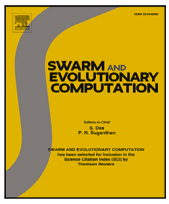
Swarm and Evolutionary Computation 101 (2026) 102288

Contents lists available at ScienceDirect

Swarm and Evolutionary Computation

journal homepage: www.elsevier.com/locate/swevo



A survey of features used for representing black-box single-objective continuous optimization

Gjorgjina Cenikj [a,b,*], Ana Nikolikj [a,b], Gašper Petelin [a,b], Niki van Stein [c], Carola Doerr [d], Tome Eftimov [a]

[a] Computer Systems Department, Jožef Stefan Institute, Ljubljana, Slovenia
[b] Jožef Stefan International Postgraduate School, Ljubljana, Slovenia
[c] Leiden Institute for Advanced Computer Science, Leiden, The Netherlands
[d] Computer Science department LIP6, Sorbonne Université, CNRS, Paris, France

ARTICLE INFO

Keywords:
Problem landscape features
Algorithm features
Problem-algorithm trajectory features
Problem classification
Algorithm selection
Algorithm configuration
Complementarity analysis

ABSTRACT

This survey examines key advancements in designing features to represent optimization problem instances, algorithm instances, and their interactions within the context of single-objective continuous black-box optimization. These features support machine learning tasks such as algorithm selection, algorithm configuration, and problem classification, and they are also used to evaluate the complementarity of benchmark problem sets. We provide a comprehensive overview of problem landscape features, algorithm features, high-level problem-algorithm interaction features, and trajectory features, including the latest works from the past five years. We also point out limitations of the current state-of-the-art and suggest directions for future research.

1. Introduction

It is widely acknowledged that different instances of an optimization problem and different performance criteria require different algorithm instances for optimal resolution. This is particularly evident in computationally difficult problems, where the choice of the algorithm used to solve it has a major impact on the quality of the final solution or the time needed to find a satisfactory one. Harnessing this performance complementarity has been the target of several approaches, with one of the most prominent ones being *per-instance algorithm selection* [1], which aims at mapping problem instances to the most suitable among a portfolio of different algorithm instances.

Per-instance algorithm selection is a challenging problem by itself, which has attracted significant research interest in recent years [1–3]. Most studies in this area use meta-learning [4] to train a supervised Machine Learning (ML) model to predict the performance of an algorithm instance or to directly predict the algorithm which is expected to work best on the input problem. This process requires input features for the ML model that are linked to the algorithm instance. The features are typically related to the problem instance landscape [5].

Other learning tasks that leverage problem landscape features include problem classification and analyzing the complementarity of benchmark problem suites. In problem classification, an ML model learns to identify *which* problem is represented or *its difficulty* based on landscape features [6–8]. A related task is to classify key problem properties like ruggedness, neutrality, and gradients. For complementarity analysis [9], unsupervised learning techniques such as clustering and dimensionality reduction are applied to evaluate the coverage of benchmark suites, which are then used to select representative learning data necessary for statistical benchmarking or developing robust ML models [10–12].

However, recently, features related to the properties/characteristics of the algorithm instances or features extracted from the interaction between an algorithm instance and a problem instance (i.e., high-level interaction features and trajectory-based features) have also been used and incorporated into various ML applications [13–16].

In this survey, we provide an overview of features used to represent problem landscapes, algorithm instances, and their interactions in the field of single-objective continuous optimization in a high-level interaction level and a trajectory-based level. We explore their applications in tasks such as algorithm selection, problem classification, and evaluating the complementarity of benchmark problem suites. Additionally, we identify research gaps and suggest directions for future development.

Throughout this survey, we use the following notation to maintain consistency across mathematical expressions. Let $\mathbf{x} = (x_1, x_2, \ldots, x_d)$ denote a candidate solution vector in a d-dimensional decision space

* Corresponding author at: Computer Systems Department, Jožef Stefan Institute, Ljubljana, Slovenia.
E-mail address: gjorgjina.cenikj@ijs.si (G. Cenikj).

https://doi.org/10.1016/j.swevo.2026.102288
Received 1 August 2025; Received in revised form 12 December 2025; Accepted 11 January 2026
Available online 14 January 2026

$\mathcal{X} \subseteq \mathbb{R}^d$, and let $f(\mathbf{x})$ represent the objective function value associated with $\mathbf{x}$. A set of n candidate solutions sampled from the search space is denoted by $\{\mathbf{x}^{(1)}, \mathbf{x}^{(2)}, \dots, \mathbf{x}^{(n)}\}$, with corresponding objective values $f(\mathbf{x}^{(1)}), f(\mathbf{x}^{(2)}), \dots, f(\mathbf{x}^{(n)})$. The trajectory of an optimization algorithm executed for a budget of b iterations (or t function evaluations), consists of all the candidate solutions explored by the algorithm $\{\mathbf{x}^{(1)}, \mathbf{x}^{(2)}, \dots, \mathbf{x}^{(t)}\}$, with corresponding objective values $f(\mathbf{x}^{(1)})$, $f(\mathbf{x}^{(2)}), \dots, f(\mathbf{x}^{(t)})$.

1.1. Relation to other surveys

While our focus is on single-objective continuous optimization, we acknowledge that a few studies have begun extending these concepts to multi-objective optimization [17,18], constrained multi-objective optimization [19,20], and mixed-integer optimization [21,22], as well as surveys focused on the automated design of algorithms [23]. However, this survey does not cover those areas, leaving room for specialized communities to present their own reviews. The methodologies for calculating and learning meta-features discussed here can also be easily adapted to inspire research in other optimization fields that currently lack extensive studies.

We also acknowledge that there are other survey papers addressing the same topic [24,25] and tutorials [18,26]. However, in this survey, we chose not to include an extensive list of references from these works (readers should check these works for a more comprehensive view). Instead, our search process and inclusion criteria were the following. We focused on recent papers and trends from the last five years, sourced from leading optimization conferences and their special sessions, such as the Genetic and Evolutionary Computation Conference (GECCO), IEEE Congress on Evolutionary Computation (IEEE CEC), Parallel Problem Solving from Nature (PPSN), and EvoStar. Additionally, we reviewed relevant journal articles from this period and examined contributions from major ML conferences, including the International Conference on Learning Representations (ICLR), International Joint Conferences on Artificial Intelligence (IJCAI), and the Conference on Neural Information Processing Systems (NeurIPS).

We included works that used optimization meta-features for at least one of the following: algorithm selection (classification, regression, or ranking), performance prediction (per-algorithm or per-instance), problem classification (problem class, difficulty, or high-level landscape properties), and complementarity or visualization of benchmark suites (for example, clustering, dimensionality reduction). We considered studies reporting results on established suites and modern generators, where feature learning or evaluation was central. We excluded works outside the primary scope (purely multi-objective, constrained multi-objective, or mixed-integer) unless they directly informed single-objective feature methods, papers without extractable experimental details (for example, no identifiable benchmarks or metrics), and optimization studies which do not include feature representations of optimization problems, algorithms, or trajectories.

Furthermore, works such as [24,25] provided valuable taxonomies but offered limited attention to algorithm features and completely omitted trajectory-based approaches, since these were not developed at the time. In contrast, our survey introduces a structured taxonomy that integrates four representation families: problem landscape features, algorithm features, high-level interaction features, and trajectory-based features, thereby broadening the scope substantially. On the other hand, tutorial contributions such as [18,26] played an important pedagogical role by explaining the basic concepts of landscape analysis and providing practical guidance on tools (for example, software for computing ELA features). However, these tutorials were not intended to provide a comprehensive synthesis of the state of the art, and they do not cover the surge of recent methods, particularly those based on deep learning (for example, CNN-based fitness maps, TransOpt, DeepELA, DoE2Vec) and trajectory-based representations (for example, DynamoRep, Opt2Vec, probing trajectories). Finally, unlike these earlier contributions, our survey emphasizes critical evaluation and open challenges. We analyze methodological issues such as over-reliance on the BBOB benchmark, lack of variance and effect size reporting, weak cross-benchmark generalization, and limited interpretability of black-box features - points that earlier surveys and tutorials did not address. We also outline forward-looking directions including invariance-aware feature design, and cross-benchmark evaluation protocols. In summary, our survey advances the field by unifying problem, algorithm, interaction, and trajectory representations, incorporating modern deep learning approaches, and highlighting methodological rigor and generalization, offering a comprehensive and up-to-date overview.

2. Machine learning pipeline using optimization features

Fig. 1 illustrates an ML pipeline that includes essential components for calculating or learning features related to problem landscapes, algorithm characteristics, and problem-algorithm interactions. It also highlights several learning tasks - such as algorithm selection, problem classification, and complementarity analysis of benchmark problem suites - where these features serve as inputs to the ML model. It is important to note that the specific features and components involved may vary depending on the particular ML task being solved.

The main components of such a pipeline are the following:

- **Problem Portfolio** - A set of optimization problem instances for which we want to find an optimal algorithm instance, analyze their coverage, or predict their interoperable properties. In the single-objective optimization domain, examples of such functions are the sphere function, ellipsoidal functions, discus, etc. These problem instances are structured in problem benchmark suites, the most commonly used ones being the Black Box Optimization Benchmarking (BBOB) [27] suite and the IEEE Congress on Evolutionary Computation (CEC) Special Sessions and Competitions on Real-Parameter Single-Objective Optimization [28–31] suites. For a sound (academic) evaluation of the ML models, the problem portfolio should cover problem instances that are diverse, challenging, and discriminating of algorithms' performances. In recent years, there has been a growing interest in the optimization field regarding the generation of new optimization problem instances in an automatic manner [32–37]. Generating novel problem instances has the potential to introduce previously unexplored optimization challenges in ongoing research.
- **Algorithm Portfolio** - A set of optimization algorithm instances that are candidates for solving an optimization problem instance. Some examples in single-objective black-box optimization are Differential Evolution [38], Covariance Matrix Adaptation Evolution Strategy (CMA-ES) [39], and Particle Swarm Optimization (PSO) [40] algorithms. In addition, modular optimization frameworks are also utilized such as ModCMA [41,42], ModDE [43], and PSO-X [44].
- **Algorithm Evaluation** - The process of running the optimization algorithm instances on the optimization problem instances and recording their performance in terms of some evaluation/performance metric. Two standard evaluation scenarios include the fixed-budget and fixed-target evaluations. In the **fixed-budget scenario,** the algorithm instance is allowed to use a predefined budget and one studies the solution quality or, if the value of the global optimum is known, the *target precision,* defined as the difference between the quality of the solution recommended by the algorithm (typically, but not necessarily, the best evaluated one) and that of the global optimum. In the **fixed-target scenario,** the algorithm is tasked with finding a solution of a certain quality, and we study the budget that is needed by the algorithms to find a solution of at least this quality. We recall that, in black-box optimization, the *budget* is often specified in terms of function evaluations or iterations, not in terms of (CPU or wall-clock) time.

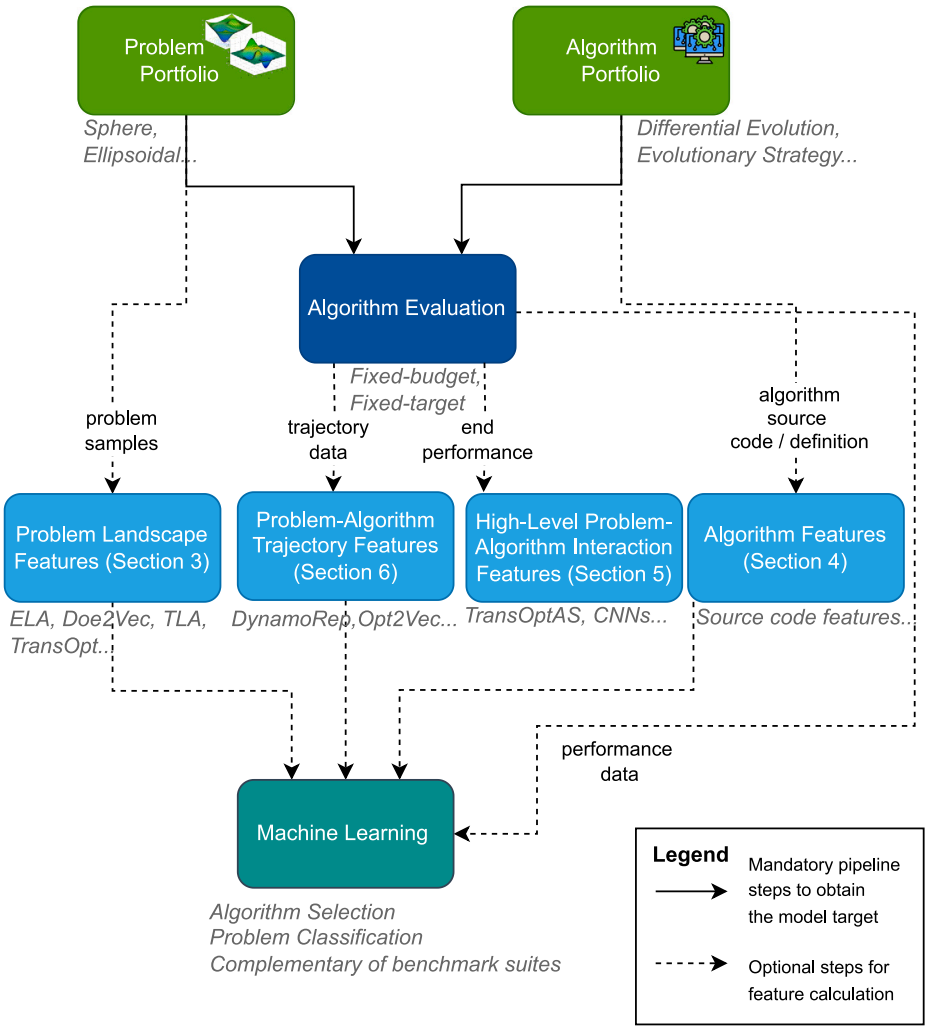


Fig. 1. A typical machine learning pipeline using optimization meta-features.

A third commonly used set of performance measure considers the *anytime performance* of the algorithms, measured through empirical cumulative distribution functions [45] or directly through empirical attainment functions [46].

- **Problem Landscape Features** - The description of an optimization problem instance in terms of numerical features. Landscape analysis is a line of research whose aim is to extract meta-features that describe the properties of the problem instances [47]. We typically distinguish between two levels of landscape features: high-level properties of the fitness landscape that are qualitative descriptors of the landscape structure (e.g., multi-modality, separability, search space homogeneity) [24] and low-level features (i.e., exploratory landscape analysis, ELA [48] such as candidate solution distribution, local search, convexity, meta-model, smoothness, ruggedness) calculated using different statistical methods from a sample of candidate solutions sampled from the decision space for a given problem instance using some sampling technique. We need to highlight here that those features are calculated to describe the whole landscape space of a problem instance, without taking care about the landscape visited/observed when an algorithm instance is run on it. Problem landscape features therefore describe intrinsic properties of optimization problems, independently of any algorithmic influence. We provide an overview of problem landscape feature construction in Section 3.
- **Algorithm Features** - These features characterize the algorithm instance, independently of the problem it is used to optimize. They are generated by analyzing the source code [49] or derived from algorithm parameters that remain constant and do not depend on the problem being solved. It is the low resourced investigated feature group. An overview of algorithm features is given in Section 4.
- **High-level Problem-Algorithm Interaction Features** - These features rely on high-level information about the optimization problems on which the algorithms are executed, rather than capturing the trajectory of the optimization runs. In essence, these features connect high-level information about a set of optimization problems with the final performance achieved by an algorithm. Unlike algorithm features, which describe only the algorithm, without any influence of the problem on which it is executed, the high-level problem-algorithm interaction features capture properties of both the problem and the algorithm.
  Different approaches exist focusing on features based on their performance [50], features based on graph embeddings [51] and graph neural networks (GNNs) [52], and features based on configuration settings for modular algorithm frameworks [53,54]. Section 5 provides an overview of the high-level problem-algorithm features.
- **Trajectory-based Features for Capturing Problem-Algorithm Interaction** - Problem landscape features characterize problem instances in terms of numerical features calculated on a sample

from the entire problem instance. This step is typically done *before* the optimization algorithm is run, meaning that additional function evaluations are required for the feature construction process. Additionally, the high-level problem-algorithm interaction features encode information about the problem landscape and the algorithm's final performance. An alternative line of work is the usage of samples generated by optimization algorithms during their execution on a problem instance. This type of representation would capture interactions between the problem instance and the optimization algorithm. An overview of trajectory-based feature constructions for capturing problem-algorithm interaction is given in Section 6. The difference between trajectory-based features and high-level problem-algorithm interaction features is in the input data used for their calculation. Both feature types are dependent on both the problem and the algorithm. However, the trajectory features are calculated using the entire trajectory of candidate solutions explored during algorithm execution. They therefore capture the search behavior of the algorithm executed on a problem. On the other hand, the high-level problem-algorithm interaction features do not use the entire trajectory as input, but instead may rely only on the final performance of the algorithm. These features therefore do not capture the search behavior, but the relation between the characteristics of a problem, and the performance of an algorithm.

- **Machine Learning** - The problem landscape features, algorithm features, high-level problem-algorithm interaction features, and trajectory-based features, or a combination of different groups can be used as input data for an ML model in different learning tasks. The three most researched tasks are presented below:
  - Algorithm Selection – One approach for training an AS model is to treat it as a *multi-class classification problem* [9], with the objective of identifying a single optimal algorithm for the specific problem instances. Alternatively, it can be treated as a *multi-label classification problem* [55], where multiple algorithms may be selected (e.g., two or three different algorithms with similar performance). Other learning scenarios include solving a *regression problem* [56–59], in which the ML model predicts the performance achieved by each algorithm, or *ranking* [60], where the aim is to rank the algorithms according to a specific evaluation metric. A comprehensive study comparing approaches for algorithm selection in single-objective optimization – through different learning tasks (multi-class classification, pairwise classification, regression) and performed with different ML algorithms (i.e., Random Forest [61], XGBoost [62], TabPFN [63], and FT-Transformer [64]) – is available in [65].
  - Problem Classification – In this task, meta-features serve as input to a model to predict either the problem represented (for problem landscape features) or the problem being solved (for problem-algorithm trajectory features) [13, 66]. Identifying this information can assist users and researchers in applying their domain knowledge to enhance the optimization process.
  - Complementarity of benchmark problem suites – In this task, various sets of problem instances are represented by problem landscape features, and their diversity is examined using unsupervised learning techniques like clustering [67], dimensionality reduction [68], and graph algorithms [10].

It is important to note that in this survey, we will focus on various methodologies used to calculate or learn problem landscape features, algorithm features, and problem-algorithm trajectory features, along with their existing applications. However, other components, such as the selection of problem and algorithm portfolios, and different approaches to ML modeling across various learning tasks, are beyond the scope of this survey.

3. Problem landscape features

In this section, we describe approaches that calculate problem landscape features using a set of candidate solutions, sampled from the decision space of the problem instance through non-adaptive or deterministic sampling techniques. These features are characterized by their aim to represent the problem instance, regardless of which algorithm is executed. Fig. 2 illustrates the general pipeline of calculating problem landscape features.

Several approaches for capturing characteristics/properties of single-objective continuous optimization problems have already been proposed. They can be broadly categorized as providing high-level features that are more interpretable and low-level features such as in the case of *Exploratory Landscape Analysis* [48], *Topological Landscape Analysis* [69], and *deep learning-based approaches* [70,71]. The latter can be learned through supervised learning (which includes algorithm performance data) or unsupervised learning (which uses only samples of the problem landscape space obtained through a sampling technique).

3.1. High-level landscape features

High-level fitness landscape analysis captures optimization problem characteristics such as the degree of global structure (none, low, medium, high), separability, variable scaling, multimodality, and search space homogeneity. For example, features related to ruggedness, neutrality, gradients, global landscape structure, deception, and searchability have been explored together with their correlations with the diversity rate of change metrics of the behavior of the particle swarm algorithm [72]. The results have shown links between particular features and PSO behavior. In contrast to some of the low-level landscape features, these features are designed to be highly interpretable.

3.2. Low-level landscape features

3.2.1. Exploratory landscape analysis

Exploratory landscape analysis (ELA) [48] is an approach to characterize black-box optimization problem instances using numerical measures that each describe a different aspect of the problem instances. ELA features can be further distinguished into *cheap* features that are computed from a fixed set of samples and *expensive* features that require additional sampling (e.g., they may require running some local search variants or other sequential sampling approaches) [73]. A convenient way to compute ELA features is provided by the *flacco* R package [74]. This package offers 343 different feature values split into 17 feature groups, including *dispersion*, *information content*, *meta-model*, *nearest better clustering*, and *principal component analysis*. Recently, a Python version of the package has been published, offering some additional set of features from local optima networks [75].

The original ELA features [48] were grouped into six categories:

- The **convexity** of an optimization problem is captured by observing the difference in the objective value of a point which is a linear combination of two randomly sampled points and the convex combination of the objective values of the two sampled points.
- The **distribution of the objective function values** is characterized by its skewness, kurtosis, and the number of peaks based on a kernel density estimation of the initial design's objective values.
- **Local search** features are extracted by running a local search algorithm and hierarchically clustering the considered solutions in order to approximate problem properties. For instance, the number of clusters is an indicator of multi-modality, while the cluster sizes are related to the basin sizes around the local optima.

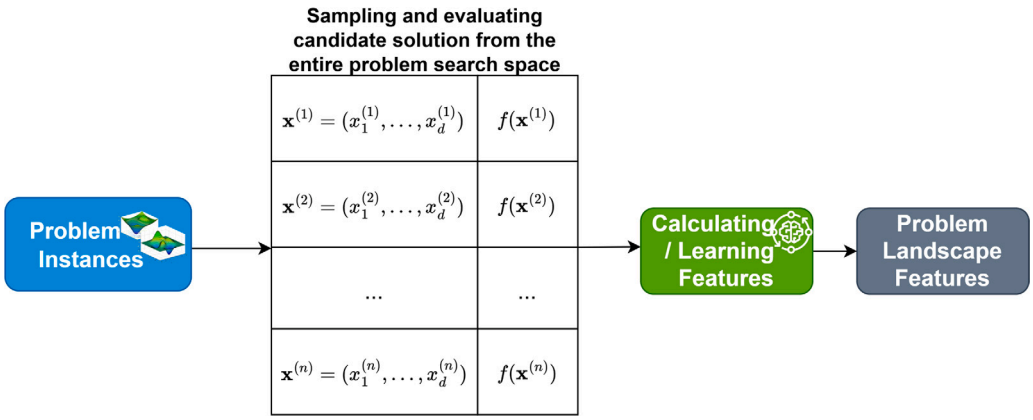


Fig. 2. The general pipeline of calculating problem landscape features.

- The **levelset** features split samples into two classes based on whether the value of the objective function falls above or below a certain threshold. Linear, quadratic, and mixture discriminant analysis is used to predict whether the objective values fall below or exceed the calculated threshold. The intuition behind this is that multi-modal functions should result in several unconnected sublevel sets for the quantile of lower values, which can only be modeled by the mixture discriminant analysis method. The extracted low-level features are based on the distribution of the resulting misclassification errors of each classifier.
- The construction of **meta-model** features requires the use of linear and quadratic regression models, with and without interaction effects, which are fitted to the sampled data. The features are then calculated by taking the adjusted coefficient of determination R2 of the models, indicating model accuracy, as well as the model coefficients. Such features are useful for extracting function characteristics, since functions with variable scaling will not allow a good fit of regression models without interaction effects, and simple unimodal functions could be approximated by a quadratic model.
- **Curvature** features estimate the gradient and Hessians from samples of the function and use their magnitudes to quantify the curvature.

Cell mapping features [76] partition the problem landscape into a finite number of cells:

- **Angle** features are based on the location of the worst and best element within each cell since opposite locations of these elements indicate a trend within a cell. More precisely, the features observe the mean and standard deviation of distances from the cell center to the best/worst element, and of the angle between these three points.
- **Convexity** features represent each cell by the element closest to the cell center and compare the objective function values of neighboring cells to capture convexity/concavity.
- **Gradient homogeneity** features compute the length of the sum of the gradients between each element and its nearest neighbor within the cell.
- **General cell mapping** features interpret the problem landscape cells as absorbing Markov chains. Cells that are visited infinitely often, once they have been visited once, are referred to as periodic cells, while all other cells are transient cells. Different types of features are then extracted based on the probability of moving from one cell to another, the number of cells of different types that appear, the size of the basins of attraction, etc.

Over time, additional ELA features have also been introduced:

- **Basic** features include minimum and maximum of the objective function, number of samples, etc. based on an initial sample of points.
- **Dispersion** features [77] compare the dispersion among all the samples and among a subset of these points.
- **Linear Model** features [74] first build several linear models with the objective variable being the dependent variable and all the remaining variables being treated as explaining variables. Based on the obtained model coefficients, statistics are calculated that are used as features.
- **Nearest Better Clustering** features [78] are computed by comparing samples to their closest neighbor and to their better closest neighbor.
- Features based on **Principal Component Analysis** [74] describe the relative amount of principal components required to explain a predetermined amount of variability.
- **Barrier Tree** features [79] describe the landscape of an optimization function using trees where saddle points are represented with regular nodes while minimums are described by leaves. Features are then generated by computing tree descriptors such as the number of levels and the distance between the leaves.
- **Information Content** features [80] try to capture information content of the landscape such as smoothness, and ruggedness. This is achieved by taking into account the distance between neighbors and changes in fitness values between them.

The success of ELA can largely be attributed to its accessibility, particularly through the ready-to-use software packages, flacco and pflacco. However, it is not a comprehensive implementation of all existing features. For instance, information epistasis [81] or autocorrelation [82], initially designed for discrete variables, can be easily adapted to continuous ones. In addition, the same is true for the use of entropy [83] as a measure.

ELA features are the most commonly used problem landscape features, however they come with a lot of limitations: their poor robustness to sampling strategy and sample size [7,11,84], and invariance to transformation in the problem space (i.e., shifting, scaling, or rotation of a problem instance) [68,85,86]. Recently, a new study highlights how many of these features are sensitive to absolute objective values (i.e., lack of shift and scale invariance), causing bias in automated algorithm selection models and hindering their ability to generalize to new problem instances [87]. To mitigate this, it has been proposed to normalize the sampled data before computing ELA features.

A potential avenue to address the sensitivity of ELA features to sampling strategy and sample size is the exploration of adaptive sampling and aggregation schemes. Instead of fixing the number of samples in advance, one could investigate starting from a small initial design (e.g., Latin hypercube or Sobol) and gradually expanding it in regions where feature estimates appear unstable. Similarly, aggregating

features across multiple resamples, possibly with normalization and variance-aware weighting, may offer more stable descriptors.

ELA features have also been used to characterize Neural Architecture Search (NAS) [88] landscapes across three typical image classification datasets: MNIST, Fashion, and CIFAR-10 [89]. This analysis reveals distinct characteristics of the NAS landscapes of these three datasets. ELA features have also been used to perform complementarity analysis of the hyperparameters optimization landscapes (HPO) and classical BBOB problems from [27], for black-box optimization [90] and for XGBoost learner [91]. Additionally, the ELA features have been used to identify cheap representative functions of 20 automotive crashworthiness problems which are expensive to evaluate [92], where it was also shown that these problems exhibit landscape features different from the classical BBOB benchmark suite.

Overall, ELA is one of the most widely used and tested approaches for characterizing black-box single-objective optimization landscapes, offering a broad set of numerical descriptors. The features are accessible in ready-to-use implementations such as *flacco* and *pflacco*. At the same time, ELA has important limitations: many features are sensitive to the sampling strategy and sample size, and several are not invariant under common transformations of the search or objective space (e.g., shifting, scaling, or rotation).

3.2.2. Topological landscape analysis

Topological Landscape Analysis [69] features characterize optimization problem instances based on Topological Data Analysis [93], which is an approach to analyzing and obtaining features from a finite set of data points, also referred to as a *point cloud*. Similarly to the ELA features, the point cloud is constructed based on a set of samples from the optimization problem. Once the samples of the objective function are collected, they are sorted by objective value and divided into multiple layers. The point cloud from each level reveals the structural characteristics specific to that layer. For each point cloud, a persistence diagram is generated, capturing the topological features within the layer. These persistence diagrams are then transformed into finite-dimensional vector representations known as *persistence images*. Persistence images enable each layer's samples to be represented as a feature vector, where pixel values from the image provide a meaningful encapsulation of the underlying structure in vector form.

This approach is especially robust to noise and supports different metrics for describing the similarity between data points. The goal of Topological Data Analysis is to discover topological structures such as spheres, torus, connected components, or even more complicated surfaces and manifolds, which persist across scales, independently of certain transformations (e.g. rotations, scaling). Discovering such structures can help characterize the point cloud using features that capture the existence of different structures across different scales.

Apart from their original version [69], an improved version called TinyTLA requiring smaller sample sizes for feature calculation has also been proposed in [94]. Here, instead of splitting the problem samples into layers, the information from the entire set of problem samples is captured using a matrix of distances between the candidate solutions ($\mathbf{x}$ values) and objective function ($f(\mathbf{x})$) values of each pair of samples. The thereby obtained distance matrix is used to further compute the persistence diagrams, persistence images, and feature representations as in the original TLA. The TinyTLA features have also been tested on the BBOB problem classification task with an extensive analysis of parameter influence and an initial evaluation for the algorithm selection task on the BBOB benchmark.

3.2.3. Deep learning-based approaches

Here, more details about low-level features learned by different types of deep neural networks (DNNs) will be provided.

Features learned by using convolutional neural networks. A set of low-level features learned by convolutional neural networks (CNNs) [95] are presented in [70,96]. To calculate them, first, for each problem instance involved in the training data (i.e., BBOB problem instances), a set of candidate solutions is generated using a Latin Hypercube sampling (other sampling techniques can also be used). The set of candidate solutions is further used to calculate a fitness map, which is a two dimensional image with a single channel in a [0,1] range. To generate the image, the objective solution values are normalized in the range of [0,1], and each candidate solution is mapped into a Cartesian plane at the location defined by the decision variables and the objective solution value defines the color range (gray-scaled) of each pixel. One weakness of this transformation is the information loss since two different candidate solutions that are close will be mapped to the same pixel. Transforming the set of candidate solutions into a two-dimensional image allows utilizing a CNN to compute low-level features for the problem instances. In [96], the CNN that has been utilized is the ShuffleNet v2 [97] due to the competitive performance achieved in object classification in computer vision. In [70], the work of using the fitness map and CNN has been extended for high-dimensional data rather than two or three dimensions. For this purpose, four dimensionality reduction techniques have been investigated to reduce the fitness map dimensionality into two dimensions that are further utilized by the CNN to learn the low-level features. Here, 24 BBOB problem classes are used with 150 problem instances per class. The training has been performed in a supervised manner in a problem classification task where the problem instances are classified into one of the three classes based on the high-level properties i.e., multimodality, global structure, and funnel structure.

Features learned using a point cloud transformer. When calculating the fitness map for high-dimensional problem instances ($d > 2$) there is a loss of information from high-dimensional to low-dimensional space. To mitigate this, point cloud transformers have also been utilized to calculate low-level features. In [70], the authors modified the original work on point cloud transformers that use the idea of convolutions that operate on the edges within a kNN-graph [98]. Here, instead of convolutions on the edges, the embedding operates on the node of the kNN-graph. This means that every candidate solution is embedded into its local neighborhood (similar candidate solutions). This has been performed since these features have been learned in a supervised manner in a problem classification task where the problem instances are classified into one of the three classes based on the high-level properties i.e., multimodality, global structure, and funnel structure.

DoE2Vec features learned using a variational autoencoder. DoE2Vec features [99] are low-level features learned in an unsupervised manner with a variational autoencoder (VAE) [100]. To learn them, first, a dataset of candidate solutions, initially within the domain of $[0,1]^d$ (d is the problem dimension) is generated using a sampling technique. Next, the candidate solutions are evaluated using a set of problem instances that are randomly generated using the random function generator from [101], which is a modification of the generator proposed in [33]. This evaluation allows the collection of training data with different complexity that covers a wide range of problem instances. Further, all objective solution values are first re-scaled within the range of [0,1] and are then used as input features to train the VAE. These features not only demonstrate promise in identifying similar surrogate problem instances that are inexpensive to evaluate but also have the potential to significantly enhance performance when used alongside traditional ELA features in problem instance classification tasks.

TransOpt features learned using a transformer. TransOpt features [102] are low-level features learned in a supervised manner by using a transformer architecture [103]. The original transformer architecture's core lies in its encoder–decoder structure, comprising multiple identical blocks for both the encoder and decoder. In the case of TransOpt

features, only the encoder has been utilized, consisting of a multi-head self-attention module and a position-wise feed-forward network. The input data for training the transformer are generated candidate solutions for each problem instance, obtained using Latin Hypercube sampling. These samples are provided to the transformer encoder, which produces a representation for each sample. A representation of the problem is then obtained by aggregating the sample representations.

This architecture has been trained in a supervised manner on two different tasks. In [102], the model is trained on the BBOB benchmark, where the task is to identify to which of the 24 BBOB problem classes an instance belongs. In this case, 24 problem classes with 999 instances (i.e., shifted, scaled) per each problem class and per dimension $d \in \{3, 5, 10, 20\}$ are used. The tested sample sizes are $50d$ and $100d$. The model consists of the previously mentioned encoder and a classification head trained to predict the 24 problem classes. It has been shown that TransOpt features can achieve accuracy rates ranging from 70% to 80% when tasked with identifying problem classes across various problem dimensions. It is important to note that even though the model is trained on the problem classification task, it can produce feature representations for unseen problem instances which can be used in different downstream tasks. A demonstrative example of this is shown in [104], where the TransOpt model is used to generate problem representations on top of which a random forest model [61] is trained for the algorithm selection task.

Deep-ELA features learned using a transformer. Deep-ELA [105] is a methodology involving the unsupervised training of transformer models to produce representations of optimization problems which are invariant to problem transformations. This is accomplished by training the model to produce the same representation for a problem instance and its augmented variant. The augmentation is accomplished through transformations that do not alter the underlying optimization problem, such as rotations and inversions of decision variables and randomization of the sequence of decision and objective variables.

Four transformers have been pre-trained on large sets of randomly generated optimization problems to grasp intricate representations of continuous single- and multi-objective landscapes. The Deep-ELA features have been evaluated in the problem classification task classifying the BBOB single-objective problem based on their high-level characteristics defined in the benchmark suite. In addition, they have been evaluated in automated algorithm selection tasks on the BBOB benchmark suite and four different multi-objective benchmark suites. A study evaluating the complementarity of Deep-ELA and classical ELA features was recently presented in [106], with a following study involving TransOpt features [107].

Random filter mappings. As opposed to using trained deep learning models to extract problem landscape features, [108] proposes the extraction of features using randomly initialized filters. The procedure starts with the initialization of a set of filters with random sizes, weights and radii. Each individual filter is then applied at randomly selected anchor points (samples of the objective function). In particular, the filter application involves the multiplication of the filter weights with the pairwise distances of a subset of the problem samples which are in the vicinity of the anchor point (have a distance which is less than the dimension-adjusted radius of the filter). The thereby obtained filter responses are aggregated to obtain the problem landscape features. These features have been evaluated for the algorithm selection task, as well as the task of problem classification of the BBOB problem instances into the 24 problem classes, and recognizing their high-level properties.

3.3. Advantages vs. Disadvantages of problem landscape features

After summarizing the latest trends in problem landscape features, Table 1 presents their advantages and disadvantages.

3.4. Summary of problem landscape features

Table 2 demonstrates a summary of the type of training tasks and inputs required for each problem landscape feature group. Additionally, the last column contains information about how the features are calculated in different studies. In particular, we report the size of the sample used for feature calculation, the number of repetitions performed when calculating the features (since some of them are stochastic and some studies perform multiple runs and aggregate the feature values), and the dimension of the problems on which the features are tested. Please note that if the study did not explicitly report that multiple repetitions were performed, we assume that the features were calculated once. Furthermore, please note that for the ELA features, we do not include an exhaustive list of studies where the features were used, due to an overwhelmingly high number of such studies. Instead, we chose 10 studies from different authors as examples. From the table, we can see that there is a lack of consistency in feature calculation. First, there is substantial variation in sample sizes between studies. Similarly, the number of repetitions varies widely: some studies rely on a single run, while others average results across 10, 15, or even 100 repetitions to mitigate stochastic variability. Another clear observation is that most empirical work has been conducted on low-dimensional problems, typically up to $d = 10$ or $d = 20$, with a few exceptions exploring higher dimensions. This suggests that while problem landscape features have been extensively explored in principle, their evaluation has been constrained to relatively small search spaces. This highlights the need for systematic evaluations of feature robustness in higher-dimensional settings, where sampling becomes more expensive but also more informative.

Future work should test how features scale across different dimensionalities, analyze their stability across varying sample sizes, and assess sensitivity to function transformations, stochasticity and repetition strategies. Moreover, standardized protocols are needed on (i) how many samples to draw per dimension, (ii) how many repetitions are required for statistically reliable estimates, and (iii) which dimensions should be included in benchmarking studies. While some studies have already addressed the sensitivity of ELA features to function transformations [85], sampling strategies and sample sizes [11,86], these questions remain relatively underexplored for more recent feature groups.

4. Algorithm features

Algorithm representations are focused on calculating features that characterize the algorithm instance. While the development of algorithm features has garnered somewhat lower research interest compared to the development of problem landscape features, in this section we present the features derived directly from the source code.

Algorithm features based on source code. The extraction of features from the algorithm source code and from the abstract syntax tree has been proposed in [49]. The features extracted from the source code capture the cyclomatic complexity, maximum indent complexity, number of lines of code, and size in bytes for the entire source code as well as aggregations of these properties across different regions of the code. On the other hand, the features extracted from the abstract syntax tree involve the conversion of the abstract syntax tree obtained during the compilation of a given algorithm into a graph representation and extracting various graph properties such as node count, edge count, transitivity, node degree, clustering coefficient, depth, etc.

In addition to source code-derived metrics, potential algorithm representations could also capture functional and structural characteristics that directly reflect the design choices of metaheuristics. For example, they could include the types of variation operators employed (e.g., mutation, crossover, or recombination), the selection and replacement mechanisms (e.g., tournament, rank-based, elitist strategies),

Table 1
Advantages vs. disadvantages of the different problem landscape feature sets.

Problem landscape features	Advantages	Disadvantages
ELA	- Easy use through R, Python, and GUI - Most commonly-used problem landscape features - Explainable to some degree	- Sensitive to sampling strategy and sample size - Noninvariant to problem shifting, scaling, and rotation - Poor generalization on new unseen instances when plugged into an ML model - Computationally costly
TLA	Invariant to shifting, scaling, and rotation	- Black-box features - Can produce a very high-dimensional feature vector, may require dimensionality reduction before usage
Learned by CNNs	- More applicable for two and three dimensional problems - Does not require any feature engineering	- Black-box features - Loss of high-dimensional information by representing the data as a two dimensional fitness map - Depends on the problem portfolio used for learning
Learned by point cloud transformer	- Applicable to all problem dimensions, though may require model retraining for higher dimensions	- Black-box features - Depends on the problem portfolios used for learning
Doe2Vec	- Does not require any feature engineering	- Black-box features
TransOpt	Does not require any feature engineering	- Black-box features
Deep-ELA	- Does not require any feature engineering	- Black-box features
Random filter mappings	- Does not require any feature engineering	- Black-box features - May produce irrelevant features

and the presence of restart or perturbation policies. Another important dimension is parameter adaptivity. Algorithms may embed self-adaptive mechanisms (e.g., step-size control in evolution strategies, cooling schedules in simulated annealing) or hyper-heuristic controllers that dynamically adjust parameters during the search. Representing whether and how such adaptivity is employed could offer valuable insights into algorithm behavior. Please note that these are potential research directions and not existing works.

4.1. Advantages vs. Disadvantages of algorithm features

Table 3 presents their advantages (pros) and disadvantages (cons) of the algorithm features based on the source code.

5. High-level problem-algorithm interaction features

These features are based on general characteristics of the optimization problems where the algorithms are applied, instead of describing how the algorithm's search process unfolds over time. Essentially, they establish a link between the problem's high-level properties and the algorithm's final performance, but they do not capture details of the algorithm's behavior during its run. Fig. 3 presents the general pipeline of calculating high-level problem-algorithm landscape features.

Features learned by using convolutional neural networks. CNN-based features were investigated in [70,96], and their details were previously discussed in Section 3. The key difference here lies in the supervised learning task. For problem landscape features, the supervised task involved classifying problem instances based on high-level properties without considering any algorithm interaction. In contrast, in this case, the model was trained for algorithm selection, aiming to choose the most suitable modular CMA-ES [39] configuration from a portfolio of 32 for each problem instance. The latent representations obtained from the CNN are subsequently used as low-level features.

TransOptAS features learned using a transformer. Similar to TransOpt features presented in Section [102], where the transformer has been trained in a problem classification task, in [71], the transformer architecture is directly trained on the algorithm selection task. In this case, the model consists of an encoder and a regression head that predicts a numerical performance indicator for 12 configurations of the PSO algorithm.

Features based on performance. Performance2Vec [50] is a methodology for constructing features that describe problem-algorithm interaction based on the performance achieved on a set of benchmark problems. Here, the vector representation consists of the algorithm instance performance obtained on each benchmark problem separately, so they can be assumed to be different features. Algorithms' performance can be defined in terms of different metrics, using simple statistics like mean or median, or more complex ranking schemes like Deep Statistical Comparison [115].

Features based on problem landscape features used by performance prediction models. This type of representation is generated by observing the importance assigned to ELA features by explainability methods applied to ML models trained for performance prediction. It involves the use of the SHAP [116] method, which can assign feature importance on a *global* level (i.e., on a set of problem instances) and on a *local* level (i.e., on a particular problem instance). A single-target regression (STR) model is trained to predict the performance of the algorithm instance using the training problem instances described by their ELA features. The model is then used for making predictions on the problem instances

Table 2
Summary of inputs, training task types and sample sizes used in the literature for problem landscape features.

Feature name	Training task type	Inputs	Sample
ELA	manually designed	Samples of optimization problems	- 250d samples; 100 repetitions; $d \in \{5\}$ [109] - 500d samples; 10 repetitions; $d \in \{2\}$ [76] - 250d samples; 10 repetitions; $d \in \{10\}$ [9] - 6d, 60d, 625d samples; $d \in \{5\}$ [84] - 1000d samples; 15 repetitions; $d \in \{2, 3, 5, 10, 20\}$ [110] - 1000d samples; 1 repetition; $d \in \{2, 5\}$ [37] - 50d, 100d samples; 1 repetition; $d \in \{3, 10\}$ [104] - 250d samples; 1 repetition; $d \in \{5\}$ [111] - 50d, 100d samples; 10 repetitions; $d \in \{2, 3, 5, 10\}$ [112] - 250d samples; 1 repetition; $d \in \{5, 10\}$ [113]
TLA	manually designed	Samples of optimization problems	- 10, 40, 100, 200, 400, 600 samples; 1 repetition; $d \in \{2, 3, 5, 10\}$ [94] - 50d samples, 1 repetition, $d \in \{10\}$ [114]
Fitness Map + CNNs	supervised	Samples of optimization problems, High-level problem properties	- 100, 500 samples; 10 repetitions; $d \in \{2, 3, 5, 10\}$ [70]
Point Cloud Transformer	supervised	Samples of optimization problems, High-level problem properties	- 100, 500 samples; 10 repetitions; $d \in \{2, 3, 5, 10\}$ [70]
DoE2Vec	self-supervised	Samples of optimization problems	- 256 samples; 1 repetition; $d \in \{5\}$ [99] - 50d samples; 1 repetition; $d \in \{10\}$problems [114]
TransOpt	supervised	Samples of optimization problems, Problem class	- 50d, 100d samples; 1 repetition; $d \in \{3, 20\}$ [102] - 50d, 100d samples; 1 repetition; $d \in \{3, 10\}$ [104]
Deep-ELA	self-supervised	Samples of optimization problems	- 25d, 50d samples; 1 repetition; $d \in \{2, 3, 5, 10\}$ [105] - 50d samples; 1 repetition; $d \in \{10\}$ [114] - 25d, 50d samples; 1 repetition; $d \in \{2, 3, 5, 10\}$ [106] - 50d samples; 1 repetition; $d \in \{3, 10\}$ [107]
Random Filter Mappings	manually designed	Samples of optimization problems	- 200d samples; 1 repetition; $d \in \{2, 3, 5, 10\}$ [108]

Table 3
Advantages vs. disadvantages of algorithm features.

Algorithms features	Advantages	Disadvantages
Based on source code	- May be used to compare different programming implementations of the algorithms and further investigate which one has better performance	- These features are ineffective for automated algorithm configuration or parameter tuning, as parameter differences are typically evident only during execution, not in the code. - Features extracted from the source code depend highly on the programming language and the specific implementation, leading to potential discrepancies even for the same algorithm.

from the test dataset. Next, using the SHAP method, the Shapley values that are the contributions of each landscape feature to the performance prediction are calculated. If the input to the model are p landscape features, averaging the Shapley values across all problems for each landscape feature separately gives a p-dimensional meta-representation of the algorithm behavior. This type of representation encodes the interactions between the landscape features and the algorithm performance. Such representations have been previously used to find algorithm instances with similar behavior on a set of benchmark problem instances [59] and to understand which module from a modular CMA-ES [39] is active based on the performance data [57].

Features using graph embeddings. Features can also be constructed by leveraging their interactions with different entities from the optimization domain. In [51], the performance prediction task was addressed as a link prediction task, which aims to identify if an algorithm can solve a given problem instance with some specified error. In this case, the methodology involved the construction of a knowledge graph (KG), where some of the nodes describe aspects of the problem instances (their corresponding problem class, high-level features, and ELA features), while other nodes are related to algorithm descriptions (their parameters). The algorithm and problem instance nodes are linked if the algorithm can solve the problem instance with some specified error threshold. Such a representation of the algorithm and problem instance properties allows the application of standard knowledge graph embeddings for deriving representations of the problem instances and algorithm instances, which can then be used for performance prediction. We need to highlight here that this approach of learning features can produce either algorithm features if the node is representing an algorithm instance or problem instance features if the node is representing a problem instance. In the case of problem instance features, they differ from the low-level landscape features described in Section 3 since they fuse information not only for the landscape but also consider the interaction of an algorithm instance on each problem instance through its end performance.

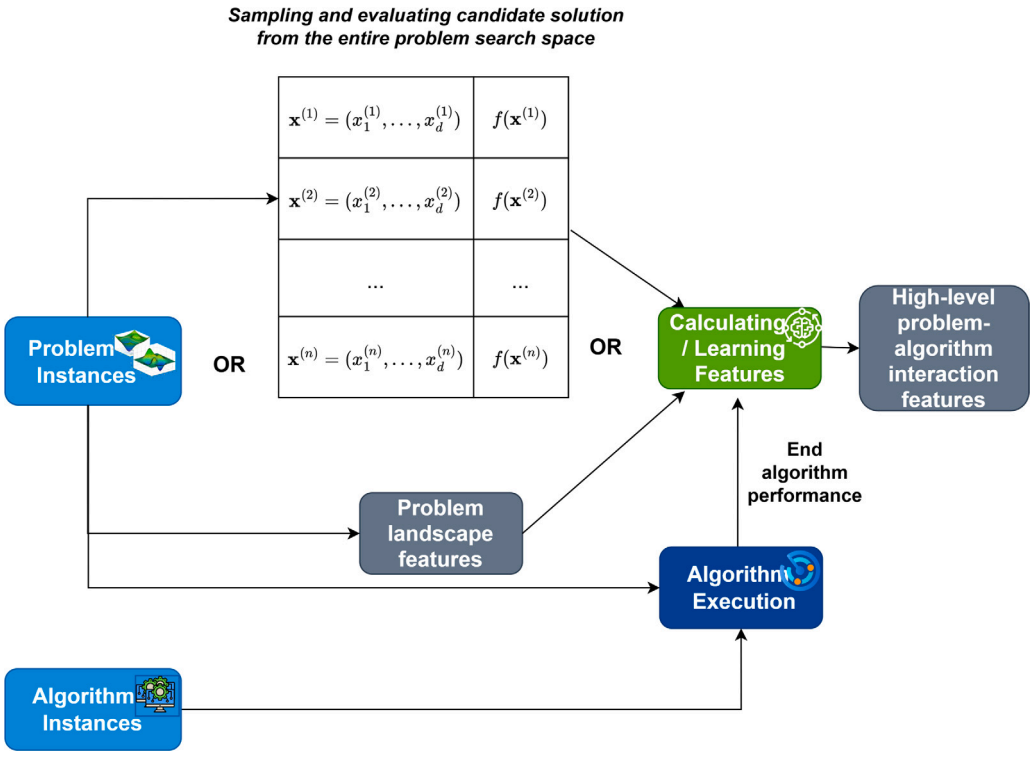


Fig. 3. The general pipeline for calculating high-level problem-algorithm interaction features.

Features using graph neural network. Parameters describing algorithm configurations, which significantly influence performance, were often overlooked by the techniques of learning algorithm features. To address this, the complex structure formed by the relationships between algorithm operators/modules, parameters, problem characteristics, and performance outcomes was represented as a graph [52]. Heterogeneous graph data structures and graph neural networks were explored to enable the prediction of optimization algorithm performance by capturing the complex dependencies between problems, algorithm configurations, and performance outcomes. Two modular frameworks, modCMA-ES and modDE, were considered. Improvements of up to 36.6% in mean square error in algorithm performance prediction over traditional tabular-based methods were achieved, and the potential of geometric learning in black-box optimization was demonstrated. This approach generates an algorithm and problem landscape features depending on the node of interest.

Features based on configuration settings for modular algorithm frameworks. Another approach for extracting features from algorithm configurations involves using the importance or contribution of operators and modules to the performance of the algorithm configuration. Two approaches are presented here:

- Features based on functional ANOVA (fANOVA) - Problem class-specific datasets are initially generated, in which modules are used as features of the modular algorithm variants, with numerous variants incorporating different module values being executed [53]. The corresponding performance on the problem class is treated as the target within each dataset, thereby capturing the relationship between the algorithmic modules and performance in that specific problem class. These datasets are then provided as input to f-ANOVA [117], through which the variance in performance is decomposed and systematically attributed to individual modules and their combinations. When variants are defined by m modules, the analysis quantifies the effect (i.e., importance) on performance for m individual modules, $\binom{m}{2}$ pairwise module interactions, and $\binom{m}{3}$ triple module interactions. Estimation of higher-order interactions is considered computationally expensive. The complete set of effects (singles, pairs, and triplets) is subsequently used as a feature vector, and by comparing these vectors across different problems, problems can be identified in which the modular frameworks exhibit similar module interactions that describe their performance.
- Features based on Shapley analysis - Similar to features generated by fANOVA, Shapley analysis can also be used to estimate the contribution of each module to performance [118]. However, a limitation of this approach is that only the contributions of individual modules are calculated, as estimating higher-order contributions is computationally expensive.

5.1. Advantages vs. Disadvantages of high-level problem-algorithm interaction features

After summarizing the latest trends in high-level problem-algorithm interaction features, Table 4 presents their advantages (pros) and disadvantages (cons).

5.2. Summary of high-level problem algorithm interaction features

Table 5 provides a structured overview of the high-level problem-algorithm interaction features studied in the literature, focusing on their training task type, the inputs required, and the sample sizes reported. Unlike problem landscape features, which primarily rely on sampled candidate solutions evaluated on the objective function, these interaction features explicitly incorporate information about algorithm behavior and performance.

The table illustrates that a variety of training paradigms are used. With respect to inputs, these features require richer information than problem landscape features. For example, Performance2Vec uses only algorithm performance vectors, whereas KG and GNN embeddings combine algorithm performance, ELA features, and configuration parameters. This integration makes them more expressive but also more expensive to compute, as they often depend on preliminary feature calculations or repeated algorithm runs.

Table 4
Advantages vs. disadvantages of high-level problem-algorithm interaction features.

High-Level p-a features	Advantages	Disadvantages
Learned by CNNs	- More applicable for two-dimensional and three-dimensional problems - Does not require any feature engineering	- Black-box features - Loss of high-dimensional information by representing the data as a two-dimensional fitness map - Depends on the algorithm portfolio used for learning
TransOptAs	- Does not require any feature engineering	- Black-box features - Depends on the algorithm portfolio used for learning - Requires algorithm execution for obtaining algorithm performance labels used for supervised training
Based on performance	- Facilitates algorithm comparison through performance vectors	- Biased to the selected portfolio of benchmark problems
Based on Shapley values	- Encodes interactions between problem landscape features and algorithm performance - Used to find similar algorithm behaviors with the assumption that the predictive models behave similarly	- Depends on the selected problem landscape features portfolio - Depends on the selected benchmark problem instances - The dependencies on the surrogate (STR) model
via Knowledge Graph	- Encodes interactions between problem landscape features and algorithm performance by also involving the graph neighborhood	- Depends on the data stored in the KG and the KG embedding method used for learning
via GNNs	- Encodes interactions between problem landscape features, algorithm configuration, and algorithm performance by involving the graph neighborhood	- Depends on the data stored in the graph and the GNN method used for learning
Based on fANOVA	- For each problem, they encode the contributions of modules and their interactions to the final performance of an algorithm.	- Depends on the available data from the pool of different configurations.
Based on SHAP	- For each problem, they encode the contributions of each separate module to the final performance of an algorithm.	- Depends on the available data from the pool of different configurations.

Table 5
Summary of inputs, training task type and sample sizes used in the literature for high-level problem-algorithm interaction features.

Feature name	Training task type	Inputs	Sample
Fitness Map + CNNs	supervised	Samples of optimization problems, algorithm performance	- 50d samples; 1 repetition; $d \in \{2\}$ [96]
TransOptAS	supervised	Samples of optimization problems, algorithm performance	- 50d samples; 1 repetition ; $d \in \{3, 10\}$ [71] - 50d samples; 1 repetition ; $d \in \{10\}$ [114]
Performance2Vec	manually designed	Algorithm performance	- No samples used; $d \in \{10\}$ [50]
Explainable Prediction Models	supervised	Algorithm performance, calculated ELA features	- 800d samples; 30 repetitions; $d \in 10$ [59]
Internal Algorithm Parameters	unsupervised	Time series of algorithm parameters during runs	- No samples used; $d \in \{5\}$ [15]
KG embeddings	self-supervised	Algorithm performance, calculated ELA features, algorithm linked with configuration parameters	- 100d samples; 100 repetitions; $d \in \{5, 30\}$ (for ELA feature calculation) [51]
GNN embeddings	supervised	Algorithm performance, calculated ELA features, algorithm linked with configuration parameters	- 100d samples; 100 repetitions; $d \in \{5, 30\}$ (for ELA feature calculation) [52]
fANOVA	manually designed	Algorithm performance	- No samples used; $d \in \{5, 30\}$ [53]

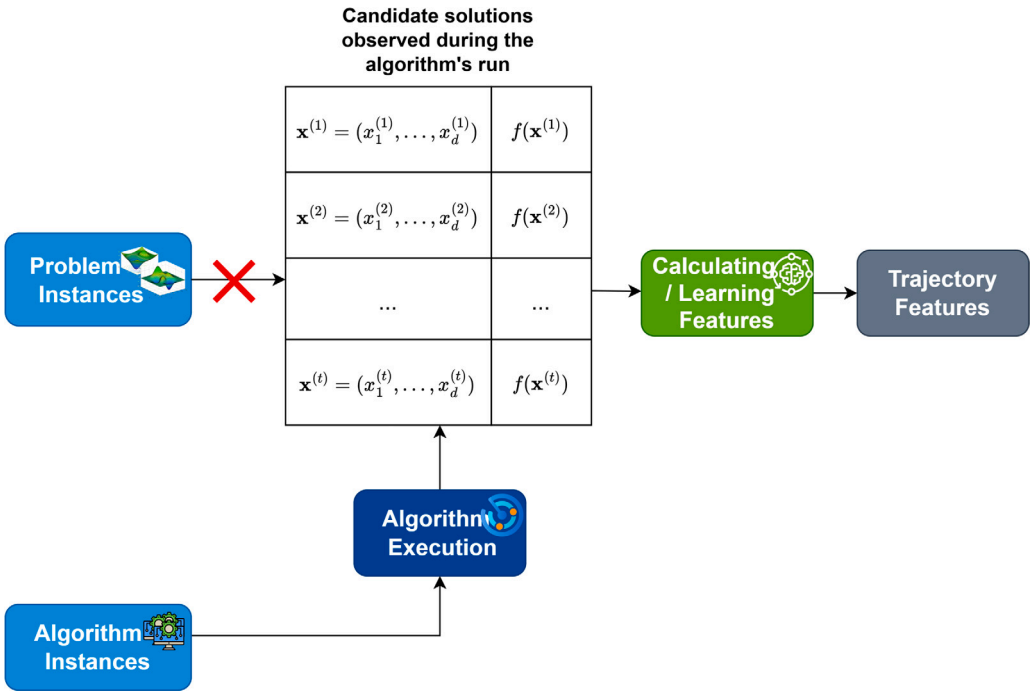


Fig. 4. The general pipeline of calculating problem-algorithm trajectory features.

6. Trajectory-based features that capture problem-algorithm interaction

The previously mentioned approaches for problem landscape features compute low-level features by generating an artificial set of candidate solutions using a sampling strategy that spread well across the whole decision space. They are hence not connected to the actual set of candidate solutions explored by an optimization algorithm instance *during* its optimization process. To address this limitation, several studies have undertaken automated algorithm selection on a per-instance and per-run basis by calculating *trajectory-based features* that represent the algorithm behavior during its run. Below, we are going to provide more details about different trajectory-based approaches for learning features that capture problem-algorithm interaction. Fig. 4 presents the general pipeline of calculating problem-algorithm trajectory features.

Trajectory-based features based on internal algorithm parameters. Some optimization algorithms contain internal parameters that are adapted throughout the optimization process, such as the step size, the evolution path, the conjugate evolution path, and the square root of the diagonal matrix holding the eigenvalues of the covariance matrix. Building on features suggested in [119], the work [15] studied the use of time-series features (using the *tsfresh* [120] library) on top of these algorithm-internal parameters. These time series features were used to identify the configuration of modular CMA-ES variants.

Trajectory-based ELA features. Trajectory-based ELA features are computed from a set of candidate solutions that are derived from the entire algorithm trajectory, rather than from a standard sampling technique (such as Latin Hypercube Sampling [121], random sampling, etc.) that is applied to the objective function. This means that the samples are extracted from the populations (i.e., sets of candidate solutions and their corresponding function values) that are generated/observed by the optimization algorithm during its execution. The trajectory-based ELA features have been first proposed in [122], under the name of *Adaptive Landscape Analysis*. In that study, the ELA features are computed using samples from the distribution that the algorithm CMA-ES uses to sample its solution candidates. The approach has been evaluated for fixed-budget performance prediction [123] of the CMA-ES [39], as well as *per-run* algorithm selection with warm-starting [124], where the features extracted from the trajectory of an initial optimization algorithm instance are used to determine whether to switch to a different algorithm instance. We need to highlight here that the trajectory-based ELA features use a subset of all candidate solutions explored by the algorithm instance during its run, without considering the iteration in which these solutions were generated. These features do therefore not capture the longitudinality of the solutions that are observed within the iterations of the algorithm execution. Another study that uses the trajectory-based ELA has been conducted to analyze the benefits of predicting the switching between different algorithm instances during the optimization trajectory run [125]. It has been also shown that they can be used for dynamic selection of the acquisition function of Bayesian Optimization [126] algorithms, improving over default static choices [127].

DynamoRep features. DynamoRep features [13,128] are capturing the problem-algorithm interaction using simple descriptive statistics extracted from the populations explored by the algorithm in each iteration of its execution on a problem instance. In particular, the minimum, maximum, mean, and standard deviation of each of the candidate solutions and the objective function values of each population are extracted. A representation of the entire trajectory is then generated by concatenating the values of these statistics extracted for each population. If the algorithm instance is run for b iterations until the stopping criteria are met on a problem instance of dimension d (i.e., number of decision variables), the entire algorithm trajectory representation would then have a size of $4b(d+1)$. The $+1$ is coming from the objective function value. DynamoRep features have been used for algorithm selection, problem classification and algorithm classification [128].

Opt2Vec features. The trajectory-based ELA and DynamoRep features are designed to analyze and represent the entire trajectory of an optimization run. However, an important aspect is the information contained within a small, specific segment of the optimization trajectory, such as a particular timestamp, iteration, or population. This is especially crucial when it comes to optimizing dynamic algorithms efficiently. For this purpose, Opt2Vec [66] features propose the usage of autoencoders to encode the information found in the interaction

between an optimization algorithm and optimization problem into an embedded subspace. These representations take the populations considered by an optimization algorithm instance in each iteration of its execution, scale the candidate solutions and objective function values, and identify a subset of informative populations. The subset of informative populations is obtained by calculating the Frobenius norm on the differences of the matrices representing consecutive populations of the algorithm's execution, and discarding one of the populations if the Forbenius norm is smaller than a predefined threshold. The proposed approach is designed to capture features for parts of the optimization problem's search space that the algorithm explores at a specific timestamp or iteration during the optimization process. The Opt2Vec features are evaluated for the task of classifying problems from the CEC benchmark suite. In particular, 30 CEC problems in three different dimensions (10,30,50) are used. Different dimensionalities of the same problem are treated as different problems, resulting in 90 problem classes used in the classification task. The Opt2Vec features achieve classification accuracies over 80% for the recognition of the problem being solved and its dimension.

Iterative-based ELA features. An indirect outcome of the Opt2vec and DynamoRep studies are the iterative-based ELA features, which involve calculating the ELA features on each population separately, and concatenating the population representations to generate a representation for the entire trajectory. In this way, the representations would capture the dynamics and behavior of the algorithm on the problem landscape, however, the dimension of the generated representation (i.e., number of features) grows linearly with the number of iterations for which the algorithm is run. The iterative ELA features have been evaluated for algorithm selection, problem classification and algorithm classification [14,128].

Local optima networks. Local Optima Networks (LONs) [16] serve as a simplified model for representing discrete (combinatorial) fitness landscapes, where local optima are nodes and search transitions are edges defined by an exploration search operator. They represent the number of local optima in the landscape, as well as their distribution and connectivity patterns. Monotonic LONs (MLONs) are a variant of LONs where transitions between local optima are considered only if fitness is non-deteriorating. To account for neutrality in the level of local optima transitions, i.e., connected components in the MLONs that have the same fitness value, compressed MLONs (CMLONs) have been proposed [129]. A compressed local optimum can be described as a grouping of interconnected nodes within the MLONs, all sharing the same fitness value. While originally proposed for combinatorial problems, LONs have been adapted for the continuous domain in [129]. In this case, a Basin-Hoping algorithm [130] is used to identify the local optima, and the MLONs and CMLONs are adapted for the continuous problems.

CMLONs have been utilized to visualize the 24 BBOB problem classes across different problem dimensions [131]. Network metrics are also calculated for each CMLON, and dimensionality reduction techniques are employed to classify and compare these problems. The findings reveal that CMLONs exhibit varied representations that are linked to the inherent properties of the problem instances and their dimensionality. Network metrics prove particularly crucial for multimodal problems in higher dimensions, where CMLONs become too intricate for meaningful visual interpretation.

Similar to LONs are the Search Trajectory Networks (STNs) [132], which are novel tools to study and visualize how population-based algorithm instances behave in continuous spaces. Inspired by LONs, which map the nodes to local optima, in STNs the nodes focus on different states from the optimization trajectory not limited to local optima. The edges signify the progression between these states. This expansion enhances network-based models' utility for understanding heuristic search algorithms. However, they have not yet been evaluated as possible sources for learning trajectory-based features.

Probing trajectories. In a recent study, researchers introduced an algorithm selector method that utilizes short probing trajectories generated by running an initial algorithm on a problem instance for a brief period [133]. These trajectories are then used to determine either the current fitness or the best-so-far fitness, which are tracked across predefined sequential iterations of the initial algorithm. Following this, time-series features are extracted using the *tsfresh* library. These time-series features are then used as input for a Random Forest classifier [61] to train the algorithm selector. Additionally, the raw probing trajectories are fed into a Rotation Forest classifier for the same purpose. This approach shows promising results comparable to those achieved using trajectory-based ELA features extracted from observed candidate solutions and a Random Forest model.

ClustOpt. A representation learning and visualization methodology is proposed [134], in which solution candidates explored by the algorithm are clustered, and the evolution of cluster memberships across iterations is tracked. In this way, a dynamic and interpretable view of the search process is provided. Additionally, two metrics — algorithm stability and algorithm similarity — are introduced to quantify the consistency of search trajectories across runs of an individual algorithm and the similarity between different algorithms, respectively. This methodology is applied to a set of ten numerical metaheuristic algorithms, through which insights into their stability and comparative behaviors are revealed, thereby enabling a deeper understanding of their search dynamics.

6.1. Advantages vs. Disadvantages of trajectory features

After summarizing the latest trends in problem-algorithm trajectory features, Table 6 presents their advantages and disadvantages.

6.2. Summary of trajectory features

Table 7 summarizes the trajectory-based feature families explored in the literature, highlighting their training task type, the inputs required, and the problem dimensions on which they have been tested. Unlike problem landscape and high-level interaction features, trajectory features explicitly exploit the sequence of solutions evaluated by an algorithm during its search process, making them especially suitable for dynamic, run-dependent characterizations of algorithm behavior. Please note that since trajectory features are based on the solutions sampled by the algorithm, we do not include the sampling information as we did for the problem landscape features.

The table shows that most trajectory-based approaches are manually designed, such as Trajectory-ELA, Iterative-ELA, LON, and Probing trajectories, where handcrafted descriptors are extracted from algorithm runs. More recent approaches, such as Opt2Vec, leverage self-supervised learning by embedding trajectories into vector spaces, while methods like ClustOpt apply unsupervised learning to cluster trajectories. This diversity reflects a growing trend to move beyond static problem landscape features toward representations that capture algorithm dynamics.

In terms of inputs, most methods require potential solutions evaluated by an algorithm during its search process, often using population-based algorithms to generate diverse trajectories. An exception is LONs, which are built from multiple runs of a local search algorithm and rely on detecting local optima and their connectivity.

The problem dimensions tested in the literature remain relatively limited. Many studies focus on low to moderate dimensions (e.g., $d \leq 20$), with some extending to $d = 30$ or $d = 50$ in the case of Opt2Vec and Iterative-ELA. This indicates that trajectory features are still predominantly explored in small-scale benchmarks, and their scalability to higher-dimensional optimization problems remains largely untested.

Table 6
Advantages vs. disadvantages of trajectory features.

Trajectory features	Advantages	Disadvantages
Based on internal parameters	- Capture the behavior of the algorithm	- Lack of comprehensive comparison of different time series features - Do not enable a comparison of different algorithms (with different configurations), limited to comparing configurations of the same algorithm
Trajectory-based ELA	- Info about the interaction across problem and algorithms (personalization)	- Does not capture the longitudinal aspect of solutions within algorithm iterations
Iterative-based ELA	- Info about a single timestamp of the optimization process, can easily be combined with ML models that will capture the longitudinality of the search process	- ELA features are sensitive on small sample sizes
DynamoRep	- cheap to compute - Despite lower computational cost, DynamoRep features yield similar performance as ELA features that are calculated at each iteration of the algorithm's execution	- Limited expressiveness - Representation size grows with number of iterations and problem dimension, may require dimensionality reduction as preprocessing step
Opt2vec	- Capture features specific to parts of the search space explored at a particular iteration - Takes into consideration the optimization problem dimension since one model is used for all problem dimensions.	- Depends on the data used to train the autoencoder - Black-box features
LON and variants	- Useful for visualization purposes	- Very costly to compute
Probing trajectories	- Potential to be utilized for per-run algorithm selection	- Requires suitably chosen probing algorithms - Difficult to transfer results from one probing algorithm to another
ClustOpt	- Useful for algorithm behavioral analysis	- Depends on the clustering algorithm and its parameters

Table 7
Summary of inputs, training task types and problem dimensions on which the trajectory features have been tested.

Feature name	Training task type	Inputs	Problem dimensions tested
Trajectory-ELA	manually designed	Potential solutions evaluated by an algorithm during its search process	- $d \in \{5\}$ [122,123] - $d \in \{5, 10\}$ [124] - $d \in \{10\}$ [125]
DynamoRep	manually designed	Potential solutions evaluated by a population-based algorithm during its search process	- $d \in \{3\}$ [13] - $d \in \{3, 5, 10, 20\}$ [128]
Opt2Vec	unsupervised	Potential solutions evaluated by a population-based algorithm during its search process, Problem class	- $d \in \{10, 30, 50\}$ [14]
Iterative-ELA	manually designed	Potential solutions evaluated by a population-based algorithm during its search process	- $d \in \{10, 30, 50\}$ [14] - $d \in \{5\}$ [128]
LON	manually designed	Multiple runs of a local search algorithm	- $d \in \{3, 5\}$ [129] - $d \in \{3, 5, 8, 12, 20\}$ [131]
Probing trajectories	manually designed	Potential solutions evaluated by a population-based algorithm during its search process	- $d \in \{10\}$ [133]
ClustOpt	unsupervised	Potential solutions evaluated by a population-based algorithm during its search process	- $d \in \{2, 5, 10\}$ [134]

7. Machine learning studies that utilized meta-features

Table 8 provides an overview of the research focused on problem, algorithm, and trajectory-based features and their applications in problem classification, algorithm selection, performance prediction, and complementarity of benchmark suites. The table is structured into three horizontal parts, one for each of these categories. Each row represents a specific class of features. The columns are divided into three primary aspects. The first aspect identifies the learning tasks in which the features are evaluated, including problem classification, algorithm selection, performance prediction, and visualization/complementarity analysis. The second and third aspects concern the collection of problem instances used in the research, which may originate from established benchmark suites such as BBOB, CEC, or Nevergrad, or be generated by problem instance generators.

We omit a detailed description of each study presented in the table, as most have been previously discussed, and instead provide a single example to illustrate how the table can be interpreted. For instance, the study [124] explores per-run algorithm selection using trajectory-ELA features and algorithm features derived from internal algorithm parameters. Accordingly, in the first part of the table, this study is listed in the cells (Trajectory-ELA, Algorithm Selection) and (Internal Algorithm Parameters, Algorithm Selection). For additional context, the same Ref. [124] appears in the second part of the table, corresponding to the BBOB and Nevergrad benchmark suites. These are the benchmark suites where the learning tasks from the first part of the table are tested.

The table results indicate that the majority of studies have utilized ELA problem landscape features with BBOB benchmark problem instances. This prevalence is expected, as developing an ML model for problem classification, algorithm performance prediction, or algorithm selection benefits from having multiple problem instances per problem class that vary by shifting or scaling. This variation allows for training the ML model on one set of problem instances and testing it on another set that differs only in shift or scale. However, using other benchmark suites, such as the CEC benchmark suites, remains challenging. These suites require the development of zero-shot ML models [135] because each problem class is represented by only a single instance.

Recent trends indicate the emergence of novel problem landscape features, such as TLA, DoE2Vec, and TransOpt, which have demonstrated predictive accuracy comparable to ELA features in problem classification tasks. Additionally, there is a growing interest in trajectory-based features, which capture information about the interaction between an algorithm instance and a problem instance.

A common limitation across all studies is the restricted generalization of the developed ML models to other benchmark suites such as CEC and Nevergrad [136], or to instances generated by problem generators. This is evident from the predominant use of the BBOB benchmark suite for learning and evaluation. To address this limitation in the future, it is essential to incorporate problem instances from diverse benchmark suites and various problem generators (which remain underexplored) into a representative learning set to enhance generalization. Additionally, as the landscape of possible problem instances continually evolves with new problem generators, future work should focus on developing ML models within continual learning frameworks [137–139].

8. Discussion and open challenges

In this section, we provide a more detailed discussion of challenges in representation learning of problem, algorithm, and trajectory features for single-objective black-box optimization.

8.1. Problem landscape features

The calculation of ELA features involves the application of a set of statistical functions to candidate solutions that are artificially sampled from the decision space of the optimization problem instance. Despite their prevalence, the calculation of ELA features can be computationally demanding for high-dimensional problems, and they have been shown to be sensitive to variations in the sample size and sampling method [84,86], as well as not being invariant to transformations such as scaling and shifting of the optimization problem [85,86]. The TLA and TinyTLA features have the desired property of being invariant to these transformations and have been demonstrated to have good predictive performance for the classification of optimization problems. While an initial study of the predictive performance of the TinyTLA features for the algorithm selection task has been conducted on the BBOB benchmark, a more comprehensive evaluation can be done, involving different algorithm portfolios and benchmark sets.

With the rapidly increasing number of studies that propose low-level landscape features based on deep learning, they come with the limitation of not allowing interpretation of why a decision is made. There is also no clear evidence of which features are the best for different learning tasks or if they are in favor of some algorithm classes. With the rapidly growing body of deep learning-based low-level landscape features, a persistent limitation is their limited interpretability: they rarely explain why a particular decision is made. Moreover, there is still no clear evidence identifying which feature sets are best for specific learning tasks, or whether certain features systematically favor particular algorithm classes. Several works have directly compared problem landscape features. A comparison between TransOpt (trained for problem classification) and classical ELA features is reported in [104]. In this work, the generalization of algorithm selection models across four benchmark suites is being evaluated with different feature groups. The results produce mixed outcomes of the superiority of one feature group over another, showing that the quality of predictions depends heavily on the benchmarks used for training and testing, and the distribution of algorithm performance. Another comparison of the DeepELA and ELA features appears in [106], where the features are evaluated for the algorithm selection task in a fixed-target setting on the BBOB benchmark. In this case, a benefit of the joint use of ELA and DeepELA features is observed. However, when the DeepELA, ELA and TransOptAS features are evaluated in a fixed-budget setting on the affine combinations of the BBOB problems [37], the ELA features are shown to be superior over the transformer-based ones [107]. The most comprehensive comparative study including the ELA, TransOptAS, DeepELA, Doe2Vec and TinyTLA features is provided in [114]. In this study, features are compared for the algorithm selection task on four different algorithm portfolios using the affine combinations of the BBOB benchmark problems [37]. The algorithm selection models are evaluated in a fixed-budget setting, on evaluation settings of increasing difficulty. The general outcome is that the ELA features remain superior to newer problem landscape features in easy evaluation settings, however, none of the features outperform a baseline model in the difficult evaluation settings. This indicates a need to rethink the standard algorithm selection setup using problem landscape features, as well as a need for improved benchmarks suites for evaluation. Overall, we can observe that various aspects of the experimental setup (problem benchmarks, method for splitting data into training and test sets, algorithms included in algorithm portfolio, sample size used for feature calculation, metrics used for capturing algorithm performance) used for comparing problem landscape features for the algorithm selection task can substantially influence on the results of the comparison. Therefore, standardization of the experimental setup is needed to ensure that comparisons of problem landscape features for the algorithm selection task are consistent, reliable, and reproducible.

Table 8
Summary of works using problem, algorithm, or trajectory-based features and their applications in the domain of algorithm selection for continuous single-objective optimization. The rows are grouped into problem landscape features, algorithm features, and trajectory-based features. The columns are split into three parts; the first part indicates the *learning scenario* where some specific features are evaluated (problem classification, algorithm selection, performance prediction, or visualization/complementarity analysis), the second and third parts correspond to the set of *problem instances* used in the study, with the first group comprising established benchmark suites (BBOB, CEC, Nevergrad) and the other problem collections obtained from instance generators.

	Learning tasks				Benchmark suites			Problem generators				
Features	Problem classification	Algorithm selection	Performance prediction	Visualization/ Complementarity	BBOB	CEC	Nevergrad	ISA	GP	TR	Affine	GKLS
Problem landscape features												
ELA	[69,76,80,84,109]	[1,9,9,60,65,104,111,114,140–142]	[37,56–59,110,118,140,141,143–146]	[11,32,37,68,85,86,101,109,113,146–148]	[1,9,11,32,37,56–60,65,68,76,80,84–86,101,104,109–111,113,118,140–143,146–148]	[11,110,113,143–145] [85,86]		[32]	[148]	[9,68,101,104]	[37,60,104,114,147]	[113]
TLA	[69,94]	[94,114]			[69]						[114]	
Fitness Map + CNNs	[70]				[70]							
Point Cloud Transformer	[70]				[70]							
DoE2Vec	[99]	[114]			[99]					[99]	[114]	
TransOpt	[102]	[104]		[104]	[102,104]						[104]	
Deep-ELA	[105]	[105–107,114]			[105,106]						[107,114]	
Random Filter Mappings	[108]	[108]			[108]							
Algorithm features												
Source Code		[49]										
High-level problem-algorithm interaction features												
Fitness Map + CNNs		[96]			[96]							
TransOptAS		[71,107,114]								[71]	[107,114]	
Performance2Vec	[50]			[50]	[50]							
Explainable Prediction Models			[59,143]		[59,143]							
Internal Algorithm Parameters	[15]	[124]	[15]		[15]							
KG embeddings			[51]		[51]							
GNN embeddings			[52]		[52]							
fANOVA				[53]	[53]							
fANOVA			[118]		[118]							
Trajectory-based features												
Trajectory-ELA	[122]	[123–125,149]			[122–125,149]		[124]					
DynamoRep	[13,128]	[128]			[13,128]							
Opt2Vec	[66]					[66]						
Iterative-ELA	[66,128]	[128]			[128]							
LON				[129,131]	[129,131]							
Probing trajectories		[133]			[133]							
ClustOpt				[134]	[134]							

8.2. Algorithm features

Algorithm features are relatively underexplored and low-resourced compared to other feature types. A promising direction is to represent algorithms as modular frameworks, similar to modCMA and modDE, by structuring them into components that reflect their operators. Such insights could then be utilized as meta-features, complementing problem-, high-level problem-algorithm interaction, or trajectory-based features, and applied to various machine learning tasks. Proper algorithm features would also allow us to go beyond the classical taxonomy-style representation of algorithms [150] to more complex descriptions that allow to cluster algorithms according to different criteria.

8.3. High-level problem-algorithm interaction features

For offline automated algorithm selection and configuration methods, these features are crucial. However, most of these features require the prior computation of ELA features and/or algorithm performance, which is itself computationally demanding. This is the case for the GNN and KG embeddings as well as the Explainable Prediction Models, requiring both ELA features and algorithm performance. On the other hand, the TransOptAS and Fitness Map + CNNs features require algorithm performance for a sufficiently large set of functions to train a neural network, while the fANOVA features require algorithm performance for a sufficiently large coverage of the parameter space of the analyzed algorithm.

Future directions should focus on model-agnostic approaches to characterize the interactions between problems and algorithms. Furthermore, since supervised machine learning is typically applied in such learning processes, the quality of the data used for training is of great importance.

8.4. Trajectory-based features

The process of automated algorithm selection and configuration could potentially benefit from features that take into account the interactions between the problem and the algorithm. Moreover, utilizing samples from the optimization algorithm's trajectory incurs no additional computational expenses, since there is no need to evaluate the objective function of the optimization problem prior to running the algorithm. A potential direction for future research would involve the generation of new features tailored specifically for online use, which would be computationally inexpensive enough to be calculated during algorithm execution. Despite the DynamoRep (statistical) and Opt2Vec (based on autoencoder) features, features from the trajectory of an optimization algorithm instance could be extracted using expert knowledge or deep learning methods such as Long Short-Term Memory Networks (LSTM), Convolutional Neural Networks (CNN), or Transformers.

While trajectory-based features provide rich information about problem–algorithm interactions, their computational costs can vary, which is crucial for assessing online feasibility and the type of tasks they are suited for. Features derived from internal algorithm parameters, probing trajectories or simple population statistics as in DynamoRep incur negligible per-iteration overhead and are well suited for online use. Iterative-based ELA are more demanding since they perform the calculation of the ELA features on every iteration of the algorithm, and Local Optima Networks or Search Trajectory Networks are even heavier, relying on repeated local search or basin-hopping steps, which limits them to offline analysis. ClustOpt features perform clustering only once across all iterations, resulting in a moderate one-time cost, making it practical for online monitoring of algorithm execution or post-hoc analysis of multiple algorithm runs. To enable early-run selection or warm-start switching, lightweight proxies such as probing trajectories features, DynamoRep or trajectory ELA features can be explored, although their predictive power is yet to be evaluated for this task.

Trajectory features also rely on the quality and diversity of samples generated by the underlying population-based algorithm. Because the statistical reliability of the learned features depends strongly on this data, the lack of standardized benchmarking introduces ambiguity in the reported results.

It is important to note that trajectory-based features are recently developed and comprehensive comparative analyses are yet to be performed. With this in mind, it is difficult to provide a definite recommendation of the use of one feature group over another.

8.5. Challenges for using the meta-features for machine learning

It seems widely accepted in the evolutionary computation community that random forest models [151] in combination with ELA problem landscape features provide acceptable results. However, we also observe a trend that recent advances in ML modeling are taken up by the community only with some delay or are neglected after some nonsatisfactory results on the BBOB benchmark suite. We are concerned that the good results achieved on this comparatively small problem collection may paint a too optimistic picture of the capability of ELA and simple random forest models. This concern seems to be confirmed by recent works pointing out the rather dissatisfying generalization ability of current algorithm selection models based on ELA features [9,60,104,114,142]. This has been demonstrated in a scenario where the algorithm selection model trained on one benchmark and evaluated on another one does not outperform a simple baseline model [9,104]. Additionally, it has been shown that algorithm selection models perform well when similar problem instances are present in the training and testing set, but fail when presented with unseen problem instances [60]. Finally, the leave-one-instance-out evaluation strategy, which is the most commonly used to evaluate algorithm selection models on the BBOB benchmark suite, has been criticized as it can produce misleading results, where meta-models achieve high performance due to spurious correlations between features and the target, rather than having genuine predictive capability [111]. The use of scale-sensitive metrics to capture algorithm performance has also raised concerns for causing a false indication of improvements over baseline models used in algorithm selection [111].

To improve the generalizability of algorithm selection models, several avenues can be explored: including newly developed features, including larger, more diverse training data, and considering contemporary ML training strategies.

To get a fair comparison of feature performance, comprehensive studies are required which evaluate different feature sets under a fixed experimental setup, using consistent datasets, splits, and evaluation protocols to enable meaningful statistical testing and reliable conclusions. Such studies should further include diverse problem benchmarks beyond BBOB, report variability and deviations in performance, establish proper baselines for comparison, and analyze the influence of sample sizes and sampling techniques. Current research relies disproportionately on a narrow set of benchmark datasets (primarily BBOB), and comparisons conducted within the same benchmark often produce dependent performance values, making many of the reported findings unsuitable for rigorous statistical testing. The need to rethink existing benchmark suites has been raised across the broader field [152].

Finally, we believe it is important to critically observe advances in ML utilizing other models based on deep learning architectures that can be evaluated in transfer learning and continual learning scenarios. Continual learning [137–139] is a ML paradigm that allows models to develop themselves adaptively by learning incrementally from dynamic data distributions and selectively adapting representations to obtain good generalizability within and between tasks. This adaptation can lead to a range of benefits such as improved robustness by handling both simple and complex data instances, reducing overfitting to specific data instances, resulting in better generalization to unseen data. By using continual learning approaches, one hopes to identify problem

instances that challenge the performance predictor assumptions and explore the uncharted regions of the problem space, ultimately leading to algorithm selection models that not only perform well on the training problem instances, but also generalize effectively to unseen problem instances.

8.6. Practical recommendations

For practitioners, the choice of feature family should primarily depend on what needs to be modeled (the problem, the algorithm, or their interaction), what data is available, the evaluation budget, the need for invariance/robustness, and whether the goal is offline modeling or online decision-making. If a well-tested and broadly applicable baseline for offline tasks (e.g., instance classification or algorithm selection) is needed and an initial sampling phase is affordable, ELA remains a sensible first choice. Deep learned landscape features can also be used when interpretability is not a primary requirement. When applying problem landscape features to the algorithm selection task, one should not expect a good generalization to unseen problems, i.e., the model will only perform well if training and testing problems are nearly identical, and the model may need to be retrained on the particular types of problems it will be applied on. When aspects of algorithm performance (rather than only landscape structure) need to be incorporated into the feature representation, high-level problem-algorithm interaction features can be effective; however, some of these are computationally expensive to compute, as detailed in previous sections. Finally, when extra pre-evaluations should be avoided and/or information computed during the run is required (e.g., early-run selection, monitoring, or switching), trajectory-based features are a natural option: DynamoRep and probing-trajectory features can be computed online with minimal overhead, whereas heavier iterative-ELA or Local Optima Network variants are better reserved for offline analysis when computational cost is less constrained. In this context, approaches such as ClustOpt and Local Optima Networks are particularly useful for visualizing algorithm behavior and relating search dynamics to structural properties of the problem.

CRediT authorship contribution statement

Gjorgjina Cenikj: Writing – review & editing, Writing – original draft, Visualization, Validation, Investigation, Formal analysis, Conceptualization. **Ana Nikolikj:** Writing – review & editing, Visualization, Validation, Conceptualization. **Gašper Petelin:** Writing – review & editing, Writing – original draft, Validation, Conceptualization. **Niki van Stein:** Writing – review & editing, Validation, Supervision, Conceptualization. **Carola Doerr:** Writing – review & editing, Writing – original draft, Validation, Supervision, Conceptualization. **Tome Eftimov:** Writing – review & editing, Writing – original draft, Visualization, Validation, Supervision, Project administration, Funding acquisition, Conceptualization.

Declaration of Generative AI and AI-assisted technologies in the writing process

During the preparation of this work the author(s) used ChatGpt in order to improve the grammar and quality of the text. After using this tool/service, the author(s) reviewed and edited the content as needed and take(s) full responsibility for the content of the publication.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Acknowledgment

We thank Marcel Wever from the Leibniz University Hannover, Germany, for his valuable feedback on an earlier version of this survey. The authors acknowledge financial support of the Slovenian Research Agency through program grant P2-0098, project grants J2-4460 and GC-0001, and young researcher grants No. PR-12393 to GC, No. PR-11263 to GP, and No. PR-12897 to AN. This work is also funded by the European Union under Grant Agreement 101187010 (HE ERA Chair AutoLearn-SI). We also acknowledge the support of the EC/EuroHPC JU and the Slovenian Ministry of HESI via the project SLAIF (grant number 101254461). We apologize if there are studies that were inadvertently omitted or overlooked.

Data availability

No data was used for the research described in the article.

References

[1] P. Kerschke, H.H. Hoos, F. Neumann, H. Trautmann, Automated algorithm selection: Survey and perspectives, Evol. Comput. 27 (1) (2019) 3–45, http://dx.doi.org/10.1162/evco_a_00242, arXiv:https://direct.mit.edu/evco/article-pdf/27/1/3/1552398/evco_a_00242.pdf.

[2] L. Kotthoff, Algorithm selection for combinatorial search problems: A survey, in: C. Bessiere, L. De Raedt, L. Kotthoff, S. Nijssen, B. O'Sullivan, D. Pedreschi (Eds.), Data Mining and Constraint Programming: Foundations of a Cross-Disciplinary Approach, Springer International Publishing, Cham, 2016, pp. 149–190, http://dx.doi.org/10.1007/978-3-319-50137-6_7.

[3] M.A. Muñoz, Y. Sun, M. Kirley, S.K. Halgamuge, Algorithm selection for black-box continuous optimization problems: A survey on methods and challenges, Inform. Sci. 317 (2015) 224–245.

[4] J. Vanschoren, Meta-learning, Autom. Mach. Learn.: Methods, Syst. Challenges (2019) 35–61.

[5] M. Gallagher, M. Munoz, Towards an improved understanding of features for more interpretable landscape analysis, in: Proceedings of the Genetic and Evolutionary Computation Conference Companion, 2024, pp. 135–138.

[6] B. Naudts, L. Kallel, A comparison of predictive measures of problem difficulty in evolutionary algorithms, IEEE Trans. Evol. Comput. 4 (1) (2000) 1–15.

[7] Q. Renau, J. Dreo, C. Doerr, B. Doerr, Expressiveness and robustness of landscape features, in: Proceedings of the Genetic and Evolutionary Computation Conference Companion, 2019, pp. 2048–2051.

[8] J. He, C. Reeves, C. Witt, X. Yao, A note on problem difficulty measures in black-box optimization: Classification, realizations and predictability, Evol. Comput. 15 (4) (2007) 435–443.

[9] U. Škvorc, T. Eftimov, P. Korošec, Transfer learning analysis of multi-class classification for landscape-aware algorithm selection, Mathematics 10 (3) (2022) http://dx.doi.org/10.3390/math10030432, URL https://www.mdpi.com/2227-7390/10/3/432.

[10] G. Cenikj, R.D. Lang, A.P. Engelbrecht, C. Doerr, P. Korošec, T. Eftimov, SELECTOR: Selecting a representative benchmark suite for reproducible statistical comparison, in: Proceedings of the Genetic and Evolutionary Computation Conference, GECCO '22, Association for Computing Machinery, New York, NY, USA, 2022, pp. 620–629, http://dx.doi.org/10.1145/3512290.3528809.

[11] R.D. Lang, A.P. Engelbrecht, An exploratory landscape analysis-based benchmark suite, Algorithms 14 (3) (2021) 78.

[12] D. Vermetten, C.-V. Dinu, M. Gallagher, A standardized benchmark set of clustering problem instances for comparing black-box optimizers, 2025, arXiv preprint arXiv:2505.09233.

[13] G. Cenikj, G. Petelin, C. Doerr, P. Korošec, T. Eftimov, DynamoRep: Trajectory-based population dynamics for classification of black-box optimization problems, in: Proceedings of the Genetic and Evolutionary Computation Conference, GECCO '23, Association for Computing Machinery, New York, NY, USA, 2023, pp. 813–821, http://dx.doi.org/10.1145/3583131.3590401.

[14] P. Korošec, T. Eftimov, Opt2vec: A continuous optimization problem representation based on the algorithm's behavior, 2023, http://dx.doi.org/10.2139/ssrn.4316939.

[15] J. de Nobel, H. Wang, T. Bäck, Explorative data analysis of time series based algorithm features of CMA-ES variants, in: Proceedings of the Genetic and Evolutionary Computation Conference, GECCO '21, Association for Computing Machinery, New York, NY, USA, 2021, pp. 510–518, http://dx.doi.org/10.1145/3449639.3459399.

[16] G. Ochoa, S. Verel, F. Daolio, M. Tomassini, Local optima networks: A new model of combinatorial fitness landscapes, Recent. Adv. the Theory Appl. Fit. Landscapes (2014) 233–262.

[17] G. Ochoa, A. Liefooghe, S. Verel, Funnels in multi-objective fitness landscapes, in: International Conference on Parallel Problem Solving from Nature, Springer, 2024, pp. 343–359.

[18] A. Liefooghe, G. Ochoa, S. Verel, B. Derbel, Pareto local optimal solutions networks with compression, enhanced visualization and expressiveness, in: Proceedings of the Genetic and Evolutionary Computation Conference, 2023, pp. 713–721.

[19] A. Liefooghe, F. Daolio, S. Verel, B. Derbel, H. Aguirre, K. Tanaka, Landscape-aware performance prediction for evolutionary multiobjective optimization, IEEE Trans. Evol. Comput. 24 (6) (2019) 1063–1077.

[20] H. Alsouly, M. Kirley, M.A. Muñoz, Online per-instance algorithm selection for constrained multi-objective optimization problems, in: Proceedings of the Genetic and Evolutionary Computation Conference Companion, 2024, pp. 559–562.

[21] R.P. Prager, H. Trautmann, Investigating the viability of existing exploratory landscape analysis features for mixed-integer problems, in: Proceedings of the Companion Conference on Genetic and Evolutionary Computation, 2023, pp. 451–454.

[22] K. Dietrich, R.P. Prager, C. Doerr, H. Trautmann, Hybridizing target-and SHAP-encoded features for algorithm selection in mixed-variable black-box optimization, in: International Conference on Parallel Problem Solving from Nature, Springer, 2024, pp. 154–169.

[23] Z. Ma, H. Guo, Y.-J. Gong, J. Zhang, K.C. Tan, Toward automated algorithm design: A survey and practical guide to meta-black-box-optimization, IEEE Trans. Evol. Comput. (2025) 1, http://dx.doi.org/10.1109/TEVC.2025.3568053.

[24] K.M. Malan, A.P. Engelbrecht, A survey of techniques for characterising fitness landscapes and some possible ways forward, Inform. Sci. 241 (2013) 148–163.

[25] K.M. Malan, A survey of advances in landscape analysis for optimisation, Algorithms 14 (2) (2021) 40.

[26] P. Kerschke, M. Preuss, Exploratory landscape analysis, in: Proceedings of the Companion Conference on Genetic and Evolutionary Computation, 2023, pp. 990–1007.

[27] N. Hansen, S. Finck, R. Ros, A. Auger, Real-parameter black-box optimization benchmarking 2009: Noiseless functions definitions, Research Report RR-6829, INRIA, 2009, URL https://hal.inria.fr/inria-00362633.

[28] J. Liang, B. Qu, P. Suganthan, A. Hernández-Díaz, Problem definitions and evaluation criteria for the CEC 2013 special session on real-parameter optimization, in: Computational Intelligence Laboratory, Zhengzhou University, Zhengzhou China and Technical Report, Nanyang Technological University, Singapore, 2013.

[29] J. Liang, B. Qu, P. Suganthan, Problem definitions and evaluation criteria for the CEC 2014 special session and competition on single objective real-parameter numerical optimization, in: Computational Intelligence Laboratory, Zhengzhou University, Zhengzhou China and Technical Report, Nanyang Technological University, Singapore, 2013.

[30] J. Liang, B. Qu, P. Suganthan, Q. Chen, Problem definitions and evaluation criteria for the CEC 2015 competition on learning-based real-parameter single objective optimization, in: Computational Intelligence Laboratory, Zhengzhou University, Zhengzhou China and Technical Report, Nanyang Technological University, Singapore, 2014.

[31] G. Wu, R. Mallipeddi, P. Suganthan, Problem definitions and evaluation criteria for the CEC 2017 competition and special session on constrained single objective real-parameter optimization, in: Computational Intelligence Laboratory, Zhengzhou University, Zhengzhou China and Technical Report, Nanyang Technological University, Singapore, 2016.

[32] M.A. Muñoz, K. Smith-Miles, Generating new space-filling test instances for continuous black-box optimization, Evol. Comput. 28 (3) (2020) 379–404.

[33] Y. Tian, S. Peng, X. Zhang, T. Rodemann, K.C. Tan, Y. Jin, A recommender system for metaheuristic algorithms for continuous optimization based on deep recurrent neural networks, IEEE Trans. Artif. Intell. 1 (1) (2020) 5–18.

[34] M. Gaviano, D.E. Kvasov, D. Lera, Y.D. Sergeyev, Software for generation of classes of test functions with known local and global minima for global optimization, ACM Trans. Math. Softw. 29 (4) (2003).

[35] R.P. Prager, K. Dietrich, L. Schneider, L. Schäpermeier, B. Bischl, P. Kerschke, H. Trautmann, O. Mersmann, Neural networks as black-box benchmark functions optimized for exploratory landscape features, in: Proceedings of the 17th ACM/SIGEVO Conference on Foundations of Genetic Algorithms, 2023, pp. 129–139.

[36] K. Dietrich, O. Mersmann, Increasing the diversity of benchmark function sets through affine recombination, in: International Conference on Parallel Problem Solving from Nature, Springer, 2022, pp. 590–602.

[37] D. Vermetten, F. Ye, T. Bäck, C. Doerr, MA-BBOB: A problem generator for black-box optimization using affine combinations and shifts, ACM Trans. Evol. Learn. Optim. (2025) http://dx.doi.org/10.1145/3673908.

[38] R. Storn, K. Price, Differential evolution – a simple and efficient heuristic for global optimization over continuous spaces, J. Global Optim. 11 (4) (1997) 341–359, http://dx.doi.org/10.1023/A:1008202821328.

[39] N. Hansen, A. Ostermeier, Completely derandomized self-adaptation in evolution strategies, Evol. Comput. 9 (2) (2001) 159–195, http://dx.doi.org/10.1162/106365601750190398.

[40] D. Wang, D. Tan, L. Liu, Particle swarm optimization algorithm: an overview, Soft Comput. 22 (2) (2017) 387–408, http://dx.doi.org/10.1007/s00500-016-2474-6.

[41] J. de Nobel, D. Vermetten, H. Wang, C. Doerr, T. Bäck, Tuning as a means of assessing the benefits of new ideas in interplay with existing algorithmic modules, in: K. Krawiec (Ed.), GECCO '21: Genetic and Evolutionary Computation Conference, Companion Volume, Lille, France, July 10-14, 2021, ACM, 2021, pp. 1375–1384, http://dx.doi.org/10.1145/3449726.3463167.

[42] S. van Rijn, H. Wang, M. van Leeuwen, T. Bäck, Evolving the structure of evolution strategies, in: 2016 IEEE Symposium Series on Computational Intelligence, SSCI, IEEE, 2016, pp. 1–8.

[43] D. Vermetten, F. Caraffini, A.V. Kononova, T. Bäck, Modular differential evolution, in: Proceedings of the Genetic and Evolutionary Computation Conference, GECCO '23, Association for Computing Machinery, New York, NY, USA, 2023, pp. 864–872, http://dx.doi.org/10.1145/3583131.3590417.

[44] C.L. Camacho-Villalón, M. Dorigo, T. Stützle, PSO-X: A component-based framework for the automatic design of particle swarm optimization algorithms, IEEE Trans. Evol. Comput. 26 (3) (2021) 402–416.

[45] N. Hansen, A. Auger, D. Brockhoff, T. Tusar, Anytime performance assessment in blackbox optimization benchmarking, IEEE Trans. Evol. Comput. 26 (6) (2022) 1293–1305, http://dx.doi.org/10.1109/TEVC.2022.3210897.

[46] M.L. nez, D. Vermetten, J. Dréo, C. Doerr, Using the empirical attainment function for analyzing single-objective black-box optimization algorithms, 2024, http://dx.doi.org/10.48550/ARXIV.2404.02031, CoRR arXiv:2404.02031.

[47] F. Zou, D. Chen, H. Liu, S. Cao, X. Ji, Y. Zhang, A survey of fitness landscape analysis for optimization, Neurocomputing 503 (2022) 129–139.

[48] O. Mersmann, B. Bischl, H. Trautmann, M. Preuss, C. Weihs, G. Rudolph, Exploratory landscape analysis, in: Proc. of Genetic and Evolutionary Computation Conference, GECCO'11, ACM, 2011, pp. 829–836, http://dx.doi.org/10.1145/2001576.2001690.

[49] D. Pulatov, M. Anastacio, L. Kotthoff, H. Hoos, Opening the black box: Automated software analysis for algorithm selection, in: I. Guyon, M. Lindauer, M. van der Schaar, F. Hutter, R. Garnett (Eds.), Proceedings of the First International Conference on Automated Machine Learning, in: Proceedings of Machine Learning Research, Vol. 188, PMLR, 2022, pp. 6/1–18, URL https://proceedings.mlr.press/v188/pulatov22a.html.

[50] T. Eftimov, G. Popovski, D. Kocev, P. Korošec, Performance2vec: A step further in explainable stochastic optimization algorithm performance, in: Proceedings of the 2020 Genetic and Evolutionary Computation Conference Companion, GECCO '20, Association for Computing Machinery, New York, NY, USA, 2020, pp. 193–194, http://dx.doi.org/10.1145/3377929.3390020.

[51] A. Kostovska, D. Vermetten, S. Džeroski, P. Panov, T. Eftimov, C. Doerr, Using knowledge graphs for performance prediction of modular optimization algorithms, in: International Conference on the Applications of Evolutionary Computation (Part of EvoStar), Springer, 2023, pp. 253–268.

[52] A. Kostovska, C. Doerr, S. Džeroski, P. Panov, T. Eftimov, Geometric learning in black-box optimization: A GNN framework for algorithm performance prediction, in: Proceedings of the Genetic and Evolutionary Computation Conference Companion, in: GECCO '25 Companion, Association for Computing Machinery, New York, NY, USA, 2025, pp. 487–490, http://dx.doi.org/10.1145/3712255.3726696.

[53] A. Nikolikj, A. Kostovska, D. Vermetten, C. Doerr, T. Eftimov, Quantifying individual and joint module impact in modular optimization frameworks, in: 2024 IEEE Congress on Evolutionary Computation, CEC, 2024.

[54] N. van Stein, D. Vermetten, A. V. Kononova, T. Bäck, Explainable benchmarking for iterative optimization heuristics, ACM Trans. Evol. Learn. 5 (2) (2025) 1–30.

[55] G. Tsoumakas, I. Katakis, Multi-label classification: An overview, Int. J. Data Warehous. Min. 3 (2009) 1–13, http://dx.doi.org/10.4018/jdwm.2007070101.

[56] A. Nikolikj, R. Trajanov, G. Cenikj, P. Korošec, T. Eftimov, Identifying minimal set of exploratory landscape analysis features for reliable algorithm performance prediction, in: 2022 IEEE Congress on Evolutionary Computation, CEC, 2022, pp. 1–8, http://dx.doi.org/10.1109/CEC55065.2022.9870439.

[57] A. Kostovska, D. Vermetten, S. Džeroski, C. Doerr, P. Korosec, T. Eftimov, The importance of landscape features for performance prediction of modular CMA-ES variants, in: Proceedings of the Genetic and Evolutionary Computation Conference, GECCO '22, Association for Computing Machinery, New York, NY, USA, 2022, pp. 648–656, http://dx.doi.org/10.1145/3512290.3528832.

[58] R. Trajanov, S. Dimeski, M. Popovski, P. Korošec, T. Eftimov, Explainable landscape analysis in automated algorithm performance prediction, in: Applications of Evolutionary Computation: 25th European Conference, EvoApplications 2022, Held As Part of EvoStar 2022, Madrid, Spain, April 20–22, 2022, Proceedings, Springer, 2022, pp. 207–222.

[59] A. Nikolikj, R. Lang, P. Korošec, T. Eftimov, Explaining differential evolution performance through problem landscape characteristics, in: International Conference on Bioinspired Optimization Methods and their Applications, Springer, 2022, pp. 99–113.

[60] G. Petelin, G. Cenikj, How far out of distribution can we go with ELA features and still be able to rank algorithms? in: 2023 IEEE Symposium Series on Computational Intelligence, SSCI, 2023, pp. 341–346, http://dx.doi.org/10.1109/SSCI52147.2023.10371880.

[61] G. Biau, E. Scornet, A random forest guided tour, Test 25 (2016) 197–227.

[62] T. Chen, C. Guestrin, Xgboost: A scalable tree boosting system, in: Proceedings of the 22nd Acm Sigkdd International Conference on Knowledge Discovery and Data Mining, 2016, pp. 785–794.

[63] N. Hollmann, S. Müller, K. Eggensperger, F. Hutter, TabPFN: A transformer that solves small tabular classification problems in a second, in: The Eleventh International Conference on Learning Representations, 2022.

[64] Y. Gorishniy, I. Rubachev, V. Khrulkov, A. Babenko, Revisiting deep learning models for tabular data, Adv. Neural Inf. Process. Syst. 34 (2021) 18932–18943.

[65] A. Kostovska, A. Jankovic, D. Vermetten, S. Džeroski, T. Eftimov, C. Doerr, Comparing algorithm selection approaches on black-box optimization problems, in: Proceedings of the Companion Conference on Genetic and Evolutionary Computation, in: GECCO '23 Companion, Association for Computing Machinery, New York, NY, USA, 2023, pp. 495–498, http://dx.doi.org/10.1145/3583133.3590697.

[66] P. Korošec, T. Eftimov, Opt2Vec-a continuous optimization problem representation based on the algorithm's behavior: A case study on problem classification, Inform. Sci. 680 (2024) 121134.

[67] K. Dietrich, D. Vermetten, C. Doerr, P. Kerschke, Impact of training instance selection on automated algorithm selection models for numerical black-box optimization, in: Proceedings of the Genetic and Evolutionary Computation Conference, 2024, pp. 1007–1016.

[68] U. Škvorc, T. Eftimov, P. Korošec, A complementarity analysis of the COCO benchmark problems and artificially generated problems, in: Proceedings of the Genetic and Evolutionary Computation Conference Companion, 2021, pp. 215–216.

[69] G. Petelin, G. Cenikj, T. Eftimov, TLA: Topological landscape analysis for single-objective continuous optimization problem instances, in: Proceedings of IEEE Symposium Series on Computational Intelligence, 2022, pp. 1698–1705, http://dx.doi.org/10.1109/SSCI51031.2022.10022126.

[70] M.V. Seiler, R.P. Prager, P. Kerschke, H. Trautmann, A collection of deep learning-based feature-free approaches for characterizing single-objective continuous fitness landscapes, in: Proceedings of the Genetic and Evolutionary Computation Conference, 2022, pp. 657–665.

[71] G. Cenikj, G. Petelin, T. Eftimov, TransOptAS: Transformer-based algorithm selection for single-objective optimization, in: Proceedings of the Genetic and Evolutionary Computation Conference, Association for Computing Machinery, New York, NY, USA, 2024, http://dx.doi.org/10.1145/3638530.3654191.

[72] A. Engelbrecht, P. Bosman, K. Malan, The influence of fitness landscape characteristics on particle swarm optimisers, Nat. Comput. (2021) 1–11.

[73] N. Belkhir, Per Instance Algorithm Configuration for Continuous Black Box Optimization (Ph.D. thesis), (2017SACLS455) Université Paris-Saclay, 2017, URL https://hal.inria.fr/tel-01669527.

[74] P. Kerschke, H. Trautmann, Comprehensive Feature-Based Landscape Analysis of Continuous and Constrained Optimization Problems Using the R-package flacco, in: N. Bauer, K. Ickstadt, K. Lübke, G. Szepannek, H. Trautmann, M. Vichi (Eds.), Applications in Statistical Computing – from Music Data Analysis To Industrial Quality Improvement, in: Studies in Classification, Data Analysis, and Knowledge Organization, Springer, 2019, pp. 93–123, http://dx.doi.org/10.1007/978-3-030-25147-5_7, URL https://link.springer.com/chapter/10.1007/978-3-030-25147-5_7.

[75] R.P. Prager, H. Trautmann, Pflacco: Feature-based landscape analysis of continuous and constrained optimization problems in python, Evol. Comput. (2022) 1–25.

[76] P. Kerschke, M. Preuss, C. Hernández Castellanos, O. Schütze, J.Q. Sun, C. Grimme, G. Rudolph, B. Bischl, H. Trautmann, Cell mapping techniques for exploratory landscape analysis, 288, 2014, pp. 115–131, http://dx.doi.org/10.1007/978-3-319-07494-8_9,

[77] M. Lunacek, D. Whitley, The dispersion metric and the CMA evolution strategy, in: Proceedings of the 8th Annual Conference on Genetic and Evolutionary Computation, 2006, pp. 477–484.

[78] M. Preuss, Improved topological niching for real-valued global optimization, in: Applications of Evolutionary Computation: EvoApplications 2012: EvoCOMNET, EvoCOMPLEX, EvoFIN, EvoGAMES, EvoHOT, EvoIASP, EvoNUM, EvoPAR, EvoRISK, EvoSTIM, and EvoSTOC, MÁlaga, Spain, April 11-13, 2012, Proceedings, Springer, 2012, pp. 386–395.

[79] C. Flamm, I.L. Hofacker, P.F. Stadler, M.T. Wolfinger, Barrier trees of degenerate landscapes, 2002.

[80] M.A. Muñoz, M. Kirley, S.K. Halgamuge, Exploratory landscape analysis of continuous space optimization problems using information content, IEEE Trans. Evol. Comput. 19 (1) (2014) 74–87.

[81] D.-I. Seo, B.-R. Moon, An information-theoretic analysis on the interactions of variables in combinatorial optimization problems, Evol. Comput. 15 (2) (2007) 169–198.

[82] E. Weinberger, Correlated and uncorrelated fitness landscapes and how to tell the difference, Biol. Cybernet. 63 (5) (1990) 325–336.

[83] J. Marín, How landscape ruggedness influences the performance of real-coded algorithms: a comparative study, Soft Comput. 16 (4) (2012) 683–698.

[84] Q. Renau, C. Doerr, J. Dreo, B. Doerr, Exploratory landscape analysis is strongly sensitive to the sampling strategy, in: Parallel Problem Solving from Nature–PPSN XVI: 16th International Conference, PPSN 2020, Leiden, the Netherlands, September 5-9, 2020, Proceedings, Part II 16, Springer, 2020, pp. 139–153.

[85] U. Škvorc, T. Eftimov, P. Korošec, Understanding the problem space in single-objective numerical optimization using exploratory landscape analysis, Appl. Soft Comput. 90 (2020) 106138, http://dx.doi.org/10.1016/j.asoc.2020.106138, URL https://www.sciencedirect.com/science/article/pii/S1568494620300788.

[86] U. Škvorc, T. Eftimov, P. Korošec, The effect of sampling methods on the invariance to function transformations when using exploratory landscape analysis, in: 2021 IEEE Congress on Evolutionary Computation, CEC, 2021, pp. 1139–1146, http://dx.doi.org/10.1109/CEC45853.2021.9504739.

[87] R.P. Prager, H. Trautmann, Nullifying the inherent bias of non-invariant exploratory landscape analysis features, in: International Conference on the Applications of Evolutionary Computation (Part of EvoStar), Springer, 2023, pp. 411–425.

[88] T. Elsken, J.H. Metzen, F. Hutter, Neural architecture search: A survey, J. Mach. Learn. Res. 20 (1) (2019) 1997–2017.

[89] B. van Stein, H. Wang, T. Bäck, Neural network design: learning from neural architecture search, in: 2020 IEEE Symposium Series on Computational Intelligence, SSCI, IEEE, 2020, pp. 1341–1349.

[90] C. Doerr, J. Dréo, P. Kerschke, Making a case for (Hyper-)parameter tuning as benchmark problems, in: M.L. nez, A. Auger, T. Stützle (Eds.), Proc. of Genetic and Evolutionary Computation Conference (GECCO, Companion Material), ACM, 2019, pp. 1755–1764, http://dx.doi.org/10.1145/3319619.3326857.

[91] L. Schneider, L. Schäpermeier, R.P. Prager, B. Bischl, H. Trautmann, P. Kerschke, HPO× ELA: Investigating hyperparameter optimization landscapes by means of exploratory landscape analysis, in: International Conference on Parallel Problem Solving from Nature, Springer, 2022, pp. 575–589.

[92] F.X. Long, B. van Stein, M. Frenzel, P. Krause, M. Gitterle, T. Bäck, Generating cheap representative functions for expensive automotive crashworthiness optimization, ACM Trans. Evol. Learn. Optim. 4 (2) (2024) http://dx.doi.org/10.1145/3646554.

[93] F. Chazal, B. Michel, An introduction to topological data analysis: Fundamental and practical aspects for data scientists, Front. Artif. Intell. Volume 4 - 2021 (2021) http://dx.doi.org/10.3389/frai.2021.667963, URL https://www.frontiersin.org/journals/artificial-intelligence/articles/10.3389/frai.2021.667963.

[94] G. Petelin, G. Cenikj, T. Eftimov, TinyTLA: Topological landscape analysis for optimization problem classification in a limited sample setting, Swarm Evol. Comput. 84 (2024) 101448, http://dx.doi.org/10.1016/j.swevo.2023.101448, URL https://www.sciencedirect.com/science/article/pii/S2210650223002201.

[95] J. Gu, Z. Wang, J. Kuen, L. Ma, A. Shahroudy, B. Shuai, T. Liu, X. Wang, G. Wang, J. Cai, et al., Recent advances in convolutional neural networks, Pattern Recognit. 77 (2018) 354–377.

[96] R.P. Prager, M.V. Seiler, H. Trautmann, P. Kerschke, Towards feature-free automated algorithm selection for single-objective continuous black-box optimization, in: 2021 IEEE Symposium Series on Computational Intelligence, SSCI, IEEE, 2021, pp. 1–8.

[97] N. Ma, X. Zhang, H.-T. Zheng, J. Sun, Shufflenet v2: Practical guidelines for efficient cnn architecture design, in: Proceedings of the European Conference on Computer Vision, ECCV, 2018, pp. 116–131.

[98] Y. Wang, Y. Sun, Z. Liu, S.E. Sarma, M.M. Bronstein, J.M. Solomon, Dynamic graph cnn for learning on point clouds, ACM Trans. Graph. (Tog) 38 (5) (2019) 1–12.

[99] B. van Stein, F.X. Long, M. Frenzel, P. Krause, M. Gitterle, T. Bäck, DoE2Vec: Deep-learning based features for exploratory landscape analysis, in: Proceedings of the Companion Conference on Genetic and Evolutionary Computation, in: GECCO '23 Companion, Association for Computing Machinery, New York, NY, USA, 2023, pp. 515–518, http://dx.doi.org/10.1145/3583133.3590609.

[100] D.P. Kingma, M. Welling, et al., An introduction to variational autoencoders, Found. Trends® Mach. Learn. 12 (4) (2019) 307–392.

[101] F.X. Long, B. van Stein, M. Frenzel, P. Krause, M. Gitterle, T. Bäck, Learning the characteristics of engineering optimization problems with applications in automotive crash, in: Proceedings of the Genetic and Evolutionary Computation Conference, 2022, pp. 1227–1236.

[102] G. Cenikj, G. Petelin, T. Eftimov, TransOpt: Transformer-based representation learning for optimization problem classification, 2023, arXiv preprint arXiv:2311.18035.

[103] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A.N. Gomez, Ł. Kaiser, I. Polosukhin, Attention is all you need, Adv. Neural Inf. Process. Syst. 30 (2017).

[104] G. Cenikj, G. Petelin, T. Eftimov, A cross-benchmark examination of feature-based algorithm selector generalization in single-objective numerical optimization, Swarm Evol. Comput. 87 (2024) 101534, http://dx.doi.org/10.1016/j.swevo.2024.101534, URL https://www.sciencedirect.com/science/article/pii/S2210650224000725.

[105] M.V. Seiler, P. Kerschke, H. Trautmann, Deep-ELA: Deep exploratory landscape analysis with self-supervised pretrained transformers for single-and multi-objective continuous optimization problems, 2024, arXiv preprint arXiv:2401.01192.

[106] M. Seiler, U. Skvorc, C. Doerr, H. Trautmann, Synergies of deep and classical exploratory landscape features for automated algorithm selection, in: Proc. of Learning and Intelligent Optimization Conference (LION) 2024, in: Lecture Notes in Computer Science, Springer, 2024, http://dx.doi.org/10.1007/978-3-031-75623-8_29.

[107] M. Seiler, U. Škvorc, G. Cenikj, C. Doerr, H. Trautmann, Learned features vs. Classical ELA on affine BBOB functions, in: International Conference on Parallel Problem Solving from Nature, Springer, 2024, pp. 137–153.

[108] G. Petelin, G. Cenikj, Random filter mappings as optimization problem feature extractors, IEEE Access 12 (2024) 143554–143571, http://dx.doi.org/10.1109/ACCESS.2024.3468723.

[109] T. Eftimov, G. Popovski, Q. Renau, P. Korošec, C. Doerr, Linear matrix factorization embeddings for single-objective optimization landscapes, in: 2020 IEEE Symposium Series on Computational Intelligence, SSCI, IEEE, 2020, pp. 775–782.

[110] M.A. Muñoz, M. Kirley, S.K. Halgamuge, A meta-learning prediction model of algorithm performance for continuous optimization problems, in: Parallel Problem Solving from Nature-PPSN XII: 12th International Conference, Taormina, Italy, September 1-5, 2012, Proceedings, Part I 12, Springer, 2012, pp. 226–235.

[111] G. Petelin, G. Cenikj, The pitfalls of benchmarking in algorithm selection: What we are getting wrong, in: Proceedings of the Companion Conference on Genetic and Evolutionary Computation, in: GECCO '25 Companion, 2025, http://dx.doi.org/10.1145/3712256.3726336.

[112] P. Kerschke, M. Preuss, S. Wessing, H. Trautmann, Low-budget exploratory landscape analysis on multiple peaks models, in: Proceedings of the Genetic and Evolutionary Computation Conference 2016, GECCO '16, Association for Computing Machinery, New York, NY, USA, 2016, pp. 229–236, http://dx.doi.org/10.1145/2908812.2908845.

[113] J. Kudela, M. Juricek, Computational and exploratory landscape analysis of the gkls generator, in: Proceedings of the Companion Conference on Genetic and Evolutionary Computation, in: GECCO '23 Companion, Association for Computing Machinery, New York, NY, USA, 2023, pp. 443–446, http://dx.doi.org/10.1145/3583133.3590653.

[114] G. Cenikj, G. Petelin, M. Seiler, N. Cenikj, T. Eftimov, Landscape features in single-objective continuous optimization: Have we hit a wall in algorithm selection generalization? Swarm Evol. Comput. 94 (2025) 101894, http://dx.doi.org/10.1016/j.swevo.2025.101894, URL https://www.sciencedirect.com/science/article/pii/S2210650225000525.

[115] T. Eftimov, P. Korošec, B. Koroušić Seljak, A Novel Approach to statistical comparison of meta-heuristic stochastic optimization algorithms using deep statistics, Inform. Sci. 417 (2017) 186–215, http://dx.doi.org/10.1016/j.ins.2017.07.015, URL https://www.sciencedirect.com/science/article/pii/S002002551631194X.

[116] S.M. Lundberg, S.-I. Lee, A unified approach to interpreting model predictions, in: Proceedings of the 31st International Conference on Neural Information Processing Systems, NIPS '17, Curran Associates Inc., Red Hook, NY, USA, 2017, pp. 4768–4777.

[117] F. Hutter, H. Hoos, K. Leyton-Brown, An efficient approach for assessing hyperparameter importance, in: International Conference on Machine Learning, PMLR, 2014, pp. 754–762.

[118] N. van Stein, D. Vermetten, A. V. Kononova, T. Bäck, Explainable benchmarking for iterative optimization heuristics, ACM Trans. Evol. Learn. Optim. 5 (2) (2025) http://dx.doi.org/10.1145/3716638.

[119] Z. Pitra, J. Repický, M. Holena, Landscape analysis of Gaussian process surrogates for the covariance matrix adaptation evolution strategy, in: Proc. of Genetic and Evolutionary Computation Conference, GECCO, ACM, 2019, pp. 691–699, http://dx.doi.org/10.1145/3321707.3321861.

[120] M. Christ, N. Braun, J. Neuffer, A.W. Kempa-Liehr, Time series feature extraction on basis of scalable hypothesis tests (tsfresh–a python package), Neurocomputing 307 (2018) 72–77.

[121] M.D. McKay, R.J. Beckman, W.J. Conover, A comparison of three methods for selecting values of input variables in the analysis of output from a computer code, Technometrics 21 (2) (1979) 239–245, http://dx.doi.org/10.1080/00401706.1979.10489755.

[122] A. Janković, C. Doerr, Adaptive landscape analysis, in: Proceedings of the Genetic and Evolutionary Computation Conference Companion, GECCO '19, Association for Computing Machinery, New York, NY, USA, 2019, pp. 2032–2035, http://dx.doi.org/10.1145/3319619.3326905.

[123] A. Jankovic, T. Eftimov, C. Doerr, Towards feature-based performance regression using trajectory data, in: Proc. of Applications of Evolutionary Computation (EvoApplications 2021), in: LNCS, Vol. 12694, Springer, 2021, pp. 601–617, http://dx.doi.org/10.1007/978-3-030-72699-7_38.

[124] A. Kostovska, A. Jankovic, D. Vermetten, J. de Nobel, H. Wang, T. Eftimov, C. Doerr, Per-run algorithm selection with warm-starting using trajectory-based features, in: International Conference on Parallel Problem Solving from Nature, Springer, 2022, pp. 46–60.

[125] D. Vermetten, H. Wang, K. Sim, E. Hart, To switch or not to switch: Predicting the benefit of switching between algorithms based on trajectory features, in: International Conference on the Applications of Evolutionary Computation (Part of EvoStar), Springer, 2023, pp. 335–350.

[126] B. Shahriari, K. Swersky, Z. Wang, R.P. Adams, N. de Freitas, Taking the human out of the loop: A review of Bayesian optimization, Proc. IEEE 104 (1) (2016) 148–175, http://dx.doi.org/10.1109/JPROC.2015.2494218.

[127] C. Benjamins, A. Jankovic, E. Raponi, K. van der Blom, M. Lindauer, C. Doerr, Towards automated design of Bayesian optimization via exploratory landscape analysis, in: Sixth Workshop on Meta-Learning at the Conference on Neural Information Processing Systems, 2022.

[128] G. Cenikj, G. Petelin, C. Doerr, P. Korošec, T. Eftimov, Beyond landscape analysis: DynamoRep features for capturing algorithm-problem interaction in single-objective continuous optimization, Evol. Comput. (2025) 1–28, http://dx.doi.org/10.1162/evco_a_00370, arXiv:https://direct.mit.edu/evco/article-pdf/doi/10.1162/evco_a_00370/2507310/evco_a_00370.pdf.

[129] J. Adair, G. Ochoa, K.M. Malan, Local optima networks for continuous fitness landscapes, in: Proceedings of the Genetic and Evolutionary Computation Conference Companion, 2019, pp. 1407–1414.

[130] D. Wales, J. Doye, Global optimization by basin-hopping and the lowest energy structures of lennard-jones clusters containing up to 110 atoms, J. Phys. Chem. A 101 (1998) http://dx.doi.org/10.1021/jp970984n.

[131] P. Mitchell, G. Ochoa, R. Chassagne, Local optima networks of the black box optimisation benchmark functions, in: Proceedings of the Companion Conference on Genetic and Evolutionary Computation, 2023, pp. 2072–2080.

[132] G. Ochoa, K.M. Malan, C. Blum, Search trajectory networks of population-based algorithms in continuous spaces, in: International Conference on the Applications of Evolutionary Computation (Part of EvoStar), Springer, 2020, pp. 70–85.

[133] Q. Renau, E. Hart, On the utility of probing trajectories for algorithm-selection, in: International Conference on the Applications of Evolutionary Computation (Part of EvoStar), Springer, 2024, pp. 98–114.

[134] G. Cenikj, G. Petelin, T. Eftimov, ClustOpt: A clustering-based approach for representing and visualizing the search dynamics of numerical metaheuristic optimization algorithms, in: 2025 IEEE Congress on Evolutionary Computation, CEC, IEEE, 2025, pp. 1–8.

[135] Y. Xian, B. Schiele, Z. Akata, Zero-shot learning-the good, the bad and the ugly, in: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, 2017, pp. 4582–4591.

[136] J. Rapin, O. Teytaud, Nevergrad - A gradient-free optimization platform, 2018, https://GitHub.com/FacebookResearch/Nevergrad.

[137] R. Aljundi, K. Kelchtermans, T. Tuytelaars, Task-free continual learning, in: Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition, 2019, pp. 11254–11263.

[138] G.M. Van de Ven, A.S. Tolias, Three scenarios for continual learning, 2019, arXiv preprint arXiv:1904.07734.

[139] L. Wang, X. Zhang, H. Su, J. Zhu, A comprehensive survey of continual learning: Theory, method and application, 2023, arXiv preprint arXiv:2302.00487.

[140] B. Lacroix, J. McCall, Limitations of benchmark sets and landscape features for algorithm selection and performance prediction, in: Proc. of Genetic and Evolutionary Computation (GECCO'19, Companion), ACM, 2019, pp. 261–262, http://dx.doi.org/10.1145/3319619.3322051.

[141] A. Jankovic, C. Doerr, Landscape-aware fixed-budget performance regression and algorithm selection for modular CMA-ES variants, in: Proceedings of the 2020 Genetic and Evolutionary Computation Conference, 2020, pp. 841–849.

[142] R. Tanabe, Benchmarking feature-based algorithm selection systems for black-box numerical optimization, IEEE Trans. Evol. Comput. 26 (6) (2022) 1321–1335.

[143] A. Nikolikj, G. Cenikj, G. Ispirova, D. Vermetten, R.D. Lang, A.P. Engelbrecht, C. Doerr, P. Korošec, T. Eftimov, Assessing the generalizability of a performance predictive model, in: Proceedings of the Companion Conference on Genetic and Evolutionary Computation, in: GECCO '23 Companion, Association for Computing Machinery, New York, NY, USA, 2023, pp. 311–314, http://dx.doi.org/10.1145/3583133.3590617.

[144] A. Nikolikj, C. Doerr, T. Eftimov, RF+ clust for leave-one-problem-out performance prediction, in: International Conference on the Applications of Evolutionary Computation (Part of EvoStar), Springer, 2023, pp. 285–301.

[145] A. Nikolikj, M. Pluháček, C. Doerr, P. Korošec, T. Eftimov, Sensitivity analysis of rf+clust for leave-one-problem-out performance prediction, in: 2023 IEEE Congress on Evolutionary Computation, CEC, 2023, pp. 1–8, http://dx.doi.org/10.1109/CEC53210.2023.10254146.

[146] A. Nikolikj, S. Džeroski, M.A. Muñoz, C. Doerr, P. Korošec, T. Eftimov, Algorithm instance footprint: Separating easily solvable and challenging problem instances, in: Proceedings of the Genetic and Evolutionary Computation Conference, GECCO '23, Association for Computing Machinery, New York, NY, USA, 2023, pp. 529–537, http://dx.doi.org/10.1145/3583131.3590424.

[147] K. Dietrich, O. Mersmann, Increasing the diversity of benchmark function sets through affine recombination, in: G. Rudolph, A.V. Kononova, H. Aguirre, P. Kerschke, G. Ochoa, T. Tuˇ sar (Eds.), Parallel Problem Solving from Nature – PPSN XVII, Springer International Publishing, Cham, 2022, pp. 590–602.

[148] F.X. Long, D. Vermetten, A. Kononova, R. Kalkreuth, K. Yang, T. Bäck, N. van Stein, Challenges of ELA-guided function evolution using genetic programming, in: Proceedings of the 15th International Joint Conference on Computational Intelligence - Volume 1: ECTA, SciTePress, 2023, pp. 119–130, http://dx.doi.org/10.5220/0012206200003595.

[149] A. Jankovic, D. Vermetten, A. Kostovska, J. de Nobel, T. Eftimov, C. Doerr, Trajectory-based algorithm selection with warm-starting, in: 2022 IEEE Congress on Evolutionary Computation, CEC, IEEE, 2022, pp. 1–8.

[150] J. Stork, A.E. Eiben, T. Bartz-Beielstein, A new taxonomy of global optimization algorithms, Nat. Comput. (2020) 1–24.

[151] A. Jankovic, G. Popovski, T. Eftimov, C. Doerr, The impact of hyper-parameter tuning for landscape-aware performance regression and algorithm selection, in: Proceedings of the Genetic and Evolutionary Computation Conference, 2021, pp. 687–696.

[152] A.V. Kononova, N. van Stein, O. Mersmann, T. Bäck, T. Bartz-Beielstein, T. Glasmachers, M. Hellwig, S. Krey, J. Kůdela, B. Naujoks, L. Papenmeier, E. Raponi, Q. Renau, J. Rook, L. Schäpermeier, D. Vermetten, D. Zaharie, Benchmarking that matters: Rethinking benchmarking for practical impact, 2025, arXiv:2511.12264, URL https://arxiv.org/abs/2511.12264.