of the sequential procedure. However, increasing the number of alternatives still has a negative effect on the average feature-set quality. Further, as opposed to the sequential procedure, there are no intermediate steps where users could interrupt the search.

## 3.3 Objective Functions – Finding Alternatives

In this section, we discuss how to find alternative feature sets. In particular, we describe how to solve the optimization problem from Section 3.1 for the different categories of feature-set quality measures from Section 2.2. We distinguish between white-box optimization (cf. Section 3.3.1), black-box optimization (cf. Section 3.3.2), and embedding alternatives (cf. Section 3.3.3).

### 3.3.1 White-Box Optimization

If the feature-set quality function $Q(s, X, y)$ is sufficiently simple, one can tackle alternative feature selection with a suitable solver for white-box optimization problems. We already showed that our notion of alternative feature sets results in 0-1 integer linear constraints (cf. Proposition 1). We now discuss several feature-selection methods with objectives that admit formulating a 0-1 integer linear problem. Appendix A.2 describes feature-selection methods we did not include in our experiments.

**Univariate filter feature selection** For univariate filter feature selection, the objective function is linear by default. In particular, these methods decompose the quality of a feature set into the qualities of the individual features:

$$\max_{s} \quad Q_{\text{uni}}(s, X, y) = \sum_{j=1}^{n} q(X_{\cdot j}, y) \cdot s_j = \sum_{j=1}^{n} q_j \cdot s_j \tag{11}$$

Here, $q(\cdot)$ typically is a bivariate dependency measure, e.g., mutual information [61] or the absolute value of Pearson correlation, to quantify the relationship between one feature and the prediction target.

For this objective, Appendix A.3 specifies the complete optimization problem, including the constraints for alternatives. Appendix A.4 describes how to potentially speed up optimization by leveraging the monotonicity of the objective. Section 3.5 proposes heuristic search methods for this objective.

Instead of an integer problem, one could formulate a weighted partial maximum satisfiability (MaxSAT) problem [5, 67], i.e., a weighted Max One problem [53]. In particular, Equation 11 is a sum of weighted binary variables, and the constraints for alternatives can be turned into SAT formulas with a cardinality encoding [108] for the sum expressions.

**Post-hoc feature importance** Technically, one can also insert values of post-hoc feature-importance scores into Equation 11. For example, one can pre-compute permutation importance [15] or SAGE scores [23] for each feature and use them as univariate feature qualities $q(X_{\cdot j}, y)$. However, such post-hoc importance scores typically evaluate the quality of each feature in the presence of other features. For example, a feature may only be important in subsets where another feature is present, due to feature interaction, but unimportant otherwise, and a post-hoc importance method like SHAP [70] may reflect both these aspects. In contrast, Equation 11 implicitly assumes feature independence and cannot adapt importance scores depending on whether other features are selected. Thus, treating pre-computed post-hoc importance scores as univariate feature qualities in the optimization objective can serve as a heuristic but may not faithfully represent the feature qualities in a particular feature subset [34].

**FCBF** The Fast Correlation-Based Filter (FCBF) [125] bases on the notion of predominance: Each selected feature's correlation with the prediction target

must exceed a user-defined threshold as well as the correlation of each other selected feature with the given one. While the original FCBF uses a heuristic search to find predominant features, we propose a formulation as a constrained optimization problem to enable a white-box optimization for alternatives:

$$\begin{aligned} \max_{s} \quad & Q_{\text{FCBF}}(s, X, y) = \sum_{j=1}^{n} q(X_{\cdot j}, y) \cdot s_j \\ \text{subject to:} \quad & \forall j_1, j_2 \in \{1, \ldots, n\},\ j_1 \neq j_2,\ (*) : s_{j_1} + s_{j_2} \leq 1 \\ \text{with } (*): \quad & q(X_{\cdot j_1}, y) \leq q(X_{\cdot j_2}, X_{\cdot j_1}) \end{aligned} \tag{12}$$

We drop the original FCBF's threshold on feature-target correlation and maximize the latter instead, as in the univariate-filter case. This change could produce large feature sets that contain many low-quality features. As a countermeasure, one can constrain the feature-set sizes, as we do in our experiments. Additionally, one could also filter out the features with low target correlation before optimization. Further, we keep FCBF's constraints on feature-feature correlation. In particular, we prevent the simultaneous selection of two features if the correlation between them is at least as high as one of the features' correlation to the target. Since the condition $q(X_{\cdot j_1}, y) \leq q(X_{\cdot j_2}, X_{\cdot j_1})$ in Equation 12 does not depend on the decision variables $s$, one can check whether it holds before formulating the optimization problem and add the corresponding linear constraint $s_{j_1} + s_{j_2} \leq 1$ only for feature pairs where it is needed.

**mRMR** Minimal Redundancy Maximal Relevance (mRMR) [93] combines two criteria, i.e., feature relevance and feature redundancy. Relevance corresponds to the dependency between features and prediction target, which should be maximized, as for univariate filters. Redundancy, in turn, corresponds to the dependency between features, which should be minimized. Both terms are averaged over the selected features. Using a bivariate dependency measure $q(\cdot)$, the objective is maximizing the following difference between relevance and redundancy:

$$\begin{aligned} \max_{s} \quad Q_{\text{mRMR}}(s, X, y) = & \frac{\sum_{j=1}^{n} q(X_{\cdot j}, y) \cdot s_j}{\sum_{j=1}^{n} s_j} \\ & - \frac{\sum_{j_1=1}^{n} \sum_{j_2=1}^{n} q(X_{\cdot j_1}, X_{\cdot j_2}) \cdot s_{j_1} \cdot s_{j_2}}{(\sum_{j=1}^{n} s_j)^2} \end{aligned} \tag{13}$$

If one knows the feature-set size $\sum_{j=1}^{n} s_j$ to be a constant $k$, the denominators of both fractions are constant, so the objective leads to a quadratic-programming problem [87, 97]. If one additionally replaces each product term $s_{j_1} \cdot s_{j_2}$ according to Equation 6, the problem becomes linear. However, there is a more efficient

linearization [83, 85], which we use in our experiments:

$$
\begin{aligned}
\max_{s} \quad & Q_{\text{mRMR}}(s, X, y) = \frac{\sum_{j=1}^{n} q(X_{\cdot j}, y) \cdot s_j}{k} - \frac{\sum_{j=1}^{n} z_j}{k \cdot (k-1)} \\
\text{subject to:} \quad \forall j_1 : \quad & A_{j_1} := \sum_{j_2 \neq j_1} q(X_{\cdot j_1}, X_{\cdot j_2}) \cdot s_{j_2} \\
\forall j : \quad & z_j \geq M \cdot (s_j - 1) + A_j \\
\forall j : \quad & z_j \in \mathbb{R}_{\geq 0} \\
\text{with indices:} \quad & j, j_1, j_2 \in \{1, \ldots, n\}
\end{aligned}
\tag{14}
$$

Here, $A_{j_1}$ is the sum of all redundancy terms related to the feature with index $j_1$, i.e., the summed dependency value between this feature and all other selected features. Thus, one can use one real-valued auxiliary variable $z_j$ for each feature instead of one new binary variable for each pair of features. $A_j$ does not need a separate variable but can be directly inserted in the subsequent constraint, so we wrote ':=' instead of '='. Since redundancy should be minimized, $z_j$ assumes the value of $A_j$ with equality if the feature with index $j$ is selected ($s_j = 1$) and is zero otherwise ($s_j = 0$). To this end, $M$ is a large positive value that deactivates the constraint $z_j \geq A_j$ if $s_j = 0$.

Since Equation 14 assumes the feature-set size $k \in \mathbb{N}$ to be user-defined before optimization, it requires fewer auxiliary variables and constraints than the more general formulation in [83, 85]. Additionally, in accordance with [87], we assign a value of zero to the self-redundancy terms $q(X_{\cdot j}, X_{\cdot j})$, effectively excluding them from the objective function. Thus, the redundancy term uses $k \cdot (k-1)$ instead of $k^2$ for averaging.

### 3.3.2 Black-Box Optimization

If feature-set quality does not have an expression suitable for white-box optimization, one has to treat it as a black-box function when searching for alternatives. This situation applies to wrapper feature-selection methods, which use prediction models to assess feature-set quality. One can optimize such black-box functions with search heuristics that systematically iterate over candidate feature sets. However, search heuristics often assume an unconstrained search space and may propose candidate feature sets that are not alternative enough. We see four ways to address this issue:

**Enumerating feature sets** Instead of using a search heuristic, one may enumerate all feature sets that are alternative enough. E.g., one can iterate over all feature sets and sort out those violating the constraints or use a solver to enumerate all valid alternatives directly. Both approaches are usually very inefficient, as there can be a vast number of alternatives.

**Sampling feature sets** Instead of considering all possible alternatives, one can also sample a limited number. E.g., one could sample from all feature sets

but remove samples that are not alternative enough. However, if the number of valid alternatives is small, this approach might need many samples. One could also sample with the help of a solver. However, uniform sampling from a constrained space is a computationally hard problem, possibly harder than determining if a valid solution exists or not [31].

**Multi-objective optimization** If one phrases alternative feature selection as a multi-objective problem (cf. Section 3.1), there are no hard constraints anymore, and one could apply a standard multi-objective black-box search procedure. However, as explained in Section 3.1, we decided to pursue a single-objective formulation with constraints.

**Adapting search** One can adapt an existing search heuristic to consider the constraints for alternatives. One idea is to prevent the search from producing feature sets that violate the constraints or at least make the latter less likely, e.g., with a penalty in the objective function. Another idea is to 'repair' feature sets in the search that violate constraints, e.g., replacing them with the most similar valid feature sets. Such solver-assisted search approaches are common in search methods for software feature models [40, 47, 119], and our following method for wrapper feature selection falls into this category as well. Finally, one could also apply solver-based repair to sampled feature sets.

**Greedy Wrapper** For wrapper feature selection in our experiments, we propose a novel hill-climbing procedure, displayed in Algorithm 1. Unlike standard hill climbing for feature selection [58], our procedure observes constraints. First, the algorithm uses a solver to find one solution that is alternative enough for the set of constraints $C$ (Line 1) and stores it as the currently best solution $S^{\text{opt}}$. Thus, the algorithm has a valid starting point and can always return a solution unless there are no valid solutions at all (Line 3). Note that the solution is not only one feature-selection decision vector $s$ but a set $S$ of them, to enable simultaneous search. For sequential search, $|S| = 1$ and $a = 0$. Also, we adapt the notion of feature-set quality $Q(S, X, y)$ in this algorithm to support a set of feature sets, encompassing the aggregation operator $\text{agg}(\cdot)$ for simultaneous search (cf. Definition 5).

Next, the algorithm tries 'swapping' two features, i.e., selecting them if they were deselected or deselecting them if they were selected (Line 7). The corresponding swap indices $j_1$ and $j_2$ start at 1 and 2, respectively (Lines 4–5). For simultaneous search, we swap the affected features in each alternative. As the swap may violate constraints, the algorithm calls the solver to find the solution $S$ that is closest to the currently best one $S^{\text{opt}}$ while satisfying the swap constraints and the constraints for alternatives $C$. To this end, we measure the similarity between feature-selection decisions with the Hamming distance [22], i.e., how many values of decision variables differ between $S$ and $S^{\text{opt}}$. Overall, we define the maximization problem for Line 7 of Algorithm 1 as follows:

**Algorithm 1:** *Greedy Wrapper* for alternative feature selection.

**Input:** Dataset $X \in \mathbb{R}^{m \times n}$,
Prediction target $y \in Y^m$,
Quality function $Q(S, X, y)$ for sets of feature sets,
Set $C$ of constraints for alternatives,
Maximum number of iterations $max\_iters \in \mathbb{N}$

**Output:** Set of feature-selection decision vectors $S = \{s^{(0)}, \ldots, s^{(a)}\}$

```
S^opt ← solve(C)                                   // Initial alternatives
iters ← 1                                  // Number of iterations = solver calls
if S^opt = ∅ then return ∅                 // No valid alternatives exist
j_1 ← 1                                    // Indices of features to be swapped
j_2 ← j_1 + 1
while iters < max_iters and j_1 < n do
    S ← solve(Equation 16)                                          // Try swap
    iters ← iters + 1
    if S ≠ ∅ and Q(S, X, y) > Q(S^opt, X, y) then                   // Swap if
     improved
        S^opt ← S
        j_1 ← 1                                  // Reset swap-feature indices
        j_2 ← j_1 + 1
    else if j_2 < n then                 // Try next swap; advance one index
        j_2 ← j_2 + 1
    else                                // Try next swap; advance both indices
        j_1 ← j_1 + 1
        j_2 ← j_1 + 1
return S^opt
```

$$
\begin{aligned}
\max_{s^{(0)},\ldots,s^{(a)}} \quad & \mathrm{sim}(S, S^{\mathrm{opt}}) = \sum_{l=0}^{a} \sum_{j=1}^{n} \left( s_j^{(l)} \leftrightarrow s_j^{(\mathrm{opt},\, l)} \right) \\
\text{subject to:} \quad & C \\
& \forall l \in \{0, \ldots, a\} : \quad s_{j_1}^{(l)} \leftrightarrow \neg s_{j_1}^{(\mathrm{opt},\, l)} \\
& \forall l \in \{0, \ldots, a\} : \quad s_{j_2}^{(l)} \leftrightarrow \neg s_{j_2}^{(\mathrm{opt},\, l)}
\end{aligned}
\tag{15}
$$

The values of $s^{(\mathrm{opt},\, l)}$, $j_1$, and $j_2$ in this problem are fixed based on Algorithm 1, while $s^{(l)}$ remains variable. Thus, we obtain a 0-1 integer linear program:

$$
\begin{aligned}
\max_{s^{(0)},\ldots,s^{(a)}} \quad & \operatorname{sim}(S, S^{\text{opt}}) \quad = \sum_{l=0}^{a} \Big( \sum_{\substack{j \in \{1,\ldots,n\} \\ s_j^{(\text{opt},\, l)}=1}} s_j^{(l)} + \sum_{\substack{j \in \{1,\ldots,n\} \\ s_j^{(\text{opt},\, l)}=0}} \Big(1 - s_j^{(l)}\Big) \Big) \\
\text{subject to:} \quad & C \\
& \forall l \in \{0,\ldots,a\}: \quad s_{j_1}^{(l)} = 1 - s_{j_1}^{(\text{opt},\, l)} \\
& \forall l \in \{0,\ldots,a\}: \quad s_{j_2}^{(l)} = 1 - s_{j_2}^{(\text{opt},\, l)}
\end{aligned}
\tag{16}
$$

If a solution $S$ for Equation 16 exists and its quality $Q(S, X, y)$ improves upon the currently best solution $S^{\text{opt}}$, the algorithm proceeds with the new solution, attempting again to swap the first and second features (Lines 10–12). Otherwise, it tries to swap another pair of features (Lines 13–17). Specifically, we assess only one solution per swap instead of exhaustively enumerating and evaluating all valid solutions involving the swap.

The algorithm terminates if it reaches a local optimum, i.e., no swap leads to an improvement, or a fixed number of iterations *max_iters* (Line 6). We define the iteration count as the number of solver calls, i.e., attempts to generate valid alternatives. This number also bounds the number of prediction models trained. However, we only train a model for valid solutions (Line 9), and not all solver invocations may yield one.

### 3.3.3 Embedding Alternatives

If feature selection is embedded into a prediction model, there is no general approach for finding alternative feature sets. Instead, one would need to embed the search for alternatives into model training as well. Thus, we leave the formulation of specific approaches open for future work. E.g., one could adapt the training of decision trees to not split on a feature if the resulting feature set of the tree was too similar to a given feature set. As another example, there are various formal encodings of prediction models, e.g., as SAT formulas [82, 102, 124], where 'training' already uses a solver. In such representations, one may directly add constraints for alternatives.

## 3.4 Time Complexity

In this section, we analyze the time complexity of alternative feature selection. In particular, we study the scalability regarding the number of features $n \in \mathbb{N}$, also considering the feature-set size $k \in \mathbb{N}$ and the number of alternatives $a \in \mathbb{N}_0$. Section 3.4.1 discusses exhaustive search, which works for arbitrary feature-selection methods, while Section 3.4.2 examines the optimization problem with univariate feature qualities (cf. Equation 11). Section 3.4.3 summarizes key results.

### 3.4.1 Exhaustive Search for Arbitrary Feature-Selection Methods

An exhaustive search over the entire search space is the arguably simplest though inefficient approach to finding alternative feature sets. This approach provides an upper bound for the time complexity of a runtime-optimal search algorithm. In this section, we assume unit costs for elementary arithmetic operations like addition, multiplication, and comparison of two numbers.

**Conventional feature selection** In general, the search space of feature selection grows exponentially with $n$, even without alternatives. In particular, there are $2^n - 1$ possibilities to form a single non-empty feature set of arbitrary size. For a fixed feature-set size $k$, there are $\binom{n}{k} = \frac{n!}{k! \cdot (n-k)!} \leq n^k$ solution candidates. In an exhaustive search, we iterate over these feature sets:

**Proposition 2** (Complexity of exhaustive conventional feature selection)**.** *Exhaustive search for one feature set of size $k \in \mathbb{N}$ from $n$ features has a time complexity of $O(n^k)$ without the cost of evaluating the objective.*

Evaluating the objective means computing the quality of each solution candidate so that we can determine the best feature set in the end. The cost of this step depends on the feature-selection method but should usually be polynomial in $n$. Even better, since feature-set quality typically only depends on selected features rather than unselected ones, this cost may be polynomial in $k \ll n$.

If we assume $k \ll n,\ k \in O(1)$, i.e., $k$ being a small constant, independent from $n$, then the complexity in Proposition 2 is polynomial rather than exponential in $n$. This assumption makes sense for feature selection, where one typically wants to obtain a small feature set from a high-dimensional dataset. However, the exponent $k$ may still render an exhaustive search practically infeasible. In terms of parameterized complexity, the problem resides in class $\mathcal{XP}$ since the complexity term has the form $O(f(k) \cdot n^{g(k)})$ [28], here with parameter $k$ and functions $f(k) = 1$, $g(k) = k$.

**Sequential search** Like conventional feature selection, sequential search for alternatives (cf. Definition 2) optimizes feature sets one at a time. However, not all size-$k$ feature sets are valid anymore. In particular, the constraints for alternatives put an extra cost on each solution candidate. Constraint checking involves iterating over all existing feature sets and features to compute the dissimilarity between sets (cf. Equation 21). This procedure entails a cost of $O(a \cdot n)$ for each new alternative and $O((a+1)^2 \cdot n)$ for the whole sequential search with $a$ alternatives. Combining this cost with Proposition 2, we obtain the following proposition:

**Proposition 3** (Complexity of exhaustive sequential search)**.** *Exhaustive sequential search (cf. Definition 3) for $a \in \mathbb{N}_0$ alternative feature sets of size $k \in \mathbb{N}$ from $n$ features has a time complexity of $O((a+1)^2 \cdot n^{k+1})$ without the cost of evaluating the objective.*

Thus, the runtime remains polynomial in $n$ if $k$ is a small constant $k \in O(1)$, which places the problem in the parameterized complexity class $\mathcal{XP}$. Due to the fixed $k$, only choosing $a \leq \binom{n}{k} \in O(n^k)$ admits valid alternatives and therefore does $a$ not affect polynomiality.

**Simultaneous search** The simultaneous-search problem (cf. Definition 4) enlarges the search space since it optimizes $a + 1$ feature sets at once. Thus, an exhaustive search over size-$k$ feature sets iterates over $O((n^k)^{a+1}) = O(n^{k \cdot (a+1)})$ solution candidates. Including the cost of constraint checking, we arrive at the following proposition:

**Proposition 4** (Complexity of exhaustive simultaneous search)**.** *Exhaustive simultaneous search (cf. Definition 5) for $a \in \mathbb{N}_0$ alternative feature sets of size $k \in \mathbb{N}$ from $n$ features has a time complexity of $O((a+1)^2 \cdot n^{k \cdot (a+1)+1})$ without the cost of evaluating the objective.*

The scalability with $n$ is worse than for exhaustive sequential search since the number of alternatives appears in the exponent now, except for a special case discussed in Appendix A.5.1. Further, Proposition 4 assumes that the constraints do not use linearization variables (cf. Equations 6 and 22), which would enlarge the search space even further. Finally, the complexity remains polynomial in $n$ if $a$ and $k$ are small and independent from $n$, i.e., $a \cdot k \in O(1)$:

**Proposition 5** (Parameterized complexity of simultaneous-search problem)**.** *The simultaneous-search problem (cf. Definition 5) for $a \in \mathbb{N}$ alternative feature sets of size $k \in \mathbb{N}$ from $n$ features resides in the parameterized complexity class $\mathcal{XP}$ for the parameter $a \cdot k$.*

### 3.4.2 Univariate Feature Qualities

**Motivation** While the assumption $a \cdot k \in O(1)$ ensures polynomial runtime regarding $n$ for arbitrary feature-selection methods, the optimization problem can still be hard without this assumption. In the following, we derive complexity results for *univariate feature qualities* (cf. Equation 11 and Appendix A.3). This feature-selection method arguably has the simplest objective function, where the quality of a feature set is equal to the sum of the individual qualities of its constituent features. This simplicity eases the transformation from and to well-known $\mathcal{NP}$-hard problems. Appendix A.5.2 discusses related work on these problems in detail.

In the following complexity analyses, we assume that the feature qualities $q(X_{\cdot j}, y)$ are given. In particular, one can pre-compute these qualities before searching alternatives and treat them as constants in the optimization problem. The complexity of this computation depends on the particular feature-quality measure and the number of data objects $m$. However, the number of features $n$ should only affect the complexity linearly due to the univariate setting.

**Min-aggregation with complete partitioning** We start with three assumptions, which we will drop later: First, we use a dissimilarity threshold of $\tau = 1$, i.e., zero overlap of feature sets. Second, all features must be part of one set. Third, we analyze the simultaneous-search problem (cf. Definition 5) with min-aggregation (cf. Equation 18). We call the combination of the first two assumptions, which implies $n = (a+1) \cdot k$ if all sets have size $k$, a *complete partitioning*. This scenario differs from $a \cdot k \in O(1)$, which yielded polynomial runtime regarding $n$ in Proposition 5.

A key factor for the hardness of partitioning is the number of solutions: There are $\left\{ {n \atop a} \right\}$ ways to partition a set of $n$ elements into $a$ non-empty subsets, a Stirling number of the second kind [36], which roughly scale like $a^n/a!$ [77], i.e., exponential in $n$ for a fixed $a$. Even if the subset sizes are fixed, the scalability regarding $n$ remains bad since it bases on a multinomial coefficient.

Our complete-partitioning scenario is a variant of the Multi-Way Number Partitioning problem: Partition a multiset of $n$ integers into a fixed number of $a$ subsets such that the sums of all subsets are as equal as possible [60]. One problem formulation, called Multiprocessor Scheduling in [35], minimizes the maximum subset sum: The goal is to assign tasks with different lengths to a fixed number of processors such that the maximum processor runtime is minimal. Multiplying task lengths with $-1$, one can turn the minimax problem of Multiprocessor Scheduling into the maximin formulation of the simultaneous-search problem with min-aggregation: The tasks become features, the negative task lengths become univariate feature qualities, and the processors become feature sets. Since Multiprocessor Scheduling is $\mathcal{NP}$-complete, even for just two partitions [35], our problem is $\mathcal{NP}$-complete as well:

**Proposition 6** (Complexity of simultaneous-search problem with min-aggregation, complete partitioning, and unconstrained feature-set size)**.** *Assuming univariate feature qualities (cf. Equation 11), a dissimilarity threshold $\tau = 1$, unconstrained feature-set sizes, and all $n$ features have to be selected, the simultaneous-search problem (cf. Definition 5) for alternative feature sets with min-aggregation (cf. Equation 18) is $\mathcal{NP}$-complete.*

Since the assumptions in Proposition 6 denote a special case of alternative feature selection, we directly obtain the following, more general proposition:

**Proposition 7** (Complexity of simultaneous-search problem with min-aggregation)**.** *The simultaneous-search problem (cf. Definition 5) for alternative feature sets with min-aggregation (cf. Equation 18) is $\mathcal{NP}$-hard.*

While Proposition 6 allowed arbitrary sets sizes, there are also existing partitioning problems for constrained $k$, e.g., called Balanced Number Partitioning or K-Partitioning. K-Partitioning with a minimax objective is $\mathcal{NP}$-hard [4] and can be transformed into our maximin objective as above:

**Proposition 8** (Complexity of simultaneous-search problem with min-aggregation, complete partitioning, and constrained feature-set size)**.** *Assuming univariate feature qualities (cf. Equation 11), a dissimilarity threshold $\tau = 1$, de-*

*sired feature-set size $k \in \mathbb{N}$, and all $n$ features have to be selected, the simultaneous-search problem (cf. Definition 5) for alternative feature sets with min-aggregation (cf. Equation 18) is $\mathcal{NP}$-complete.*

**Min-aggregation with incomplete partitioning** We now allow that some features may not be part of any feature set while we keep the assumption of zero feature-set overlap. The problem of finding such an *incomplete partitioning* still is $\mathcal{NP}$-complete in general (cf. Appendix A.5.3 for the proof):

**Proposition 9** (Complexity of simultaneous-search problem with min-aggregation, incomplete partitioning, and constrained feature-set size)**.** *Assuming univariate feature qualities (cf. Equation 11), a dissimilarity threshold $\tau = 1$, desired feature-set size $k \in \mathbb{N}$, and* not *all $n$ features have to be selected, the simultaneous-search problem (cf. Definition 5) for alternative feature sets with min-aggregation (cf. Equation 18) is $\mathcal{NP}$-complete.*

**Min-aggregation with overlapping feature sets** The problem with $\tau < 1$, i.e., set overlap, also is $\mathcal{NP}$-hard in general (cf. Appendix A.5.3 for the proof):

**Proposition 10** (Complexity of simultaneous-search problem with min-aggregation, $\tau < 1$, and constrained feature-set size)**.** *Assuming univariate feature qualities (cf. Equation 11), a dissimilarity threshold $\tau < 1$, and desired feature-set size $k \in \mathbb{N}$, the simultaneous-search problem (cf. Definition 5) for alternative feature sets with min-aggregation (cf. Equation 18) is $\mathcal{NP}$-hard.*

**Sum-aggregation** In contrast to the previous $\mathcal{NP}$-hardness results for min-aggregation, sum-aggregation (cf. Equation 17) with $\tau = 1$ admits polynomial-time algorithms (cf. Appendix A.5.3 for the proof):

**Proposition 11** (Complexity of search problems with sum-aggregation and $\tau = 1$)**.** *Assuming univariate feature qualities (cf. Equation 11) and a dissimilarity threshold $\tau = 1$, the problems for (1) sequential search (cf. Definition 3) and (2) simultaneous search with sum-aggregation (cf. Definition 5 and Equation 17) have a time complexity of $O(n \cdot \log n)$.*

This feasibility result applies to an arbitrary number of alternatives $a$ and arbitrary feature-set sizes. The key reason for polynomial runtime is that sum-aggregation does not require balancing the feature sets' qualities. Thus, $\tau = 1$ allows many solutions with the same summed objective value. While at least one of these solutions also optimizes the objective with min-aggregation, most do not. Hence, it is not a contradiction that optimizing with min-aggregation is considerably harder.

### 3.4.3 Summary

We showed that the simultaneous-search problem for alternative feature sets is $\mathcal{NP}$-hard in general (cf. Proposition 7). We also placed it in the parameterized

complexity class $\mathcal{XP}$ (cf. Proposition 5), having $a$ and $k$ as the parameters that drive the hardness of the problem. For univariate feature qualities and min-aggregation, we obtained more specific $\mathcal{NP}$-hardness results for (1) complete partitioning, i.e., $\tau = 1$ and $(a+1) \cdot k = n$ (cf. Proposition 8), (2) incomplete partitioning, i.e., $(a+1) \cdot k < n$ (cf. Proposition 9) and (3) feature set overlap, i.e., $\tau < 1$ (cf. Proposition 10). In contrast, we also inferred polynomial runtime for univariate feature qualities, sum-aggregation, and $\tau = 1$ (cf. Proposition 11).

## 3.5 Heuristic Search for Univariate Feature Qualities

In this section, we propose heuristic search methods for univariate feature qualities (cf. Equation 11 and Section A.3) and the Dice dissimilarity (cf. Equation 3) as $d(\cdot)$. These heuristics complement the solver-based search that we discussed in Section 3.3.1. The proposed heuristics may be faster than exact optimization at the expense of lower feature-set quality. In particular, we describe *Greedy Replacement* (cf. Section 3.5.1), which is a sequential search method, and *Greedy Balancing* (cf. Section 3.5.2), which is a simultaneous search method. Additionally, Appendix A.6 introduces *Greedy Depth Search*. All three heuristics leverage that the univariate objective sums up the individual qualities $q_j$ of selected features and does not consider interactions between features.

### 3.5.1 Greedy Replacement

*Greedy Replacement* is our first heuristic for alternative feature selection with univariate feature qualities. This heuristic conducts a sequential search.

**Algorithm** Algorithm 2 outlines *Greedy Replacement*. It retains a fixed subset of features in each alternative while iteratively replacing the remaining ones. We start by sorting the features decreasingly based on their qualities $q_j$ (Line 1). For a fixed feature-set size $k$, a dissimilarity threshold $\tau$, and using the Dice dissimilarity (cf. Equation 3), one subset with $\lfloor (1-\tau) \cdot k \rfloor$ features can be contained in all alternatives without violating the dissimilarity threshold (cf. Equation 8). Thus, our algorithm indeed selects the $\lfloor (1-\tau) \cdot k \rfloor$ features with highest quality in each alternative $s^{(\cdot)}$ (Lines 2–7). We fill the remaining spots in the sets by iterating over the alternatives and remaining features (Lines 8–15). For each alternative, we select the $\lceil \tau \cdot k \rceil$ highest-quality features not used in any prior alternative, thereby satisfying the dissimilarity threshold. We continue this procedure until we reach the desired number of alternatives $a$ or until there are not enough unused features to form further alternatives (Line 9).

**Example 1** (Algorithm of *Greedy Replacement*)**.** With $n = 10$ features, feature-set size $k = 5$, and $\tau = 0.4$, each feature set must differ by $\lceil \tau \cdot k \rceil = 2$ features from the other feature sets. The original feature set $s^{(0)}$ consists of the top $k = 5$ features regarding quality $q_j$. The first alternative $s^{(1)}$ consists of the top $\lfloor (1-\tau) \cdot k \rfloor = 3$ features plus the sixth- and seventh-best feature. The second alternative $s^{(2)}$ consists of the top three features plus the eighth- and ninth-best

**Algorithm 2:** *Greedy Replacement* for alternative feature selection.

**Input:** Univariate feature qualities $q \in \mathbb{R}^n$,
Feature-set size $k \in \mathbb{N}$,
Number of alternatives $a \in \mathbb{N}_0$,
Dissimilarity threshold $\tau \in (0, 1]$

**Output:** List of feature-selection decision vectors $s^{(\cdot)}$

```
1  indices ← sort_indices(q, order=descending)   // Order by qualities
2  s ← {0}^n                                     // Initial selection for all alternatives
3  feature_position ← 1                          // Index of index of current feature
4  while feature_position ≤ ⌊(1 − τ) · k⌋ do
5      j ← indices[feature_position]             // Index feature by quality
6      s_j ← 1
7      feature_position ← feature_position + 1
8  l ← 0                                         // Number of current alternative
9  while l ≤ a and l ≤ (n − k)/⌈τ · k⌉ do
10     s^(l) ← s                                 // Select top ⌊(1 − τ) · k⌋ features
11     for _ ← 1 to ⌈τ · k⌉ do                   // Select remaining ⌈τ · k⌉ features
12         j ← indices[feature_position]
13         s_j^(l) ← 1
14         feature_position ← feature_position + 1
15     l ← l + 1
16 return s^(0), ..., s^(l)
```

one. The algorithm cannot continue beyond $l = 2$ since there are not enough unused features to form further alternatives in the same manner.

In general, the $l$-th alternative consists of the top $\lfloor (1-\tau) \cdot k \rfloor$ features plus the features $k + (l-1) \cdot \lceil \tau \cdot k \rceil + 1$ to $k + l \cdot \lceil \tau \cdot k \rceil$ in descending quality order.

**Time complexity** Sorting $n$ feature qualities (Line 1) has a time complexity of $O(n \cdot \log n)$. Next, the algorithm iterates over the features and processes each feature at most once. In particular, after selecting a feature in an alternative, *feature_position* increases by 1. The maximum value of this variable depends on $a$ and $k$ (Line 9) but cannot exceed the total number of features $n$. For each *feature_position*, the algorithm conducts a constant number of update operations (Lines 11–14). Assuming each array access takes $O(1)$, the total update cost is $O(n)$. Further, each alternative $s^{(l)}$ gets initialized as the selection $s$ of the top $\lfloor (1-\tau) \cdot k \rfloor$ features (Line 10), which the algorithm determines once before the main loop (Lines 2–7). Initializing $a$ arrays costs $O(a \cdot n)$. Since the algorithm can only yield $a < n$ alternatives, the overall time complexity is $O(n^2)$, i.e., polynomial in $n$.

**Quality** While not optimizing exactly, *Greedy Replacement* still offers an approximation guarantee relative to exact search methods:

**Proposition 12** (Approximation quality of *Greedy Replacement*)**.** *Assume non-negative univariate feature qualities of $n$ features (cf. Equation 11), $a \in \mathbb{N}_0$ alternatives, the Dice dissimilarity (cf. Equation 3) as $d(\cdot)$, a dissimilarity threshold $\tau \in (0, 1]$, desired feature-set size $k \in \mathbb{N}$, and $k + a \cdot \lceil \tau \cdot k \rceil \leq n$. Under these conditions,* Greedy Replacement *reaches at least a fraction of $\frac{\lfloor (1-\tau) \cdot k \rfloor}{k}$ of the optimal objective values of the optimization problems for (1) sequential search (cf. Definition 3), (2) simultaneous search with sum-aggregation (cf. Definition 5 and Equation 17), and (3) simultaneous search with min-aggregation (cf. Definition 5 and Equation 18).*

*Proof.* For univariate feature qualities, the quality of a feature set is the sum of the qualities of the contained features. *Greedy Replacement* includes the $\lfloor (1 - \tau) \cdot k \rfloor$ highest-quality features in each alternative of size $k$, while the remaining $\lceil \tau \cdot k \rceil$ features may have an arbitrary quality. In comparison, the optimal original, i.e., unconstrained, feature set of size $k$ contains the top $k$ features, which are the union of the top $\lfloor (1 - \tau) \cdot k \rfloor$ features and the next-best $\lceil \tau \cdot k \rceil$ features. Due to quality sorting, each of the next-best $\lceil \tau \cdot k \rceil$ features has at most the quality of each of the top $\lfloor (1 - \tau) \cdot k \rfloor$ features, i.e., contributes the same or less to the summed quality of the feature set. Hence, assuming non-negative qualities, each alternative yielded by *Greedy Replacement* has at least a quality of $\lfloor (1 - \tau) \cdot k \rfloor / k$ relative to the optimal original feature set of size $k$ since the $\lfloor (1 - \tau) \cdot k \rfloor$ highest-quality features are part of both feature sets. Next, the optimal original feature set of size $k$ upper-bounds the quality of any feature set of size $k$. Consequently, alternative feature sets found by exact sequential or simultaneous search can also not be better. Thus, the quality bound of the heuristic solution relative to the exact solution also applies to the minimum and sum of qualities over multiple alternative feature sets. □

In particular, *Greedy Replacement* yields a constant-factor approximation for the three optimization problems mentioned in Proposition 12. The condition $k + a \cdot \lceil \tau \cdot k \rceil \leq n$ describes scenarios where *Greedy Replacement* can yield all desired alternatives, i.e., does not run out of unused features. As the heuristic has polynomial runtime, alternative feature selection lies in the complexity class $\mathcal{APX}$ [54] under the specified conditions:

**Proposition 13** (Approximation complexity of alternative feature selection)**.** *Assume non-negative univariate feature qualities of $n$ features (cf. Equation 11), $a \in \mathbb{N}_0$ alternatives, the Dice dissimilarity (cf. Equation 3) as $d(\cdot)$, a dissimilarity threshold $\tau \in [0, 1]$, desired feature-set size $k \in \mathbb{N}$, and $k + a \cdot \lceil \tau \cdot k \rceil \leq n$. Under these conditions, the optimization problems for (1) sequential search (cf. Definition 3), (2) simultaneous search with sum-aggregation (cf. Definition 5 and Equation 17), and (3) simultaneous search with min-aggregation (cf. Definition 5 and Equation 18) reside in the complexity class $\mathcal{APX}$.*

For $\tau = 1$, *Greedy Replacement* even yields the same objective values as exact sequential search and exact simultaneous search with sum-aggregation since it becomes identical to a procedure we outlined in our complexity analysis earlier (cf. Proposition 11). In contrast, for arbitrary $\tau$, the following example shows that the heuristic can be worse than exact sequential search for as few as $a = 2$ alternatives:

**Example 2** (Quality of *Greedy Replacement* vs. exact search)**.** Consider $n = 6$ features with univariate feature qualities $q = (9, 8, 7, 3, 2, 1)$, feature-set size $k = 2$, number of alternatives $a = 2$, the Dice dissimilarity (cf. Equation 3) as $d(\cdot)$, and dissimilarity threshold $\tau = 0.5$, which permits an overlap of one feature between sets here. Exact sequential search and exact simultaneous search, for min- and sum-aggregation, yield the selection $s^{(0)} = (1, 1, 0, 0, 0, 0)$, $s^{(1)} = (1, 0, 1, 0, 0, 0)$, and $s^{(2)} = (0, 1, 1, 0, 0, 0)$, with a summed quality of $17+16+15 = 48$. *Greedy Replacement* yields the selection $s^{(0)} = (1, 1, 0, 0, 0, 0)$, $s^{(1)} = (1, 0, 1, 0, 0, 0)$, and $s^{(2)} = (1, 0, 0, 1, 0, 0)$, with a summed quality of $17 + 16 + 12 = 45$.

While the first two feature sets are identical between exact and heuristic search, the quality of $s^{(2)}$ is lower for the heuristic (12 vs. 15). In particular, by always selecting all the top $\lfloor (1 - \tau) \cdot k \rfloor$ features, the heuristic misses out on feature sets only involving some or none of them.

For min-aggregation in simultaneous search, $a = 1$ alternative already suffices for the heuristic being potentially worse than exact search:

**Example 3** (Quality of *Greedy Replacement* vs. min-aggregation)**.** Consider $n = 6$ features with univariate feature qualities $q = (9, 8, 7, 3, 2, 1)$, feature-set size $k = 3$, number of alternatives $a = 1$, the Dice dissimilarity (cf. Equation 3) as $d(\cdot)$, and dissimilarity threshold $\tau = 0.5$, which permits an overlap of one feature between sets here. Exact simultaneous search with min-aggregation yields the selection $s^{(0)} = (1, 1, 0, 0, 1, 0)$ and $s^{(1)} = (1, 0, 1, 1, 0, 0)$, with a quality of $\min\{19, 19\} = 19$. *Greedy Replacement* and exact sequential search yield the selection $s^{(0)} = (1, 1, 1, 0, 0, 0)$ and $s^{(1)} = (1, 0, 0, 1, 1, 0)$, with a quality of $\min\{24, 14\} = 14$. Exact simultaneous search with sum-aggregation may yield either of these two solutions or the selection $s^{(0)} = (1, 1, 0, 1, 0, 0)$ and $s^{(1)} = (1, 0, 1, 0, 1, 0)$, all with the same sum-aggregated quality of 38 but different min-aggregated quality.

In particular, *Greedy Replacement* does not balance feature-set qualities since it is a sequential search method. We alleviate this issue with the heuristic *Greedy Balancing* (cf. Section 3.5.2).

**Limitations** Proposition 12 and Examples 2, 3 already showed the potential quality loss of *Greedy Replacement* compared to an exact search for alternatives. Further, the heuristic only works as long as some features have not been part of any feature set yet, i.e., $k + a \cdot \lceil \tau \cdot k \rceil \leq n$. Once the heuristic runs out of unused features, one would need to switch the search method. Thus, to obtain

a high number of alternatives $a$ with the heuristic, the following conditions are beneficial: The number of features $n$ should be high, the feature-set size $k$ should be low, and the dissimilarity threshold $\tau$ should be low. These conditions align well with typical feature-selection scenarios where $k \ll n$.

Another drawback is that *Greedy Replacement* assumes a very simple notion of feature-set quality. If the latter becomes more complex than a sum of univariate qualities, quality-based feature ordering (Line 1) may be impossible or suboptimal. Further, *Greedy Replacement* cannot accommodate additional constraints on feature sets, e.g., based on domain knowledge. Finally, the heuristic assumes the same size $k$ for all feature sets.

### 3.5.2 Greedy Balancing

*Greedy Balancing* adapts *Greedy Replacement* to obtain more balanced feature-set qualities by employing a simultaneous search method. In particular, it tries to distribute the individual feature qualities evenly to alternatives.

**Algorithm** Algorithm 3 outlines *Greedy Balancing*. First, we check whether the algorithm should terminate early, i.e., whether the number of features $n$ is not high enough to satisfy the desired user parameters $k$, $a$, and $\tau$ (Line 1). Next, we select the first $\lfloor (1-\tau) \cdot k \rfloor$ features in each alternative like in *Greedy Replacement* (cf. Algorithm 2), i.e., we pick the features with the highest quality $q_j$ (Lines 2–10).

For the remaining spots in the alternatives, we use a Longest Processing Time (LPT) heuristic (Lines 11–23). Such heuristics are common for Multiprocessor Scheduling and Balanced Number Partitioning problems [4, 21, 65] (cf. Section A.5.2). In particular, we continue iterating over features by decreasing quality. We assign each feature to the alternative that currently has the lowest summed quality $Q^{(l)}$ and whose size $k$ has not been reached yet. We continue this procedure until all alternatives have reached size $k$ (Line 13).

**Example 4** (Algorithm of *Greedy Balancing*)**.** Consider $n = 6$ features with univariate feature qualities $q = (9, 8, 7, 3, 2, 1)$, feature-set size $k = 4$, number of alternatives $a = 1$, and dissimilarity threshold $\tau = 0.5$. The features with qualities 9 and 8 become part of both feature sets, $s^{(0)}$ and $s^{(1)}$, since $\lfloor (1-\tau) \cdot k \rfloor = 2$ (Lines 2–10). At this point, both alternatives have the same relative quality $Q^{(0)} = Q^{(1)} = 0$, which ignores the quality of the shared features. Now the LPT heuristic becomes active (Lines 11–23). The feature with quality 7 is added to $s^{(0)}$, which causes $Q^{(0)} > Q^{(1)}$ (i.e., $7 > 0$). Thus, the feature with quality 3 is added to $s^{(1)}$. As $Q^{(0)} > Q^{(1)}$ (i.e., $7 > 3$) still holds, the feature with quality 2 becomes part of $s^{(1)}$ as well. Because $s^{(1)}$ has reached size $k = 4$, the feature with quality 1 is added to $s^{(0)}$, even if the latter still has a higher relative quality (i.e., $7 > 5$). Now both alternatives have reached their desired size and $n = 6 = \lceil 0.5 \cdot 4 \rceil \cdot 1 + 4 = \lceil \tau \cdot k \rceil \cdot a + k$ (Line 13). Thus, the algorithm terminates. The solution consists of $s^{(0)} = (1, 1, 1, 0, 0, 1)$ and $s^{(1)} = (1, 1, 0, 1, 1, 0)$.

**Algorithm 3:** *Greedy Balancing* for alternative feature selection.

**Input:** Univariate feature qualities $q \in \mathbb{R}^n$,
Feature-set size $k \in \mathbb{N}$,
Number of alternatives $a \in \mathbb{N}_0$,
Dissimilarity threshold $\tau \in [0, 1]$

**Output:** List of feature-selection decision vectors $s^{(0)}, \ldots, s^{(a)}$

**if** $\lceil \tau \cdot k \rceil \cdot a + k > n$ **then return** $\emptyset$
*indices* $\leftarrow$ sort_indices($q$, order=descending) `// Order by qualities`
**for** $l \leftarrow 0$ **to** $a$ **do** `// Initial selection for all alternatives`
    $s^{(l)} \leftarrow \{0\}^n$
*feature_position* $\leftarrow 1$ `// Index of index of current feature`
**while** *feature_position* $\leq \lfloor (1 - \tau) \cdot k \rfloor$ **do** `// Select top features`
    $j \leftarrow$ *indices*[*feature_position*] `// Index feature by quality`
    **for** $l \leftarrow 0$ **to** $a$ **do** `// Same features in all alternatives`
        $s_j^{(l)} \leftarrow 1$
    *feature_position* $\leftarrow$ *feature_position* $+ 1$
**for** $l \leftarrow 0$ **to** $a$ **do**
    $Q^{(l)} \leftarrow 0$ `// Relative quality of each alternative`
**while** *feature_position* $\leq \lceil \tau \cdot k \rceil \cdot a + k$ **do** `// Fill all positions`
    $Q_{\min} \leftarrow \infty$ `// Find alternative with lowest quality`
    $l_{\min} \leftarrow -1$
    **for** $l \leftarrow 0$ **to** $a$ **do**
        **if** $Q^{(l)} < Q_{min}$ ***and*** $\sum_{j=1}^{n} s_j^{(l)} < k$ **then** `// Check cardinality`
            $Q_{\min} \leftarrow Q^{(l)}$
            $l_{\min} \leftarrow l$
    $j \leftarrow$ *indices*[*feature_position*] `// Index feature by quality`
    $s_j^{(l_{\min})} \leftarrow 1$ `// Add to lowest-quality, non-full alternative`
    $Q^{(l_{\min})} \leftarrow Q^{(l_{\min})} + q_j$ `// Update quality of that alternative`
    *feature_position* $\leftarrow$ *feature_position* $+ 1$
**return** $s^{(0)}, \ldots, s^{(a)}$

**Time complexity** Like *Greedy Replacement*, *Greedy Balancing* has an up-front cost of $O(n \cdot \log n)$ for sorting feature qualities (Line 2) and then iterates over $O(n)$ *feature_position*s (Lines 6 and 13). For each *feature_position*, the algorithm iterates over $a$ alternatives (Lines 8 and 16) and conducts a constant number of operations each, which yields total update costs of $O(a \cdot n)$. This figure assumes cardinality checks (Line 17) can be done in $O(1)$, e.g., by storing the current feature-set sizes. There is also a total cost of $O(a \cdot n)$ for array initialization (Lines 3–4). Since $a < n$ (Line 1), the overall time complexity of *Greedy Balancing* is $O(n^2)$, as for *Greedy Replacement*.

**Quality** *Greedy Balancing* selects the same features as *Greedy Replacement* and only changes their assignment to the feature sets. Thus, the summed feature-set quality remains the same, while the minimum feature-set quality may be higher due to balancing. Hence, the quality guarantee of *Greedy Replacement* (cf. Proposition 12) holds here as well:

**Proposition 14** (Approximation quality of *Greedy Balancing*)**.** *Assume non-negative univariate feature qualities of $n$ features (cf. Equation 11), $a \in \mathbb{N}_0$ alternatives, the Dice dissimilarity (cf. Equation 3) as $d(\cdot)$, a dissimilarity threshold $\tau \in [0, 1]$, desired feature-set size $k \in \mathbb{N}$, and $k + a \cdot \lceil \tau \cdot k \rceil \leq n$. Under these conditions,* Greedy Balancing *reaches at least a fraction of $\frac{\lfloor (1-\tau) \cdot k \rfloor}{k}$ of the optimal objective values of the optimization problems for (1) sequential search (cf. Definition 3), (2) simultaneous search with sum-aggregation (cf. Definition 5 and Equation 17), and (3) simultaneous search with min-aggregation (cf. Definition 5 and Equation 18).*

For min-aggregation in the objective, *Greedy Balancing* can even be better than exact sequential search, as Example 3 shows, where the heuristic search would yield the same solution as exact simultaneous search with min-aggregation. However, the heuristic can also be worse than exact sequential search and exact simultaneous search, as Example 2 shows, where *Greedy Balancing* would yield the same solution as *Greedy Replacement*.

**Limitations** *Greedy Balancing* shares several limitations with *Greedy Replacement*, e.g., it may be worse than exact search, assumes univariate feature qualities, and does not work if the number of features $n$ is too low relative to $k$, $a$, and $\tau$. In the latter case, *Greedy Balancing* yields no solution due to its simultaneous nature, while *Greedy Replacement* yields at least some alternatives. However, if running out of features is not an issue, *Greedy Balancing* has the advantage of more balanced feature-set qualities. Also, one could easily adapt *Greedy Balancing* to yield the largest feasible number of alternatives in case $a$ alternatives are infeasible.

# 4 Related Work

In this section, we review related work from the fields of feature selection (cf. Section 4.1), subgroup discovery (cf. Section 4.2), clustering (cf. Section 4.3), subspace clustering and subspace search (cf. Section 4.4), explainable artificial intelligence (cf. Section 4.5), and Rashomon sets (cf. Section 4.6). To the best of our knowledge, searching for optimal alternative feature sets in the sense of this paper is novel. However, there is literature on optimal alternatives outside the field of feature selection. Also, there are works on finding multiple, diverse feature sets.

## 4.1 Feature Selection

**Conventional feature selection** Most feature-selection methods only yield one solution [14], though some exceptions exist. Nevertheless, none of the following approaches searches for optimal alternatives in our sense.

[106] proposes a genetic algorithm that iteratively updates a population of multiple feature sets. To foster diversity, the algorithm's fitness criterion does not only consider feature-set quality but also a penalty on feature-set overlap in the population. However, users cannot control the admissible overlap, i.e., there is no parameter comparable to $\tau$. In contrast, the genetic algorithm's parameter for the population size corresponds to the number of alternatives.

[30] employs multi-objective genetic algorithms to obtain prediction models with different complexity and diverse feature sets. However, the two objectives are prediction performance and feature-set size, while diversity only influences the genetic selection step under particular circumstances.

[80] clusters features and forms alternatives by picking one feature from each cluster. However, they do this to reduce the number of features for subsequent model selection and model evaluation, not as a guided search for alternatives.

**Ensemble feature selection** Ensemble feature selection [100, 104] combines feature-selection results, e.g., obtained by different feature-selection methods or on different samples of the data. Fostering diverse feature sets might be a subgoal to improve prediction performance, but it is usually only an intermediate step. This focus differs from our goal of finding optimal alternatives.

[123] obtains feature sets or rankings on bootstrap samples of the data. Next, an aggregation strategy creates one or multiple diverse feature sets. The authors propose using k-medoid clustering and frequent itemset mining for the latter. While these approaches allow to control the number of feature sets, there is no parameter for their dissimilarity. Also, aggregation builds on bootstrap sampling instead of being allowed to form arbitrary alternatives.

[69] builds an ensemble prediction model from classifiers trained on different feature sets. To this end, a genetic algorithm iteratively evolves a population of feature sets. Diversity is one of multiple fitness criteria, with the Hamming distance quantifying the dissimilarity of feature sets. However, since feature diversity is only one of several objectives, users cannot control it directly.

[105] uses different univariate feature-quality measures to initialize a population of multiple feature sets. Next, they iteratively form new feature sets by choosing those features from two existing feature sets that are only in either set. This procedure does not guarantee how diverse the final feature sets are.

[41] computes feature relevance separately for each class and then combines the top features. This procedure can yield alternatives but does not enforce dissimilarity. Also, the number of alternatives is fixed to the number of classes.

**Statistically equivalent feature sets** Approaches for statistically equivalent feature sets [14, 63] use statistical tests to determine features or feature sets that are equivalent for predictions. E.g., a feature may be independent of the target given another feature. A search algorithm conducts multiple such tests and outputs equivalent feature sets or a corresponding feature grouping.

Our notion of alternatives differs from equivalent feature sets in several aspects. In particular, building optimal alternatives from equivalent feature sets is not straightforward. Depending on how the statistical tests are configured, there can be an arbitrary number of equivalent feature sets without explicit quality-based ordering. Instead, we always provide a fixed number of alternatives. Also, our alternatives need not have equivalent quality but should be optimal under constraints. Further, our dissimilarity threshold allows controlling overlap between feature sets instead of eliminating all redundancies.

**Constrained feature selection** We define alternatives via constraints on feature sets. There already is work on other kinds of constraints in feature selection, e.g., for feature cost [91], feature groups [126], or domain knowledge [8, 37]. These approaches are orthogonal to our work, as such constraints do not explicitly foster optimal alternatives. At most, they might implicitly lead to alternative solutions [8]. Further, most of the approaches are tied to particular constraint types, while our integer-programming formulation supports such constraints besides the ones for alternatives. [8] is an exception in that regard since it models feature selection as a Satisfiability Modulo Theories (SMT) optimization problem [12, 88], which also admits our constraints for alternatives.

## 4.2 Subgroup Discovery

[66] presents six strategies to foster diversity in subgroup set discovery, which searches for interesting regions in the data space, i.e., combinations of conditions on feature values, rather than only selecting features. Three strategies yield a fixed number of alternatives and the other three a variable number. The strategies become part of beam search, i.e., a heuristic search procedure, while we mainly consider exact optimization. Also, the criteria for alternatives differ from ours. The strategy *fixed-size description-based selection* prunes subgroups with the same quality as previously found ones if they differ by at most one feature-value condition. In contrast, we require dissimilarity independent from the quality, have a flexible dissimilarity threshold, and support simultaneous besides sequential search for alternatives. Another strategy, *variable-size*

*description-based selection*, limits the total number of subgroups a feature may occur in but does not constrain subgroup overlap per se. The four remaining strategies in [66] have no obvious counterpart in our feature-selection scenario.

## 4.3 Clustering

Finding alternative solutions has been addressed extensively in the field of clustering. [11] gives a taxonomy and describes algorithms for alternative clustering. Our problem definition in Sections 3.1 and 3.2 is, on a high level, inspired by the one in [11]: Find multiple solutions that maximize quality while minimizing similarity. [11] also distinguishes between singular/multiple alternatives and sequential/simultaneous search. They mention constraint-based search for alternatives as one of several solution paradigms. Further, feature selection can help to find alternative clusterings [111]. Nevertheless, the problem definition for alternatives in clustering and feature selection is fundamentally different. First, the notion of dissimilarity differs, as we want to find differently composed feature sets while alternative clustering targets at different assignments of data objects to clusters. Second, our objective function, i.e., feature-set quality, relates to a supervised prediction scenario while clustering is unsupervised.

Two exemplary approaches for alternative clustering are *COALA* [9] and *MAXIMUS* [10]. COALA [9] imposes *cannot-link constraints* on pairs of data objects rather than constraining features: Data objects from the same cluster in the original clustering should be assigned to different clusters in the alternative clustering. In each step of its iterative clustering procedure, COALA compares the quality of an action observing the constraints to another one violating them. Based on a threshold on the quality ratio, either action is taken. MAXIMUS [10] employs an integer program to formulate dissimilarity between clusterings. In particular, it wants to maximize the dissimilarity of the feature-value distributions in clusters between the clusterings. The output of the integer program leads to constraints for a subsequent clustering procedure.

## 4.4 Subspace Clustering and Subspace Search

Finding multiple useful feature sets plays a role in subspace clustering [39, 48, 79] and subspace search [33, 86, 112]. These approaches strive to improve the results of data-mining algorithms by using subspaces, i.e., feature sets, rather than the full space, i.e., all features. While some subspace approaches only consider individual subspaces, others explicitly try to remove redundancy between subspaces [39, 48, 86] or foster subspace diversity [33, 112]. In particular, [48] surveys subspace-clustering approaches yielding multiple results and discusses the redundancy aspect. However, subspace clustering and -search approaches differ from alternative feature selection in at least one of the following aspects:

First, the objective differs, i.e., definitions of subspace quality deviate from feature-set quality in our scenario. Second, definitions of subspace redundancy may consider dissimilarity between projections of the entire data, i.e., data objects with feature values, into subspaces, while our notion of dissimilarity purely

bases on binary feature-selection decisions. Third, controlling dissimilarity in subspace approaches is often less user-friendly than with our parameter $\tau$. E.g., dissimilarity might be a regularization term in the objective rather than a hard constraint, or there might not be an explicit control parameter at all.

## 4.5 Explainable Artificial Intelligence (XAI)

In the field of XAI, alternative explanations might provide additional insights into predictions, enable users to develop and test different hypotheses, appeal to different kinds of users, and foster trust in the predictions [56, 118]. In contrast, obtaining significantly different explanations for the same prediction might raise doubts about how meaningful the explanations are [49]. Finding diverse explanations has been studied for various explainers, e.g., for counterfactuals [24, 50, 74, 78, 99, 115], criticisms [55], and semifactual explanations [2]. There are several approaches to foster diversity, e.g., ensembling different kinds of explanations [107], considering multiple local minima [115], using a search algorithm that maintains diversity [24], extending the optimization objective [2, 55, 78], or introducing constraints [50, 74, 99]. The last option is similar to the way we enforce alternatives. Of the various mentioned approaches, only [2, 74, 78] introduce a parameter to control the diversity of solutions. Of these three works, only [74] offers a user-friendly dissimilarity threshold in $[0, 1]$, while the other two approaches employ a regularization parameter in the objective.

Despite similarities, all the previously mentioned XAI techniques tackle different problems than alternative feature selection. In particular, they provide local explanations, i.e., target at prediction outcomes for individual data objects and build on feature values. In contrast, we are interested in the global prediction quality of feature sets. For example, counterfactual explanations [38, 110, 114] alter feature *values as little as possible* to produce an alternative prediction *outcome*. In contrast, alternative feature sets might alter the feature *selection significantly* while trying to maintain the original prediction *quality*.

## 4.6 Rashomon Sets

A Rashomon set is a set of prediction models that reach a certain, e.g., close-to-optimal, prediction performance [32]. Despite similar performance, these models may still assign different feature importance scores, leading to different explanations [62]. Thus, Rashomon sets may yield partial information about alternative feature sets. However, approaches for Rashomon sets do not explicitly search for alternative feature sets as a whole, i.e., feature sets satisfying a dissimilarity threshold relative to other sets. Instead, these approaches focus on the range of each feature's importance over prediction models. Further, our notion of alternatives is not bound to model-based feature importance but encompasses a broader range of feature-selection methods. Finally, we use importance scores from one instead of multiple models to find importance-based alternatives.

# 5 Experimental Design

In this section, we introduce our experimental design. After a brief overview of its goal and components (cf. Section 5.1), we describe its components in detail: evaluation metrics (cf. Section 5.2), methods (cf. Section 5.3), and datasets (cf. Section 5.4). Finally, we briefly outline our implementation (cf. Section 5.5).

## 5.1 Overview

We conduct experiments with 30 binary-classification datasets. As evaluation metrics, we consider feature-set quality and runtime. We compare five feature-selection methods, representing different notions of feature-set quality. Also, we train prediction models with the resulting feature sets and analyze prediction performance. To find alternatives, we consider simultaneous and sequential search, both with solver-based and heuristic search methods. We systematically vary the two user parameters for searching alternatives, i.e., the number of alternatives $a$ and the dissimilarity threshold $\tau$.

## 5.2 Evaluation Metrics

**Feature-set quality** We evaluate feature-set quality with two metrics. First, we report the *objective value* $Q(s, X, y)$ of the feature-selection methods, which guided the search for alternatives. Second, we train prediction models with the found feature sets. We report *prediction performance* in terms of the Matthews correlation coefficient (MCC) [71]. This coefficient is insensitive to class imbalance, reaches its maximum of 1 for perfect predictions, and is 0 for random guessing as well as constant predictions.

We conduct a stratified five-fold cross-validation. In particular, the search for alternatives and model training only use the training data, while we employ the test data for evaluation: For the test-set objective value, we compute the objective on the test set but with the feature selection from the training set. For the test-set prediction performance, we predict on the test set but use a prediction model trained with these features on the training set.

**Runtime** We consider two metrics related to runtime.

First, we analyze the *optimization time*. For white-box feature-selection methods in solver-based search for alternatives, we sum the measured runtime of solver calls. We exclude the time for computing feature qualities and feature dependencies for the objective since one can compute these values once per dataset and then reuse them in each solver call. For *Greedy Wrapper* feature selection and the heuristic search methods for alternatives, we measure the runtime of the corresponding search algorithms. For *Greedy Wrapper*, this search procedure involves multiple solver calls and trainings of a prediction model.

Second, we examine the *optimization status*, which can take four values for the solver-based search. If the solver finished before the timeout, it either found an *optimal* solution or proved the problem *infeasible*, i.e., no solution exists.

If the solver reached its timeout, it either found a *feasible* solution without proving its optimality or found no valid solution, though one might exist, so the problem is *not solved*. For the heuristic search methods, we only use *not solved* and *feasible* as statuses, as these search methods are neither guaranteed to find the optimum nor do they prove infeasibility if they terminate early.

## 5.3 Methods

We employ multiple methods for making predictions (cf. Section 5.3.1), feature selection (cf. Section 5.3.2), and searching alternatives (cf. Section 5.3.3).

### 5.3.1 Prediction

As prediction models, we use decision trees [16], which admit learning complex, non-linear dependencies from the data. Preliminary experiments with random forests [15] and k-nearest neighbors yielded similar insights. We leave the trees' hyperparameters at their defaults, except for using information gain instead of Gini impurity as the split criterion, to be consistent with our filter feature-selection methods. Note that tree models also select features themselves, so they may not use all features from the alternative feature sets. However, this is not an issue for our study. We are interested in which performance the models achieve if limited to certain feature sets, not how they use each available feature.

### 5.3.2 Feature Selection (Objective Functions)

We search for alternatives under different notions of feature-set quality in the objective function. We choose five well-known feature-selection methods that are easy to parameterize and cover the different categories from Section 2.2 except *embedded*, as explained in Section 3.3.3. However, we use feature importance from an embedded method as post-hoc importance scores. Four feature-selection methods allow a white-box formulation of the optimization problem, while *Greedy Wrapper* is black-box. With each feature-selection method, we enforce fixed feature-set sizes of $k \in \{5, 10\}$.

**Filter feature selection** We evaluate three filter methods, all using mutual information [61] as the dependency measure $q(\cdot)$. This measure can capture arbitrary dependencies rather than, e.g., just linear ones. *MI* denotes a univariate filter (cf. Equation 11), while *FCBF* (cf. Equation 12) and *mRMR* (cf. Equation 14) are multivariate. We normalize the mutual-information values per dataset and cross-validation fold to improve comparability: For *FCBF* and *MI*, we scale the individual features' qualities with a constant such that the overall objective value is in $[0, 1]$. For *mRMR*, we min-max-normalize all mutual-information values to $[0, 1]$, so the overall objective is in $[-1, 1]$.

**Wrapper feature selection** As a wrapper method, we employ our hill-climbing procedure *Greedy Wrapper* (cf. Algorithm 1) with $max\_iters = 1000$.

To evaluate feature-set quality in the wrapper, we apply a stratified 80:20 hold-out split and train decision trees. $Q(s, X, y)$ corresponds to the prediction performance in terms of MCC on the 20% validation part.

**Post-hoc feature importance** As a post-hoc importance measure called *Model Gain*, we use importance scores from *scikit-learn's* decision trees. There, importance expresses a feature's contribution towards optimizing the split criterion of the tree, for which we choose information gain. These importances are normalized to sum up to 1 by default. We plug them into Equation 11, i.e., treat them like univariate filter scores, though they actually originate from trees trained with all features and thus are not univariate.

#### 5.3.3 Alternatives (Constraints)

**Overview** In our evaluation, we categorize search methods for alternatives in two dimensions that are orthogonal to each other: Solver-based vs. heuristic and sequential vs. simultaneous. Also, we analyze the impact of the user parameters $a$ and $\tau$.

**Solver-based search methods** For the four feature-selection methods with white-box objectives, we use the integer-programming solver *SCIP* [13] to solve the underlying optimization problems exactly. Given sufficient solving time, these alternatives are globally optimal. For *Greedy Wrapper*, the search procedure (Algorithm 1) is heuristic (though still solver-based, so we place it in this category) and might not cover the entire search space. There, the solver only assists in finding valid solutions but does not optimize quality.

For each feature selection method, we analyze *sequential* (cf. Definition 3) and *simultaneous* (cf. Definition 5) solver-based search for alternatives. For the latter, we employ sum-aggregation (cf. Equation 17) and min-aggregation (cf. Equation 18) in the objective. In figures and tables, we use the abbreviations *seq.*, *sim. (sum)*, and *sim. (min)*.

**Heuristic search methods** We also evaluate heuristic search methods, which do not use a solver (cf. Section 3.5). In particular, we employ *Greedy Replacement* (cf. Algorithm 2), which is a sequential search method, and *Greedy Balancing* (cf. Algorithm 3), which is a simultaneous search method. In figures and tables, we use the abbreviations *rep.* and *bal.*. Since these heuristics assume univariate feature qualities (cf. Equation 11), we only combine them with the univariate feature-selection methods *MI* and *Model Gain*.

**Search parametrization** We vary the parameters of the search systematically: We evaluate $a \in \{1, \ldots, 10\}$ alternatives for sequential search methods and $a \in \{1, \ldots, 5\}$ for simultaneous search methods due to the higher runtime of the latter. For the dissimilarity threshold $\tau$, we analyze all possible sizes of the feature-set overlap in the Dice dissimilarity (cf. Equations 3 and 8). Thus,

for $k = 5$, we consider $\tau \in \{0.2, 0.4, 0.6, 0.8, 1.0\}$, corresponding to an overlap of four to zero features. For $k = 10$, we consider $\tau \in \{0.1, 0.2, \ldots, 1.0\}$. We exclude $\tau = 0$, which would allow returning duplicate feature sets.

**Timeout** In solver-based search, we employ a timeout to enable a large-scale evaluation and account for the high variance of solver runtime. In particular, we grant each solver call 60 s multiplied by the number of feature sets sought. Thus, solver-based sequential search conducts multiple solver calls with 60 s timeout each, while solver-based simultaneous search conducts one solver call with proportionally more time, e.g., 300 s for five feature sets. For 84% of the feature sets in our evaluation, the solver finished before the timeout.

**Competitors for search methods** As discussed in Section 4, related work pursues different objective functions, operates with different notions of alternatives, and may only target particular feature-selection methods. All these points prevent a meaningful comparison to our search methods. E.g., a feature set deemed alternative in related work may violate our constraints for alternatives. Further, we can still put the feature-set quality into perspective by comparing alternatives to each other. In particular, the original feature set, i.e., without constraints for alternatives, serves as a natural reference point.

## 5.4 Datasets

We use datasets from the Penn Machine Learning Benchmarks (PMLB) [90, 98]. To harmonize evaluation, we only consider binary-classification datasets, though alternative feature selection also works for regression and multi-class problems. We exclude datasets with less than 100 data objects since they may entail a high uncertainty when assessing feature-set quality. Also, we exclude datasets with less than 15 features to leave room for alternatives. Next, we exclude one dataset with 1000 features, which would dominate the overall runtime. Finally, we manually exclude datasets that seem to be duplicated or modified versions of other datasets. Consequently, we obtain 30 datasets with 106 to 9822 data objects and 15 to 168 features (cf. Table 2). The datasets do not contain any missing values. Categorical features have an ordinal encoding by default.

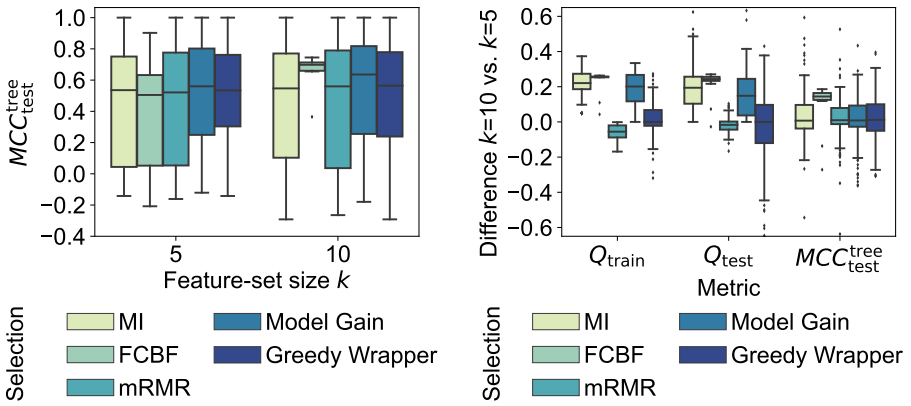
## 5.5 Implementation and Execution

We implemented our experimental pipeline in Python 3.8, using *scikit-learn* [92] for machine learning and the integer-programming solver *SCIP* [13] via the package *OR-Tools* [94] for solver-based search. The code is available on *GitHub*[1] and additionally backed up in the *Software Heritage archive*[2]. A requirements file in our repository specifies the versions of all dependencies. Further, we

[1] `https://github.com/Jakob-Bach/Alternative-Feature-Selection`
[2] swh:1:dir:6b679eb1b901c281b7c7e7fdc9dbdaec2f627c7a

Table 2: Datasets from PMLB used in our experiments. $m$ denotes the number of data objects and $n$ the number of features. In dataset names, we replaced *GAMETES_Epistasis* with *GE_* and *GAMETES_Heterogeneity* with *GH_* to reduce the table's width.

| Dataset | $m$ | $n$ |
|---|---|---|
| backache | 180 | 32 |
| chess | 3196 | 36 |
| churn | 5000 | 20 |
| clean1 | 476 | 168 |
| clean2 | 6598 | 168 |
| coil2000 | 9822 | 85 |
| credit_a | 690 | 15 |
| credit_g | 1000 | 20 |
| dis | 3772 | 29 |
| GE_2_Way_20atts_0.1H_EDM_1_1 | 1600 | 20 |
| GE_2_Way_20atts_0.4H_EDM_1_1 | 1600 | 20 |
| GE_3_Way_20atts_0.2H_EDM_1_1 | 1600 | 20 |
| GH_20atts_1600_Het_0.4_0.2_50_EDM_2_001 | 1600 | 20 |
| GH_20atts_1600_Het_0.4_0.2_75_EDM_2_001 | 1600 | 20 |
| hepatitis | 155 | 19 |
| Hill_Valley_with_noise | 1212 | 100 |
| horse_colic | 368 | 22 |
| house_votes_84 | 435 | 16 |
| hypothyroid | 3163 | 25 |
| ionosphere | 351 | 34 |
| molecular_biology_promoters | 106 | 57 |
| mushroom | 8124 | 22 |
| ring | 7400 | 20 |
| sonar | 208 | 60 |
| spambase | 4601 | 57 |
| spect | 267 | 22 |
| spectf | 349 | 44 |
| tokyo1 | 959 | 44 |
| twonorm | 7400 | 20 |
| wdbc | 569 | 30 |




(a) Test-set prediction performance by feature-set size $k$.

(b) Difference in feature-set quality between $k = 10$ and $k = 5$ by evaluation metric. Y-axis is truncated to improve readability.

Figure 1: Distribution of feature-set quality over datasets and cross-validation folds, by feature-selection method. Results from the original feature sets of solver-based sequential search.

released the methods for alternative feature selection as the Python package *alfese*[3] on *PyPI* to ease reuse. Finally, we published all experimental data[4].

Our experimental pipeline parallelizes over datasets, cross-validation folds, and feature-selection methods, while each of these experimental tasks runs single-threaded. We ran the pipeline on a server with 160 GB RAM and an *AMD EPYC 7551* CPU, having 32 physical cores and a base clock of 2.0 GHz. With this hardware, the parallelized pipeline run took approximately 249 hours, i.e., about 10.4 days.

# 6 Evaluation

In this section, we evaluate our experiments. After comparing feature-selection methods without constraints (cf. Section 6.1), we discuss the parametrization for searching alternatives: the search methods (cf. Section 6.2) and the two user parameters $a$ and $\tau$ (cf. Section 6.3). Section 6.4 summarizes key findings.

## 6.1 Feature-Selection Methods

**Prediction performance** The five feature-selection methods in our experiments employ different objective functions $Q(s, X, y)$, so comparing objective

[3] `https://pypi.org/project/alfese/`
[4] `https://doi.org/10.35097/4ttgrpx92p30jwww`

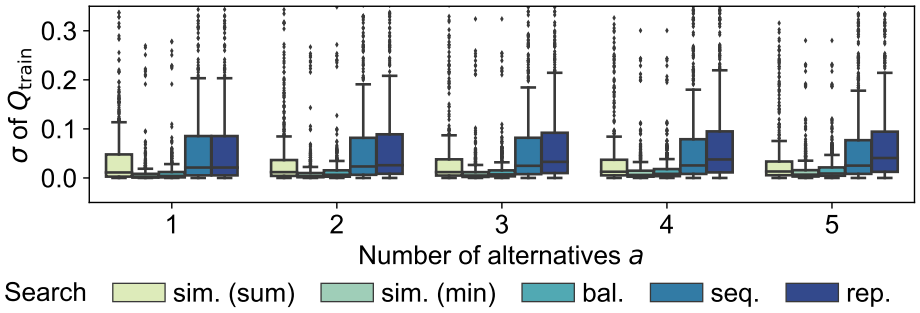
values between them does not make sense. However, we can analyze the prediction performance of the obtained feature sets. Figure 1a displays the distribution of a decision tree's test-set MCC on the original feature sets, i.e., without constraints, for the feature-selection methods. The mean test-set MCC is 0.53 for *Model Gain* and *Greedy Wrapper*, 0.47 for *MI*, 0.46 for *mRMR*, and 0.43 for *FCBF*. Thus, the analyses of alternative feature sets in Sections 6.2 and 6.3 focus on *Model Gain* while still discussing the remaining feature-selection methods.

The univariate, model-free method *MI* keeps up surprisingly well with more sophisticated methods. It uses the same objective function as *Model Gain* but obtains its feature qualities from a bivariate dependency measure rather than a prediction model.

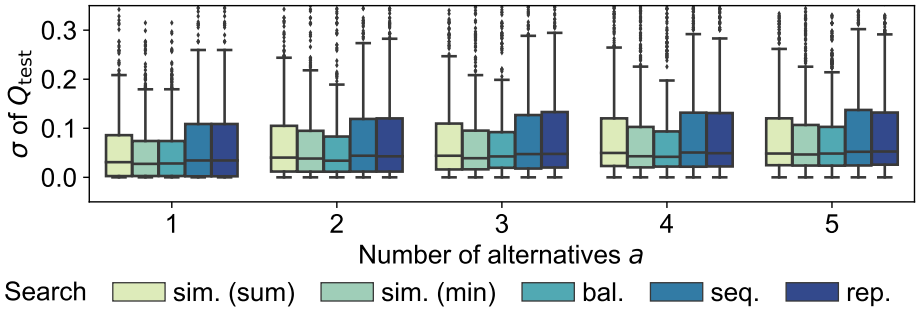
*Greedy Wrapper* uses prediction performance to assess feature-set quality but employs a search heuristic instead of optimizing globally. In particular, it performed 661 iterations on average to determine the original feature sets. However, the number of possible feature sets is higher, e.g., already $\binom{15}{5} = 3003$ for the lowest-dimensional dataset in our evaluation (cf. Table 2) and $k = 5$. The still high prediction performance comes at the expense of high runtime (cf. Table 5), so we prefer *Model Gain* for later evaluations.

*FCBF*'s results may be taken with a grain of salt: The original feature set in solver-based sequential search is already infeasible, i.e., no solution satisfied the constraints, in 71% of the cases for *FCBF* but never for the other feature-selection methods. Over all solver-based search runs, even 89% of the feature sets were infeasible for *FCBF* but only 18% for *Model Gain*. In particular, the combination of feature-correlation constraints in our formulation of *FCBF* (cf. Equation 12) with a fixed feature-set size $k$ may make the problem infeasible, especially if $k$ gets larger.

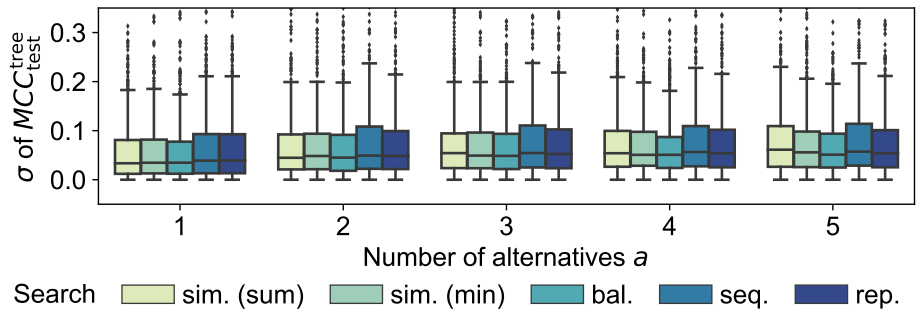
**Influence of feature-set size $k$** One could expect larger feature sets to exhibit a higher feature-set quality than smaller ones, but the picture in our experiments is more nuanced. In particular, quality may not increase proportionally with $k$ or may even decrease. As Figure 1b shows for the original feature sets of solver-based sequential search, *MI* and *Model Gain* exhibit an increase of the training-set objective value $Q_{\text{train}}$ from $k = 5$ to $k = 10$, i.e., the difference depicted in Figure 1b is positive. As these objectives are monotonic and the feature qualities are non-negative, a decrease in the training-set objective value is impossible. In contrast, *Greedy Wrapper* with its black-box objective does not necessarily benefit from more features. The latter insight also applies to *mRMR*, which normalizes its objective with the number of selected features and penalizes feature redundancy. For *FCBF*, the fraction of feasible feature sets changes considerably from $k = 5$ to $k = 10$, so the overall quality between these two settings should not be compared. As Figure 1b also displays, the benefit of larger feature sets is even less clear for prediction performance. In particular, all feature-selection methods except *FCBF* show a median difference in test-set MCC close to zero when comparing $k = 5$ to $k = 10$.




(a) Training-set objective value.




(b) Test-set objective value.




(c) Test-set prediction performance.

Figure 2: Distribution of standard deviation of feature-set quality within search runs over datasets, cross-validation-folds, and $\tau$, by search method for alternatives and number of alternatives $a$. Results with *Model Gain* as the feature-selection method and $k = 5$. Excludes search settings where at least one combination of search method and $a$ yielded no valid solution. Y-axes are truncated to improve readability.

## 6.2 Search Methods for Alternatives

**Variance in feature-set quality** As expected, the search method influences how much the training-set objective value $Q$ varies between multiple alternatives obtained for the same experimental settings, i.e., within one search run for alternatives. Figure 2a visualizes this result for *Model Gain* as the feature-selection method and $k = 5$. The figure shows how the standard deviation of the training-set objective value within individual search runs for alternatives is distributed over other experimental settings, e.g., datasets and cross-validation folds. In particular, the quality of multiple alternatives varies more if they are found by solver-based sequential search rather than solver-based simultaneous search. For solver-based simultaneous search, min-aggregation yields considerably more homogeneous feature-set quality than sum-aggregation. These findings apply to all white-box feature-selection methods but not *Greedy Wrapper*.

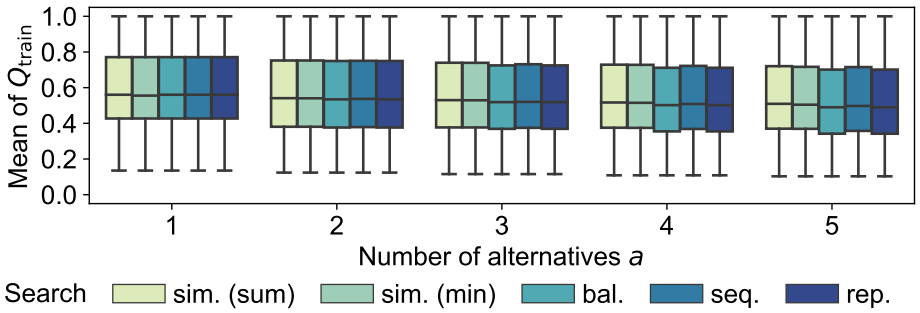
The heuristic search method *Greedy Balancing* yields a small variance of training-set objective value within search runs, only slightly higher than for solver-based simultaneous search with min-aggregation. In contrast, *Greedy Replacement* rather mimics solver-based sequential search, having a substantial variance of quality. Additionally, the variance of *Greedy Replacement* noticeably grows with the number of alternatives $a$.

As Figures 2b and 2c show, the variance of feature-set quality differs considerably less between the search methods on the test set, for the objective value as well as prediction performance. This effect might result from overfitting: Even if the training-set quality is similar, some alternatives might generalize better than others. Thus, this variance caused by overfitting could alleviate the effect caused by the choice of search method.

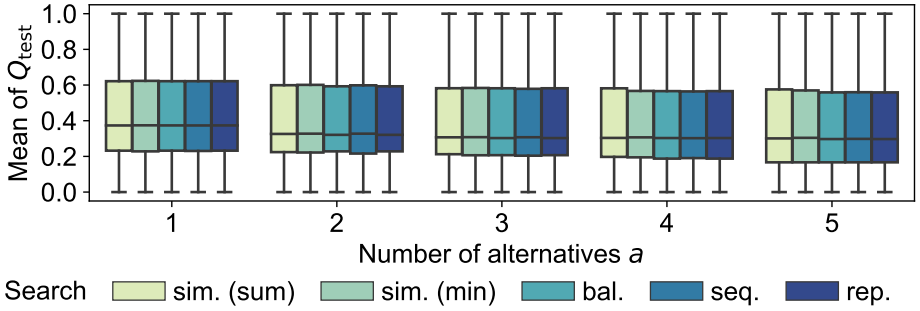
**Average value of feature-set quality** While obtaining alternatives of homogeneous quality can be one goal of simultaneous search, another selling point would be reaching higher average quality than sequential search. However, this potential advantage rarely materialized in our experiments. In particular, Figure 3a compares the distribution of the mean training-set objective value in search runs with *Model Gain* as the feature-selection method and $k = 5$. We observe that all search methods yield very similar overall distributions of average feature-set quality. Comparing solver-based sequential and simultaneous search on each experimental setting separately and then aggregating also shows a mean quality difference close to zero. Further, outliers can occur in both directions, i.e., either solver-based search method may yield higher quality.

The mean test-set objective value in Figure 3b and the mean test-set prediction performance in Figure 3c also exhibit a negligible quality difference between the search methods. Finally, the other four feature-selection methods do not show a general quality advantage of solver-based simultaneous search either.

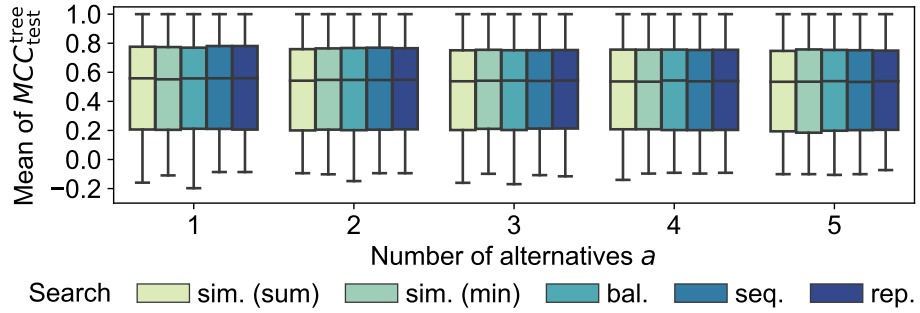
**Quality difference of heuristics** We now look closer at the quality difference between solver-based and heuristic search. Figure 4 compares the mean training feature-set quality within search runs for each experimental setting




(a) Training-set objective value.

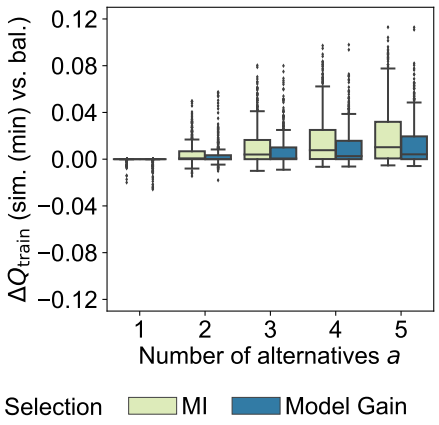



(b) Test-set objective value.

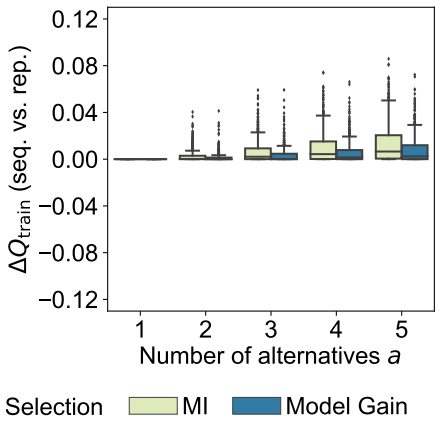



(c) Test-set prediction performance.

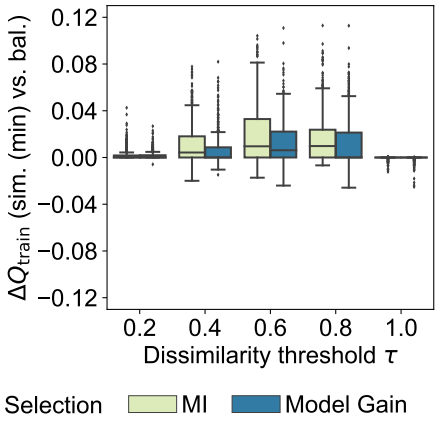
Figure 3: Distribution of mean feature-set quality within search runs over datasets, cross-validation-folds, and $\tau$, by search method for alternatives and number of alternatives $a$. Results with *Model Gain* as the feature-selection method and $k = 5$. Excludes search settings where at least one combination of search method and $a$ yielded no valid solution. Y-axes are truncated to improve readability.



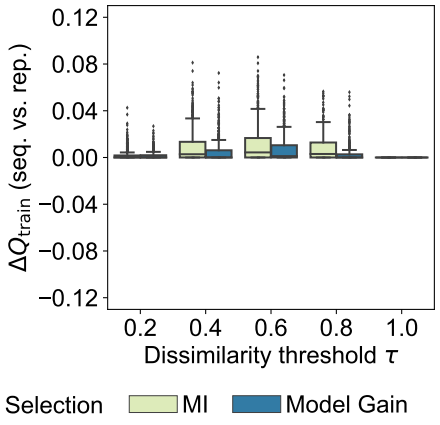

(a) Difference between solver-based simultaneous (min-aggregation) search and *Greedy Balancing*, by the number of alternatives $a$.




(b) Difference between solver-based sequential search and *Greedy Replacement*, by the number of alternatives $a$.




(c) Difference between solver-based simultaneous (min-aggregation) search and *Greedy Balancing*, by the dissimilarity threshold $\tau$.




(d) Difference between solver-based sequential search and *Greedy Replacement*, by the dissimilarity threshold $\tau$.

Figure 4: Distribution of difference in mean training feature-set quality within a search run between solver-based and heuristic search methods over remaining experimental settings, by feature-selection method. Results with $k = 5$. Excludes search settings where at least one combination of search method and $a$ yielded no valid solution.

separately. In particular, we compare solver-based simultaneous search with min-aggregation to *Greedy Balancing* (cf. Figures 4a and 4c) and solver-based sequential search to *Greedy Replacement* (cf. Figures 4b and 4d). Positive values in Figure 4 express that solver-based search yields higher quality; negative values favor heuristic search. The latter can occur if solver-based search yields suboptimal solutions due to timeouts, as we analyze later (cf. Table 3).

Figures 4a and 4b compare solver-based and heuristic search over the number of alternatives $a$. For $a = 1$, solver-based and heuristic search yield the same mean training feature-set quality except when timeouts occur. The more alternatives are desired, the more advantageous a solver-based search is quality-wise. As in prior analyses, the picture is less clear on the test set, which shows a smaller quality difference here. Further, the difference in mean training feature-set quality between *Greedy Balancing* and solver-based simultaneous search (cf. Figure 4a) grows faster with $a$ than between *Greedy Replacement* and solver-based sequential search (cf. Figure 4b). As an explanation, consider that simultaneous search may generally develop a quality advantage over sequential search for more alternatives. *Greedy Balancing* selects the same features as *Greedy Replacement*, i.e., a sequential search heuristic, and only distributes them differently into feature sets, yielding the same mean feature-set quality.

Figures 4c and 4d compare solver-based and heuristic search over the dissimilarity threshold $\tau$. Unlike for $a$, the difference in mean training feature-set quality between solver-based and heuristic search does not increase over the whole range of $\tau$ but shows an increase followed by a decrease. In particular, $\tau = 1$ allows the two heuristic search methods to reach the same mean training feature-set quality as the solver-based methods. This observation corresponds to our theoretical result that optimizing the summed quality of alternatives with $\tau = 1$ admits polynomial-time algorithms (cf. Proposition 11).

**Optimization status** Suboptimal search results are one reason why solver-based simultaneous search does not consistently beat solver-based sequential search quality-wise. For *Greedy Wrapper*, the search is heuristic per se and does not cover the entire search space. For all feature-selection methods, the solver can time out. Table 3 shows that solver-based simultaneous search has a higher likelihood of timeouts than solver-based sequential search, likely due to the larger size of the optimization problem (cf. Table 1). In particular, for up to five alternatives and $k = 5$, all solver-based sequential searches for *FCBF*, *MI*, and *Model Gain* finished within the timeout, i.e., yielded the optimal feature set or ascertained infeasibility, while *mRMR* had about 10% timeouts. In contrast, for solver-based simultaneous search with sum-aggregation, all feature-selection methods experience timeouts: 1–3% of the searches for *FCBF*, *MI*, and *Model Gain*, and 67% of the searches for *mRMR* found a feasible solution but could not prove optimality. Such timeout-affected simultaneous solutions can be worse than optimal sequential solutions.

*mRMR* is especially prone to suboptimal solutions, likely because it has a more complex objective (cf. Equation 14) than *MI* and *Model Gain*. *FCBF* often

Table 3: Frequency of optimization statuses (cf. Section 5.2) over datasets, cross-validation folds, $a$, and $\tau$, by feature-selection method and search method for alternatives. Results with $k = 5$, $a \in \{1, 2, 3, 4, 5\}$, and excluding *Greedy Wrapper*, which does not use the solver for optimizing. Each row adds up to 100%.

| Feature sel. | Search | Optimization status | | | |
|---|---|---|---|---|---|
| | | Infeasible | Not solved | Feasible | Optimal |
| FCBF | seq. | 74.51% | 0.00% | 0.00% | 25.49% |
| FCBF | sim. (min) | 73.07% | 0.00% | 1.60% | 25.33% |
| FCBF | sim. (sum) | 73.07% | 0.00% | 2.19% | 24.75% |
| MI | bal. | 0.00% | 9.20% | 90.80% | 0.00% |
| MI | rep. | 0.00% | 9.20% | 90.80% | 0.00% |
| MI | seq. | 4.93% | 0.00% | 0.00% | 95.07% |
| MI | sim. (min) | 4.67% | 0.00% | 9.33% | 86.00% |
| MI | sim. (sum) | 4.67% | 0.00% | 3.04% | 92.29% |
| Model Gain | bal. | 0.00% | 9.20% | 90.80% | 0.00% |
| Model Gain | rep. | 0.00% | 9.20% | 90.80% | 0.00% |
| Model Gain | seq. | 4.93% | 0.00% | 0.00% | 95.07% |
| Model Gain | sim. (min) | 4.67% | 0.00% | 5.28% | 90.05% |
| Model Gain | sim. (sum) | 4.67% | 0.00% | 1.87% | 93.47% |
| mRMR | seq. | 4.88% | 0.00% | 9.55% | 85.57% |
| mRMR | sim. (min) | 4.67% | 0.00% | 48.64% | 46.69% |
| mRMR | sim. (sum) | 4.67% | 0.00% | 67.04% | 28.29% |

results in infeasible optimization problems since its constraints, which prevent the selection of redundant features (cf. Equation 12), might prevent finding any valid feature set of size $k$. Min-aggregation instead of sum-aggregation in solver-based simultaneous search exhibits more timeouts for *MI* and *Model Gain* but less for *FCBF* and *mRMR*. Still, solver-based sequential search incurs fewer timeouts for all of these four feature-selection methods.

Also, note that the fraction of timeouts in solver-based simultaneous search strongly depends on the number of alternatives $a$, as Table 4 displays: For $k = 5$ and sum-aggregation, roughly 8% of the white-box searches timed out for $a = 1$, but 20% for $a = 3$ and 30% for $a = 5$. While we grant solver-based simultaneous searches proportionally more time for multiple alternatives (cf. Section 5.3.3), the observed increase in timeouts suggests that runtime increases super-proportionally with $a$, as we analyze later.

For the heuristic search methods, Table 3 shows that *Greedy Replacement* more often did not find a valid alternative (9.20%) than solver-based sequential search (4.93%). A similar phenomenon occurred for *Greedy Balancing* (9.20%) compared to solver-based simultaneous search (4.67%). Such a result can be expected since both heuristic search methods stop early as soon as each feature is part of at least one alternative.

Table 4: Frequency of optimization statuses (cf. Section 5.2) over datasets, cross-validation folds, feature-selection methods, and $\tau$, by number of alternatives $a$. Results from solver-based simultaneous search with sum-aggregation, $k = 5$, and excluding *Greedy Wrapper*. Each row adds up to 100%.

| $a$ | Optimization status | | |
|---|---|---|---|
| | Infeasible | Feasible | Optimal |
| 1 | 16.10% | 7.60% | 76.30% |
| 2 | 17.50% | 13.27% | 69.23% |
| 3 | 20.00% | 20.20% | 59.80% |
| 4 | 27.00% | 21.43% | 51.57% |
| 5 | 28.23% | 30.17% | 41.60% |

Table 5: Mean optimization time over datasets, cross-validation folds, $a$, and $\tau$, by feature-selection method and search method for alternatives. Results with $k = 5$ and $a \in \{1, 2, 3, 4, 5\}$.

| Feature selection | Optimization time | | | | |
|---|---|---|---|---|---|
| | Bal. | Rep. | Seq. | Sim. (min) | Sim. (sum) |
| FCBF | — | — | 0.22 s | 11.41 s | 12.62 s |
| Greedy Wrapper | — | — | 52.62 s | 68.39 s | 70.36 s |
| MI | 0.00 s | 0.00 s | 0.03 s | 47.39 s | 24.56 s |
| Model Gain | 0.00 s | 0.00 s | 0.03 s | 30.38 s | 19.08 s |
| mRMR | — | — | 33.59 s | 156.00 s | 189.25 s |

**Optimization time** As Table 5 shows, solver-based sequential search is faster on average than solver-based simultaneous search for all five feature-selection methods. In particular, the difference is up to three orders of magnitude for the four white-box feature-selection methods. Further, *FCBF*, *MI*, and *Model Gain* experience a dramatic increase in optimization time with the number of alternatives $a$ in solver-based simultaneous search, as Table 6 displays. In contrast, the runtime increase is considerably less for solver-based sequential search, which shows an approximately linear trend with the number of alternatives.

Table 5 also shows that the optimization time of the heuristic search methods for *MI* and *Model Gain* is negligible. In particular, *Greedy Replacement* and *Greedy Balancing* never took longer than 1 ms per search run for alternatives. These results highlight the runtime advantage of the heuristics, particularly of *Greedy Balancing* for simultaneous search.

An interesting question for practitioners is how the runtime relates to $n$, the number of features in the dataset. One could expect a positive correlation since the problem instance increases with $n$. Roughly speaking, this trend appears in our experimental data indeed. However, the observed trend is rather noisy, particularly for solver-based simultaneous search, and some higher-dimensional

Table 6: Mean optimization time over datasets, cross-validation folds, and $\tau$, by number of alternatives and feature-selection method. Results from solver-based simultaneous search with sum-aggregation and $k = 5$.

| $a$ | Optimization time | | | | |
|---|---|---|---|---|---|
| | FCBF | Greedy Wrapper | MI | Model Gain | mRMR |
| 1 | 0.45 s | 28.44 s | 0.03 s | 0.02 s | 44.68 s |
| 2 | 0.86 s | 41.76 s | 0.09 s | 0.08 s | 117.62 s |
| 3 | 2.97 s | 62.70 s | 0.30 s | 0.27 s | 208.14 s |
| 4 | 13.32 s | 96.65 s | 3.68 s | 3.47 s | 258.19 s |
| 5 | 45.52 s | 122.26 s | 118.72 s | 91.58 s | 317.63 s |

datasets even show lower average runtimes than lower-dimensional ones. This result indicates that other factors than $n$ influence runtime as well, e.g., other experimental settings or the solver's internal search heuristics.

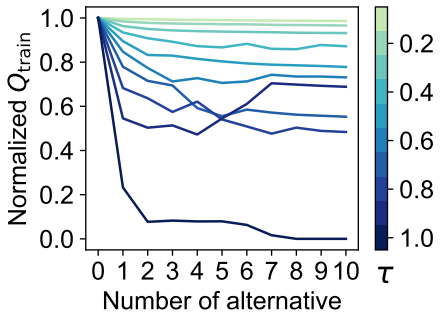
Based on all results described in this section, we focus on solver-based sequential search in the next section. In particular, it was significantly faster than solver-based simultaneous search while yielding similar feature-set quality.

## 6.3 User Parameters $a$ And $\tau$

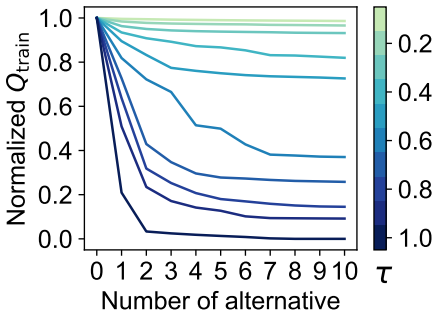
**Feature-set quality** Higher values of the two user parameters introduce more (for $a$) or stronger (for $\tau$) constraints into the optimization problem of alternative feature selection. Thus, one would expect a corresponding decrease in feature-set quality. Figure 5 illustrates this trend for *Model Gain* as the feature-selection method and $k = 10$. Since the maximum feature-set quality varies among datasets, we max-normalize quality in this figure. In particular, we set the highest feature-set quality in each search run for alternatives to 1 and scale the other feature-set qualities accordingly. For prediction performance in terms of MCC, we shift its range from $[-1, 1]$ to $[0, 1]$ before normalization.

Figure 5 shows that multiple alternatives may have a similar quality. Further, the training-set objective value (cf. Figure 5a) decreases most from the original feature set, i.e., the zeroth alternative, to the first alternative, but less beyond. Also, the decrease strongly depends on the dissimilarity threshold $\tau$. For a low dissimilarity threshold like $\tau = 0.1$, the training-set objective value barely drops over the number of alternatives. Additionally, note that Figure 5 averages the normalized feature-set quality over multiple datasets. In our experiments, datasets with more features tend to experience a smaller decrease in quality over $a$ and $\tau$. As higher-dimensional datasets offer more options for alternatives, this observation makes sense. However, this effect is not guaranteed since datasets with many features could also contain many useless features instead of interesting alternatives.

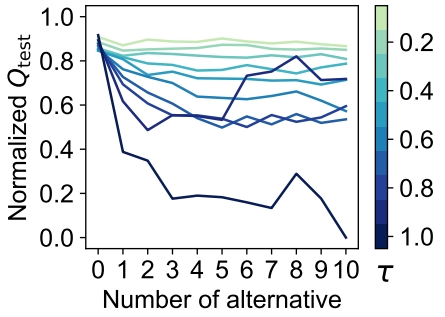
The overall decrease in quality is slightly less pronounced for the test-set objective value (Figure 5c) than on the training set (Figure 5a) since overfitting



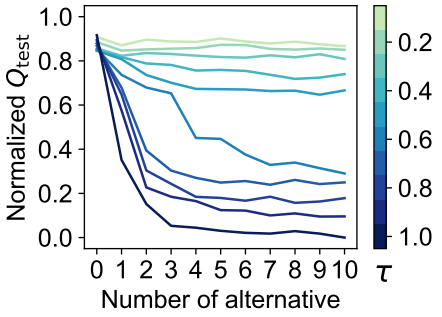

(a) Training-set objective value. Infeasible feature sets excluded.



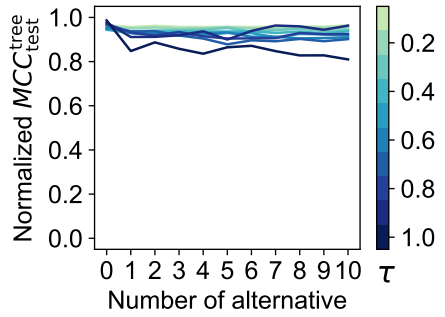

(b) Training-set objective value. Infeasible feature sets assigned a quality of 0.



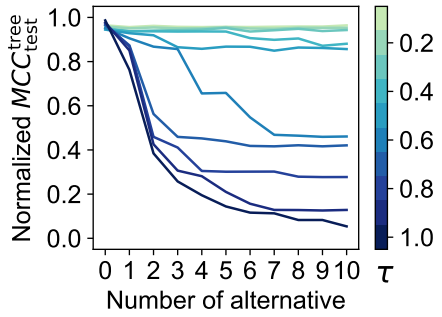

(c) Test-set objective value. Infeasible feature sets excluded.




(d) Test-set objective value. Infeasible feature sets assigned a quality of 0.

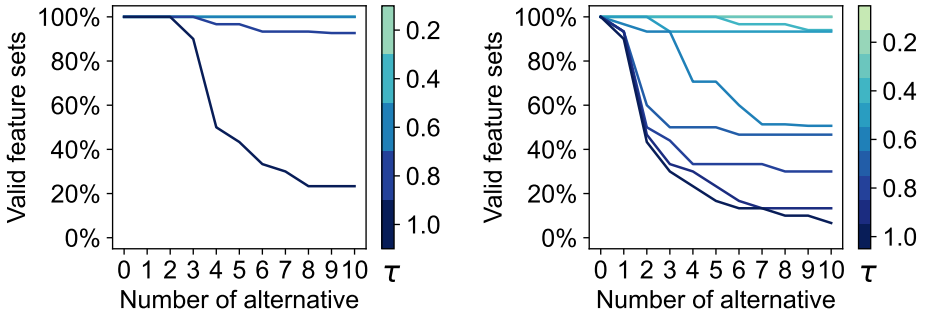



(e) Test-set prediction performance. Infeasible feature sets excluded.




(f) Test-set prediction performance. Infeasible feature sets assigned a quality of 0.

Figure 5: Mean feature-set quality over datasets and cross-validation folds, max-normalized per search run for alternatives, by the number of alternative and dissimilarity threshold $\tau$. Results from solver-based sequential search with *Model Gain* as the feature-selection method and $k = 10$.




(a) Feature-set size $k = 5$.

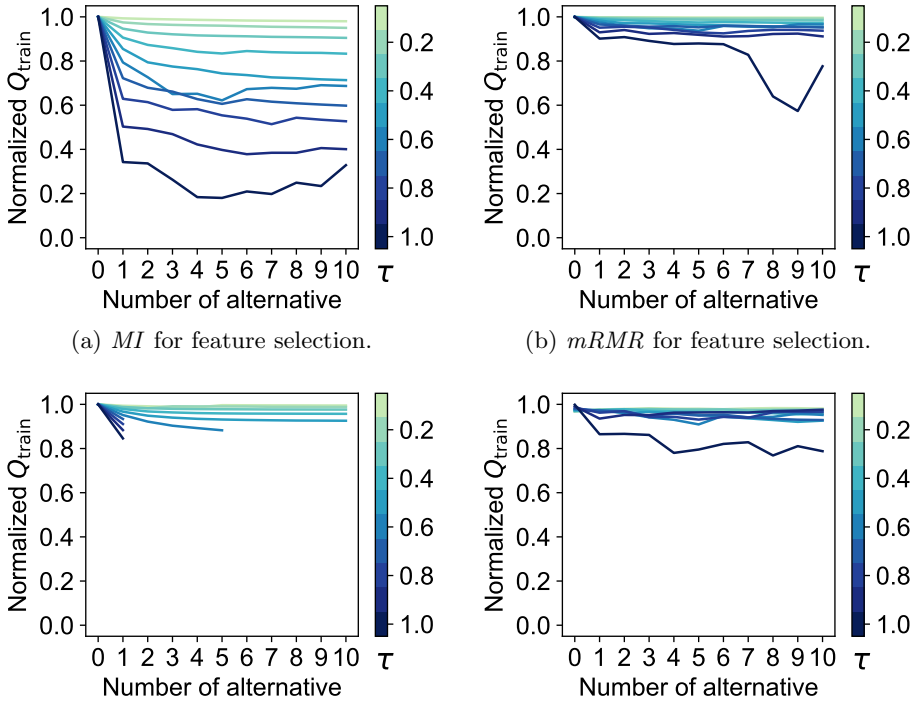
(b) Feature-set size $k = 10$.

Figure 6: Frequency of optimization runs yielding a valid feature set over datasets and cross-validation folds, by the number of alternative and dissimilarity threshold $\tau$. Results from solver-based sequential search with *Model Gain* as the feature-selection method.

might occur. In particular, the original feature set can even have lower test-set quality than the subsequent alternatives. The trend becomes even less clear for prediction performance, which varies little over $a$ and $\tau$ in our experiments (cf. Figure 5e). In general, the optimization objective $Q$ may only partially indicate actual prediction performance since the former may use a simplified feature-set quality criterion. Indeed, the correlation between optimization objective $Q$ and prediction MCC is only weak to moderate in our experiments.

**Optimization status** The previous observations refer to the quality of the found feature sets. However, the more alternatives one desires and the more they should differ, the likelier an infeasible optimization problem is. Figure 6 visualizes the fraction of valid feature sets over the number of alternatives and dissimilarity threshold $\tau$, showing the expected trend. Additionally, Figures 5b, 5d, and 5f display the same data as Figures 5a, 5c, and 5e but with the quality of infeasible feature sets set to zero instead of excluding these feature sets from evaluation. Consequently, the decrease in feature-set quality over $a$ and $\tau$ is noticeably stronger. In contrast, if only considering valid feature sets, the mean quality in our experiments can increase over the number of alternatives, e.g., as visible in Figures 5a and 5c for $\tau = 0.9$. This counterintuitive phenomenon can occur because some datasets run out of valid feature sets sooner than others, so the average quality may be determined for different sets of datasets at each number of alternatives.

**Influence of feature-selection method** While we discussed *Model Gain* before, the decrease in objective value over $a$ and $\tau$ occurs to different extents for the other feature-selection methods in our experiments, as Figure 7 displays. In this figure, we shifted the objectives of *Greedy Wrapper* and *mRMR* to $[0, 1]$ before max-normalization since their original range is $[-1, 1]$. *MI* (cf. Figure 7a)




(a) *MI* for feature selection.

(b) *mRMR* for feature selection.

(c) *FCBF* for feature selection.

(d) *Greedy Wrapper* for feature selection.

Figure 7: Mean training-set objective value over datasets and cross-validation folds, max-normalized per search run for alternatives, by the number of alternative and dissimilarity threshold $\tau$. Results from solver-based sequential search with $k = 10$. Infeasible feature sets excluded.

shows a similar behavior as *Model Gain* (cf. Figure 5a), which may result from both feature-selection methods using the same objective function, though with different feature qualities. In contrast, *mRMR* (cf. Figure 7b) exhibits a considerably smaller effect of increasing $\tau$. For *FCBF* (cf. Figure 7c), the additional constraints on feature-feature correlation (cf. Equation 12) cause many infeasible results (cf. Table 3), so we cannot determine the average feature-set quality for some combinations of $a$ and $\tau$. For *Greedy Wrapper* (cf. Figure 7d), $a$ and $\tau$ barely make any difference, which may be explained by the heuristic, inexact search procedure.

## 6.4 Summary

**Feature-selection methods (cf. Section 6.1)** Among the feature-selection methods, *Model Gain* yielded the best average prediction performance. The simple univariate *MI* also turned out competitive, while *Greedy Wrapper* and *mRMR* required high optimization times, and our constraint-based version of

*FCBF* yielded many infeasible solutions. Selecting $k = 10$ instead of $k = 5$ features had only a small impact on prediction performance, so users may stick to smaller feature-set sizes if such a setting benefits interpretability.

**Search methods for alternatives (cf. Section 6.2)** Solver-based simultaneous search, particularly with min-aggregation, considerably reduced the variance of the training-set objective value over alternatives compared to solver-based sequential search, as we desired. However, results were less clear on the test set and when using prediction performance to measure feature-set quality. Further, the average quality of alternatives was similar to solver-based sequential search. In addition, the latter was considerably faster and led to fewer solver timeouts, particularly when increasing the number of alternatives. Also, sequential search allows users to stop searching after each alternative.

The heuristic search methods *Greedy Replacement* and *Greedy Balancing* for univariate feature qualities achieved a good feature-set quality relative to solver-based search, particularly for a low number of alternatives and on the test set. For a high number of alternatives, the training feature-set quality may differ more, and the heuristics may stop early despite the existence of further alternatives. As a positive point, both the heuristics' runtime was negligible. Also, *Greedy Balancing* achieved a low variance of training-set objective value between alternatives, similar to solver-based simultaneous search with min-aggregation.

**User parameters $a$ and $\tau$ (cf. Section 6.3)** Feature-set quality tended to decrease with a larger number of alternatives $a$ and dissimilarity threshold $\tau$, so these parameters give users control over alternatives. The decrease was highest from the original feature set to the first alternative but smaller beyond, resulting in multiple alternatives of similar quality. Also, the decrease was more prominent on the training set than on the test set. Further, the strength of this decrease depended on the feature-selection method; *MI* and *Model Gain* showed the largest effect. Independent from the feature-selection method, the frequency of infeasible solutions increased with $a$ and $\tau$ due to stronger constraints.

# 7 Conclusions and Future Work

In this section, we summarize our work (cf. Section 7.1) and give an outlook on potential future work (cf. Section 7.2).

## 7.1 Conclusions

Feature-selection methods are a valuable tool to foster interpretable predictions. Conventional feature-selection methods typically yield only one feature set. However, users may be interested in obtaining multiple, sufficiently diverse feature sets of high quality. Such alternative feature sets may provide alternative explanations for predictions from the data.

In this article, we defined alternative feature selection as an optimization problem. We formalized alternatives via constraints that are independent of the feature-selection method, can be combined with other constraints on feature sets, and allow users to control diversity according to their needs with two parameters, i.e., the number of alternatives $a$ and a dissimilarity threshold $\tau$. Further, we discussed how to integrate different categories of conventional feature-selection methods as objectives. We also analyzed the complexity of this optimization problem and proved $\mathcal{NP}$-hardness, even for simple notions of feature-set quality. Additionally, we showed that the problem gives way to a constant-factor approximation under certain conditions, and we proposed corresponding heuristic search methods. Finally, we evaluated alternative feature selection with 30 classification datasets and five feature-selection methods. We compared sequential and simultaneous search for alternatives, both with solver-based and heuristic search methods, and varied the number of alternatives as well as the dissimilarity threshold for alternatives.

## 7.2 Future Work

**Feature selection (objective function)** One could search for alternatives with other feature-selection methods than the five we analyzed. In particular, we implemented only one procedure to find alternatives for wrapper feature selection (cf. Section 3.3.2). Embedded feature selection, which we did not evaluate, would also need adapted search methods for alternatives (cf. Section 3.3.3).

**Alternatives (constraints)** One could vary the definition of alternatives, e.g., the set-dissimilarity measure (cf. Section 3.2.1), the quality aggregation for simultaneous alternatives (cf. Appendix A.1), or the overall optimization problem (cf. Section 3.1). While we made general and straightforward decisions for each of these points, particular applications might demand other formalizations of alternatives. E.g., one could use soft instead of hard constraints.

**Time complexity** Appendix A.5.4 discusses how one could extend our complexity analysis of alternative feature selection (cf. Section 3.4).

**Runtime** Our experiments (cf. Section 6.2) and theoretical analyses (cf. Section 3.2.2) revealed that exact simultaneous search scales poorly with the number of alternatives. One could conceive a more efficient problem formulation. Further, one could limit the solver runtime and take the intermediate results once the timeout is reached. We already used a fixed timeout in our experiments, but studying the exact influence of timeouts on feature-set quality is an open topic. Next, one could use a different solver, e.g., one for non-linear optimization, so the auxiliary variables from Equation 6 become superfluous. Finally, one could develop further simultaneous heuristics (cf. Section 3.5.2).

**Datasets** In the current article, we conducted a broad quantitative evaluation of alternative feature selection on datasets from various domains. (cf. Section 5.4). While we uncovered several general trends, the existence and quality of alternatives naturally depend on the dataset. Thus, practitioners may employ alternative feature selection in domain-specific case studies and evaluate the alternative feature sets qualitatively, thereby assessing their usefulness for interpreting predictions.

# A Appendix

In this section, we provide supplementary materials. Section A.1 discusses possible aggregation operators for the objective of the simultaneous-search problem (cf. Definition 5). Section A.2 discusses additional objective functions for multivariate filter feature selection (cf. Section 3.3.1). Section A.3 provides complete definitions of the alternative-feature-selection problem (cf. Section 3.2) for the univariate objective (cf. Equation 11). Section A.4 proposes how to speed up optimization for the univariate objective (cf. Equation 11). Section A.5 complements the complexity analysis (cf. Section 3.4). Section A.6 proposes another heuristic search method for the univariate objective (cf. Equation 11), complementing Section 3.5.

## A.1 Aggregation Operators for the Simultaneous-Search Problem

In this section, we discuss operators to aggregate the feature-set quality of multiple alternatives in the objective of the simultaneous-search problem (cf. Definition 5).

**Sum-aggregation** The arguably simplest way to aggregate the qualities of multiple feature sets is to sum them up, which we call *sum-aggregation*:

$$\max_{s^{(0)},\ldots,s^{(a)}} \sum_{l=0}^{a} Q(s^{(l)}, X, y) \tag{17}$$

While this objective fosters a high average quality of feature sets, it does not guarantee that the alternatives have similar quality:

**Example 5** (Sum-aggregation)**.** Consider $n = 6$ features with univariate feature qualities (cf. Equation 11) $q = (9, 8, 7, 3, 2, 1)$, feature-set size $k = 3$, number of alternatives $a = 2$, the Dice dissimilarity (cf. Equation 3) as $d(\cdot)$, and dissimilarity threshold $\tau = 0.5$, which permits an overlap of one feature between sets here (cf. Equation 8). Exact sequential search (cf. Definition 3) yields the selection $s^{(0)} = (1, 1, 1, 0, 0, 0)$, $s^{(1)} = (1, 0, 0, 1, 1, 0)$, and $s^{(2)} = (0, 1, 0, 1, 0, 1)$, with a summed quality of $24 + 14 + 12 = 50$. One possible exact simultaneous-search (cf. Definition 5) solution consists of the feature sets $s^{(0)} = (1, 1, 0, 1, 0, 0)$,

$s^{(1)} = (1, 0, 1, 0, 1, 0)$, and $s^{(2)} = (0, 1, 1, 0, 0, 1)$, with a summed quality of $20 + 18 + 16 = 54$. Another possible exact simultaneous-search solution is $s^{(0)} = (1, 1, 0, 0, 0, 1)$, $s^{(1)} = (1, 0, 1, 0, 1, 0)$, and $s^{(2)} = (0, 1, 1, 1, 0, 0)$, with a summed quality of $18 + 18 + 18 = 54$.

This example allows several insights. First, exact sequential search yields worse quality than exact simultaneous search here, i.e., 50 vs. 54. Second, the feature-set qualities of the sequential solution, i.e., 24, 14, and 12, differ significantly. Third, exact simultaneous search can yield multiple solutions whose feature-set quality is differently balanced. Here, the feature-set qualities in the second simultaneous-search solution, i.e., 18, 18, and 18, are more balanced than in the first, i.e., 20, 18, and 16. However, both solutions are equally optimal for sum-aggregation.

**Min-aggregation** To actively foster balanced feature-set qualities in simultaneous search, we propose *min-aggregation* in the objective:

$$\max_{s^{(0)},\ldots,s^{(a)}} \min_{l \in \{0,\ldots,a\}} Q(s^{(l)}, X, y) \tag{18}$$

In the terminology of social choice theory, this objective uses an egalitarian rule instead of a utilitarian one [81]. In particular, *min-aggregation* maximizes the quality of the worst selected alternative. Thereby, it incentivizes all alternatives to have high quality and implicitly balances their quality.

Note that optimizing the objective with either sum-aggregation or min-aggregation does not necessarily optimize the other. We already showed a solution optimizing sum-aggregation but not min-aggregation (cf. Example 5). In the following, we demonstrate the other direction:

**Example 6** (Min-aggregation)**.** Consider $n = 6$ features with univariate feature qualities (cf. Equation 11) $q = (11, 10, 6, 5, 4, 1)$, feature-set size $k = 3$, number of alternatives $a = 1$, the Dice dissimilarity (cf. Equation 3) as $d(\cdot)$, and dissimilarity threshold $\tau = 0.5$, which permits an overlap of one feature between sets here (cf. Equation 8). One simultaneous-search (cf. Definition 5) solution with min-aggregation (cf. Equation 18) is $s^{(0)} = (1, 1, 0, 0, 1, 0)$ and $s^{(1)} = (1, 0, 1, 1, 0, 0)$, with a summed quality of $25 + 22 = 47$. Another solution is $s^{(0)} = (1, 1, 0, 0, 0, 1)$ and $s^{(1)} = (1, 0, 1, 1, 0, 0)$, with a summed quality of $22 + 22 = 44$.

While both solutions have the same minimum feature-set quality, only the first solution optimizes the objective with sum-aggregation. In particular, min-aggregation permits reducing the quality of feature sets as long as the latter remains above the minimum quality of all sets.

From the technical perspective, Equation 18 has the disadvantage of being non-linear regarding the decision variables $s^{(0)}, \ldots, s^{(a)}$. However, we can

linearize it with one constraint per feature set and an auxiliary variable $Q_{\min}$:

$$\begin{aligned} \max_{s^{(0)},\ldots,s^{(a)}} \quad & Q_{\min} \\ \text{subject to:} \quad \forall l \in \{0,\ldots,a\}: \quad & Q_{\min} \leq Q(s^{(l)}, X, y) \\ & Q_{\min} \in \mathbb{R} \end{aligned} \tag{19}$$

As we maximize $Q_{\min}$, this variable will implicitly assume the actual minimum value of $Q(s^{(l)}, X, y)$ with equality since the solution would not be optimal otherwise. This situation relieves us from introducing further auxiliary variables that are usually necessary when linearizing maximum or minimum expressions [76].

**Further approaches for balancing quality** Min-aggregation provides no control or guarantee of how much the feature-set qualities will actually differ between alternatives since it only incentivizes high quality for all sets. One can alleviate this issue by adapting the objective or constraints. First, related work on Multi-Way Number Partitioning also uses other objectives for balancing [60, 64] (cf. Section A.5.2). E.g., one could minimize the difference between maximum and minimum feature-set quality. Second, one could use sum-aggregation but constrain the minimum or maximum quality of sets, or the difference between the qualities. However, such constraint-based approaches introduce one or several parameters bounding feature-set quality, which are difficult to determine a priori. Third, one could treat balancing qualities as another objective besides maximizing the summed quality. One can then optimize two objectives simultaneously, filtering results for Pareto-optimal solutions or optimizing a weighted combination of the two objectives. In both cases, users may need to define an acceptable trade-off between the objectives. It is an open question if a solution that jointly optimizes min- and sum-aggregation always exists. If yes, then optimizing a weighted combination of the two objectives would also optimize each of them on its own, assuming positive weights.

## A.2 Further Objectives for Multivariate Filter Methods

While Section 3.3.1 already addressed FCBF and mRMR as multivariate filter feature-selection methods, we discuss the objectives of CFS and Relief here.

**CFS** Correlation-based Feature Selection (CFS) [43, 44] follows a similar principle as mRMR but uses the ratio instead of the difference between a relevance term and a redundancy term for feature-set quality. Using a bivariate dependency measure $q(\cdot)$ to quantify correlation, the objective is as follows:

$$Q_{\text{CFS}}(s, X, y) = \frac{\sum_{j=1}^{n} q(X_{\cdot j}, y) \cdot s_j}{\sqrt{\sum_{j=1}^{n} s_j + \sum_{j_1=1}^{n} \sum_{\substack{j_2=1 \\ j_2 \neq j_1}}^{n} q(X_{\cdot j_1}, X_{\cdot j_2}) \cdot s_{j_1} \cdot s_{j_2}}} \tag{20}$$

One can square this objective to remove the square root in the denominator [85]. Nevertheless, the objective remains non-linear in the decision variables $s$ since

it involves a fraction and multiplications between variables. However, one can linearize the objective with additional variables and constraints [84, 85], allowing to formulate alternative feature selection for CFS as a linear problem.

**Relief** Relief [57, 96] builds on the idea that data objects with a similar value of the prediction target should have similar feature values, but data objects that differ in their target should differ in their feature values. Relief assigns a score to each feature by sampling data objects and quantifying the difference in feature values and target values compared to their nearest neighbors. We deem Relief to be multivariate since the nearest-neighbor computations involve all features instead of considering them independently. However, the resulting feature scores can directly be put into the univariate objective (cf. Equation 11) to obtain a linear problem. One can also use Relief scores in CFS to consider feature redundancy [43, 44], which the default Relief does not.

## A.3 Complete Specifications of the Optimization Problem for the Univariate Objective

In this section, we provide complete specifications of the alternative-feature-selection problem for sequential and simultaneous search as integer linear problem. In particular, we combine all relevant definitions and equations from Section 3. We use the objective of univariate filter feature selection (cf. Equation 11). The corresponding feature qualities $q(\cdot)$ are constants in the optimization problem. Further, we use the Dice dissimilarity (cf. Equation 8) to measure feature-set dissimilarity for alternatives. The dissimilarity threshold $\tau \in [0, 1]$ is a user-defined constant. Finally, we assume fixed, user-defined feature-set sizes $k \in \mathbb{N}$.

**Sequential-search problem** In the sequential case (cf. Definition 3 and Equation 9), only one feature set $F_s$ is variable in the optimization problem, while the existing feature sets $F_{\bar{s}} \in \mathbb{F}$ with their selection vectors $\bar{s}$ are constants.

$$\begin{aligned} \max_{s} \quad & Q_{\text{uni}}(s, X, y) = \sum_{j=1}^{n} q(X_{\cdot j}, y) \cdot s_j \\ \text{subject to:} \quad \forall F_{\bar{s}} \in \mathbb{F}: \quad & \sum_{j=1}^{n} s_j \cdot \bar{s}_j \leq (1 - \tau) \cdot k \\ & \sum_{j=1}^{n} s_j = k \\ & s \in \{0, 1\}^n \end{aligned} \tag{21}$$

**Simultaneous-search problem** In the simultaneous case (cf. Definition 5 and Equation 10), all feature sets are variable. $a \in \mathbb{N}_0$ denotes the number of alternatives, which corresponds to the number of feature sets minus one.

Next, we introduce auxiliary variables to linearize products between variables (cf. Equation 6). Finally, we use sum-aggregation (cf. Equation 17) over alternatives in the objective here.

$$
\begin{aligned}
\max_{s^{(0)},\ldots,s^{(a)}} \quad & \sum_l Q_{\text{uni}}(s^{(l)}, X, y) = \sum_l \sum_j q(X_{\cdot j}, y) \cdot s_j^{(l)} \\
\text{subject to:} \quad \forall l_1\ \forall l_2: \quad & \sum_j t_j^{(l_1,l_2)} \leq (1-\tau) \cdot k \\
\forall l_1\ \forall l_2\ \forall j: \quad & t_j^{(l_1,l_2)} \leq s_j^{(l_1)} \\
\forall l_1\ \forall l_2\ \forall j: \quad & t_j^{(l_1,l_2)} \leq s_j^{(l_2)} \\
\forall l_1\ \forall l_2\ \forall j: \quad & 1 + t_j^{(l_1,l_2)} \geq s_j^{(l_1)} + s_j^{(l_2)} \\
\forall l: \quad & \sum_j s_j^{(l)} = k \\
\forall l: \quad & s^{(l)} \in \{0,1\}^n \\
\forall l_1\ \forall l_2: \quad & t^{(l_1,l_2)} \in \{0,1\}^n \\
\text{with indices:} \quad & l \in \{0,\ldots,a\} \\
& l_1 \in \{1,\ldots,a\} \\
& l_2 \in \{0,\ldots,l_1-1\} \\
& j \in \{1,\ldots,n\}
\end{aligned}
\tag{22}
$$

## A.4 Pre-Selection for the Univariate Objective

In this section, we describe how to potentially speed up the optimization of the univariate objective (cf. Equation 11) by *pre-selection* if the user-defined feature-set sizes $k$ and the number of alternatives $a$ are small.

The univariate objective is monotonic in the features' qualities $q(X_{\cdot j}, y)$ and the selection decisions $s_j$. In particular, the objective is non-decreasing when replacing a feature with another one of higher quality. Further, unless some feature qualities are negative, selecting more features does not decrease the objective. Sum-aggregation (cf. Equation 17) and min-aggregation (cf. Equation 18) for the simultaneous-search problem are monotonic as well.

Thus, assuming $(a+1) \cdot k < n$, it suffices to use the $(a+1) \cdot k$ highest feature qualities when searching for an optimal solution out of $a+1$ feature sets. Due to monotonicity, the remaining feature qualities cannot improve the objective, so one can drop them before optimization. We call this step *pre-selection*. While there might also be optimal solutions using the dropped features, their objective value cannot be higher than with pre-selection. For example, such solutions can arise in case of multiple identical qualities or for min-aggregation in the objective (cf. Example 6). Also, the optimal solution might not contain all pre-selected features, i.e., pre-selection over-approximates the set of selected features.

One can conduct pre-selection before using a solver or any other search mechanism, e.g., exhaustive search. The latter generally has polynomial runtime

regarding $n$ assuming small, constant $a$ and $k$, i.e., $a \cdot k \in O(1)$ (cf. Section 3.4.1). With pre-selection, the pure search cost would even become independent from $n$, i.e., $O(1)$ under that assumption. However, one would need to determine the highest feature qualities first, e.g., by sorting all qualities in $O(n \cdot \log n)$ or iteratively determining the maximum quality in $O((a+1) \cdot k \cdot n)$.

## A.5 Time Complexity

In this section, we provide details for our analysis of time complexity (cf. Section 3.4). In particular, we discuss a special case of exhaustive simultaneous search (cf. Section A.5.1), outline related work (cf. Section A.5.2), provide proofs (cf. Section A.5.3), and describe future work (cf. Section A.5.4).

### A.5.1 A Special Case of Exhaustive Simultaneous Search

The complexity of exhaustive simultaneous search is lower than in Proposition 4 for the special case $0 < \tau \cdot k \leq 1$, i.e., if feature sets need to differ in only one feature. There, each feature set is an alternative to each other unless both sets are identical. Thus, each set of $a+1$ distinct feature sets constitutes a valid solution, and further constraint checking is unnecessary. Hence, instead of iterating over sets of feature sets, one can iterate over individual feature sets and maintain a buffer containing the $a+1$ feature sets with the highest quality. For each feature set iterated over, one needs to determine if its quality is higher than the lowest feature-set quality in the buffer and replace it if yes. This procedure has a runtime of $O((a+1) \cdot n^k)$ without the cost of evaluating the objective. I.e., unlike in Proposition 4, the number of alternatives $a$ is not part of the exponent anymore, and the cost corresponds to the search for one feature set times the cost of updating the buffer. For large $a$, one can implement the buffer as a heap, thereby reducing the linear factor regarding $a$ to a logarithmic one.

### A.5.2 Related Work

In this section, we discuss related work on $\mathcal{NP}$-hard problems that resemble alternative feature-selection with univariate feature qualities (cf. Equation 11), providing background for Section 3.4.2.

**Integer programming** The univariate objective and several other feature-selection methods allow us to phrase alternative feature selection as a 0-1 integer linear program (cf. Section 3.3.1). Integer Programming is $\mathcal{NP}$-complete in general, even for binary decision variables [35, 51]. Thus, alternative feature selection with a white-box objective suitable for Integer Programming resides in $\mathcal{NP}$. However, it could still be easier since alternative feature selection only uses particular constraint types instead of expressing arbitrary integer linear problems. Vice versa, the membership in $\mathcal{NP}$ based on Integer Programming assumes a particular encoding of alternative feature selection, i.e., each constraint is stored separately and counts towards the problem's input size. If

we instead define the input size only as the number of features $n$ or the total encoding length of the objective function plus parameters $a$, $k$, and $\tau$, the problem could be harder than $\mathcal{NP}$, e.g., for a high number of alternatives. In particular, increasing the number of alternatives would increase the encoding length logarithmically but the cost of constraint checking quadratically.

**Multi-way number partitioning / multiprocessor scheduling** The literature provides different formulations of Multi-Way Number Partitioning and Multiprocessor Scheduling. In particular, different objectives formalize the notion of balanced subset sums and can lead to different optimal solutions [60, 64]. The maximin formulation we use for min-aggregation in the simultaneous-search problem (cf. Definition 5 and Equation 18) is one such notion.

There are several exact algorithms to solve Multi-Way Number Partitioning, e.g., using branch-and-bound approaches that might have exponential runtime [45, 103, 117]. For a fixed number of partitions, the problem is weakly $\mathcal{NP}$-complete since it admits pseudo-polynomial algorithms [35, 59]. Such algorithms run in polynomial time if the input numbers are bounded to a particular size known in advance. Since our feature qualities typically are real numbers, one would need to scale and discretize them to apply such an algorithm. Also, for an arbitrary number of partitions, the problem is strongly $\mathcal{NP}$-complete, so no pseudo-polynomial algorithm can exist unless $\mathcal{P} = \mathcal{NP}$ [35].

However, $\mathcal{NP}$-completeness does not exclude the existence of approximation routines that run in polynomial time and have a guaranteed quality relative to the optimal solution. For example, [1, 27, 122] present such algorithms for the maximin formulation of Multi-Way Number Partitioning, which corresponds to our objective with min-aggregation. In particular, [1, 122] describe polynomial-time approximation schemes (PTAS), which can provide a solution arbitrarily close to the optimum. However, the runtime depends on the desired approximation ratio and can grow exponentially the more precision is desired. Unless $\mathcal{P} = \mathcal{NP}$, the strong $\mathcal{NP}$-completeness of the problem prevents the existence of a fully polynomial-time approximation scheme (FPTAS), which would only polynomially depend on the precision of approximation [1, 122]. However, an FPTAS does exist for each fixed number of partitions [101]. Further, besides approximations, the problem also has polynomial-time exact algorithms if certain parameters of the problem are fixed, e.g., the number of unique numbers to be partitioned or the largest number [73]. Thus, the problem is fixed-parameter tractable ($\mathcal{FPT}$) for an appropriate definition of 'parameter'.

**Balanced number partitioning / k-partitioning** While the previous approaches considered sets of arbitrary sizes, there are number-partitioning problems with constrained $k$ as well, e.g., called Balanced Number Partitioning or K-Partitioning. The problem formulations differ in their objective and cardinality constraints, e.g., if equalities or inequalities are used.

For the minimax objective, [4, 72, 127] propose heuristic algorithms, some

with approximation guarantees. [4] also provides a bound of the objective value relative to the unconstrained case. Further, there is a PTAS for each fixed set size $k$ [72]. Finally, the problem exhibits a polynomial-time exact algorithm for $k = 2$ [25, 26] and an FPTAS for $k = n/2$ [121].

One can also loosen the cardinality constraints by requiring $\leq k$ instead of $= k$. Further, the cardinality $k$ might vary between partitions. This generalized problem is strongly $\mathcal{NP}$-hard but has heuristics running in polynomial time [52]. In particular, [20] provides an efficient PTAS (EPTAS).

As another problem formulation, [21, 46, 65] use a maximin objective as we do. This objective was rarely addressed in combination with cardinality constraints in the literature [65]. Also, all these three references use $\leq k$ constraints instead of $= k$. Again, this problem is strongly $\mathcal{NP}$-hard [46], but [21, 46, 65] propose approximation algorithms, partly with quality guarantees.

**Other partitioning problems** There are other $\mathcal{NP}$-complete problems that partition elements into non-overlapping subsets [35]. E.g., Partition [51] asks if one can partition a set of elements with positive integer weights into two subsets with the same subset sum. 3-Partition [35] demands a partitioning into three-element subsets with an identical, predefined subset sum of the elements' positive integer weights. In contrast to these two problems, we do not require alternative feature sets to have the same quality.

**Bin covering** Bin Covering [3] distributes elements with individual weights into bins such that the number of bins is maximal and the summed weights in each bin surpass a predefined limit. [64] noted a relationship between Multi-Way Number Partitioning and Bin Covering, which may improve solution approaches for either problem [116, 117]. In our case, we could maximize the number of alternatives such that each feature set's quality exceeds a threshold.

**Multiple knapsack** The simultaneous-search problem with sum-aggregation, $\tau = 1$, and univariate feature qualities is a special case of the Multiple Knapsack problem [19]. The latter involves knapsacks, i.e., sets with individual capacities, and elements with individual weights and profits. The goal is to assign elements to knapsacks such that the summed profit of selected elements is maximal. Each element can be assigned to at most one knapsack, and the weights of all elements in the knapsack must not violate its capacity. This problem is strongly $\mathcal{NP}$-complete in general, though it exhibits a PTAS [19]. However, our problem is a special case where the feature qualities act as profits, the feature-set sizes are capacities, and each feature has a weight of 1. These uniform weights enable the polynomial-runtime result stated in Proposition 11.

### A.5.3 Proofs

In this section, we provide proofs for propositions from Section 3.4.2.

**Proof of Proposition 9**

*Proof.* Let $I$ be an arbitrary problem instance of the simultaneous-search problem with min-aggregation, univariate feature qualities, a complete-partitioning scenario, and a fixed feature-set size $k$ (cf. Proposition 8). We add a new feature $f'$ to $I$ and keep the parameters $a$, $k$, and $\tau$ as before, obtaining an instance $I'$ of the incomplete-partitioning scenario since one feature will not be selected. We set the quality $q'$ of $f'$ to be lower than all other feature qualities in $I$. Since the univariate objective monotonically increases in the selected feature qualities, selecting feature $f'$ in a solution of $I'$ does not have any benefit since $f'$ would replace a feature with higher quality. If $f'$ is not selected, this solution of $I'$ also solves $I$. However, if the qualities of the alternatives are not equal, $f'$ might still be chosen in a set that does not have the minimum quality of all sets since only the latter determines the objective value (cf. Example 6). In that case, we replace $f'$ with the remaining unselected feature; the objective value remains the same, and the solution becomes valid for $I$. Thus, in any case, we can easily transform a solution for $I'$ to a solution for $I$.

Overall, an algorithm for incomplete partitioning instances can also solve arbitrary complete-partitioning instances with negligible computational overhead. Thus, a polynomial-time algorithm for incomplete partitioning could also solve complete partitioning polynomially. However, the latter problem is $\mathcal{NP}$-complete (cf. Proposition 8), so incomplete partitioning has to be $\mathcal{NP}$-hard. Since checking a solution for incomplete partitioning needs only polynomial time, we obtain membership in $\mathcal{NP}$ and thereby $\mathcal{NP}$-completeness. □

**Proof of Proposition 10**

*Proof.* Let $I$ be an arbitrary problem instance of the simultaneous-search problem with min-aggregation, univariate feature qualities, a complete-partitioning scenario, the Dice dissimilarity (cf. Equation 3) as $d(\cdot)$, and a fixed feature-set size $k$ (cf. Proposition 8). We create a new problem instance $I'$ by adding a new feature $f'$ and increasing the feature-set size to $k' = k + 1$. Further, we set $\tau' = (k' - 1)/k'$, thereby allowing an overlap of at most one feature between feature sets. Also, we choose $f'$ to have a considerably higher quality $q'$ than all other features. Our goal is to force the selection of $f'$ in all feature sets, no matter which other features are selected. One possible choice is $q' = \sum_{j=1}^{n} q_j + \varepsilon$ for a small $\varepsilon \in \mathbb{R}_{>0}$ This quality $q'$ of $f'$ is higher than of any feature set not containing it. Thus, a solution for $I'$ contains $f'$ in each feature set, while the remaining features are part of exactly one feature set. Hence, we can remove $f'$ to get feature sets of size $k = k' - 1$ that constitute an optimal solution for the original problem instance $I$.

This transformation shows how an algorithm for problem instances with $\tau < 1$ can help solve arbitrary problem instances with $\tau = 1$. Given the $\mathcal{NP}$-completeness of the latter problem (cf. Proposition 8), we obtain $\mathcal{NP}$-hardness of the former. □

One can transfer this reduction from $\tau' = (k' - 1)/k'$ to all other $\tau > 0$. In particular, for a given $k$, there is only a finite number of $\tau$ values leading to different set overlaps, e.g., $\tau = \{0, 1/k, \ldots, (k-1)/k, 1\}$ for the Dice dissimilarity. The proof for the highest overlap except $\tau = 0$ requires creating an instance $I'$ with $\tau' = 1/k$ from an instance with $\tau = 1$. For this purpose, $k^2 - k$ features need to be added since $\tau' = k/k' = k/(k + k^2 - k) = 1/k$. I.e., $k$ out of $k' = k^2$ features need to form a complete partitioning, while the remaining $k^2 - k$ features occur in each feature set and will be removed after solving $I'$. The number of features to be added is polynomial in $k$ and thereby also polynomial in $n$.

**Proof of Proposition 11**

*Proof.* We discuss the simultaneous-search problem (cf. Definition 5) with sum-aggregation (cf. Equation 17) first. We leverage the monotonicity of the univariate objective with sum-aggregation. In particular, this objective cannot decrease when selecting features of higher quality. Thus, we order all features decreasingly by their quality, which yields the complexity of $O(n \cdot \log n)$. Next, we pick features in this order without replacement and assign them to sets until we have the user-defined number of alternatives with the user-defined feature-set sizes. Apart from observing cardinality constraints, the actual assignment of the selected features to sets does not matter quality-wise since swapping features between sets does not change the summed objective. Thus, one can fill the feature sets in an arbitrary order. Each assignment runs in $O(1)$, e.g., using arrays to store feature-set membership, yielding $O(n)$ for all features. Without cardinality constraints, only the number of alternatives needs to be satisfied. Further, if all features need to be selected, i.e., for a complete partitioning, one need not sort the features. Finally, if only a small fraction of features needs to be selected, one might slightly improve complexity to $O(k \cdot n)$ by iteratively picking the maximum instead of sorting all qualities.

For the sequential-search problem (cf. Definition 3), we conduct the same quality-sorting procedure. In contrast to the simultaneous-search problem, the actual assignment of features to sets matters since the sets have an explicit order. In particular, each alternative should get the remaining highest-quality features until its user-defined size is reached. The complexity is still dominated by sorting and therefore $O(n \cdot \log n)$. □

### A.5.4 Future Work

In this section, we outline future work on alternative feature selection from the complexity-theory perspective, supplementing the Sections 3.4 and 7.2.

**Scenarios of alternative feature selection** Our prior complexity analyses focused on special cases of alternative feature selection. E.g., while we obtained $\mathcal{NP}$-hardness for min-aggregation with feature-set overlap (cf. Proposition 10), an analysis of sum-aggregation with overlap is open, even for the sequential-search problem. Sum-aggregation admits polynomial runtime for

$\tau = 1$ (cf. Proposition 11), but this result might not extend to $\tau < 1$. In particular, $\tau < 1$ increases the number of solution candidates, which could negatively affect the runtime.

Further, our complexity analyses mostly assumed univariate feature qualities (cf. Equation 11). Other feature-selection methods can reside in different complexity classes.

**Complexity classes** For analyzing other scenarios of alternative feature selection, several questions spring to mind. First, one could establish a complexity result like $\mathcal{NP}$-hardness or membership in $\mathcal{P}$. In the former case, there might be pseudo-polynomial approaches or (F)PTAS. As a first step in that direction, we showed membership in complexity class $\mathcal{APX}$ under certain conditions (cf. Proposition 13), i.e., there are polynomial-time algorithms yielding constant-factor approximations. One may attempt to tighten the quality bound we derived. Further, there might be efficient exact or approximate algorithms for certain types of problem instances, e.g., satisfying additional assumptions regarding the feature-set quality or the parameters $k$, $a$, and $\tau$. Finally, while we placed alternative feature selection in the parameterized complexity class $\mathcal{XP}$ (cf. Proposition 5), one might prove membership or hardness for more specific parameterized complexity classes.

**Related problem formulations** We only focused on the optimization problem of alternative feature selection until now. Another interesting question is how many alternatives exist for a given $n$, $k$, and $\tau$, regardless of their quality. Also, given the number of alternatives as well, it would be interesting to have an exact or approximate estimate for the number of valid solutions for alternative feature selection, i.e., sets of feature sets. While both these estimates are straightforward for $\tau = 1$, allowing arbitrary $\tau$ poses a larger challenge. Finally, one could re-formulate alternative feature selection similar to Bin Covering (cf. Section A.5.2) and analyze this problem in detail.

## A.6 Greedy Depth Search for the Univariate Objective

In this section, we propose another heuristic search method for univariate feature qualities (cf. Equation 11 and Section A.3), complementing the methods discussed in Section 3.5. In particular, the new method *Greedy Depth Search* is a sequential search method that generalizes *Greedy Replacement* and allows to obtain more than $\frac{n-k}{\lceil \tau \cdot k \rceil}$ alternatives.

**Algorithm** Algorithm 4 outlines *Greedy Depth Search*. As in the other two heuristics, we start by sorting the features decreasingly according to their qualities $q_j$ (Line 1). However, instead of keeping the same $\lfloor (1-\tau) \cdot k \rfloor$ features in each alternative and only replacing the remaining ones, we now allow all features to be replaced. In particular, we may exhaustively iterate over all feature sets, depending on the number of alternatives $a$. Thus, we maintain not only one

**Algorithm 4:** *Greedy Depth Search* for alternative feature selection.

**Input:** Univariate feature qualities $q \in \mathbb{R}^n$,
Feature-set size $k \in \mathbb{N}$,
Number of alternatives $a \in \mathbb{N}_0$,
Dissimilarity threshold $\tau \in [0, 1]$

**Output:** List of feature-selection decision vectors $s^{(\cdot)}$

```
1  indices ← sort_indices(q, order=descending)   // Order by qualities
2  feature_positions ← {0}^k                     // Indices of indices of features
3  for p ← 1 to k do                             // Start with top k features
4      feature_positions[p] ← p                  // Ordered by qualities as well
5  l ← 0                                         // Number of current alternative
6  has_next_solution ← true
7  while l ≤ a and has_next_solution do
8      s^(l) ← {0}^n
9      for p ← 1 to k do  // Select k features, indexed by quality
10         j ← indices[feature_positions[p]]
11         s_j^(l) ← 1
12     if is_valid_alternative(s^(l), {s^(0), ..., s^(l−1)}) then
13         l ← l + 1      // Else, s^(l) overwritten in next iteration
14     p ← k      // Update feature positions, starting with last
15     while p ≥ 1 do
16         position ← feature_positions[p]
17         if position < n + p − k then  // Position can be increased
18             for Δ_p ← 0 to k − p do // Also update later positions
19                 feature_positions[p + Δ_p] ← position + Δ_p + 1
20             p ← −1                    // Position update finished
21         else                        // Position cannot be increased
22             p ← p − 1 // Also update at least one prior position
23     if p = 0 then // Updating positions further would violate n
24         has_next_solution ← false
25 return s^(0), ..., s^(l)
```

feature position as before but a length-$k$ array of the feature positions for the current feature set (Lines 2–4). This array represents feature indices regarding the sorted qualities and is sorted increasingly, which prevents evaluating the same feature set, only with different feature order, multiple times.

In the main loop of the algorithm, we find alternatives sequentially (Lines 7–24). For each potential alternative, we select the features based on the position array (Lines 8–11). We check the resulting feature set against the constraints for alternatives (Line 12) and only store it if it is valid. This check was unnecessary in the other two heuristics, which only formed valid alternatives by design.

Next, we update the feature positions for the next potential alternative (Lines 14–24). First, we try to replace the lowest-quality feature in the current feature set by advancing one position in the sorted qualities. This step may not be possible, as the feature set may already contain the feature with the overall lowest quality, i.e., position $n$ in the array of sorted qualities (Line 17). In this case, we try to replace the second-lowest-quality feature in the current feature set by advancing its position. If this action is impossible as well, we iterate further over positions in the current feature set by increasing quality (Line 22). Once we find a feature position that we can increase, we also advance all subsequent, i.e., lower-quality, positions accordingly. Hence, the feature positions remain sorted by decreasing quality (Lines 18–19).

We repeat the main loop until we reach the desired number of alternatives $a$ or until we cannot update any feature position without exceeding the number of features $n$, i.e., we cannot form another alternative (Lines 7 and 23).

**Example 7** (Algorithm of *Greedy Depth Search*)**.** Consider $n = 6$ features with univariate feature qualities $q = (9, 8, 7, 3, 2, 1)$, feature-set size $k = 4$, number of alternatives $a = 1$, and dissimilarity threshold $\tau = 0.5$, which permits an overlap of two features between sets here. Note that the features are already ordered by quality here, i.e., $\mathit{indices} = (1, 2, 3, 4, 5, 6)$ (Line 1). Next, the algorithm initializes $\mathit{feature\_positions} = (1, 2, 3, 4)$ (Line 2–4). $s^{(0)}$ contains these $k$ features, i.e., $s^{(0)} = (1, 1, 1, 1, 0, 0)$. Given that there are no other alternatives yet, this feature set is valid (Line 12)) and the algorithm moves on to $l = 1$.

For forming $s^{(1)}$, the position-update step (Lines 14–24) first tries to only replace the lowest-quality feature in the alternative, i.e., $\mathit{feature\_positions} = (1, 2, 3, 5)$ and $\mathit{feature\_positions} = (1, 2, 3, 6)$. However, neither of these feature sets constitutes a valid alternative regarding $s^{(0)}$. Thus, the algorithm attempts to replace the feature with the second-lowest quality as well, evaluating $\mathit{feature\_positions} = (1, 2, 4, 5)$ and $\mathit{feature\_positions} = (1, 2, 4, 6)$. However, the overlap with $s^{(0)}$ is still too large. The next value is $\mathit{feature\_positions} = (1, 2, 5, 6)$, which yields the valid alternative $s^{(1)} = (1, 1, 0, 0, 1, 1)$.

*Greedy Replacement* would terminate now since the options for replacing the $\lceil \tau \cdot k \rceil = 2$ lowest-quality features are exhausted. In contrast, *Greedy Depth Search* attempts to replace the third-lowest-quality feature, starting with $\mathit{feature\_positions} = (1, 3, 4, 5)$. This feature set is not a valid alternative, and neither are the subsequent feature sets with $\mathit{feature\_positions} = (1, 3, 4, 6)$, $\mathit{feature\_positions} = (1, 3, 5, 6)$, etc. After more iterations, the algorithm also re-

places the highest-quality feature, starting with *feature_positions* $= (2, 3, 4, 5)$. Eventually, the algorithm reaches *feature_positions* $= (3, 4, 5, 6)$, which yields the valid alternative $s^{(2)} = (0, 0, 1, 1, 1, 1)$. After obtaining $s^{(2)}$, there is no valid update of the feature positions left (Line 23). Thus, the algorithm terminates.

**Time complexity** The runtime behavior differs from the other two heuristics. In particular, *Greedy Replacement* has the same runtime cost between subsequent alternatives since it directly creates valid alternatives by design. In contrast, *Greedy Depth Search* iterates over all possible feature sets, and the runtime between valid alternatives may vary. For each values of *feature_positions*, the algorithm creates a feature selection in $O(k \cdot n)$ (Lines 8–11), checks constraints in $O(a \cdot n)$ (Line 12), and updates the position array in $O(k^2)$ (Lines 14–24). However, there are $O(n^k)$ potential *feature_positions*, and *Greedy Depth Search* may exhaustively iterate over them. This cost is comparable to exhaustive conventional feature selection (cf. Proposition 2) and exhaustive sequential search (cf. Proposition 3). Unlike the latter, the search does not restart for each alternative, i.e., it only considers each feature set once instead of $a + 1$ times.

On the positive side, *Greedy Depth Search* can yield more alternatives than *Greedy Replacement* with its $O(n^2)$ cost or *Greedy Balancing* with its $O(a \cdot n^2)$ cost. Nevertheless, in scenarios where the latter two are applicable, i.e., $k + a \cdot \lceil \tau \cdot k \rceil \leq n$, they have a lower cost than *Greedy Depth Search*. In particular, *Greedy Depth Search* needs $O(n^{\lceil \tau \cdot k \rceil})$ iterations to cover the options for replacing the worst $\lceil \tau \cdot k \rceil$ features in size-$k$ feature sets, which is the search space of the other two heuristics. In particular, the cost disadvantage relative to the other two heuristics grows with the dissimilarity threshold $\tau$. As a remedy, one may use *Greedy Replacement* for as many alternatives as possible and then continue with *Greedy Depth Search*, initializing the *feature_positions* (Line 2–4) based on the results of the former heuristic.

**Quality** *Greedy Depth Search* initially yields the same solutions as *Greedy Replacement*. Thus, *Greedy Depth Search* also yields a constant-factor approximation of the optimal solution in case $k + a \cdot \lceil \tau \cdot k \rceil \leq n$ (cf. Proposition 12). The quality analysis becomes more involved for further alternatives since these do not contain all top $\lfloor (1 - \tau) \cdot k \rfloor$ features anymore, on which our proof of Proposition 12 builds. Thus, we leave this analysis open for future work. The quality of alternatives may not even be monotonically decreasing anymore, as the following example shows:

**Example 8** (Non-monotonic quality of *Greedy Depth Search*)**.** Consider $n = 4$ features with univariate feature qualities $q = (9, 8, 7, 1)$, feature-set size $k = 2$, number of alternatives $a = 3$, and dissimilarity threshold $\tau = 0.5$, which permits an overlap of one feature between sets here. *Greedy Depth Search* yields the the selection $s^{(0)} = (1, 1, 0, 0)$, $s^{(1)} = (1, 0, 1, 0)$, $s^{(2)} = (1, 0, 0, 1)$, and $s^{(3)} = (0, 1, 1, 0)$, with the corresponding feature-set qualities 17, 16, 10, and 15.

**Limitations** Like *Greedy Balancing* and *Greedy Replacement*, *Greedy Depth Search* assumes univariate feature qualities and may be worse than exact search. As a sequential procedure, it does not balance the alternatives' qualities. It may yield more alternatives than the former two heuristics but has a higher and more variable runtime.

# References


[1] Noga Alon et al. "Approximation schemes for scheduling on parallel machines". In: *J. Sched.* 1.1 (1998), pp. 55–66. DOI: 10.1002/(SICI)1099-1425(199806)1:1<55::AID-JOS2>3.0.CO;2-J.

[2] André Artelt and Barbara Hammer. ""Even if ..." – Diverse Semifactual Explanations of Reject". In: *Proc. SSCI*. Singapore, Singapore, 2022, pp. 854–859. DOI: 10.1109/SSCI51031.2022.10022139.

[3] Susan F. Assmann et al. "On a Dual Version of the One-Dimensional Bin Packing Problem". In: *J. Algorithms* 5.4 (1984), pp. 502–525. DOI: 10.1016/0196-6774(84)90004-X.

[4] Luitpold Babel, Hans Kellerer, and Vladimir Kotov. "The k-Partitioning Problem". In: *Math. Methods Oper. Res.* 47.1 (1998), pp. 59–82. DOI: 10.1007/BF01193837.

[5] Fahiem Bacchus, Matti Järvisalo, and Ruben Martins. "Maximum Satisfiability". In: *Handbook of Satisfiability*. 2nd ed. IOS Press, 2021. Chap. 24, pp. 929–991. DOI: 10.3233/FAIA201008.

[6] Jakob Bach. "Leveraging Constraints for User-Centric Feature Selection". PhD thesis. Karlsruhe Institute of Technology (KIT), 2025. DOI: 10.5445/IR/1000178649.

[7] Jakob Bach and Klemens Böhm. "Alternative feature selection with user control". In: *Int. J. Data Sci. Anal.* (2024). DOI: 10.1007/s41060-024-00527-8.

[8] Jakob Bach et al. "An Empirical Evaluation of Constrained Feature Selection". In: *SN Comput. Sci.* 3.6 (2022). DOI: 10.1007/s42979-022-01338-z.

[9] Eric Bae and James Bailey. "COALA: A Novel Approach for the Extraction of an Alternate Clustering of High Quality and High Dissimilarity". In: *Proc. ICDM*. Hong Kong, China, 2006, pp. 53–62. DOI: 10.1109/ICDM.2006.37.

[10] Eric Bae, James Bailey, and Guozhu Dong. "A clustering comparison measure using density profiles and its application to the discovery of alternate clusterings". In: *Data Min. Knowl. Disc.* 21.3 (2010), pp. 427–471. DOI: 10.1007/s10618-009-0164-z.

[11] James Bailey. "Alternative Clustering Analysis: A Review". In: *Data Clustering: Algorithms and Applications*. 1st ed. CRC Press, 2014. Chap. 21, pp. 535–550. DOI: 10.1201/9781315373515.

[12] Clark Barrett and Cesare Tinelli. "Satisfiability Modulo Theories". In: *Handbook of Model Checking*. 1st ed. Springer, 2018. Chap. 11, pp. 305–343. DOI: 10.1007/978-3-319-10575-8_11.

[13] Ksenia Bestuzheva et al. *The SCIP Optimization Suite 8.0*. Tech. rep. Zuse Institute Berlin, 2021. URL: http://nbn-resolving.de/urn:nbn:de:0297-zib-85309.

[14] Giorgos Borboudakis and Ioannis Tsamardinos. "Extending greedy feature selection algorithms to multiple solutions". In: *Data Min. Knowl. Disc.* 35.4 (2021), pp. 1393–1434. DOI: 10.1007/s10618-020-00731-7.

[15] Leo Breiman. "Random Forests". In: *Mach. Learn.* 45.1 (2001), pp. 5–32. DOI: 10.1023/A:1010933404324.

[16] Leo Breiman et al. *Classification and Regression Trees*. 1st ed. Chapman and Hall, 1984. DOI: 10.1201/9781315139470.

[17] Diogo V. Carvalho, Eduardo M. Pereira, and Jaime S. Cardoso. "Machine Learning Interpretability: A Survey on Methods and Metrics". In: *Electronics* 8.8 (2019). DOI: 10.3390/electronics8080832.

[18] Girish Chandrashekar and Ferat Sahin. "A survey on feature selection methods". In: *Comput. Electr. Eng.* 40.1 (2014), pp. 16–28. DOI: 10.1016/j.compeleceng.2013.11.024.

[19] Chandra Chekuri and Sanjeev Khanna. "A Polynomial Time Approximation Scheme for the Multiple Knapsack Problem". In: *SIAM J. Comput.* 35.3 (2005), pp. 713–728. DOI: 10.1137/S0097539700382820.

[20] Lin Chen et al. "An Efficient PTAS for Parallel Machine Scheduling with Capacity Constraints". In: *Proc. COCOA*. Hong Kong, China, 2016, pp. 608–623. DOI: 10.1007/978-3-319-48749-6_44.

[21] Shi Ping Chen, Yong He, and Guohui Lin. "3-Partitioning Problems for Maximizing the Minimum Load". In: *J. Comb. Optim.* 6.1 (2002), pp. 67–80. DOI: 10.1023/A:1013370208101.

[22] Seung-Seok Choi, Sung-Hyuk Cha, and Charles C. Tappert. "A Survey of Binary Similarity and Distance Measures". In: *J. Syst. Cybern. Inf.* 8.1 (2010), pp. 43–48. URL: http://www.iiisci.org/Journal/pdv/sci/pdfs/GS315JG.pdf.

[23] Ian Covert, Scott M. Lundberg, and Su-In Lee. "Understanding Global Feature Contributions With Additive Importance Measures". In: *Proc. NeurIPS*. Virtual-only Conference, 2020, pp. 17212–17223. URL: https://proceedings.neurips.cc/paper_files/paper/2020/hash/c7bf0b7c1a86d5eb3be2c722cf2cf746-Abstract.html.

[24] Susanne Dandl et al. "Multi-Objective Counterfactual Explanations". In: *Proc. PPSN*. Leiden, The Netherlands, 2020, pp. 448–469. DOI: 10.1007/978-3-030-58112-1_31.

[25] Mauro Dell'Amico, Manuel Iori, and Silvano Martello. "Heuristic Algorithms and Scatter Search for the Cardinality Constrained $P||C_{max}$ Problem". In: *J. Heuristics* 10.2 (2004), pp. 169–204. DOI: 10.1023/B:HEUR.0000026266.07036.da.

[26] Mauro Dell'Amico and Silvano Martello. "Bounds for the cardinality constrained $P||C_{max}$ problem". In: *J. Sched.* 4.3 (2001), pp. 123–138. DOI: 10.1002/jos.68.

[27] Bryan L. Deuermeyer, Donald K. Friesen, and Michael A. Langston. "Scheduling to Maximize the Minimum Processor Finish Time in a Multiprocessor System". In: *SIAM J. Algebraic Discrete Methods* 3.2 (1982), pp. 190–196. DOI: 10.1137/0603019.

[28] Rodney G. Downey, Michael R. Fellows, and Ulrike Stege. "Parameterized Complexity: A Framework for Systematically Confronting Computational Intractability". In: *Contemporary Trends in Discrete Mathematics: From DIMACS and DIMATIA to the Future*. Štiřín Castle, Czech Republic, 1997, pp. 49–99. DOI: 10.1090/dimacs/049/04.

[29] Leo Egghe. "New Relations Between Similarity Measures for Vectors Based on Vector Norms". In: *J. Am. Soc. Inf. Sci. Technol.* 60.2 (2009), pp. 232–239. DOI: 10.1002/asi.20949.

[30] Christos Emmanouilidis et al. "Selecting Features in Neurofuzzy Modelling by Multiobjective Genetic Algorithms". In: *Proc. ICANN*. Edinburgh, United Kingdom, 1999, pp. 749–754. DOI: 10.1049/cp:19991201.

[31] Stefano Ermon, Carla Gomes, and Bart Selman. "Uniform Solution Sampling Using a Constraint Solver As an Oracle". In: *Proc. UAI*. Catalina Island, CA, USA, 2012, pp. 255–264. URL: https://www.auai.org/uai2012/papers/160.pdf.

[32] Aaron Fisher, Cynthia Rudin, and Francesca Dominici. "All Models are Wrong, but Many are Useful: Learning a Variable's Importance by Studying an Entire Class of Prediction Models Simultaneously". In: *J. Mach. Learn. Res.* 20.177 (2019). URL: https://jmlr.org/papers/v20/18-760.html.

[33] Edouard Fouché, Florian Kalinke, and Klemens Böhm. "Efficient subspace search in data streams". In: *Inf. Syst.* 97 (2021). DOI: 10.1016/j.is.2020.101705.

[34] Daniel Fryer, Inga Strümke, and Hien Nguyen. "Shapley Values for Feature Selection: The Good, the Bad, and the Axioms". In: *IEEE Access* 9 (2021), pp. 144352–144360. DOI: 10.1109/ACCESS.2021.3119110.

[35] Michael R. Garey and David S. Johnson. *Computers and Intractibility: A Guide to the Theory of NP-Completeness*. 24th ed. W. H. Freeman and Company, 2003. URL: https://www.worldcat.org/title/440655898.

[36] Ronald L. Graham, Donald E. Knuth, and Oren Patashnik. *Concrete Mathematics: A Foundation for Computer Science*. 2nd ed. Addison-Wesley, 1994. URL: https://www.worldcat.org/title/1085703509.

[37] William Christopher Groves. "Toward Automating and Systematizing the Use of Domain Knowledge in Feature Selection". PhD thesis. University of Minnesota, 2015. URL: https://hdl.handle.net/11299/175444.

[38] Riccardo Guidotti. "Counterfactual explanations and how to find them: literature review and benchmarking". In: *Data Min. Knowl. Disc.* (2022), pp. 1–55. DOI: 10.1007/s10618-022-00831-6.

[39] Stephan Günnemann et al. "Detection of Orthogonal Concepts in Subspaces of High Dimensional Data". In: *Proc. CIKM*. Hong Kong, China, 2009, pp. 1317–1326. DOI: 10.1145/1645953.1646120.

[40] Jianmei Guo and Kai Shi. "To Preserve or Not to Preserve Invalid Solutions in Search-Based Software Engineering: A Case Study in Software Product Lines". In: *Proc. ICSE*. Gothenburg, Sweden, 2018, pp. 1027–1038. DOI: 10.1145/3180155.3180163.

[41] D. S. Guru et al. "An alternative framework for univariate filter based feature selection for text categorization". In: *Pattern Recognit. Lett.* 103 (2018), pp. 23–31. DOI: 10.1016/j.patrec.2017.12.025.

[42] Isabelle Guyon and André Elisseeff. "An Introduction to Variable and Feature Selection". In: *J. Mach. Learn. Res.* 3.Mar (2003), pp. 1157–1182. URL: https://www.jmlr.org/papers/v3/guyon03a.html.

[43] Mark A. Hall. "Correlation-based Feature Selection for Machine Learning". PhD thesis. University of Waikato, 1999. URL: https://hdl.handle.net/10289/15043.

[44] Mark A. Hall. *Correlation-based Feature Selection of Discrete and Numeric Class Machine Learning*. Tech. rep. University of Waikato, Department of Computer Science, 2000. URL: https://hdl.handle.net/10289/1024.

[45] Mohamed Haouari and Mahdi Jemmali. "Maximizing the minimum completion time on parallel machines". In: *4OR* 6.4 (2008), pp. 375–392. DOI: 10.1007/s10288-007-0053-5.

[46] Yong He et al. "k-Partitioning Problems for Maximizing the Minimum Load". In: *Comput. Math. Appl.* 46.10-11 (2003), pp. 1671–1681. DOI: 10.1016/S0898-1221(03)90201-X.

[47] Christopher Henard et al. "Combining Multi-Objective Search and Constraint Solving for Configuring Large Software Product Lines". In: *Proc. ICSE*. Florence, Italy, 2015, pp. 517–528. DOI: 10.1109/ICSE.2015.69.

[48] Juhua Hu and Jian Pei. "Subspace multi-clustering: a review". In: *Knowl. Inf. Sys.* 56.2 (2018), pp. 257–284. DOI: 10.1007/s10115-017-1110-9.

[49] Sarthak Jain and Byron C. Wallace. "Attention is not Explanation". In: *Proc. NAACL-HLT*. Minneapolis, MN, USA, 2019, pp. 3543–3556. DOI: 10.18653/v1/N19-1357.

[50] Amir-Hossein Karimi et al. "Model-Agnostic Counterfactual Explanations for Consequential Decisions". In: *Proc. AISTATS*. Online, 2020, pp. 895–905. URL: https://proceedings.mlr.press/v108/karimi20a.html.

[51] Richard M. Karp. "Reducibility among Combinatorial Problems". In: *Complexity of Computer Computations*. 1st ed. Plenum Press, 1972. Chap. 9, pp. 85–103. DOI: 10.1007/978-1-4684-2001-2_9.

[52] Hans Kellerer and Vladimir Kotov. "A 3/2-approximation algorithm for $k_i$-partitioning". In: *Oper. Res. Lett.* 39.5 (2011), pp. 359–362. DOI: 10.1016/j.orl.2011.06.005.

[53] Sanjeev Khanna, Madhu Sudan, and David P. Williamson. "A Complete Classification of the Approximability of Maximization Problems Derived from Boolean Constraint Satisfaction". In: *Proc. STOC*. El Paso, TX, USA, 1997, pp. 11–20. DOI: 10.1145/258533.258538.

[54] Sanjeev Khanna et al. "On Syntactic Versus Computational Views of Approximability". In: *SIAM J. Comput.* 28.1 (1998), pp. 164–191. DOI: 10.1137/S0097539795286612.

[55] Been Kim, Rajiv Khanna, and Oluwasanmi Koyejo. "Examples are not Enough, Learn to Criticize! Criticism for Interpretability". In: *Proc. NIPS*. Barcelona, Spain, 2016. URL: https://proceedings.neurips.cc/paper_files/paper/2016/hash/5680522b8e2bb01943234bce7bf84534-Abstract.html.

[56] Mi-Young Kim et al. "A Multi-Component Framework for the Analysis and Design of Explainable Artificial Intelligence". In: *Mach. Learn. Knowl. Extr.* 3.4 (2021), pp. 900–921. DOI: 10.3390/make3040045.

[57] Kenji Kira and Larry A. Rendell. "The Feature Selection Problem: Traditional Methods and a New Algorithm". In: *Proc. AAAI*. San Jose, CA, USA, 1992, pp. 129–134. URL: https://aaai.org/papers/00129-aaai92-020-the-feature-selection-problem-traditional-methods-and-a-new-algorithm/.

[58] Ron Kohavi and George H. John. "Wrappers for feature subset selection". In: *Artif. Intell.* 97.1-2 (1997), pp. 273–324. DOI: 10.1016/S0004-3702(97)00043-X.

[59] Richard E. Korf. "Multi-Way Number Partitioning". In: *Proc. IJCAI*. Pasadena, CA, USA, 2009, pp. 538–543. URL: https://www.ijcai.org/Proceedings/09/Papers/096.pdf.

[60] Richard E. Korf. "Objective Functions for Multi-Way Number Partitioning". In: *Proc. SoCS*. Atlanta, GA, USA, 2010, pp. 71–72. DOI: 10.1609/socs.v1i1.18172.

[61] Alexander Kraskov, Harald Stögbauer, and Peter Grassberger. "Estimating mutual information". In: *Phys. Rev. E* 69.6 (2004). DOI: 10.1103/PhysRevE.69.066138.

[62] Gabriel Laberge et al. "Partial Order in Chaos: Consensus on Feature Attributions in the Rashomon Set". In: *J. Mach. Learn. Res.* 24.364 (2023). URL: http://jmlr.org/papers/v24/23-0149.html.

[63] Vincenzo Lagani et al. "Feature Selection with the R Package MXM: Discovering Statistically Equivalent Feature Subsets". In: *J. Stat. Software* 80.7 (2017). DOI: 10.18637/jss.v080.i07.

[64] Alexander Lawrinenko. "Identical Parallel Machine Scheduling Problems: Structural patterns, bounding techniques and solution procedures". PhD thesis. Friedrich-Schiller-Universität Jena, 2017. URL: https://nbn-resolving.org/urn:nbn:de:gbv:27-dbt-20170427-0956483.

[65] Alexander Lawrinenko, Stefan Schwerdfeger, and Rico Walter. "Reduction criteria, upper bounds, and a dynamic programming based heuristic for the max–min $k_i$-partitioning problem". In: *J. Heuristics* 24.2 (2018), pp. 173–203. DOI: 10.1007/s10732-017-9362-9.

[66] Matthijs van Leeuwen and Arno Knobbe. "Diverse subgroup set discovery". In: *Data Min. Knowl. Disc.* 25.2 (2012), pp. 208–242. DOI: 10.1007/s10618-012-0273-y.

[67] Chu Min Li and Felip Manyà. "MaxSAT, Hard and Soft Constraints". In: *Handbook of Satisfiability*. 2nd ed. IOS Press, 2021. Chap. 23, pp. 903–927. DOI: 10.3233/FAIA201007.

[68] Jundong Li et al. "Feature Selection: A Data Perspective". In: *ACM Comput. Surv.* 50.6 (2017). DOI: 10.1145/3136625.

[69] Kai Liu and Jin Tian. "Subspace Learning with an Archive-Based Genetic Algorithm". In: *Proc. IEEM*. Changsha, China, 2018, pp. 181–188. DOI: 10.1007/978-981-13-3402-3_20.

[70] Scott M. Lundberg and Su-In Lee. "A Unified Approach to Interpreting Model Predictions". In: *Proc. NIPS*. Long Beach, CA, USA, 2017. URL: https://proceedings.neurips.cc/paper_files/paper/2017/hash/8a20a8621978632d76c43dfd28b67767-Abstract.html.

[71] Brian W. Matthews. "Comparison of the predicted and observed secondary structure of T4 phage lysozyme". In: *Biochim. Biophys. Acta - Protein Struct.* 405.2 (1975), pp. 442–451. DOI: 10.1016/0005-2795(75)90109-9.

[72] Wil Michiels et al. "Computer-assisted proof of performance ratios for the Differencing Method". In: *Discrete Optim.* 9.1 (2012), pp. 1–16. DOI: 10.1016/j.disopt.2011.10.001.

[73] Matthias Mnich and René van Bevern. "Parameterized complexity of machine scheduling: 15 open problems". In: *Comput. Oper. Res.* 100 (2018), pp. 254–261. DOI: 10.1016/j.cor.2018.07.020.

[74] Kiarash Mohammadi et al. "Scaling Guarantees for Nearest Counterfactual Explanations". In: *Proc. AIES*. Virtual Event, USA, 2021, pp. 177–187. DOI: 10.1145/3461702.3462514.

[75] Christoph Molnar, Giuseppe Casalicchio, and Bernd Bischl. "Interpretable Machine Learning – A Brief History, State-of-the-Art and Challenges". In: *Proc. ECML PKDD Workshops*. Ghent, Belgium, 2020, pp. 417–431. DOI: 10.1007/978-3-030-65965-3_28.

[76] MOSEK ApS. *MOSEK Modeling Cookbook : Mixed integer optimzation*. Accessed: 2022-10-18. 2022. URL: https://docs.mosek.com/modeling-cookbook/mio.html.

[77] L. Moser and M. Wyman. "Stirling numbers of the second kind". In: *Duke Math. J.* 25.1 (1958), pp. 29–43. DOI: 10.1215/S0012-7094-58-02504-3.

[78] Ramaravind K. Mothilal, Amit Sharma, and Chenhao Tan. "Explaining Machine Learning Classifiers through Diverse Counterfactual Explanations". In: *Proc. FAT**. Barcelona, Spain, 2020, pp. 607–617. DOI: 10.1145/3351095.3372850.

[79] Emmanuel Müller et al. "Relevant Subspace Clustering: Mining the Most Interesting Non-Redundant Concepts in High Dimensional Data". In: *Proc. ICDM*. Miami Beach, FL, USA, 2009, pp. 377–386. DOI: 10.1109/ICDM.2009.10.

[80] Inga M. Müller. "Feature selection for energy system modeling: Identification of relevant time series information". In: *Energy AI* 4 (2021). DOI: 10.1016/j.egyai.2021.100057.

[81] Roger B. Myerson. "Utilitarianism, Egalitarianism, and the Timing Effect in Social Choice Problems". In: *Econometrica* 49.4 (1981), pp. 883–897. DOI: 10.2307/1912508.

[82] Nina Narodytska et al. "Learning Optimal Decision Trees with SAT". In: *Proc. IJCAI*. Stockholm, Sweden, 2018, pp. 1362–1368. DOI: 10.24963/ijcai.2018/189.

[83] Hai Nguyen, Katrin Franke, and Slobodan Petrović. "Optimizing a Class of Feature Selection Measures". In: *Proc. DISCML*. Vancouver, BC, Canada, 2009. URL: https://www.researchgate.net/publication/231175763.

[84] Hai Thanh Nguyen, Katrin Franke, and Slobodan Petrović. "Improving Effectiveness of Intrusion Detection by Correlation Feature Selection". In: *Proc. ARES*. Krakow, Poland, 2010, pp. 17–24. DOI: 10.1109/ARES.2010.70.

[85] Hai Thanh Nguyen, Katrin Franke, and Slobodan Petrović. "Towards a Generic Feature-Selection Measure for Intrusion Detection". In: *Proc. ICPR*. Istanbul, Turkey, 2010, pp. 1529–1532. DOI: 10.1109/ICPR.2010.378.

[86] Hoang Vu Nguyen, Emmanuel Müller, and Klemens Böhm. "4S: Scalable Subspace Search Scheme Overcoming Traditional Apriori Processing". In: *Proc. Big Data*. Santa Clara, CA, USA, 2013, pp. 359–367. DOI: 10.1109/BigData.2013.6691596.

[87] Xuan Vinh Nguyen et al. "Effective Global Approaches for Mutual Information Based Feature Selection". In: *Proc. KDD*. New York, NY, USA, 2014, pp. 512–521. DOI: 10.1145/2623330.2623611.

[88] Robert Nieuwenhuis and Albert Oliveras. "On SAT Modulo Theories and Optimization Problems". In: *Proc. SAT*. Seattle, WA, USA, 2006, pp. 156–169. DOI: 10.1007/11814948_18.

[89] Uchechukwu F. Njoku et al. "Wrapper Methods for Multi-Objective Feature Selection". In: *Proc. EDBT*. Ioannina, Greece, 2023, pp. 697–709. DOI: 10.48786/edbt.2023.58.

[90] Randal S. Olson et al. "PMLB: a large benchmark suite for machine learning evaluation and comparison". In: *BioData Min.* 10 (2017). DOI: 10.1186/s13040-017-0154-4.

[91] Pavel Paclík et al. "On Feature Selection with Measurement Cost and Grouped Features". In: *Proc. SSPR /SPR*. Windsor, ON, Canada, 2002, pp. 461–469. DOI: 10.1007/3-540-70659-3_48.

[92] Fabian Pedregosa et al. "Scikit-learn: Machine Learning in Python". In: *J. Mach. Learn. Res.* 12.85 (2011), pp. 2825–2830. URL: http://jmlr.org/papers/v12/pedregosa11a.html.

[93] Hanchuan Peng, Fuhui Long, and Chris Ding. "Feature Selection Based on Mutual Information Criteria of Max-Dependency, Max-Relevance, and Min-Redundancy". In: *IEEE Trans. Pattern Anal. Mach. Intell.* 27.8 (2005), pp. 1226–1238. DOI: 10.1109/TPAMI.2005.159.

[94] Laurent Perron and Vincent Furnon. *OR-Tools*. Accessed: 2022-10-18. Google, 2022. URL: https://developers.google.com/optimization/.

[95] Marco Tulio Ribeiro, Sameer Singh, and Carlos Guestrin. ""Why Should I Trust You?" Explaining the Predictions of Any Classifier". In: *Proc. KDD*. San Francisco, CA, USA, 2016, pp. 1135–1144. DOI: 10.1145/2939672.2939778.

[96] Marko Robnik-Šikonja and Igor Kononenko. "An adaptation of Relief for attribute estimation in regression". In: *Proc. ICML*. Nashville, TN, USA, 1997, pp. 296–304. URL: https://www.researchgate.net/publication/2635627.

[97] Irene Rodriguez-Lujan et al. "Quadratic Programming Feature Selection". In: *J. Mach. Learn. Res.* 11.49 (2010), pp. 1491–1516. URL: http://jmlr.org/papers/v11/rodriguez-lujan10a.html.

[98] Joseph D. Romano et al. *PMLB v1.0: An open source dataset collection for benchmarking machine learning methods*. arXiv:2012.00058v3 [cs.LG]. 2021. DOI: 10.48550/arXiv.2012.00058.

[99] Chris Russell. "Efficient Search for Diverse Coherent Explanations". In: *Proc. FAT**. Atlanta, GA, USA, 2019, pp. 20–28. DOI: 10.1145/3287560.3287569.

[100] Yvan Saeys, Thomas Abeel, and Yves Van de Peer. "Robust Feature Selection Using Ensemble Feature Selection Techniques". In: *Proc. ECML PKDD*. Antwerp, Belgium, 2008, pp. 313–325. DOI: 10.1007/978-3-540-87481-2_21.

[101] Sartaj K. Sahni. "Algorithms for Scheduling Independent Tasks". In: *J. ACM* 23.1 (1976), pp. 116–127. DOI: 10.1145/321921.321934.

[102] André Schidler and Stefan Szeider. "SAT-based Decision Tree Learning for Large Data Sets". In: *Proc. AAAI*. Virtual Conference, 2021, pp. 3904–3912. DOI: 10.1609/aaai.v35i5.16509.

[103] Ethan L. Schreiber, Richard E. Korf, and Michael D. Moffitt. "Optimal Multi-Way Number Partitioning". In: *J. ACM* 65.4 (2018), pp. 1–61. DOI: 10.1145/3184400.

[104] Borja Seijo-Pardo et al. "Ensemble feature selection: Homogeneous and heterogeneous approaches". In: *Knowl.-Based Syst.* 118 (2017), pp. 124–139. DOI: 10.1016/j.knosys.2016.11.017.

[105] Arvind Kumar Shekar, Patricia Iglesias Sánchez, and Emmanuel Müller. "Diverse Selection of Feature Subsets for Ensemble Regression". In: *Proc. DaWaK*. Lyon, France, 2017, pp. 259–273. DOI: 10.1007/978-3-319-64283-3_19.

[106] Umair F. Siddiqi, Sadiq M. Sait, and Okyay Kaynak. "Genetic Algorithm for the Mutual Information-Based Feature Selection in Univariate Time Series Data". In: *IEEE Access* 8 (2020), pp. 9597–9609. DOI: 10.1109/ACCESS.2020.2964803.

[107] Wilson Silva, Kelwin Fernandes, and Jaime S. Cardoso. "How to produce complementary explanations using an Ensemble model". In: *Proc. IJCNN*. Budapest, Hungary, 2019. DOI: 10.1109/IJCNN.2019.8852409.

[108] Carsten Sinz. "Towards an Optimal CNF Encoding of Boolean Cardinality Constraints". In: *Proc. CP*. Sitges, Spain, 2005, pp. 827–831. DOI: 10.1007/11564751_73.

[109] Gustavo Sosa-Cabrera et al. "Feature selection: a perspective on inter-attribute cooperation". In: *Int. J. Data Sci. Anal.* 17.2 (2024), pp. 139–151. DOI: 10.1007/s41060-023-00439-z.

[110] Ilia Stepin et al. "A Survey of Contrastive and Counterfactual Explanation Generation Methods for Explainable Artificial Intelligence". In: *IEEE Access* 9 (2021), pp. 11974–12001. DOI: 10.1109/ACCESS.2021.3051315.

[111] Vinh Thanh Tao and JongHyeok Lee. "A Novel Approach for Finding Alternative Clusterings Using Feature Selection". In: *Proc. DASFAA*. Busan, South Korea, 2012, pp. 482–493. DOI: 10.1007/978-3-642-29038-1_35.

[112] Holger Trittenbach and Klemens Böhm. "Dimension-based subspace search for outlier detection". In: *Int. J. Data Sci. Anal.* 7.2 (2019), pp. 87–101. DOI: 10.1007/s41060-018-0137-7.

[113] Felix Ulrich-Oltean, Peter Nightingale, and James Alfred Walker. "Selecting SAT Encodings for Pseudo-Boolean and Linear Integer Constraints". In: *Proc. CP*. Haifa, Israel, 2022. DOI: 10.4230/LIPIcs.CP.2022.38.

[114] Sahil Verma et al. *Counterfactual Explanations for Machine Learning: A Review*. arXiv:2010.10596v3 [cs.LG]. 2022. DOI: 10.48550/arXiv.2010.10596.

[115] Sandra Wachter, Brent Mittelstadt, and Chris Russell. "Counterfactual Explanations without Opening the Black Box: Automated Decisions and the GDPR". In: *Harv. J. Law Technol.* 31.2 (2017), pp. 841–887. URL: https://jolt.law.harvard.edu/assets/articlePDFs/v31/Counterfactual-Explanations-without-Opening-the-Black-Box-Sandra-Wachter-et-al.pdf.

[116] Rico Walter and Alexander Lawrinenko. "Lower bounds and algorithms for the minimum cardinality bin covering problem". In: *Eur. J. Oper. Res.* 256.2 (2017), pp. 392–403. DOI: 10.1016/j.ejor.2016.06.068.

[117] Rico Walter, Martin Wirth, and Alexander Lawrinenko. "Improved approaches to the exact solution of the machine covering problem". In: *J. Sched.* 20.2 (2017), pp. 147–164. DOI: 10.1007/s10951-016-0477-x.

[118] Danding Wang et al. "Designing Theory-Driven User-Centric Explainable AI". In: *Proc. CHI*. Glasgow, United Kingdom, 2019. DOI: 10.1145/3290605.3300831.

[119] Jules White et al. "Automated diagnosis of feature model configurations". In: *J. Syst. Software* 83.7 (2010), pp. 1094–1107. DOI: 10.1016/j.jss.2010.02.017.

[120] Wallace Alvin Wilson. "On Semi-Metric Spaces". In: *Am. J. Math.* 53.2 (1931), pp. 361–373. DOI: 10.2307/2370790.

[121] Gerhard J. Woeginger. "A comment on scheduling two parallel machines with capacity constraints". In: *Discrete Optim.* 2.3 (2005), pp. 269–272. DOI: 10.1016/j.disopt.2005.06.005.

[122] Gerhard J. Woeginger. "A polynomial-time approximation scheme for maximizing the minimum machine completion time". In: *Oper. Res. Lett.* 20.4 (1997), pp. 149–154. DOI: 10.1016/S0167-6377(96)00055-7.

[123] Adam Woznica, Phong Nguyen, and Alexandros Kalousis. "Model Mining for Robust Feature Selection". In: *Proc. KDD*. Beijing, China, 2012, pp. 913–921. DOI: 10.1145/2339530.2339674.

[124] Jinqiang Yu et al. "Learning Optimal Decision Sets and Lists with SAT". In: *J. Artif. Intell. Res.* 72 (2021), pp. 1251–1279. DOI: 10.1613/jair.1.12719.

[125] Lei Yu and Huan Liu. "Feature Selection for High-Dimensional Data: A Fast Correlation-Based Filter Solution". In: *Proc. ICML*. Washington DC, USA, 2003, pp. 856–863. URL: https://aaai.org/papers/icml03-111-feature-selection-for-high-dimensional-data-a-fast-correlation-based-filter-solution/.

[126] Ming Yuan and Yi Lin. "Model selection and estimation in regression with grouped variables". In: *J. R. Stat. Soc. B* 68.1 (2006), pp. 49–67. DOI: 10.1111/j.1467-9868.2005.00532.x.

[127] Jilian Zhang, Kyriakos Mouratidis, and HweeHwa Pang. "Heuristic Algorithms for Balanced Multi-Way Number Partitioning". In: *Proc. IJCAI*. Barcelona, Spain, 2011, pp. 693–698. DOI: 10.5591/978-1-57735-516-8/IJCAI11-122.