Information Sciences 680 (2024) 121134

Contents lists available at ScienceDirect

Information Sciences

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

Opt2Vec - a continuous optimization problem representation based on the algorithm's behavior: A case study on problem classification

Peter Korošec *, Tome Eftimov

Computer Systems Department, Jožef Stefan Institute, Jamova cesta 39, Ljubljana, SI-1000, Slovenia

ARTICLE INFO

Keywords:
Continuous optimization
Problem representation
Autoencoder
Dynamic problem characterization
Anytime problem identification

ABSTRACT

Characterization of the optimization problem is a crucial task in many recent optimization research topics (e.g., explainable algorithm performance assessment, and automated algorithm selection and configuration). The state-of-the-art approaches use exploratory landscape analysis to represent the optimization problem, where for each one, a set of features is extracted using a set of candidate solutions sampled by a sampling strategy over the whole decision space. This paper proposes a novel representation of continuous optimization problems by encoding the information found in the interaction between an algorithm and an optimization problem. The new problem representation is learned using the information from the states/positions in the optimization run trajectory (i.e., the candidate solutions visited by the algorithm). With the novel representation, the problem can be characterized dynamically during the optimization run, instead of using a set of candidate solutions from the whole decision space that have never been observed by the algorithm. The novel optimization problem representation is called Opt2Vec and uses an autoencoder type of neural network to encode the information found in the interaction between an optimization algorithm and optimization problem into an embedded subspace. The Opt2Vec representation efficiency is shown by enabling different optimization problems to be successfully identified using only the information obtained from the optimization run trajectory.

1. Introduction

An essential aspect for various learning tasks in optimization, such as problem classification [1,2], automated algorithm performance prediction [3–6], automated algorithm selection [7–9], and automated algorithm configuration [10], is to possess a comprehensive representation of the landscape characteristics of optimization problems. The significance of possessing effective problem representations is connected to choosing unbiased and diverse problem instances, which aids in mitigating bias during performance assessment analyses [11]. Moreover, the capability to automatically identify the problem being solved anytime in the optimization process is crucial for ensuring the efficiency of the optimization process [4,8,10].

Nowadays, when tackling an optimization problem, a fundamental consideration emerges: determining the appropriate representation for the problem and identifying which landscape features should be derived from the data. The decision between encompassing the entire search space or focusing on specific regions typically explored by algorithms lacks a definitive answer, as the selection of

* Corresponding author.
E-mail address: peter.korosec@ijs.si (P. Korošec).

https://doi.org/10.1016/j.ins.2024.121134
Received 26 September 2023; Received in revised form 27 June 2024; Accepted 28 June 2024

Available online 3 July 2024

representation relies on the specific application scenario. For static analysis within the problem landscape, a static representation of the entire search space might be adequate. However, to explore the interactions between optimization problems and algorithms or to provide personalized insights into algorithm behavior for a specific problem, information from the visited or observed parts of the search space becomes indispensable.

Our contribution: In this paper, we propose a novel methodology for representing optimization problems using the optimization algorithms' behavior, called Opt2Vec. The proposed methodology is developed to represent parts from the optimization problem search space that are visited by the algorithm in a particular timestamp/iteration of the optimization process. It encodes the anytime information of how a given algorithm interacts with that problem, while at the same time ensuring invariance to simple transformations like shifting and scaling. These are all significant contributions that remove the need for extra time-consuming calculations of features for further machine learning tasks and at the same time inherently provide information about correlations between problem instance and algorithm instance features, which are lacking when performing standard calculations of static landscape features. Due to invariance to simple transformations, the representation is more robust to characteristics of problem instances and/or algorithm behavior, which usually do not influence algorithm performance. Capturing interactions between algorithm and problem instances could provide important information for effective dynamic algorithm configuration. The effectiveness of the Opt2Vec representation is evaluated in the task of problem classification to explore if it captures enough characteristics of the problem so it can be used for anytime identification of the optimization problem being solved.

The main contributions of the paper are the following:

- Opt2Vec is a novel continuous optimization problem representation based on the algorithm's behavior. It offers an efficient and concise way to obtain relevant information about a problem. This learned representation can serve as input data for other machine learning and deep learning tasks.
- The optimization problem being solved can be identified solely from the current state/position (population) in the optimization run. The new representation, Opt2Vec, uses only scaled information about the solutions' population, making it invariant to simple transformations like shifting and scaling. Hence, Opt2Vec is well-suited for dynamically characterizing the optimization problem at any time.
- The scalability of the proposed Opt2Vec representations extends across various problem dimensions, effectively capturing the characteristics of the problem being solved regardless of its dimensionality.
- To the best of our knowledge, this is a first attempt to apply representation learning on individual population of optimization process trajectory. The embedding vectors of Opt2Vec representation provided enough information for successful (over 80% accuracy) problem classification from individual population in the optimization run and significantly outperformed compared exploratory landscape analysis-based representations.

The paper is organized as follows: Section 2 gives an overview of the related work. Section 3 introduces the Opt2Vec representations, a novel way of representing an optimization problem that encodes the information from the current state/position in the optimization run trajectory. Section 4 defines an optimization problem classification task that will be used to evaluate the introduced representations. Section 5 shows the effectiveness of the proposed representations in classifying test problems of various dimensions. Finally, the discussion and conclusion with ideas for further work are presented in sections 6 and 7.

2. Related work

To describe the characteristics of the optimization problems, fitness landscape analysis (FLA) is used [12]. One approach to doing FLA is to characterize the optimization problems based on some high-level characteristics of the problem landscape (e.g., separability, multimodality) [13], which requires an expert's knowledge to identify them. Another way of performing FLA is to calculate low-level analytical features from a set of candidate solutions sampled with some sampling technique from the whole decision space of the problem landscape [14,15].

The most commonly used approach for extracting low-level landscape features is known as exploratory landscape analysis (ELA) [14]. ELA calculates a set of mathematical and statistical numerical features for each problem instance from a set of candidate solutions selected by using some sampling strategy (e.g., uniform sampling or Latin hypercube sampling). In 2016, the R package flacco [16] was released, which provides an easy way to calculate many of these proposed features. Currently, it provides 343 different landscape features, which can be split into cheap and expensive based on the computational cost required to calculate them. In many recent studies, only cheaper features are explored [4,17]. It has been also shown that the ELA calculation is sensitive to the sampling strategy and sample size used [18,19] and is not invariant to simple problem transformation such as shifting and scaling [17,20].

Recently, another approach known as topological landscape analysis (TLA) has been proposed to extract low-level analytical features for single-objective continuous optimization problems [21,22]. These features provide a fresh perspective on optimization problems by quantifying their similarity based on the presence of topological structures. The findings demonstrate that these topologically inspired features are independent of the existing ELA landscape feature groups. As a result, they capture distinct and complementary information about the problems under study. Both ELA and TLA features are mostly applicable to a static learning task, where they should be calculated in an offline setting before using them in the learning process. This comes as a result of their calculation which is a time-consuming task [23].

Other approaches to calculating low-level landscape features are known as feature-free approaches that use a deep neural architecture to learn the low-level landscape features [24,25]. In [24], a collection of deep-learning-based features have been proposed based on using convolutional neural networks and transformers that use an alternative representation of the set of candidate solutions in the form of point clouds and 2D images as input to extract low-level analytical features. DoE2Vec are other deep-learning features learned by using a variational autoencoder using large training data of candidate solutions selected with some sampling strategy from the decision space of a set of problems. These features can be efficiently used to find similar problems. Moreover, when used in conjunction with classical ELA features in problem classification tasks, they lead to significant performance improvements.

All the above-mentioned approaches (ELA, TLA, and feature-free) calculate the low-level features by using an artificial set of candidate solutions obtained by some sampling strategy, which is not related to the set of candidate solutions that are visited by an algorithm during its optimization process. However, numerous research papers highlight the lack of a guaranteed correlation between landscape features calculated using artificial samples and the performance of the algorithms. Notably, similar landscape features can lead to significantly divergent algorithm performances [26,27].

To go beyond this, several studies performed per-instance and per-run automated algorithm selection by calculating the ELA features using all candidate solutions that were observed during an optimization algorithm run on a particular problem [28–31]. In such cases, the whole information observed by the algorithm during its trajectory (i.e., all visited candidate solutions instead of sampling them with some sampling strategy from the decision space) is used to calculate the ELA features. One drawback here is that even though the calculation is done by the visited candidates' solution by merging them all together into one big sample, the longitudinal dimension of how they were visited (i.e., the time-series component) is not taken into the calculation process. These studies are a step into what is called trajectory-based feature extractions, or more specifically trajectory-based ELA calculation. Recently, another study has been proposed where four statistical features (i.e., mean, standard deviation, min, and max) have been calculated on each decision variable, and the objective value from the problem through all iterations of the optimization run, called DynamoRep features [32]. Next, contacting the four statistics through all iterations provides us with representations about the entire trajectory of the optimization run. The study demonstrates that the proposed DynamoRep features capture enough information to accurately identify the problem class on which the optimization algorithm was running.

To represent the trajectories that represent the interactions between an algorithm and problem instance, there are also approaches rooted in continuous optimization and complex networks. Local Optima Networks (LONs) [33] simplify discrete fitness landscapes, representing local optima as nodes and transitions as edges through exploration search operators. For continuous landscapes, a related model called compressed monotonic LON (CMLON) [34] groups interconnected nodes sharing the same fitness within the monotonic LON (MLON). CMLONs, applied to visualize 24 BBOB problem classes across dimensions [35], display varied representations linked to problem properties. Calculated network metrics and dimensionality reduction techniques classify and compare these problems, with crucial insights for multimodal problems in higher dimensions, where CMLONs become intricate for visual interpretation. Resembling LONs, Search Trajectory Networks (STNs) [36] are innovative instruments for examining and visualizing the behavior of population-based algorithm instances in continuous spaces. Derived from the concept of Local Optima Networks (LONs), where nodes represent local optima, STNs shift the focus to various states within the optimization trajectory, extending beyond local optima. The edges in STNs represent the progression between these states, augmenting the usefulness of network-based models in comprehending heuristic search algorithms.

Despite the progress in analyzing fitness landscapes based on trajectories, trajectory-based ELA, DynamoRep, and features derived from CMLON or STN concentrate on capturing a representation for the entire optimization run trajectory. However, a crucial aspect that remains unexplored, and is the focus of this study, is understanding the amount of information contained in a specific, small segment of the optimization run trajectory (such as a particular timestamp, iteration, or population). This can become particularly significant for effective dynamic algorithm configuration.

3. The Opt2Vec methodology

The proposed Opt2Vec methodology aims to capture the dynamics between an optimization algorithm and a problem instance by analyzing populations generated at each iteration during optimization. This involves scaling the collected data samples (i.e., populations) to ensure inherent invariance to basic transformations such as shifting and scaling. By doing so, the methodology mitigates the influence of these transformations on machine learning feature calculations, which could otherwise erroneously treat two samples as distinct when they should be considered the same. Subsequently, unsupervised learning, in the form of autoencoders, is employed to embed the scaled, captured data into a representation suitable for further machine learning tasks. It will be demonstrated that such a representation of a single scaled data sample contains sufficient information for successful problem classification. The ultimate goal is that the successful classification of a problem instance at any point during the optimization run will enable leveraging knowledge from past algorithm performances on similar problem instances to dynamically configure algorithm hyper-parameters for optimal performance in the remainder of the optimization run.

The proposed Opt2Vec methodology consists of the following steps.

- Data pre-processing, which consists of two steps.
  - Data scaling, which aims to make data less sensitive to simple transformations seen across different problem instances.
  - Data reduction, which aims to decrease the volume of data for the upcoming learning step, enhancing both its quality and processing speed.
- Learning the Opt2Vec representations using an autoencoder network architecture on the pre-processed data.

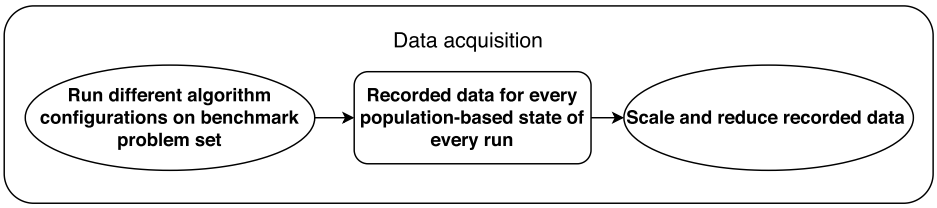


Fig. 1. The data acquisition pipeline for computing Opt2Vec continuous optimization problem representations consists of running various algorithms, recording the trajectory of populations, and finally scaling and reducing data to be used in the computing task. In the presented diagram ellipses represent actions, while squares represent states.

In the following section, the data generation process will be presented, as it is essential to gather suitable data for experimentation within the proposed Opt2Vec methodology. In subsequent sections, a comprehensive explanation of each of the mentioned steps will be provided.

3.1. Data generation

Since we are interested in computing a problem representation for different states/positions of the trajectory, a population-based algorithm is suitable to generate data for learning the Opt2Vec representations. Each optimization run (i.e., an algorithm is executed on a particular problem instance) consists of many iterations of the optimization algorithm until some stopping condition is met. During the optimization run, the candidate solutions that form the population and their offspring are recorded for every iteration. In this way, the optimization algorithm's behavior is recorded concerning how its population changes during the optimization process (i.e., optimization trajectory). The population and their offspring at every iteration represent the different states/positions during the trajectory of the optimization run. In addition, each population from each run is labeled with the information about the optimization problem (i.e., problem name and dimension) that is being solved from which the data is collected. Such kind of data management allows us to investigate whether populations generated by the optimization algorithm in a single iteration of the run carry sufficient information to successfully train a model that can classify the optimization problem on which the optimization algorithm is being executed. The learning task can be assumed as an anytime classification of the problem being solved.

3.2. Data pre-processing

The process of computing Opt2Vec continuous optimization problem representations begins with the pre-processing (i.e., scaling and reduction) of data to be used in the computing task. The general data acquisition pipeline is presented in Fig. 1. In more detail, data scaling and data reduction are explained in the following sections.

3.2.1. Data scaling

Before computing the problem representations, the collected data is scaled to improve invariance to simple transformations that can exist between the problem landscape spaces of different problem instances. This is a really important step since when characterizing the problem we should not be interested in solutions' absolute values (which can be always shifted and/or scaled, while still preserving problem characteristics relevant to optimization algorithm), but in the relations between solutions (i.e., the decision variables and the objective value) that define the problem (or sampled part of it) landscape characteristics.

For easier explanation, let us assume the following simple multimodal problem $f(x) = -(1.4 - 3.0 * x) * sin(18.0 * x)$ where $x \in X = [0, 1]$. On this problem, we apply a simple transformation in the following form $f_i(x) = a_i * f(x - b_i) + c_i$, where $a_i \neq 0$ represents scaling factor, b_i represents shifting value in search space, while c_i represents a shift in objective space. For problem instance $i = 1$ we selected the following values $a_1 = 2$, $b_1 = -0.5$, $c_1 = 0$ and for instance $i = 2$ we selected values $a_1 = 4$, $b_1 = 0.5$, $c_1 = 4$. To ensure that we sample from the same search space of the problem we also change the range for each of the instances in the following way, $X_i = X + b_i$ (i.e., the range is shifted for the value of b_i). The resulting problem instances are presented in Fig. 2, where we can observe that with applied transformations they preserve the same landscape characteristics.

Next, if we apply min-max scaling on the whole problem (i.e., *min* and *max* for the decision space and the objective value are selected from all candidate solutions that are visited across all iterations), both instances (f_1 and f_2) would transform to the same problem instance (shown in Fig. 2 as f_{scaled}). Nevertheless, our objective is to perform anytime problem classification, even during the optimization run. Due to the nature of the task, we lack precise information about the exact *min* and *max* values of the problem beforehand. Consequently, it is not feasible to apply such scaling to all candidate solutions within each population during the optimization run. For this purpose, the *min* and *max* values are obtained within each population, where the information provided by the individual population is used for scaling and as such the problem is characterized by the sampled space of the population. Such kind of scaling allows us to describe the landscape characteristics of the problem based on the population (i.e., the current state/position of the trajectory of the optimization run, shown as sampled points in Fig. 2). Consequently, it skews the correctness of the representation as a whole (see $f_{\text{population}}$ in Fig. 2), but on the other side, it will provide scaled information about the current state of the algorithm (i.e., the set of visited candidate solutions). This kind of scaling allows us a personalized characterization of the problem (independently of the simple transformations) and the algorithm state (i.e., exploration or exploitation phase) at every state/point in the search process.

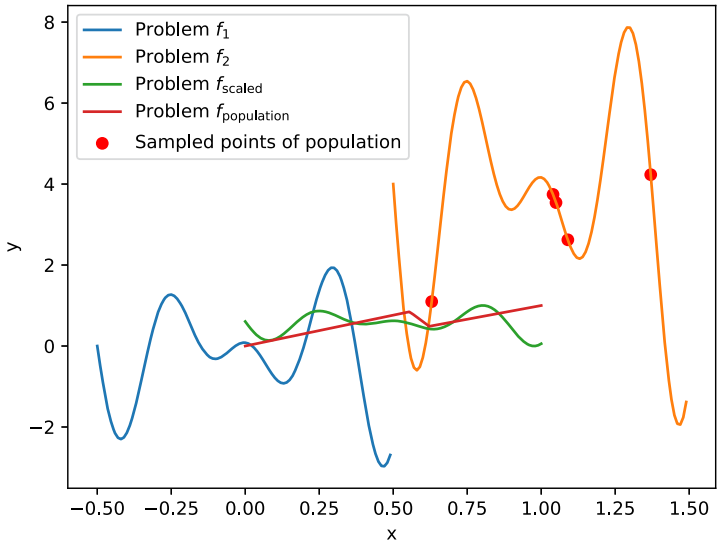


Fig. 2. Multimodal problem presented with shifted and scaled instances f_1 and f_2, and as scaled version f_{scaled}, where we can observe that with applied transformations they preserve the same landscape characteristics. In addition, an example of a population with five solutions is presented through sample points, and its representation is shown as $f_{\text{population}}$.

By conducting this pre-processing, we ensure the same problem representation (assuming optimization algorithm would have the same trajectory behavior) regardless of shifting and scaling applied to the problem (e.g., in the above multimodal problem example, the values $a \neq 0$, b, and c). Even more importantly such landscape characteristics can be found in any function that can lead to characteristics that are transferable to unseen problem instances. Furthermore, all the necessary computations for problem representation are limited to the current population, leading to minimal space and time complexity requirements.

The min-max scaling data values from all populations are in the range $[0, 1]$. The population at each iteration from each run (as seen by the optimization algorithm at a specific iteration/state of the trajectory) can be represented through the scaled matrix, A, of dimension $m \times n$, where m is the population size and n is the solution length (i.e., number of search (decision) variables plus its evaluation (objective) value, which is the result from evaluating the solution on that problem). For example, let us assume that a parabola problem in $n-1$ dimensions, $y = \sum_{j=1}^{n-1} x_j^2$, is optimized by an algorithm. It follows that x_j, $j = 1, \ldots, n-1$, are the decision variables that describe the solution, and y corresponds to the solution's evaluation result (i.e., objective value). In this way, each row of the matrix is filled by the solution's objective value, y, and its decision variables, x_j, $j = 1, \ldots, n-1$. Because the decision variables and the objective values can be in different data ranges, scaling is performed twice within each population separately, once for the decision variables and once for the objective values. Finally, we end up with as many scaled matrices as there are iterations from the optimization run trajectory, where each one represents an interaction between the algorithm and the problem at a specific state from the trajectory of the optimization process.

We would like to emphasize that the task of scaling is not to transform the problem landscape to some "equal", alternative representation, but to remove the problem specifics that can often mask the problem's relevant characteristics. For instance, we are not interested in problem parameters a, b, and c for the above multimodal problem example, but in the scaled representation f_{scaled}.

3.2.2. Data reduction

Due to the nature of how population-based optimization algorithms behave, the population considerably changes with every iteration at the beginning of the search process (i.e., when the algorithms are performing exploration, or exploring a whole search space for regions with potential for good solutions). Later on, the changes become increasingly smaller (i.e., when the algorithm is performing exploitation, or searching for the best/optimal solution within the identified regions) until only minor changes are applied, if any at all. Based on this, one can assume that at the beginning of the optimization process, almost every new population contains new information about the optimization problem search space. With smaller changes being applied with each iteration means that new information becomes diminished with every subsequent population. This does not mean that information from the exploitation phase should be discarded since it provides valuable information about the local topology of the function. However, the amount of redundant information is much higher in the exploitation phase, since often only a small amount of candidate solutions of the population changes.

To reduce the amount of data for learning the Opt2Vec representations, only populations that contain either new or relevant information about the optimization problem search space should be selected. For this purpose, the Frobenius norm, used as a measure of difference or information relevancy, is applied to all scaled matrices obtained from the optimization process trajectories. It is represented by the formula:

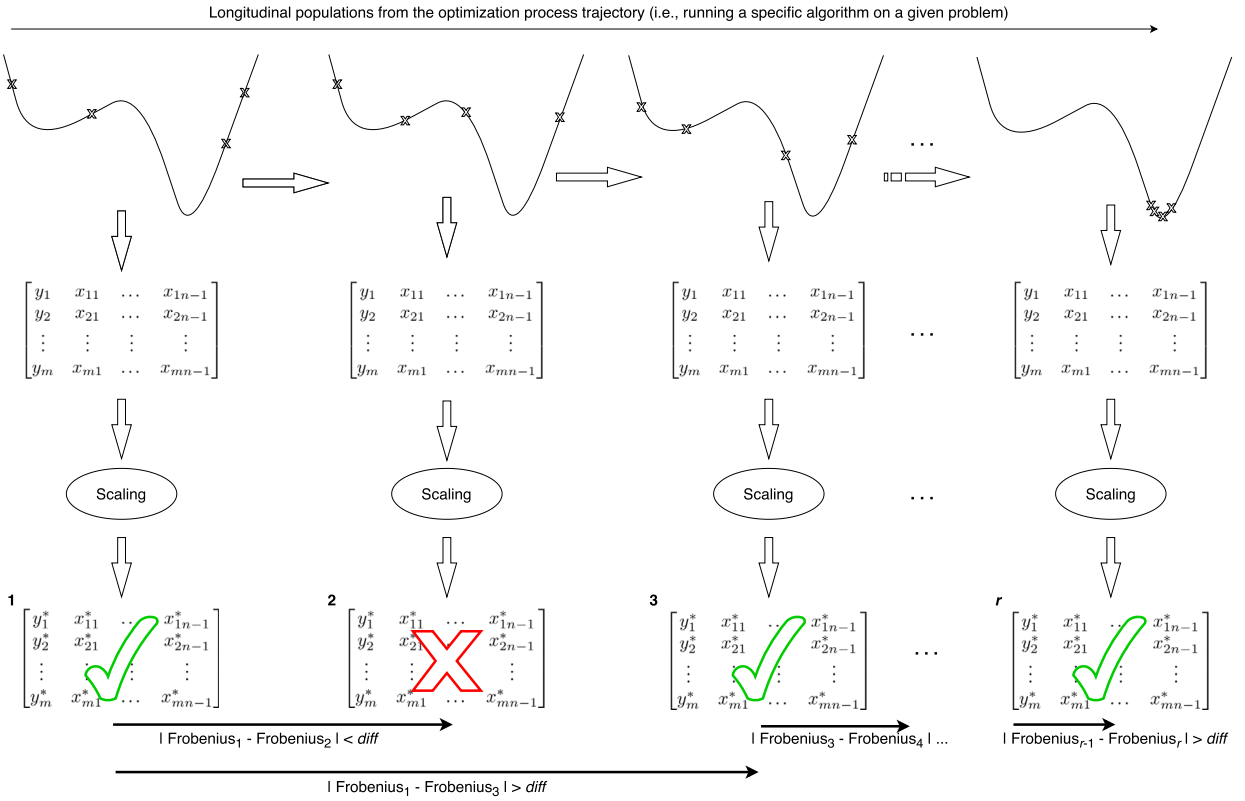


Fig. 3. The data pre-processing pipeline consisting of data generation by recording populations, applying scaling, and data reduction using differences of Frobenius norms.

$$\|A\|_F = \sqrt{\sum_{i=1}^{m}\sum_{j=1}^{n}|a_{ij}|^2}.$$

Here, a_{ij} is the element of the scaled matrix A of dimension $m \times n$, where $i = 1, \ldots, m$ and $j = 1, \ldots, n$. When comparing two matrices, the Frobenius norm is normally applied to the differences between them. However, in our case, the order of rows (i.e., decision variables and objective value) is not always the same since it can change from iteration to iteration. So, it follows that two populations with the same solutions in different orders could have a high Frobenius norm, although they are the same from the optimization algorithm's perspective. To get around this issue, the Frobenius norm is calculated for every scaled matrix. We would like to note that some other measure of difference could be used (e.g., Hausdorff distance [37]), however, the primary goal is to reduce the amount of data to be processed in training based on some similarity measure, so we consider our proposed approach appropriate for the task.

If the difference between two Frobenius norms is smaller than some predefined value, *diff*, the matrices are considered too similar, so only one of them needs to be retained for training purposes. There is no need to store two similar matrices since the second one does not bring any new information and can only deteriorate the performance of machine learning methods by unnecessarily increasing training data size and providing bias to such training data instances. So, the last retained scaled matrix is compared with the next scaled matrix calculated from the optimization process trajectory. This process is applied to all scaled matrices from the beginning to the end of the optimization run, going through the whole optimization process trajectory. In this way, only adjacent scaled matrices from the optimization process trajectory run that are deemed sufficiently different are retained, forming a dataset. Each retained scaled matrix is considered a data instance. For better presentation, the whole data pre-processing and reduction process for a single optimization run is presented in Fig. 3.

The data instances can be large, depending on the number of candidate solutions in the population and the optimization problem's dimensionality. This is a potentially significant problem for machine learning approaches since it can reduce their performance and robustness while increasing time complexity. For this purpose, a representation learning concept that resulted in the embedded Opt2Vec representations is proposed.

3.3. Learning the Opt2Vec representations

Next, the unsupervised learning technique used for learning the embedded Opt2Vec continuous optimization problem representations is explained in more detail. The so-called original Opt2Vec representations are introduced as a special case of the Opt2Vec representations. Finally, a classification model used to test their utility is explained in more detail.

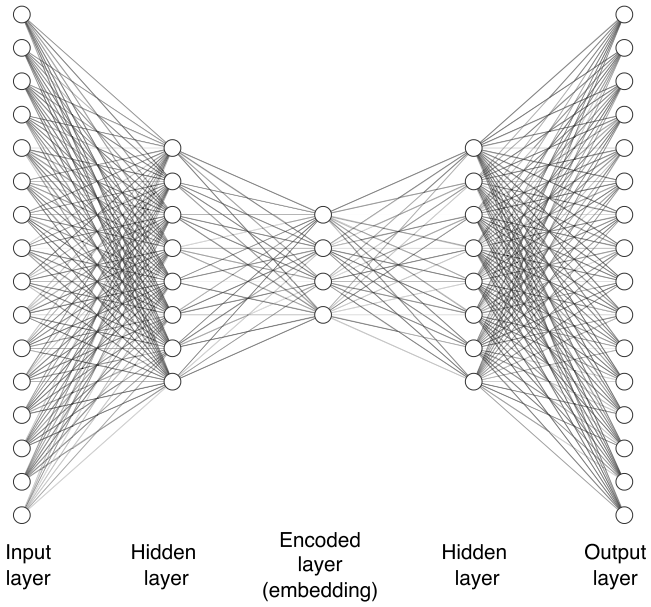


Fig. 4. The autoencoder's architecture consists of two encodings and two decoding steps. In the middle (Encoded) layer, the values represent the embedding for the input.

3.3.1. The autoencoder

One of the most successful and efficient unsupervised representation learning methods is to utilize autoencoders [38]. Autoencoders are artificial neural networks with a symmetrical structure. They are trained to reconstruct the input data onto the output layer. The middle layer in the architecture provides an encoded vector (embedding) of the input data or dimensionality reduction. In parallel, the autoencoder also generates a close representation of the original input data using only the reduced encoding information. The autoencoders can perform automated feature fusion by applying nonlinear techniques. A taxonomy of different architectures for nonlinear feature fusion and use cases is presented in [39]. Autoencoders are primarily used in natural language processing and computer vision [40,41], where they have been utilized to transform the data instance in another reduced space with little or no loss of information. Since our focus is to represent the problem landscape at the current state of the optimization run (i.e., population visited by the algorithm at that iteration) and not the entire trajectory, we have not explored other representations that use sequential data (iterations) such as long short-term memory architectures or transformers. We have decided to use a standard, basic autoencoder, as a solid base on which the methodology can be tested and leave performance-based enhancements, in the form of more advanced neural networks, for future work.

For learning the embedded Opt2Vec representation, the autoencoder's architecture is composed of two encodings and two decoding steps (Fig. 4). This architecture has been selected for evaluating the proof of concept of the proposed approach, without doing a network architecture search which is one direction for future work. All the layers, except the output layer, are fully connected to the next layer using the Rectified Linear Unit, "relu", activation function. The output layer has a "sigmoid" activation function, so the outcome is always between 0 and 1, as is the case for the input, i.e., scaled data instance where every value is between 0 and 1. The input layer's width is equal to the longest vector obtained from the data instance. The widths of the hidden layers are determined as a geometrical progression between the input and embedded layer. In the case of the encoder part the widths decrease, while in the decoder part, the widths increase. At each step, the information is compressed by the same factor. For example for an input layer of width 111 and the encoded layer width of 9, the width of the hidden layer would be 32. For decoding, the layers are the inverse of the encoding layers. Finally, the loss function is defined as the mean squared error between the input and the output vector.

3.3.2. Training the embedded Opt2Vec representations

Once the data is collected, the autoencoder architecture is used to train a model that further defines the embedded Opt2Vec representations. The collected data consists of datasets (i.e., separate runs) where each of them consists of data instances (i.e., retained scaled matrices) calculated from one optimization algorithm run (i.e., containing the information of one optimization process trajectory). Datasets are split into the train, validate, and test sets so the embedded Opt2Vec representation transferability can be checked. Since data instances are of varying sizes (i.e., due to different problem dimensions and population sizes), the matrices are padded with zeros (Table 1) to meet the predefined maximal matrix width and height. The padded matrices are then used as input data for the autoencoder. The actual input is a vector, where the padded matrix is flattened by concatenating the rows of the matrix into a single vector. Due to the split on the dataset level, each dataset (one optimization run) can only belong to one of the three sets (train, validation, test) used for learning to avoid overfitting. In this way, we can mimic the autoencoder's behavior when applied to data from a new optimization run while investigating the methodology's transferability, e.g., not looking at the autoencoder's behavior within optimization runs but between them. The training result is a nonlinear transformation of the input data into condensed, encoded

Table 1
A padded matrix with zeros.

solution							
result	variables						
y_1	x_{11}	x_{12}	x_{13}	x_{14}	x_{15}	x_{16}	0
y_2	x_{21}	x_{22}	x_{23}	x_{24}	x_{25}	x_{26}	0
y_3	x_{31}	x_{32}	x_{33}	x_{34}	x_{35}	x_{36}	0
y_4	x_{41}	x_{42}	x_{43}	x_{44}	x_{45}	x_{46}	0
y_5	x_{51}	x_{52}	x_{53}	x_{54}	x_{55}	x_{56}	0
y_6	x_{61}	x_{62}	x_{63}	x_{64}	x_{65}	x_{66}	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0

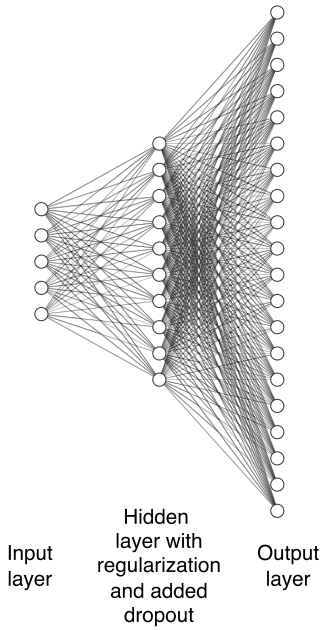


Fig. 5. A three-layer, fully connected architecture of the classification neural network is used to classify the representations presented at the input into optimization problem classes on the output.

subspace (defined by the encoding layers), which defines the embedded Opt2Vec representation. The novel problem representation is based on an algorithm's behavior (i.e., personalizing the interaction that happens between the problem and the algorithm, and not treating the problem alone discarding the algorithm's behavior).

3.3.3. The original Opt2Vec representation

The original Opt2Vec representation can be considered a special case of the Opt2Vec representation, where no transformations are applied to the data instances (i.e., the autoencoder step is skipped). For example, if the population size and problem dimension are set to 10 (i.e., $m = 10$ and $n = 11$), a vector of length 110 ($m * n$), which is a concatenation of the rows of the scaled matrix, defines the original Opt2Vec representation.

4. Optimization problem classification

To test the utility of the Opt2Vec representations, a three-layer, fully connected classification neural network (Fig. 5) has been used to classify them into optimization problem classes (i.e., a combination of the problem name and its dimension). The basic architecture is defined by the input layer width (e.g., length of the Opt2Vec representation vector) and output layer width (number of optimization problem classes). Between them, there is a hidden layer whose width is in geometric progression to the width of the input and output layers. The activation function is again set to "relu". Simultaneously, "softmax" is used as an activation function for the output layer, so only one class (optimization problem and its dimension) can be identified at a time (i.e., multi-class classification). The neural network was selected as a classifier for several reasons. First, the encoder part acts as an input to the classifier. This enables us to easily combine feature extractor and classifier into one deeper neural network. For these experiments, this was not done for the practical reason of step-by-step comparison between different approaches but will be considered in the future. Second, due to large amounts of data, the common off-line machine learning classification algorithms are not efficient at learning. Third, the goal is not to

Table 2
CEC 2014 Competition functions used in experiments.

Family	Function
Unimodal Functions	Rotated High Conditioned Elliptic Function
	Rotated Bent Cigar Function
	Rotated Discus Function
Simple Multimodal Functions	Shifted and Rotated Rosenbrock's Function
	Shifted and Rotated Ackley's Function
	Shifted and Rotated Weierstrass Function
	Shifted and Rotated Griewank's Function
	Shifted Rastrigin's Function
	Shifted and Rotated Rastrigin's Function
	Shifted Schwefel's Function
	Shifted and Rotated Schwefel's Function
	Shifted and Rotated Katsuura Function
	Shifted and Rotated HappyCat Function
	Shifted and Rotated HGBat Function
	Shifted and Rotated Expanded Griewank's plus Rosenbrock's Function
	Shifted and Rotated Expanded Schaffer's F6 Function
Hybrid Functions	Hybrid Function 1 (N = 3)
	Hybrid Function 2 (N = 3)
	Hybrid Function 3 (N = 4)
	Hybrid Function 4 (N = 4)
	Hybrid Function 5 (N = 5)
	Hybrid Function 6 (N = 5)
Composition Functions	Composition Function 1 (N = 5)
	Composition Function 2 (N = 3)
	Composition Function 3 (N = 3)
	Composition Function 4 (N = 5)
	Composition Function 5 (N = 5)
	Composition Function 6 (N = 5)
	Composition Function 7 (N = 3)
	Composition Function 8 (N = 3)

find the best classification algorithm but to see if populations carry sufficient information to make relevant predictions. A categorical cross-entropy loss is used as the loss function. The training of the classifier has been performed in a standard feed-forward way.

5. Experimental studies

The optimization algorithm for generating the training data was implemented in Java 8, ELA feature calculations were implemented in R, while all other experimental codes were implemented in Python 3.7 using the Tensorflow 2.2 library.

5.1. Experimental setup

For data acquisition, the test problems from the CEC 2014 Competition [42] (see Table 2) have been chosen as the test optimization problems. Three dimensionalities ($D = 10, 30, 50$) have been selected for all 30 test problems, resulting in 90 test problem classes used in the classification task. We need to point out here that different dimensionalities of the same problem are treated as different problems.

For the optimization algorithm, a basic Differential Evolution (DE) algorithm [43] has been selected, where its mutation strategy (Rand/1/Bin, Rand/1/Exp, Rand/2/Bin, Rand/2/Exp, Best/1/Bin, Best/1/Exp, Best/2/Bin, Best/2/Exp, Best/3/Bin, Rand/Rand/Bin, RandToBest/1/Bin, RandToBest/1/Exp), scaling factor $F = (0, 1)$ and crossover probability $Cr = (0, 1)$ hyper-parameters have been randomly set. For each problem and its dimensionality two random configurations (i.e., algorithm hyper-parameters settings) have been selected, where for each one, 25 runs were performed. Such a setup resulted in executing 4,500 optimization runs. In practice, the training data can be gathered from any of the previous optimization runs done by different optimization algorithms that have stored information about their trajectory runs. So no additional experiments would be needed. So our selection of DE is only a use case that will be used to test the feasibility of the proposed approach. Ideally, the data would be gathered from various optimization algorithms, so different heuristics would be used to describe the problem and its characteristics.

To speed up the time to run the experiments, the stopping criteria for all runs have been set to one of the following conditions: $D * 10,000$ problem evaluations, 100 iterations without improvement, or an optimum has been achieved (i.e., a solution within 10^{-8} of the actual optimum value). In the following sections, the size of the population ($population\_size$) was set equal to the dimension of the problem. The DE pseudocode is presented in Algorithm 1.

The calculated offspring for every iteration has been used from all algorithm runs, which meant that much more information about the problem space was gathered than when using candidate solutions from populations. Offspring within each iteration has been scaled and selected for training based on observed differences in the Frobenius norm (Section 3.2), where the required difference

Algorithm 1 Different Evolution Algorithm.

Input: $strategy$, F, Cr, $population\_size$
Output: best solution
P = Create and initialize the population of $population\_size$;
while stopping condition(s) not true **do**
  for each individual, $x_i \in P$ **do**
    Randomly select individuals;
    Create an offspring, x_i', by applying $strategy$ with scaling factor F on selected individuals;
    Update x_i' using binomial crossover on x_i with probability Cr;
    Evaluate the fitness, $f(x_i')$;
    if $f(x_i')$ is better than $f(x_i)$ **then**
      Replace x_i' with x_i in P;
    end if
  end for
end while
Return the individual with the best fitness as the solution;

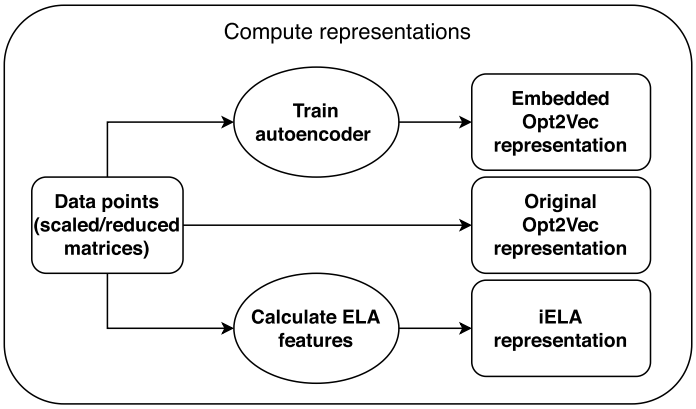


Fig. 6. The basic pipeline for computing the three compared representations (Embedded Opt2Vec, Original Opt2Vec, and iELA) from the gathered data using autoencoder or computing ELA features, respectively. In the presented diagram ellipses represent actions, while squares represent states.

diff has been set to 0.5. Finally, we end up with $1,398,130$ different data instances coming from all the optimization runs, which are classified into one of the 90 test problems.

The data gathered from running all $4,500$ optimization runs was stored in 484 GB of compressed files for all three dimensions $D = 10, 30, 50$. After performing all the pre-processing steps the size of the data was reduced to 11 GB for the classification task with $diff = 0.5$, while an additional 34 GB of compressed data was produced with respect to $diff = 0.1$ and 0.3 required for sensitivity analysis. The increase in size for the latter is in the fact that with smaller *diff* values more populations differ, so more of them need to be stored.

For training purposes, the datasets (each optimization run is a separate dataset of retained scaled matrices) have been randomly split into train, validation, and test sets with ratios of $82 : 12 : 6$, respectively. Since the split has been performed on the dataset level (optimization runs), a fully stratified split is not possible since each run contains a different number of data instances (i.e., retained scaled matrices). Not being able to stratify the split means that we cannot guarantee that each class (problem name and dimension) is represented with the same ratio of data instances in each of the three sets. For this purpose, we repeated each experiment ten times with different random splits to obtain a more robust evaluation.

5.2. Benchmarking Opt2Vec problem representations

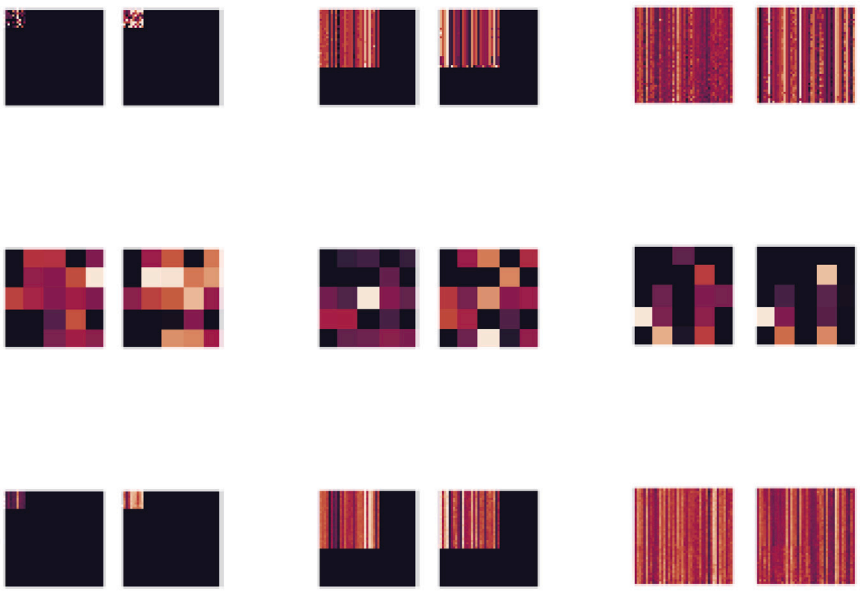
To assess the effectiveness of the Opt2Vec representations, we conducted a benchmark in the problem classification task. The comparison involved two types of Opt2Vec representations: the ones learned by autoencoders, called Embedded Opt2Vec, and the original Opt2Vec representations, referred to as Original Opt2Vec. Additionally, we compared the Opt2Vec representations with an iterative trajectory-based ELA representation, which we will refer to as iELA. The basic pipeline for computing the three compared representations (i.e., Embedded Opt2Vec, Original Opt2Vec, and iELA) is shown in Fig. 6. A more detailed description of the Embedded Opt2Vec and the iELA representations is given in the following subsections.

5.2.1. The embedded Opt2Vec representation

Computing the Embedded Opt2Vec representation is done using a trained autoencoder. For the autoencoder, the optimizer "adamax" [44] has been used with a batch size of $1,024$ and a maximum of 300 allowed epochs. The early stopping condition was set to 10% of maximal epochs without any improvement in the validation set loss function. For better visualization, embedding lengths of size l^2 were selected, where the embedding length is denoted as $l = 3, 4, \ldots, 10$. From here, it follows that we learned

Table 3
Average MSE for autoencoder on test set data.

l	$D = 10$	$D = 30$	$D = 50$	$D = 10, 30$	D =10-50
3	0.1843	0.2580	0.3130	0.2012	0.2119
4	0.1528	0.2440	0.3078	0.1908	0.2089
5	0.1349	0.2313	0.2982	0.1801	0.2045
6	0.1226	0.2172	0.2900	0.1709	0.2000
7	0.1110	0.2101	0.2846	0.1656	0.1971
8	0.0974	0.2091	0.2806	0.1643	0.1945
9	0.0822	0.2071	0.2777	0.1626	0.1925
10	0.0668	0.2045	0.2748	0.1608	0.1914



(a) $D = 10$ (b) $D = 30$ (c) $D = 50$

Fig. 7. Visualized examples of input (top), encoded (middle), and decoded (bottom) vectors for different dimensionalities for the Embedded Opt2Vec representation. The black color represents the zero value, while other colors represent values in the range (0, 1].

the Embedded Opt2vec representations vectors with sizes $9, 16, \ldots, 100$. The autoencoder was trained for all embedding lengths l on dimensionalities $D = 10$, $D = 30$, and $D = 50$, separately, and their combinations ($D = 10, 30$ and $D = 10, 30, 50$). The training mean squared errors (MSEs) averaged across all ten splits are presented in Table 3.

When the embedding length l increases, which means that we are learning representations with a larger size, the MSE is reduced for all dimensions (Table 3). This outcome is expected since the original data needs to be less compressed when l is large. From the results, it follows that the MSE for $D = 30$ and $D = 50$ is greater than $D = 10, 30$ and $D = 10, 30, 50$ because, for both combinations, the lower dimensional matrices need to be padded with zeros.

Fig. 7 presents several examples of the input, the embedding (i.e., the Embedded Opt2Vec representation), and the decoded vectors, which provide more insights into the autoencoder's performance on a single training instance. The examples correspond to $l = 5$, which means that we have the Embedded Opt2Vec representations of size 25. The input vectors used for learning them are coming from the combination of all dimensionalities $D = 10, 30, 50$. The black color represents the zero value, while other colors represent values in the range (0, 1]. It can be observed that the dimensions are perfectly preserved with encoding/decoding. Although actual data instance decoding is not perfect, it still maintains the major characteristics of the original input.

5.2.2. Iterative trajectory-based ELA representation

The ELA representation has been selected for comparison since it is a state-of-the-art representation for continuous optimization problems. In our experiments to perform fair benchmarking, we introduce iterative trajectory-based ELA calculation (iELA), where the ELA features are computed using the set of the candidate solution at every iteration (i.e., retained scaled matrix with check-marks in Fig. 3). It is important to highlight the key distinction between the trajectory-based ELA calculation and the iELA calculation. In the trajectory-based ELA approach, all candidate solutions visited throughout all iterations of the optimization run are grouped together to compute the ELA features. On the other hand, the iELA approach calculates the ELA features exclusively from the candidate solutions present in the current population at that specific state of the optimization run's trajectory. As a result, the trajectory-based

Table 4
ELA features used with different dimensionalities.

ELA feature	dimensionality				
	10	30	50	10, 30	10 – 50
basic.lower_max	✓	✓	✓	✓	✓
basic.upper_min	✓	✓	✓	✓	✓
basic.objective_max	✓	✓	✓		
basic.cells_filled		✓	✓		
basic.costs_runtime	✓	✓	✓		
disp.ratio_mean_02		✓	✓		
disp.ratio_mean_05		✓	✓		
disp.ratio_mean_10		✓	✓		
disp.ratio_mean_25		✓	✓		
disp.diff_mean_02	✓	✓	✓	✓	✓
disp.diff_mean_05	✓	✓	✓	✓	✓
disp.diff_mean_10	✓	✓	✓	✓	✓
disp.diff_mean_25	✓	✓	✓	✓	✓
disp.diff_median_02	✓	✓	✓	✓	✓
disp.diff_median_05	✓	✓	✓	✓	✓
disp.diff_median_10	✓	✓	✓	✓	✓
disp.diff_median_25	✓	✓	✓		
disp.costs_runtime	✓	✓	✓		
ela_level.costs_runtime	✓			✓	✓
ela_meta.costs_runtime	✓	✓	✓		
ela_meta.lin_simple.intercept	✓	✓	✓	✓	✓
nbc.nn_nb.sd_ratio		✓			
nbc.nn_nb.mean_ratio		✓			
nbc.nn_nb.cor		✓			
nbc.dist_ratio.coeff_var		✓			
nbc.costs_runtime		✓			
number of used features	16	25	20	11	11

ELA encompasses information about the entire trajectory, while the iELA only represents information pertaining to a particular state or position of the optimization run's trajectory. The iELA calculation is performed in order to test if the information from a single state from the trajectory is enough for ELA to capture which problem is being solved. At the same time, it allows us a fair comparison with the proposed novel Opt2Vec representations that are designed for such learning scenarios.

For this purpose, we have focused on all groups of landscape features that can be found in the R package flacco [16] (except the cell-mapping-based ELA group). After calculating all the ELA features for all data instances (i.e., retained scaled matrices), only those features that returned non "NA" values for all data instances have been used (see results in Table 4). Having "NA" values for some ELA features is a common issue with ELA features calculation since many feature calculations are sensitive to samplings and their characteristics [18]. The last line in Table 4 represents the number of features used (marked with ✓) for the specified dimensionality that defines the iELA representations. We would like to note that the potential influence of scaling on the calculation of ELA features was not investigated in this work.

5.3. Problem classification

For the classification neural network, the optimizer "adamax" was used with a batch size of $1,024$ and a maximum of 300 allowed epochs. The early stopping condition was set to 10% of the maximal epochs without any improvement in the validation set loss function. Weights were initialized using the He uniform variance scaling initializer [45]. All the experiments for training the neural network classifier using scaled inputs have been performed on the same ten splits used for training the autoencoder. Three sets of experiments have been performed to see if the generated samples can be classified into the appropriate classes (problem name and dimension).

For the first experiment, several sizes of the Embedded Opt2Vec representations were used to evaluate the sensitivity of the classifier with regard to embedding length l. Experiments were performed for $D = 10$, $D = 30$, $D = 50$, and their combinations ($D = 10, 30$ and $D = 10, 30, 50$) with input vectors lengths for the classifiers of 110, 930, 2550, 930, and 2550, respectively.

In the second set of experiments, a classifier was trained using the Original Opt2Vec representations as an input vector for the classifier.

In the last set of experiments, instead of the Original Opt2Vec representations as input vectors, iELA representations were used as input data for training the classifier. A pipeline of how the representations were evaluated is shown in Fig. 8.

5.3.1. Classification results

The classification results obtained from ten random splits (Fig. 9) show that using the flacco library (i.e., iELA representations calculated from the data instances), the parts of a problem cannot be correctly described. In every dimensionality or combination thereof, the classification accuracy is below the accuracy obtained by both Opt2Vec representations. This lower accuracy most prob-

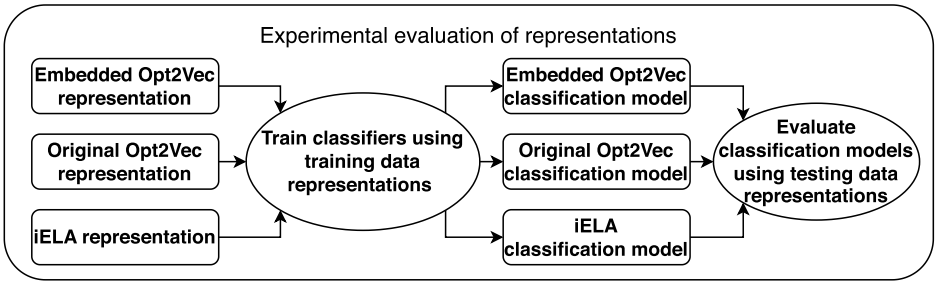


Fig. 8. A pipeline for evaluation of the representations, where classifiers are first trained using respective representations. Trained classification models are then evaluated using test data. Training and test data consist of the same data instances for each representation, so the comparison is fair. In the presented diagram ellipses represent actions, while squares represent states.

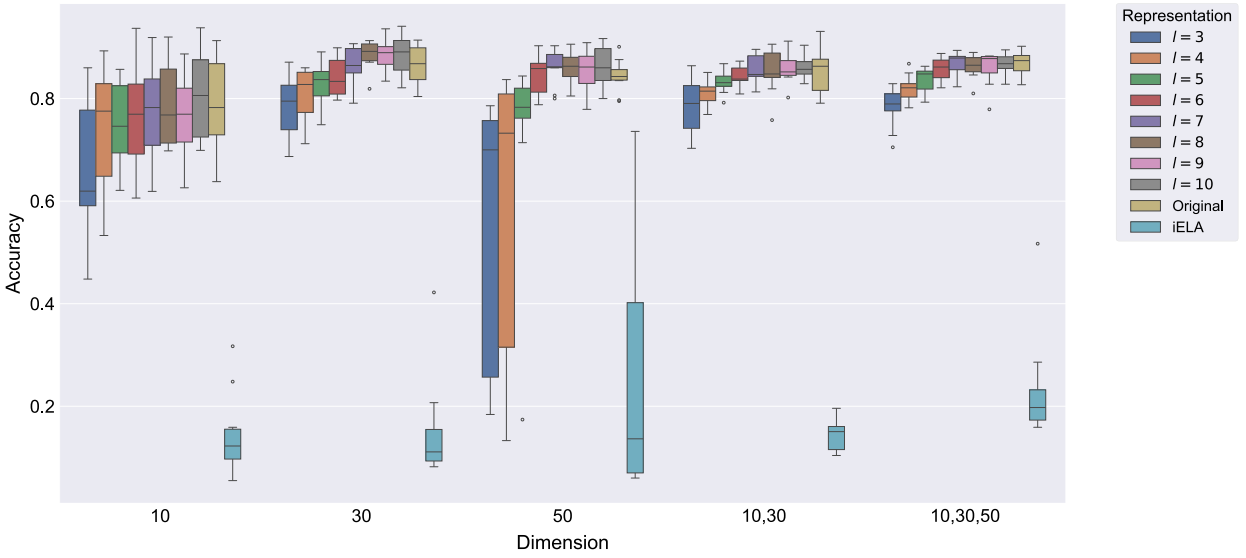


Fig. 9. Embedding length sensitivity analysis of test data using box plots. The results for the Embedded Opt2Vec with embedding lengths $l = 3, \ldots, 10$, Original Opt2Vec, and iELA representations are shown. The Y-axis represents the accuracy that each of the models achieved concerning the dimensions presented on the x-axis.

ably occurs because the data instance samples are too small or correlated, which results in "NA" values for many of the calculated features (as is often the case and an issue with calculating ELA features). Please refer to Table 4 to see which ELA features were without such values for each dimensionality. This indicates that ELA features are more useful when the sampling is done from the whole search space discarding the algorithm's behavior, however, they are not suitable to represent parts from the search space visited by the algorithm. For dimension, $D = 10$ can be seen that the Embedded Opt2Vec representations with size 25 or higher (i.e., corresponding to embedding length $l \geq 5$) provide similar classification accuracy and robustness (height of the box plots) with the Original Opt2Vec representations. For dimensions $D = 30$ and $D = 50$ the average accuracy of the Embedded Opt2Vec representations (for $l \geq 8$) can even slightly outperform the Original Opt2Vec representation.

Further, we wanted to see if there is any correlation between autoencoder losses and achieved classification accuracies. The most interesting results can be observed when the performance of the classification accuracy is poor. For this reason, we selected the Embedded Opt2Vec with $l = 3$ and $D = 50$. Table 5 presents the autoencoder losses and achieved classification accuracies for all ten splits in such a scenario. The bold lines indicate split instances, where the classification performance is lower than 0.5. Looking at the bold lines, one can observe that they have the worst MSE values out of all split instances. This indicates that the quality of the Embedded Opt2Vec representation influences the classification performance. This effect is expected since encoding vectors of length 2,550 into vectors of size 9 (l^2) without losing information represent a significant challenge.

Finally, we evaluated the classification accuracy achieved by all of the compared approaches. Tables 6–8 show the average classification accuracy for the training, the validation, and the test set, respectively. The results for the training sets and validation sets are presented so one can observe any noticeable overfitting in the training. The reduction in accuracy from training to validation set for Opt2Vec approaches is approximately $10 - 15\%$, while in the case of iELA representation, the accuracy drops approximately $25 - 50\%$. This is an indication that the iELA representation is much more affected by overfitting compared to the Opt2Vec. From the perspective of classification accuracy, we are the most interested in the test sets, since they are the most representative of the expected performance. The bold values in Table 8 represent the best accuracy for each dimension achieved on the test set. In contrast, values annotated with stars represent better accuracies than the results obtained using the Original Opt2Vec representations. From

Table 5
Autoencoder losses vs classification accuracy for $l = 3$ and $D = 50$.

split instance	autoencoding MSE			classification accuracy		
	train	validation	test	train	validation	test
1	**0.320**	**0.311**	**0.339**	**0.184**	**0.143**	**0.240**
2	0.281	0.295	0.285	0.844	0.792	0.748
3	0.273	0.303	0.300	0.857	0.779	0.763
4	**0.322**	**0.307**	**0.319**	**0.187**	**0.144**	**0.184**
5	0.276	0.283	0.298	0.880	0.800	0.679
6	0.276	0.283	0.309	0.881	0.860	0.786
7	**0.316**	**0.331**	**0.353**	**0.174**	**0.172**	**0.307**
8	0.279	0.294	0.302	0.851	0.846	0.721
9	0.275	0.305	0.294	0.864	0.825	0.760
10	**0.321**	**0.306**	**0.331**	**0.177**	**0.189**	**0.226**

Table 6
Average classification accuracy results for the training sets.

representation	$D = 10$	$D = 30$	$D = 50$	$D = 10, 30$	D =10-50
Opt2Vec ($l = 3$)	0.8744	0.8980	0.5899	0.9026	0.8963
Opt2Vec ($l = 4$)	0.9086	0.9413	0.6853	0.9304	0.9350
Opt2Vec ($l = 5$)	0.9213	0.9551	0.8728	0.9423	0.9513
Opt2Vec ($l = 6$)	0.9328	0.9651	0.9685	0.9486	0.9578
Opt2Vec ($l = 7$)	0.9413	0.9690	0.9751	0.9573	0.9623
Opt2Vec ($l = 8$)	0.9478	0.9713	0.9772	0.9602	0.9663
Opt2Vec ($l = 9$)	0.9569	0.9717	0.9794	0.9663	0.9694
Opt2Vec ($l = 10$)	0.9616	0.9754	0.9817	0.9668	0.9713
Original Opt2Vec	0.9692	0.9984	1.0000	0.9919	0.9969
iELA	0.4542	0.6782	0.8065	0.5357	0.5359

Table 7
Average classification accuracy results for the validation sets.

representation	$D = 10$	$D = 30$	$D = 50$	$D = 10, 30$	D =10-50
Opt2Vec ($l = 3$)	0.7998	0.7872	0.5550	0.7769	0.7711
Opt2Vec ($l = 4$)	0.8292	0.8211	0.5951	0.8112	0.8132
Opt2Vec ($l = 5$)	0.8140	0.8167	0.7297	0.8287	0.8264
Opt2Vec ($l = 6$)	0.8333	0.8473	0.8423	0.8481	0.8375
Opt2Vec ($l = 7$)	0.8389	0.8725	0.8630	0.8477	0.8470
Opt2Vec ($l = 8$)	0.8446	0.8739	0.8660	0.8521	0.8462
Opt2Vec ($l = 9$)	0.8643	0.8668	0.8511	0.8691	0.8505
Opt2Vec ($l = 10$)	0.8759	0.8686	0.8669	0.8521	0.8454
Original Opt2Vec	0.8732	0.8770	0.8430	0.8680	0.8672
iELA	0.1336	0.1651	0.5554	0.1671	0.1941

Table 8
Average classification accuracy results for the test sets.

representation	$D = 10$	$D = 30$	$D = 50$	$D = 10, 30$	D =10-50
Opt2Vec ($l = 3$)	0.6630	0.7846	0.5414	0.7833	0.7837
Opt2Vec ($l = 4$)	0.7385	0.8082	0.5935	0.8117	0.8205
Opt2Vec ($l = 5$)	0.7494	0.8297	0.7282	0.8335	0.8371
Opt2Vec ($l = 6$)	0.7634	0.8423	0.8461*	0.8420	0.8578
Opt2Vec ($l = 7$)	0.7692	0.8646	0.8612*	0.8560*	**0.8703**
Opt2Vec ($l = 8$)	0.7872*	**0.8855**	0.8588*	0.8528	0.8625
Opt2Vec ($l = 9$)	0.7695	0.8854*	0.8527*	0.8590*	0.8604
Opt2Vec ($l = 10$)	**0.8051**	0.8828*	**0.8626**	**0.8627**	0.8676
Original Opt2Vec	0.7831	0.8663	0.8439	0.8543	0.8699
iELA	0.1452	0.1502	0.2505	0.1419	0.2323

the results, we can conclude that the Embedded Opt2Vec representations with sizes greater or equal to 36 (i.e., $l \geq 6$) provide similar or better performance than the Original Opt2Vec representations on the test sets. The iELA representations perform much poorer compared to the Opt2Vec representations, indicating that iELA representation is not appropriate for the evaluated learning scenario.

Table 9
Classification macro F1 score results for the test sets.

representation	$D = 10$	$D = 30$	$D = 50$	$D = 10, 30$	D =10-50
Opt2Vec ($l = 3$)	0.4713	0.6302	0.3872	0.5722	0.5549
Opt2Vec ($l = 4$)	0.5441	0.6823	0.4519	0.6276	0.6145
Opt2Vec ($l = 5$)	0.5586	0.7119	0.5608	0.6381	0.6269
Opt2Vec ($l = 6$)	0.5728	0.7454	0.6125	0.6582	0.6586
Opt2Vec ($l = 7$)	0.5723	0.7661	0.6396	0.6666	0.6680
Opt2Vec ($l = 8$)	0.5885	0.7711	0.6428	0.6676	0.6666
Opt2Vec ($l = 9$)	0.5862	0.7787	0.6561	0.6866	0.6651
Opt2Vec ($l = 10$)	0.6047	0.7748	0.6549	0.6805	0.6755
Original Opt2Vec	0.6132	0.7786	0.6321	0.7098	0.7002
iELA	0.0421	0.0710	0.1580	0.0548	0.0794

To take into account also the type of errors (false positive and false negative) we have also included the analysis of test sets results using macro F1 score. From the Table 9 we can see a similar pattern as observed with the analysis based on average accuracy where the performance increases with higher l values and the performance of all Opt2Vec representations drastically outperform iELA representation.

To obtain more insight into the performance differences between the selected Embedded Opt2Vec, the Original Opt2Vec, and the iELA representations, a confusion matrix for the same test set with $l = 10$ and $D = 50$ is presented (Fig. 10). From the figures, it is clear that the iELA representations produce a poor confusion matrix with basically none of the problems being correctly identified, while both Opt2Vec representations produce similar confusion matrices. This is another clear indication that the Embedded Opt2Vec representation with $l = 10$ provided very similar (in many cases even better, see Table 8) performance with much smaller representations (i.e., smaller vector size).

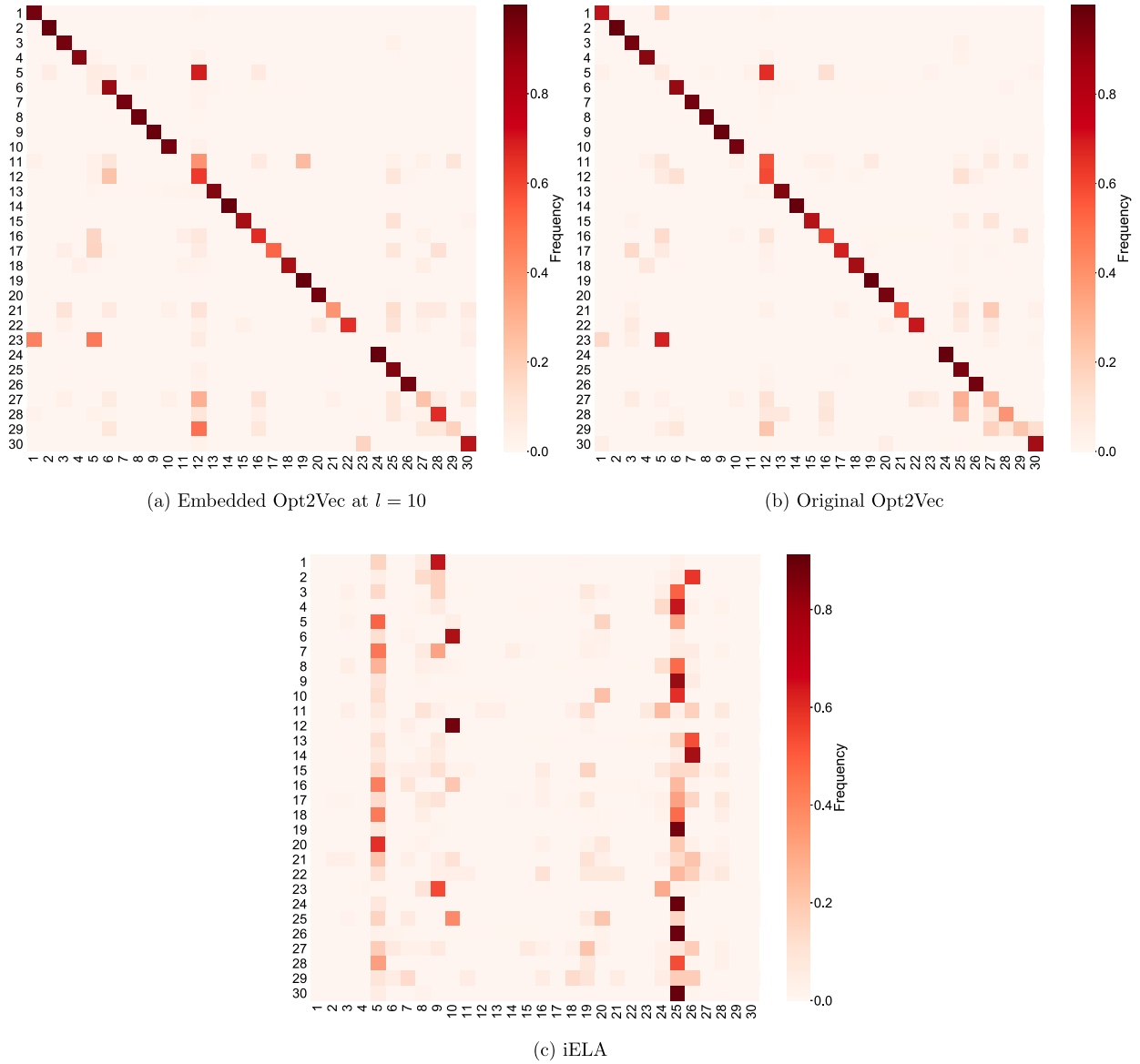
5.3.2. Data acquisition sensitivity analysis

An additional experiment was conducted to understand the impact of the Frobenius norm threshold on the data acquisition process. We investigated this by analyzing the influence of selecting different thresholds between Frobenius norms in the acquisition process on the process of learning the Opt2Vec representations and their classification. The tested thresholds between the Frobenius norms (i.e., *diff*) were: 0.1, 0.3, and 0.5. The reason for selecting lower *diff* values is to see if the reduced required difference (and consequent higher number of training data) has any significant influence on the performance of the proposed Opt2Vec representations. The influence of selected thresholds was evaluated using offspring data acquired from problems in $D = 30$. The selected thresholds had a huge impact on the number of data instances that were collected. Namely, for $diff = 0.1$ the number of data instances was $4,882,067$, for $diff = 0.3$ the number of data instances was $1,343,085$, while for $diff = 0.5$ the number of data instances was $514,889$. The increased number of data instances had a large influence on the time complexity of all learning representation tasks, especially on calculating the ELA features. Fig. 11 shows how the average classification accuracy over all ten splits changes with respect to the applied differences. There is also very little difference in the achieved accuracy for all representations except the iELA representations. Although there is a noticeable improvement, it is still far too little to get even close to the other performances.

To evaluate the approach's efficiency regarding sampling coarseness when using different thresholds between Frobenius norms, the models learned on the 10 splits when $diff = 0.5$ were tested on all data instances acquired with $diff = 0.1$. The test was conducted with problems of dimension $D = 30$. So the autoencoder learned on each split using the data collected with $diff = 0.5$ has been utilized to encode the data collected with $diff = 0.1$. Next, for the encoded representations with $diff = 0.1$, the classification model learned using the same split and data collected with $diff = 0.5$ has been utilized to make the predictions. With this, we have tested the transferability of the proposed approach to different samplings concerning the selected threshold *diff*. We need to note here that all of the training instances from the split using a 0.5 threshold can also be present in the dataset collected using a 0.1 threshold, which can further lead to overfitting. However, in our case, such data represents at most 8% of the dataset collected using a 0.1 threshold, so we have neglected it for this experiment. Table 10 presents the classification results obtained using the ten models (trained using the data collected with $diff = 0.5$) evaluated on all data instances acquired with $diff = 0.1$. The classification models were trained using both Opt2Vec representations as input data. The iELA representation was skipped since the feature portfolio calculated for all samples with $diff = 0.1$ was different from the one calculated for all samples with $diff = 0.5$, making such comparison infeasible and provides another reason why iELA representation is less suitable in a such learning scenario.

Based on the average classification accuracies across all ten models (i.e., evaluated on all data instances acquired with $diff = 0.1$), the Original Opt2Vec representation returned the best accuracy. It even achieved a 5% higher accuracy than the averaged accuracy obtained from the acquired test sets where $diff = 0.5$.

When the Embedded Opt2Vec representations with $l = 3$ are used, the average accuracy across all ten models decreases compared to other embedding lengths. This decrease suggests that the embedding space with $l = 3$ is too small, and it is not possible to capture the necessary features from the original space. The average classification accuracy across all ten models obtained when the Embedded Opt2Vec representations with $l \geq 4$ are similar (i.e., almost the same) to the average accuracy obtained from the acquired test sets ($diff = 0.5$). Therefore, the test set results indicate that classification performance is not very sensitive to using different thresholds *diff* if the Opt2Vec representation preserves enough information (i.e., in our experiment with $l \geq 4$).



(a) Embedded Opt2Vec at $l = 10$

(b) Original Opt2Vec

(c) iELA

Fig. 10. Confusion matrix illustrating the performance differences between the selected Embedded Opt2Vec representations with $l = 10$, the Original Opt2Vec representations, and the iELA representations, using the same test set for all 30 test problems and $D = 50$.

Table 10
Sensitivity of Opt2Vec models with Frobenius norm differences of 0.5 on 0.1 data.

split instance	Original	$l = 3$	$l = 4$	$l = 5$	$l = 6$	$l = 7$	$l = 8$	$l = 9$	$l = 10$
1	0.935	0.729	0.795	0.794	0.865	0.877	0.828	0.839	0.893
2	0.927	0.655	0.831	0.781	0.787	0.829	0.885	0.889	0.879
3	0.943	0.777	0.817	0.768	0.824	0.893	0.880	0.839	0.866
4	0.913	0.724	0.792	0.849	0.862	0.881	0.868	0.867	0.860
5	0.935	0.753	0.788	0.835	0.800	0.896	0.901	0.880	0.890
6	0.926	0.709	0.734	0.876	0.843	0.867	0.885	0.908	0.899
7	0.910	0.658	0.817	0.870	0.863	0.842	0.861	0.870	0.868
8	0.928	0.728	0.830	0.840	0.858	0.860	0.883	0.868	0.845
9	0.867	0.750	0.809	0.832	0.869	0.839	0.877	0.831	0.904
10	0.901	0.731	0.836	0.815	0.849	0.886	0.897	0.891	0.902
mean	0.918	0.721	0.805	0.826	0.842	0.867	0.877	0.868	0.881
test set (0.5)	0.866	0.785	0.808	0.830	0.842	0.865	0.886	0.885	0.883

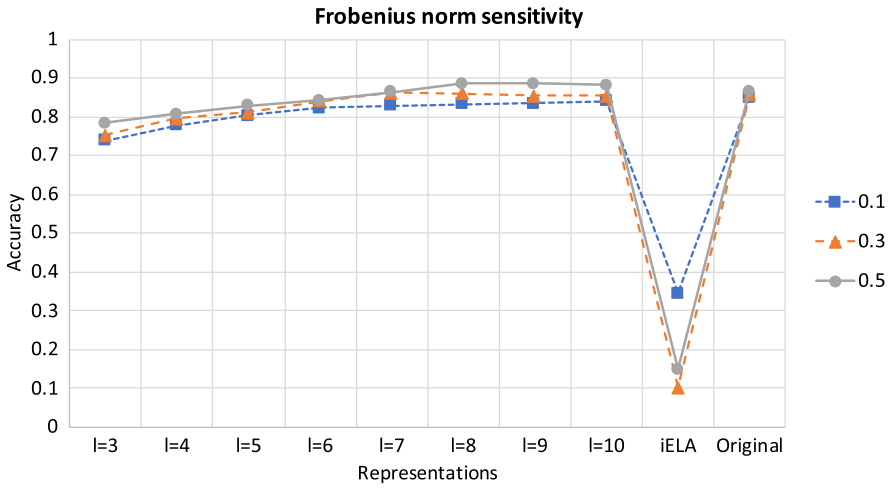


Fig. 11. Frobenius norm sensitivity analysis on $D = 30$ for the following *diff* = 0.1, 0.3 and 0.5 values. The average classification accuracy (y-axis) is shown with respect to the Embedded Opt2Vec of lengths $l = 3, \ldots, 10$, the Original Opt2Vec, and the iELA representations (x-axis).

6. Discussion

The Opt2Vec representations capture sufficient information from individual populations (i.e., a particular state of the trajectory of the optimization run) and characterize a part of the problem landscape instance that can further be used for anytime identification of the actual problem that is being solved. It needs to be noted that the characteristics obtained using the Opt2Vec representations are based on scaled data, where scaling is done within a single population. Such representation could provide valuable additional information to an optimization algorithm during the optimization in respect of what type of problem instance the algorithm is currently solving. This information could also lead to a better selection of the most appropriate algorithm configuration for efficiently solving the problem (based on previous experience).

In our experiments, the Opt2Vec representations were classified into 90 classes, consisting of 30 different optimization problems and three dimensionalities 10, 30, and 50, which resulted in an accuracy of 80% to 87%. The results were obtained using offspring data that was min-max scaled. Hence, the approach is invariant to simple transformations (shifting or scaling) of the problem. This result is significant since it allows us to efficiently determine the problem while using only information from one state of the trajectory of the algorithm run. The only required pre-processed step of the proposed methodology is the scaling of the data (i.e., the Original Opt2Vec representation) and the calculation of the Embedded Opt2Vec representation, which are both trivial tasks. This represents a major time-saving in comparison to the calculation of the most commonly-used ELA features, which has been shown by the experiments that they are not suitable for representing different parts from the search space that are visited by the algorithm across different iterations and are not relevant for such kind of learning task. The optimization problem classification performed by the learned model is also a computationally simple task.

Training autoencoders can be a time-consuming task (i.e., can take from minutes to days, depending on the size of the training dataset), except for the Original Opt2Vec representation, where no training is required. However, it needs to be made only once, and the learned model can be used to generate the Opt2Vec representations for new datasets. Similarly, training a classification model can be a time-consuming task and largely depends on the amount of data that is used for training. But the advantage is that it needs to be done only once and can be later adapted to new data, which can be a less time-consuming task with higher learning start, slope, and asymptote [46]. Finally, the learned Opt2Vec representations can be used as part of a classification process to identify the optimization problem that is being solved regardless of the dimensionality (by simply padding the lower dimension data with zeros). An additional advantage of using the Embedded Opt2Vec representations is the reduced input size when combining them as input for machine learning approaches, which can be much more efficient in time and accuracy.

Furthermore, we need to point out that the samples found in the training, validation, and test sets did not belong to the same run. The learned models were checked to see if they were transferable to new optimization runs. The results indicate that this approach has strong transfer learning capabilities that achieve a test set accuracy above 80%. It can also distinguish between different continuous optimization problems in the Original and Embedded Opt2Vec representation space.

7. Conclusion

Automatic characterization of an optimization problem holds significant importance in the field of optimization research. Given that evaluating real-world problems can be time-consuming, it becomes crucial to minimize the number of evaluations required to accurately characterize the problem. In this study, we showed that continuous optimization problems can be characterized with information from individual populations visited by the algorithm during its run (i.e., different states of the trajectory of the optimization run) that allow us for highly accurate anytime identification of the problem being solved. This representation of the problem

landscape characteristics, called Opt2Vec, does not require any additional evaluations, except the ones made by the optimization algorithm during its run. The Opt2Vec representations are based on an autoencoder type of neural network, which can encode the gathered information into a condensed subspace without any significant loss of information.

The experimental results demonstrated that the Opt2Vec representations achieved classification accuracies of over 80% for the problem being solved and its dimension. In comparison to a modified version of the current state-of-the-art trajectory-based ELA technique for fitness landscape analysis (i.e., iterative trajectory-based ELA, iELA), the results indicated that the Opt2Vec representation outperformed it. This outcome suggests that the iELA representation is not well-suited for capturing the characteristics of a single state in the trajectory of the optimization process.

For future work, we are planning to investigate different research directions. Firstly, we would like to test the approach when data from different optimization algorithms is used to learn the Opt2Vec representations to see if this approach extends also over different families of optimization algorithms and not just the DE tested in this study. This will allow us to investigate the influence of different algorithm families on Opt2Vec performance in different machine-learning tasks. Another direction is related to finding more suitable autoencoder architecture to achieve better machine learning task performance (e.g., Graph Autoencoder, where geometrical landscape properties found in population could probably be better preserved). Further, since the Opt2Vec representations are coming with the cost that they are black-box optimization, we plan to extend the research concerning their interpretability. We aim to examine the suitability of the proposed Opt2Vec representation for both automated algorithm selection and configuration learning tasks. Additionally, we intend to extend these tasks by not limiting the learning of the Opt2Vec representations to a single population but by incorporating several populations from sequential iterations. Last but not least, we are planning to use different deep learning architectures such as transformers, and long short-term memory networks to find features that can describe various particularities of the algorithm behavior.

CRediT authorship contribution statement

Peter Korošec: Writing – review & editing, Writing – original draft, Visualization, Validation, Software, Resources, Project administration, Methodology, Investigation, Funding acquisition, Formal analysis, Data curation, Conceptualization. **Tome Eftimov:** Writing – review & editing, Software, Project administration, Funding acquisition.

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.

Data availability

Data will be made available on request.

Acknowledgement

The authors thank the Slovenian Research and Innovation Agency for their financial support (research core funding No. P2-0098, and projects No. J2-4460 and N2-0239).

Appendix A. Problem classification using conventional ELA features

To demonstrate the advantages of the proposed representations, we conducted an additional experiment. We evaluated the performance of conventional ELA features, which are computed from artificial candidate solutions generated by some sampling technique, rather than being derived from observations during the algorithm's execution.

For this purpose, we have generated 10,000 samples for each of the 30 CEC 2014 problems in a $D = 10, 30, 50$ dimensional space, resulting in a total of 300,000 data instances described by ELA features for each problem dimension separately, with 10,000 instances from each problem class. To calculate the ELA, we have used the Sobol sampling technique [47] by using a $50 \times D$ sample size (minimum recommended sample size for robust ELA feature calculation [48]). The used ELA features have been selected based on their non-missing values during the iterative ELA calculation process (on the test data). Subsequently, we created a balanced dataset for multi-class classification with 80:20 split for training and validation sets, where each instance has been labeled with its corresponding problem class. Using this dataset, we trained the same multi-class classifier (for each problem dimension, separately) that has been used for Opt2Vec and iELA representations as defined in Section 5.3 to predict the problem class based on the calculated conventional ELA features. We have evaluated the models using the same test datasets employed for models trained with Opt2Vec and iELA features. This experiment has also explored the potential of transfer learning by assessing if artificial samples generated for a specific problem could help identify the problem being solved by using a single iteration of real observed candidate solutions produced by the algorithm instance. As presented in Table A.11 the mean classification accuracy across all 10 test splits is below 5% in all three selected dimensions, indicating that the conventional ELA approach is inadequate for this task, underscoring the necessity for novel representations like Opt2Vec. This experiment also points out that landscape features that are proposed only to describe the fitness landscape in a problem space are not suitable for working with trajectory data.

Table A.11
Average classification accuracy results for the conventional ELA features.

dataset	$D = 10$	$D = 30$	$D = 50$
train	0.793	0.852	0.808
test	0.017	0.048	0.007

References

[1] 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.
[2] Q. Renau, J. Dréo, C. Doerr, B. Doerr, Towards explainable exploratory landscape analysis: extreme feature selection for classifying bbob functions, in: EvoApplications, 2021, pp. 17–33.
[3] R. Trajanov, S. Dimeski, M. Popovski, P. Korošec, T. Eftimov, Explainable landscape-aware optimization performance prediction, in: 2021 IEEE Symposium Series on Computational Intelligence (SSCI), IEEE, 2021, pp. 01–08.
[4] 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.
[5] 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, 2022, pp. 648–656.
[6] 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), IEEE, 2022, pp. 1–8.
[7] B. Bischl, P. Kerschke, L. Kotthoff, M. Lindauer, Y. Malitsky, A. Fréchette, H. Hoos, F. Hutter, K. Leyton-Brown, K. Tierney, et al., Aslib: a benchmark library for algorithm selection, Artif. Intell. 237 (2016) 41–58.
[8] P. Kerschke, H. Trautmann, Automated algorithm selection on continuous black-box problems by combining exploratory landscape analysis and machine learning, Evol. Comput. 27 (1) (2019) 99–127.
[9] D. Vermetten, H. Wang, T. Bäck, C. Doerr, Towards dynamic algorithm selection for numerical black-box optimization: investigating bbob as a use case, in: Proceedings of the 2020 Genetic and Evolutionary Computation Conference, 2020, pp. 654–662.
[10] N. Belkhir, J. Dréo, P. Savéant, M. Schoenauer, Per instance algorithm configuration of cma-es with limited budget, in: Proceedings of the Genetic and Evolutionary Computation Conference, 2017, pp. 681–688.
[11] 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, 2022, pp. 620–629.
[12] 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.
[13] M. Wang, B. Li, G. Zhang, X. Yao, Population evolvability: dynamic fitness landscape analysis for population-based metaheuristic algorithms, IEEE Trans. Evol. Comput. 22 (4) (2018) 550–563, https://doi.org/10.1109/TEVC.2017.2744324.
[14] O. Mersmann, B. Bischl, H. Trautmann, M. Preuss, C. Weihs, G. Rudolph, Exploratory landscape analysis, in: Proceedings of the 13th Annual Conference on Genetic and Evolutionary Computation, 2011, pp. 829–836.
[15] 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) (2015) 74–87, https://doi.org/10.1109/TEVC.2014.2302006.
[16] P. Kerschke, H. Trautmann, The r-package flacco for exploratory landscape analysis with applications to multi-objective optimization problems, in: 2016 IEEE Congress on Evolutionary Computation (CEC), IEEE, 2016, pp. 5262–5269.
[17] 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.
[18] Q. Renau, C. Doerr, J. Dreo, B. Doerr, Exploratory landscape analysis is strongly sensitive to the sampling strategy, in: International Conference on Parallel Problem Solving from Nature, Springer, 2020, pp. 139–153.
[19] R.D. Lang, A.P. Engelbrecht, An exploratory landscape analysis-based benchmark suite, Algorithms 14 (3) (2021) 78.
[20] 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.
[21] G. Petelin, G. Cenikj, T. Eftimov, Tla: topological landscape analysis for single-objective continuous optimization problem instances, in: 2022 IEEE Symposium Series on Computational Intelligence (SSCI), IEEE, 2022, pp. 1698–1705.
[22] 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.
[23] R. Tanabe, Towards exploratory landscape analysis for large-scale optimization: a dimensionality reduction framework, arXiv preprint arXiv:2104.10301, 2021.
[24] 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.
[25] B. van Stein, F.X. Long, M. Frenzel, P. Krause, M. Gitterle, T. Bäck, Doe2vec: deep-learning based features for exploratory landscape analysis, arXiv preprint arXiv:2304.01219, 2023.
[26] 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.
[27] F.X. Long, D. Vermetten, B. van Stein, A.V. Kononova, Bbob instance analysis: landscape properties and algorithm performance across problem instances, in: International Conference on the Applications of Evolutionary Computation (Part of EvoStar), Springer, 2023, pp. 380–395.
[28] A. Janković, C. Doerr, Adaptive landscape analysis, in: Proceedings of the Genetic and Evolutionary Computation Conference Companion, 2019, pp. 2032–2035.
[29] A. Jankovic, T. Eftimov, C. Doerr, Towards feature-based performance regression using trajectory data, in: International Conference on the Applications of Evolutionary Computation (Part of EvoStar), Springer, 2021, pp. 601–617.
[30] 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), 2022, pp. 1–8.
[31] 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: G. Rudolph, A.V. Kononova, H. Aguirre, P. Kerschke, G. Ochoa, T. Tušar (Eds.), Parallel Problem Solving from Nature – PPSN XVII, Springer International Publishing, Cham, 2022, pp. 46–60.
[32] 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.
[33] G. Ochoa, S. Verel, F. Daolio, M. Tomassini, Local optima networks: a new model of combinatorial fitness landscapes, Recent Adv. Theory Appl. Fit. Landsc. (2014) 233–262.

[34] 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.
[35] 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.
[36] 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.
[37] T. Birsan, D. Tiba, One hundred years since the introduction of the set distance by dimitrie Pompeiu, in: System Modeling and Optimization: Proceedings of the 22nd IFIP TC7 Conference Held from July 18–22, 2005, in Turin, Italy 22, Springer, 2006, pp. 35–39.
[38] D. Charte, F. Charte, S. García, M.J. del Jesus, F. Herrera, A practical tutorial on autoencoders for nonlinear feature fusion: taxonomy, models, software and guidelines, Inf. Fusion 44 (2018) 78–96.
[39] D. Charte, F. Charte, M.J. del Jesus, F. Herrera, An analysis on the use of autoencoders for representation learning: fundamentals, learning task case studies, explainability and challenges, Neurocomputing (2020).
[40] W.-J. Jia, Y.-D. Zhang, Survey on theories and methods of autoencoder, Comput. Syst. Appl. 5 (2018).
[41] A. Voulodimos, N. Doulamis, A. Doulamis, E. Protopapadakis, Deep learning for computer vision: a brief review, Comput. Intell. Neurosci. (2018) 2018.
[42] J.J. Liang, B.Y. Qu, P.N. Suganthan, Problem definitions and evaluation criteria for the cec 2014 special session and competition on single objective real-parameter numerical optimization, Computational Intelligence Laboratory, Zhengzhou University, Zhengzhou China and Technical Report, Nanyang Technological University, Singapore 635, 2013.
[43] K. Price, R.M. Storn, J.A. Lampinen, Differential Evolution: A Practical Approach to Global Optimization, Springer Science & Business Media, 2006.
[44] D.P. Kingma, J. Ba, Adam: a method for stochastic optimization, arXiv preprint arXiv:1412.6980, 2014.
[45] K. He, X. Zhang, S. Ren, J. Sun, Delving deep into rectifiers: surpassing human-level performance on imagenet classification, in: Proceedings of the IEEE International Conference on Computer Vision, 2015, pp. 1026–1034.
[46] E. Olivas, J. Guerrero, M. Martinez-Sober, J. Magdalena-Benedito, A. Serrano L¢pez, Handbook of Research on Machine Learning Applications and Trends: Algorithms, Methods, and Techniques: Algorithms, Methods, and Techniques, Information Science Reference, 2009, https://books.google.si/books?id=gFpKXO8H_6YC.
[47] I. Sobol, The distribution of points in a cube and the accurate evaluation of integrals, (in Russian), Vychisl. Mat. Mater. Phys. 7 (1967) 784–802.
[48] 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, 2016, pp. 229–236.

Further reading

[49] A. Dumka, D. V, J. Loganathan, Data dissemination for green-vanets communication: an opportunistic optimization approach, Int. J. Pervasive Comput. Commun. 17 (1) (2021) 89–108.