Artificial Neural Networks as Feature Extractors in Continuous Evolutionary Optimization

Stephen Friess[§], Peter Tiňo[§], Zhao Xu[‡], Stefan Menzel[†], Bernhard Sendhoff[†] and Xin Yao[§*]
[§] CERCIA, School of Computer Science, University of Birmingham, UK
[‡] NEC Laboratories Europe GmbH, 69115 Heidelberg, Germany
[†] Honda Research Institute Europe GmbH, 63073 Offenbach a.M., Germany
[*] Southern University of Science and Technology, Shenzhen, China
{shf814, p.tino, x.yao}@cs.bham.ac.uk, zhao.xu@neclab.eu, {stefan.menzel, bernhard.sendhoff}@honda-ri.de

Abstract—**Recent years have seen the advancement of data-driven paradigms in population-based and evolutionary optimization. This reflects on one hand the mere abundance of available data, but on the other hand also progresses in the refinement of previously available machine learning methods. Surprisingly, deep pattern recognition methods emerging from the studies of neural networks have only been sparingly applied. This comes unexpected, as the complex data generated by evolutionary search algorithms can be considered tedious and intractable for manual analysis with mere practical intuitions. In this work, we therefore explore opportunities to employ deep networks to directly learn problem characteristics of continuous optimization problems. Particularly, with data obtained during initial runs of an optimization algorithm. We find that a graph neural network, trained upon a graph representation of continuous search spaces, shows in comparison to more traditional approaches higher validation accuracy and retrieves characteristics within the latent space which are better at distinguishing different continuous optimization problems. We hope that our study can pave the way towards new approaches which allow us to learn problem-dependent algorithm components and recall these from predictions of inputs generated during the run-time of an optimization algorithm.**

Index Terms—**Feature learning, representation learning, algorithm selection, graph neural networks, knowledge transfer.**

I. Introduction

Within the study of natural computing methods, research on neural and evolutionary computation have historically been strongly intertwined (e.g. [1]–[3]). This comes not unexpected, as the processing of data generated by evolutionary approaches can benefit from incorporating flexible predictive methods. Further, the study of combined neural and evolutionary computation methods also allows one to define analogue models to study effects and phenomena in biology and complex systems research [4], [5]. Thus, fertilizing synergies to test theoretical models in the natural sciences through means of computation. However, while the application of evolutionary optimization to modern deep pattern recognition methods [6] has regained notable interest within the recent years [7]–[11], the reverse direction has surprisingly been ignored.

This comes as a surprise, as population-based optimization routines produce over their run-time abundant data which can be considered to be complex and intractable for study by mere intuition of the practitioner. The strength of modern deep pattern recognition methods comes particularly helpful to this regard, as they mimic the feature learning capabilities of biological systems [12], [13], thus can be flexibly trained for different tasks, to learn representations which help them to efficiently process and differentiate given input data. In this work, we therefore explore opportunities to employ these strengths of deep pattern recognition methods to learn characteristics from data generated from evolutionary searches, capable of differentiating continuous optimization problems. With the goal of establishing a pipeline which allows us to learn and predict operators as inductive biases for evolutionary search algorithms.

The remainder of this paper is structured as follows: In Sec. II we review related work, with a special emphasis on methods which have been proposed for the analysis of the search behavior of evolutionary optimization algorithms in Sec. II-A and a review on recent advances towards feature-free algorithm selection in Sec. II-B. From reviewing these two lines of research, we argue for the advancements of methods for the continuous domain in Sec. II-C. We therefore propose over Sec. III, Sec. IV and Sec. V an approach based-upon a partitioning step of continuous search spaces using unsupervised clustering methods, and further deep neural network classifiers to learn problem characteristics from data generated during evolutionary searches to differentiate continuous optimization problems. In a subsequent experimental study in Sec. VI, we discuss advantages and disadvantages of these different approaches, specifically in regards to incorporating structural information about the neighborhood relationship between partition cells through cartesian maps and graphs, compare their performances, and analyze their behavior based upon different pre-processing steps, as well as benchmark functions of different properties. Finally, in Sec. VII we conclude our work and provide an outlook on future opportunities to employ our framework for practical problems.

II. Related Work

Relevant to our work, we can identify two lines of research which hybridize neural computing methods for pattern recognition with evolutionary search algorithms: The first one attempts to analyze behavioral optimization data using various supervised and unsupervised learning methods. The second one employs neural networks for classification and regression as a way to predict solvers and solutions based upon direct inputs generated from optimization problems.

A. Data Analytics for Algorithm Behavior

This line of research originates from early attempts at moving away from theoretical models of search behavior to more pragmatic ones which enable the analysis of algorithms through means of empirical measures [14]. While in principle, this has been already done in the past by relating analytically calculated properties of white-box optimization problems with algorithm performance (e.g.: [15]), the approach taken in [14] attempts to develop a new tool based upon directly assessing the behavior of an algorithm by keeping track of the problem-dependent movement of candidates solutions in the search space as a whole (c.f. Fig. 1).

Central to their method is the use of a self-organizing map (SOM) [16] which is a clustering technique that superimposes a neighborhood structure upon the cluster nodes, such that every cluster can be identified through a set of tuples $(n_1, \cdots, n_N)$ of size N within a regular structured coordinate system. Usual implementations of the self-organized map rely upon two-dimensional and rectangular coordinate systems with $N = 2$. Ref. [14] uses the self-organized map as a way to model the population structure within the search space of continuous single-objective optimization problems. In an initial training phase, the self-organized map is fitted to the start population. Based upon the obtained map, one can associate fitness values, solution counts and solution distances to the nearest cluster centroid. Subsequently, the evolutionary algorithm is iterated for another generation and a second training phase is initiated to refit the map to the new population. Based upon changes to cluster assignments, as well as population and fitness density, Ref. [14] define empirical measures quantifying exploration and exploitation behavior. Note, that their focus is solely to describe different search behaviors of evolutionary algorithms from a qualitative perspective.

More loosely based upon their work, Ref. [17] follows up this approach using behavioral optimization data. However, this time with a clear focus on learning features which allow them to explicitly differentiate the behavior of different evolutionary algorithms and optimization problems, and not to describe them. Likewise they adapt the self-organized map within their work, but instead to obtain a low-dimensional representation of the search space self. Thus, in advance fit it to training data uniformly and exhaustively generated within the search space. Experiments are setup with a fixed initial population and subsequently the evolutionary algorithm or problem of interest is varied and the generated offspring population is recorded. Using the offspring generation, the cluster assignments on the self-organized map are recorded and post-processing is done using PCA and the slow-feature analysis technique. Where the latter is explicitly used to construct a feature space in which the different evolutionary algorithms and optimization problems separate. Within their experiments, Ref. [17] can show that they are able to construct feature spaces which

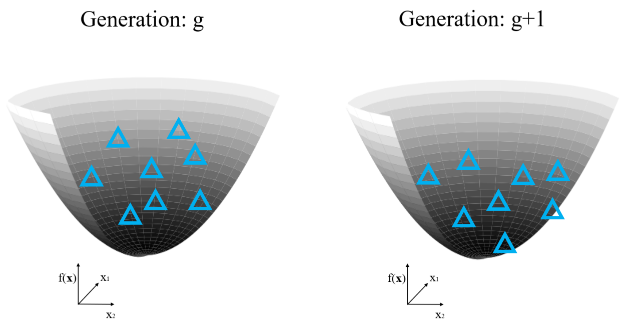

Fig. 1. In works on algorithm behavior studies, a key idea is that algorithms and problems can be characterized by the specific changes imposed on the solution distribution from $P_g \to P_{g+1}$, through the interplay between algorithm and problem over successive generations $g \to g+1$.

can sufficiently separate different evolutionary algorithms. However, they are incapable of sufficiently separating different optimization problems for given fixed optimization algorithms, unless the given problems contain obvious asymmetries. Notably, their work neglects any information about fitness values or distances of the candidate solutions to their assigned cluster node.

Work from Ref. [18] has been further following up the slow-feature analysis based approach. However, motivated by particular drawbacks of the former, their work explores the question whether or not a modern deep network architecture can be used to learn features capable of separating different evolutionary algorithms. Likewise, to previous work, the self-organized map is used once again as a way of obtaining a low-dimensional representation of a search space. However, they explicitly harness the two-dimensional structure of the self-organized map by recording generational changes in the assigned number of candidate solutions to every cluster node in a matrix representation. Once these are obtained, they can subsequently be labeled and fed for training to a neural network architecture based upon convolutional layers [19]. Note, that the latter are used in an attempt to explicitly exploit the two dimensional structure of the input matrices. Their study shows, that they can indeed achieve competitive results to the previous SFA-based method, by means of obtaining a latent space in which behavioral data of different algorithm is significantly disentangled. However, note their work does not reconsider the problem of distinguishing different continuous optimization problems.

B. Feature-Free Algorithm Selection

The application of pattern recognition methods has a longer tradition within the field of evolutionary optimization. E.g., in the field of hyper-heuristics and combinatorial optimization, they can be used to learn a mapping from a known problem state to the best known optimal solver [20]. They may also find their application within traditional studies of algorithm selection pipelines [21], [22]. However, in many cases obtain-

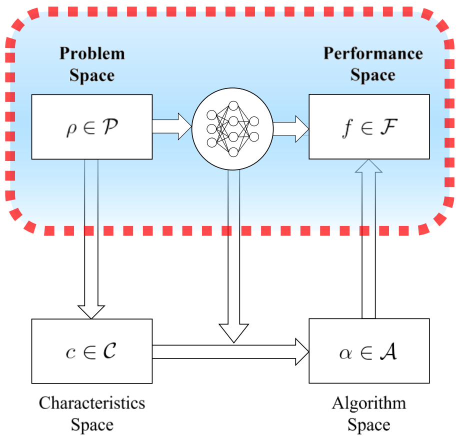

Fig. 2. Recent works on algorithm selection frameworks have attempted to short-cut the traditional pipeline, by introducing a direct mapping from problem space to performance space, where the calculation of problem characteristics is encapsulated into deep neural network architectures, from which directly the best performing algorithms and solutions are predicted.

ing good and descriptive characteristics is the problem which needs to be solved in the first place [23].

Recent ambitions within the field of combinatorial optimization have attempted to short-cut this step by employing deep neural networks (c.f. Fig. 2). Thus, the step of calculating problem characteristics is done implicitly by the network architecture, which is trained to predict algorithms and solutions to efficiently solve optimization problems in the first place. From an intelligent systems design point of view, this reflects the previously mentioned notions which features also have in biological cognitive systems [12], [13]. Particularly, the recent work of [24], explicitly uses deep neural networks to predict the optimal solver for traveling salesperson problems (TSP). However, instead of calculating problem characteristics of the different TSP instances, they generate 512x512 raster images of visual representations from plots of point clouds, minimum spanning trees and nearest neighbor graphs. Based upon these generated images, a CNN-based classifier is trained to predict the best known solver for a given TSP problem instance from two available ones. Through their results they can prove, that with the feature-free neural network based approach they can achieve results which are competitive and partly surpassing classical algorithm selection frameworks.

A similar and parallel line of work [25], [26] has investigated the use of sequential models, particularly implementations of LSTMs [27], to solve 1-d bin packing problems. In principle, with an interest in either predicting the optimal solvers for a given problem instance or optimal solutions. For the former case, they can show that their approach is capable of achieving higher performance than the single best solver (SBS). While for the latter, they find that their method can predict solutions very accurately to heuristics used to generate the training data, and at times even generates them with comparably higher performance.

C. Synopsis

Summarizing the reviewed work, deep pattern recognition methods found an interest in either enabling the description of search behavior from evolutionary algorithms, or using them such that the calculation of characteristics in algorithm selection framework is encapsulated by the feature learning capabilities of neural networks. However, surprisingly most of the latter reviewed work in regards to feature-free algorithm selection [24]–[26] has only been done within the domain of combinatorial optimization. Even though, prior research on methods to describe algorithm behavior [14], [17], [18] can be considered to have laid a certain foundation for further investigation. Therefore, in the following we advance this work by investigating whether we can propose a pipeline to learn features capable of distinguishing different continuous optimization problems from procedural optimization data. Showing the efficiency of such an approach, could enable us to likewise integrate it into a framework to learn and predict problem-tailored algorithm components from inputs generated during the run-time of an optimization algorithm.

In our investigation, we adopt the procedure of partitioning the search space. However, we lay a special focus on a comparative investigation, by imposing different kinds of neighborhood relationships upon the retrieved partitions. By keeping track of problem-specific changes within each cells of the search space partitions, we subsequently train classifiers based upon specialized neural network architectures to learn features capable of separating the different continuous optimization problems within a latent space. We also further include in our work a channel for fitness values, as previously work solely based upon changing solution counts has shown ambiguity for symmetric functions [17] for the task of problem identification.

III. Partitioning the Search Space

In the following, we will elaborate on different methods to partition the search space of continuous optimization problems. Specifically, with an emphasis on the different ways how to retrieve search space partitions through unsupervised clustering methods, and ways to impose a neighborhood relationship on them. The necessity of such a step might not seem obvious within low dimensions, as one might be simply inclined to equally divide the space among each axis into p pieces. However, as the number of partitions would scale exponentially with p^d at high dimensions d through this approach, it would become infeasible if one still wants to maintain a sufficient resolution.

Our principle approach is outlined in the top row of Fig. 3. For a search space volume $\chi \subset \mathbb{R}^d$, we first generate $D_T = \{\mathbf{x}_1, \cdots, \mathbf{x}_N\}$ samples uniformly random within the search space volume. This dataset D_T can then be subsequently used as training data for clustering methods to partition the search space homogeneously. The number of preset clusters N_C can be seen as regulating the resolution of the retrieved partition. Specifically, we investigate in the

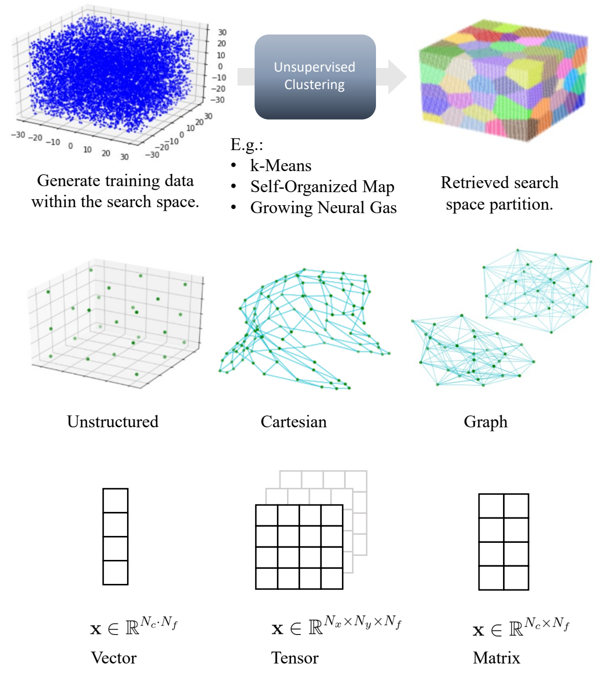
Unsupervised Clustering
Generate training data within the search space.
E.g.:
• k-Means
• Self-Organized Map
• Growing Neural Gas
Retrieved search space partition.
Unstructured
Cartesian
Graph
$\mathbf{x} \in \mathbb{R}^{N_c \cdot N_f}$
Vector
$\mathbf{x} \in \mathbb{R}^{N_x \times N_y \times N_f}$
Tensor
$\mathbf{x} \in \mathbb{R}^{N_c \times N_f}$
Matrix

Fig. 3. Top row: Explanation of the search space partitioning through unsupervised clustering. Middle row: Neighborhood-relationships which can be enforced through different techniques (i.e. k-Means, SOM, GNG & Delaunay). Bottom row: Different data formats which are obtained using the different neighborhood relationships.

following as clustering methods *k-means*, the *self-organized map* and the *growing neural gas*.

A. Unstructured Partitions

Applying the k-means algorithm to a given training dataset D_T, with a preset value $k = N_C$ usually results in the algorithm retrieving unstructured clusters after N iterations with centroids $\{\boldsymbol{\mu}_i\}_{i=1}^{N_C}$ and without any further neighborhood relationship being imposed on them. The cluster centroids are usually initialized randomly on the dataset and iteratively updated such that

$$\boldsymbol{\mu}_i = (\Sigma_n r_{nk} \cdot \mathbf{x}_n)/(\Sigma_n r_{nk}), \tag{1}$$

where $r_{nk} = 1$ if the closest $\boldsymbol{\mu}_j$ for given $\boldsymbol{x}_n$ has $j = k$, otherwise it is 0. As the k-means algorithm is usually part of many introductory literature [28] as well as standard software packages and libraries for machine learning and statistics [29], we neglect in the following any further more elaborate discussion of it.

B. Structured Partitions as Maps

Structured clusters can be retrieved using the self-organized map (SOM) technique [16]. As mentioned, this approach has been used in prior work [17], [18] as a way to partition the search space, originally motivated by work on modeling solution populations [14] within continuous evolutionary optimization. In its usual formulation, the SOM imposes a 2-d grid structure upon the clusters, such that the total number of clusters is $N_c = N_x \cdot N_y$ and the clusters can be identified through tuples $n_c = (n_x, n_y)$ with $n_c \in [1, N_x] \times [1, N_y]$. In the recursive formulation [16], each of the N_c clusters is identified by a centroid $\boldsymbol{\mu}_i$ (sometimes also called weights or model vectors), being likewise to k-means randomly initialized on the training dataset D_T, and updated at each iteration t for a given training data point $\mathbf{x}$ by

$$\boldsymbol{\mu}_i(t+1) = \boldsymbol{\mu}_i(t) + h_{ci}(t)[\mathbf{x}(t) - \boldsymbol{\mu}_i(t)], \tag{2}$$

where c is the index of the best matching unit (BMU), i.e. likewise to the k-means algorithm $c = \arg\min_i(||\mathbf{x}(t) - \boldsymbol{\mu}_i(t)||)$, and i being the index of its topological neighbors. Here, h_{ci} is a neighborhood function with

$$h_{ci}(t) = \alpha(t)\, \exp(-||\boldsymbol{\mu}_c - \boldsymbol{\mu}_j||^2/2\sigma^2(t)), \tag{3}$$

where $\sigma(t)$ and $\alpha(t)$ are monotonically decreasing functions of t. For the former, according to literature [16] its exact form does not matter, as long as $\sigma(t)$ is a monotonically decreasing function with it's value being about half of the grid diameter in the beginning and reduced after about 1000 steps to only a fraction of it. The use of the SOM to partition a high-dimensional space can be motivated as an attempt to topologically 'fold' a low-dimensional space into a higher dimensional one (c.f. central panel of Fig. 3) .

C. Structured Partitions as Graphs

1) Delaunay Triangulations: The cluster centroids $\{\boldsymbol{\mu}_i\}_{i=1}^{N_C}$ retrieved on the basis of the k-means algorithm can be re-interpreted as nodes of a graph structure. To construct a graph, one simply considers for each cluster with centroid $\boldsymbol{\mu}_i$ its associated decision volume $V(i)$, finds all neighboring volumes $V(j)$ and subsequently builds an adjacency matrix $\mathbf{A}$, with $a_{ij} = 1$ for neighboring pairs (i, j) and $a_{ij} = 0$ for unneighbored pairs (i, j). This procedure is known as Delaunay triangulation. In principle, implementing this procedure into an algorithmic form is not trivial and requires a less simplified approach. However, library implementations are available [30].

2) The Growing Neural Gas: The growing neural gas (GNG) [31] can be considered to be a variation of the former SOM [16]. However, its focus is on evolving a graph of vertices and edges (V, E) which describe the topology of the given dataset D_T. Thus, in principle the total number of clusters and edges can dynamically change during the training process. Likewise to k-means and the SOM, the training starts with N_c clusters with positions $\boldsymbol{\mu}_i$ being randomly initialized on the dataset D_T. Based upon a randomly drawn data point $\mathbf{x} \in D_T$, the nearest cluster $\boldsymbol{\mu}_1$ and second-nearest cluster $\boldsymbol{\mu}_2$ are determined. If the cluster $\boldsymbol{\mu}_1$ has edges, the ages of the edges are incremented and an error variable $\Delta\text{error}(1) = ||\boldsymbol{\mu}_1 - \mathbf{x}||^2$ is calculated. The cluster $\boldsymbol{\mu}_1$ and its topological neighbors $\boldsymbol{\mu}_n$ are subsequently moved towards the drawn data point $\mathbf{x}$ by fractions ϵ_b and ϵ_n with

$$\Delta w_{s_1} = \epsilon_b(\mathbf{x} - \boldsymbol{\mu}_1) \ \text{ and } \ \Delta w_{s_n} = \epsilon_n(\mathbf{x} - \boldsymbol{\mu}_n), \tag{4}$$

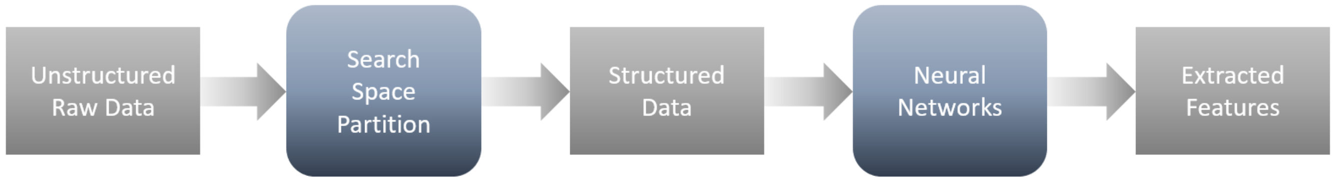

Fig. 4. Illustration of the data post-processing pipeline. Unstructured raw data descriptive of a solution population P_g^r at generation g and run r in the form of tuples $(\mathbf{x}, f(\mathbf{x}))$ of candidate solutions and fitness values are converted by a search space partitioning method into a structured data format $\mathbf{z}$, which subsequently can be fed to an adequate neural network architecture for feature extraction.

analogue to the self-organized map. If $\boldsymbol{\mu}_1$ and $\boldsymbol{\mu}_2$ possess an edge, its age is set to 0 and if no edge exists, it is created anew. Edges with an age larger than a_{max} are subsequently removed, and likewise, clusters without an edge are removed from the gas. After a certain number of iterations λ, the gas will insert a new cluster $\boldsymbol{\mu}_r$. This is done, by selecting the cluster $\boldsymbol{\mu}_q$ with the highest error, and subsequently inserting a new cluster half-way at $\boldsymbol{\mu}_r = \frac{1}{2}(\boldsymbol{\mu}_f + \boldsymbol{\mu}_q)$ between the neighbor with highest error $\boldsymbol{\mu}_f$. Subsequently, the old edges are removed and new ones are created. Errors are of q and f are lowered by a multiplicative factor α. The new cluster r subsequently inherits the updated error of q. At last, the neural gas decreases all errors by multiplication with a constant d. The algorithm terminates as soon as it has achieved a predefined network size or performance goal.

IV. The Data Post-Processing Pipeline

The full data post-processing pipeline is illustrated in Fig. 4. From running an evolutionary optimization algorithm on different single-objective optimization problems of the form $f : \chi \subset \mathbb{R}^d \to \mathbb{R}$, we first extract raw data in the form of tuples $(\mathbf{x}, f(\mathbf{x}))$ of generated solutions $\mathbf{x} \in \mathbb{R}^d$ and fitness values $f(\mathbf{x})$. We further organize these into tuples (P_g^r, y), where P_g^r is the population, it is the set of all $(\mathbf{x}, f(\mathbf{x}))$, at generation g and run r and y is a label for a particular optimization problem.

By applying a search space partition of N_c clusters to a solution population P_g^r, we obtain a structured data format in either the form of a vector $\mathbf{z} \in \mathbb{R}^{N_c \cdot N_f}$ for an unstructured k-means partition, the form of a tensor $\mathbf{z} \in \mathbb{R}^{N_x \times N_y \times N_f}$ for a structured map, or a feature matrix $\mathbf{z} \in \mathbb{R}^{N_c \times N_f}$ for structured graphs, where the latter are also further supplied with an adjacency matrix $\mathbf{A}$. Note that N_f is the number of cluster node features, and $N_x \cdot N_y = N_c$. After having converted all solution populations P_g^r into representations $\mathbf{z}_g^r$, we can use these for the subsequent feature learning and extraction step.

Note that in our approach we further process these by explicitly taking the differences $\Delta \mathbf{z}^r = \mathbf{z}_0^r - \mathbf{z}_1^r$. We also consider at most only two features with $N_f = 2$. I.e, the sum of all solutions associated to a cluster and the sum of all fitness values associated to a cluster. Effectively, we can interpret the $\Delta \mathbf{z}^r$ as finite differences, i.e. discrete derivatives in generational change $\Delta g = 1$, of total solutions and total fitness per cluster.

V. Neural Nets for Feature Learning

Depending upon the previously used search partition method, we choose in the next step of the processing pipeline the neural network architecture most suitable to process the obtained data format.

A. *Processing of Vector Data*

We use for vector data $\mathbf{z} \in \mathbb{R}^{N_c \cdot N_f}$ the multilayer perceptron (MLP) [28] with stacked dense layers of the form

$$\mathbf{h}^{(n)} = \sigma^{(n)}(\mathbf{W}^{(n)}\mathbf{h}^{(n-1)}), \tag{5}$$

where $\mathbf{h}^{(n)}$ is the output of the n-th hidden layer, where we have for the input layer $h^{(0)} = \mathbf{z}$, non-linear activation functions $\sigma^{(n)}$, in our case either ReLU($\mathbf{x}$):=max($\mathbf{0}, \mathbf{x}$) or SoftMax($\mathbf{x}$) = exp($\mathbf{x}$)/ $\sum_j$exp(x_j) and $\mathbf{W}^{(n)}$ being a trainable weight matrix.

B. *Processing of Tensor Data*

For tensor data $\mathbf{z} \in \mathbb{R}^{N_x \times N_y \times N_f}$ extracted using the SOM, we use the convolutional neural network (CNN), which is a special architecture that has been designed to process tensorial data, e.g. such as time-series and images [19]. Key ingredient of it are name-giving convolution operations [32] which can be written as

$$\mathbf{H}_{i,j,k}^{(n)} = \sigma\left(\Sigma_{l,m,p}\mathbf{H}_{(i-1)\times s+l,(j-1)\times s+m,p}^{(n-1)}\mathbf{W}_{l,m,p,k}^{(n)}\right) \tag{6}$$

where $\mathbf{H}^{(n-1)}$ is an input tensor, the parameter is s a so called stride and $\mathbf{W}^{(n-1)}$ is a kernel with trainable weights which can be parametrized by (l, m, p), and k being an index for the number of pre-defined filters. Further, we use pooling layers which are defined by

$$\mathbf{H}_{i,j,k}^{(n)} = \max_{m,n}\{\mathbf{H}_{(i-1)\times s+m,(j-1)\times s+n,\,k}^{(n-1)}\}. \tag{7}$$

Both operations are employed in specialized layers as a means to capture the underlying structural correlations hidden within the training data. We neglect a more elaborate discussion and refer to available literature instead [32], in favor of fostering in the following a comparison to their novel analogues for graph data.

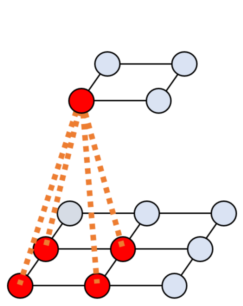
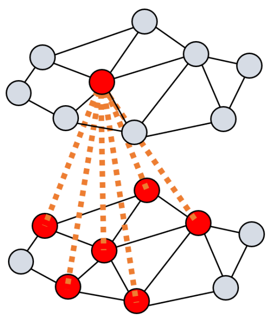

Fig. 5. In traditional convolution layers (left side), filters have a smaller dimension than the given input domain, and can aggregate features from patches of pre-defined arbitrary size from the input. In Kipf & Welling's graph convolution [33], filters have the same dimension as the input graph, and only aggregate features from the direct neighborhood of a given node.

C. Processing of Graph Data

For graph data represented by feature matrices $\mathbf{z} \in \mathbb{R}^{N_c \times N_f}$, we use in our work the recently developed techniques for graph neural networks (GNN) [34], [35]. While a variety of methods [35], [36] have been developed within the recent years, we will employ within our work particularly operations which have been defined in analogy to traditional operations, and became comparably popular. Specifically, we consider graph convolutions [33] as defined by

$$\mathbf{H}^{(n)} = \sigma^{(n)}(\tilde{\mathbf{D}}^{-\frac{1}{2}}\tilde{\mathbf{A}}\tilde{\mathbf{D}}^{-\frac{1}{2}}\mathbf{H}^{(n-1)}\mathbf{W}^{(n)}), \tag{8}$$

with a weight matrix $\mathbf{W}^{(n)} \in \mathbb{R}^{N_f \times N_F}$, further $\mathbf{H}^{(0)} = \mathbf{z}$, the adjacency matrix $\tilde{\mathbf{A}} = \mathbf{A} + \mathbf{I}$ with self-connections, as well as the degree matrix $\tilde{\mathbf{D}}_{ii} = \sum_j \tilde{\mathbf{A}}_{ij}$. The first four multiplicative terms can be interpreted as an aggregation operation over all features of the neighbors of a node (c.f. Fig. 5). Further, we also use pooling layers, based upon a graph coarsening step using Graclus algorithm [37], [38]

$$[\mathbf{A}_0^*, \cdots, \mathbf{A}_l^*; \mathbf{P}] = GraphCoarsening(\mathbf{A}, l), \tag{9}$$

with coarsened adjacency matrices $\mathbf{A}_j^*$ of size $N_C^{*j} \times N_C^{*j}$, with $N_C^{*j} = N_C^*/2^j$, where $j = 0, \cdots, l$ indicates the coarsening level up until $l \leq \ln(N_C^*)/\ln(2)$ and a permutation matrix $\mathbf{P}$ likewise of dimension $N_C^* \times N_C^*$. Note, that the Graclus algorithm extends any given input graph with N_C nodes by adding Δ feature-less fake nodes, such that $N_C^* = N_C + \Delta$ and further permuting the original node arrangement, such that the graph can be converted into a balanced binary tree. Thus, the original feature matrix must be converted by means of applying a permutation matrix $\mathbf{P}$ such that $\mathbf{X}^* = \mathbf{PX}$. Based upon the permuted feature matrix, which is ordered according to a balanced binary tree, pooling operations are then simply conducted branch-wise in analogy to 1D signals with

$$\mathbf{H}_{i,j}^{*(n)} = \max\{\mathbf{H}_{2i-1:2i,j}^{*(n-1)}\}. \tag{10}$$

Note that, even though recent work [39] has been questioning the utility of pooling operations in graph neural networks, we insist on including them nevertheless into our work to keep the analogy to traditional architectures. This is a reasonable decision, as there does not seem to be a clear consensus established upon this topic yet.

VI. Experimental Studies

The neural network architectures we use for feature extraction within our study are elaborated in Tbl. I. We implement them based upon available standard frameworks [40] and use custom implementations [33], as well as available ones [38] for the new layer-wise operations. We choose our GNN architecture in analogy to the tried-and-tested architecture for the CNN from Ref. [18], however take for both the liberty of using as classification layers our MLP architecture with bottleneck. We train each network using the adam optimizer [41] for 1000 epochs in a classification task, with a training to cross-validation dataset split of 80-20, a batch size of 250, using the categorical cross-entropy loss

$$\mathcal{L} = \sum_i -y_i \log(\hat{y}_i) \tag{11}$$

with $\hat{y}_i$ being the network output for a given class and y_i being the true label. For the prior search space partitioning step, we generate data uniformly random within a volume of $[-30, 30]^d$ with in total 10,000 training data points. Note that many of the standard single-objective optimization problems are defined on variable search space sizes. To accommodate for them within our experiments, we rescale any extracted population data to the size of the aforementioned training volume. Note, that this is a reasonable decision, as changing the search space sizes of the problems self would otherwise distort their original properties. Subsequently, using the obtained partition, we apply the data post-processing pipeline as elaborated in Sec. IV. For the unsupervised clustering methods to partition the search space, we use optimized implementations [29], [42] and train each for 1000 epochs when there is no self-termination implemented by default. The number of clusters

TABLE I
THE NEURAL NETWORK ARCHITECTURES USED FOR THE DIFFERENT DATA TYPES WITHIN OUR STUDY, WITH THE NUMBER OF CLASSES $\#C$ AND * INDICATING THE VISUALIZED LAYER.

Data Type	Vector	Tensor	Graph
Input Size	$N_c \cdot N_f$	$N_x \times N_y \times N_f$	$N_c \times N_f$
Layer0	Dense(10)*	Conv(5×5×25)	GraphConv(25)
Layer1	ReLU	ReLU	ReLU
Layer2	Dense(50)	MaxPooling(2×2)	MaxPooling(4)
Layer3	ReLU	Conv(3×3×16)	GraphConv(16)
Layer4	-	ReLU	ReLU
Layer5	-	MaxPooling(2×2)	MaxPooling(4)
Layer6	-	Dense(10)*	Dense(10)*
Layer5	-	ReLU	ReLU
Layer6	-	Dense(50)	Dense(50)
Layer7	-	ReLU	ReLU
Output	SoftMax(#C)	SoftMax(#C)	SoftMax(#C)

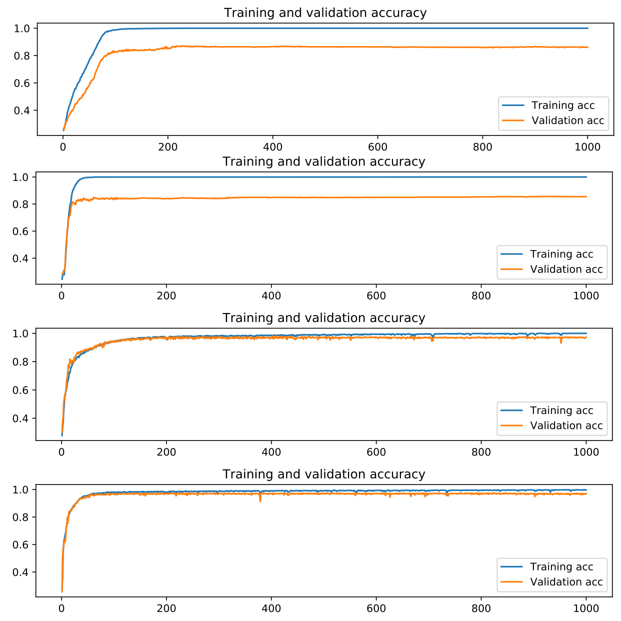

Fig. 6. Accuracy over the training for the MLP and CNN, as well as the GNN architecture for the GNG and Delaunay triangulation as input (from top to bottom).

is set to $N_C = 100$ for each.

As evolutionary algorithm of choice we use within our study the classic $(\mu{+}\lambda)$ Evolution Strategy [43], [44] with $\mu = 10$ and $\lambda = 10$, with strategy parameters $\sigma_i{\in}[0.1, 4]$, mutation and crossover probability of 0.5, which we randomly initialize on the entire search space with the given benchmark functions being of dimensionality d = 3, and further generate for each function 1000 pairs of parent and offspring generation.

A. *Comparison of Network Performances*

In the following, we compare the performances of our approaches in terms of training stability, achieved accuracy and cluster seperation. We train the networks upon data generated from symmetric function set in the upper half of Tbl. II. For the combined solution and fitness channel (S+F), we find that all networks exhibit stable training performance (c.f. Fig. 6) and are capable of achieving high accuracies in the range of $\sim$80$-$90%. Achieved values for the networks are listed in Tbl. III. Note that within prior available work [17], [18], different search spaces are used and results are only discussed qualitatively. Thus, these do not enable a direct quantitative comparison.

TABLE II
BENCHMARK FUNCTIONS USED WITHIN OUR STUDY. UPPER HALF: SYMMETRIC FUNCTIONS. LOWER HALF: ASYMMETRIC FUNCTIONS.

Name	Function Expression	Search Space
Ackley	$-20e^{-0.2\sqrt{\frac{1}{d}\Sigma_{i=1}^{d}x_i^2}} + 20 + e^1$ $-e^{-\frac{1}{d}\Sigma_{i=1}^{d}\cos(2\pi x_i)}$	$[-32.768, 32.768]^d$
Griewank	$\frac{1}{4000}\Sigma_{i=1}^{d}x_i^2{-}\Pi_{i=1}^{d}\cos(\frac{x_i}{\sqrt{i}}){+}1$	$[-600, 600]^d$
Rastrigin	$10d{+}\Sigma_{i=1}^{d}[x_i^2{-}10\cos(2\pi x_i)]$	$[-5.12, 5.12]^d$
Sphere	$\Sigma_{i=1}^{d}x_i^2$	$[-5.12, 5.12]^d$
Bohachevsky	$x_i^2{+}2x_{i+1}^2{-}0.3\cos(3\pi x_i)$ $-0.4\cos(4\pi x_{i+1}){+}0.7$	$[-100, 100]^d$
Elliptic	$\Sigma_{i=1}^{d}(10^6)^{\frac{i-1}{d-1}}x_i^2$	$[-100, 100]^d$
Rosenbrock	$\Sigma_{i=1}^{d-1}(1{-}x_i)^2{+}100(x_{i+1}{-}x_i^2)^2$	$[-5, 10]^d$
Schwefel	$\Sigma_{i=1}^{d}\left(\Sigma_{j=1}^{i}x_j\right)^2$	$[-65.536, 65.536]^d$

Overall, we find that the GNNs, trained upon graph-representations of the search space, obtained through the GNG and Delaunay triangulations, exhibit highest training performance on the validation sets with accuracies of about $\approx$ 96%. Followed up by the CNN and MLP with about $\approx$ 84%. Looking at the obtained feature spaces in Fig. 7, all compared methods exhibit clearly a high ability to seperate data inputs generated on the different optimization problems. However, one may argue, that the GNNs have a slightly better capability in separating the clusters. Note, that by comparing the unstructured as well as Delaunay-based approach, we can cross-check that the higher performance of the GNNs can be particularly attributed to considering additional knowledge about the structure of neighboring partition cells. As otherwise, the partition cells are in both approaches the same.

TABLE III
ACCURACY VALUES AVERAGED OVER 10 ITERATIONS FROM THE NEURAL NETWORK ARCHITECTURES USED FOR THE DIFFERENT DATA TYPES WITHIN OUR STUDY. FOR THE GNNS, (1) INDICATES INPUT DATA FROM THE GNG, WHILE (2) INDICATES THE DELAUNAY TRIANGULATION.

Architecture	Accuracy (S)	Accuracy (F)	Accuracy (S+F)
MLP	30.06 ± 0.75	83.89 ± 1.45	84.19 ± 0.88
CNN	28.00 ± 2.77	67.19 ± 2.90	84.20 ± 1.54
GNN[1]	26.95 ± 1.48	94.01 ± 0.49	96.90 ± 0.47
GNN[2]	27.06 ± 1.24	93.90 ± 0.92	96.46 ± 0.90

A particularly interesting question is to which regard the solution channel (S) and fitness channel (F) contribute to the training of the networks. We therefore trained all networks separately on each channel and collected likewise accuracy values averaged over 10 training runs on each. The resulting values are listed in Tbl. III. We find, that training the networks solely based upon changes in the solution channel (S) makes them incapable of separating the inputs from the symmetric function set. With accuracies being only in the range of $\sim$ 26$-$30%. The bulk of performance gain in the network training can therefore be attributed to the fitness channel (F). With the difference in accuracy for the MLP and the GNNs to the combined channel (S+F) being only about $\approx$1$-$3%. But arguably, the inclusion of the solution channel still contributes to performance improvements. This is most striking for the CNN, where the accuracy gain is about $\approx$ 17%. While this seems surprising at first glance, considering the fact, that by means of 'folding' the SOM into the higher dimensional space, neighborhood relationships are created which don't reflect the actual structure of the search space, including the solution channel (S) therefore can be considered as helping the network

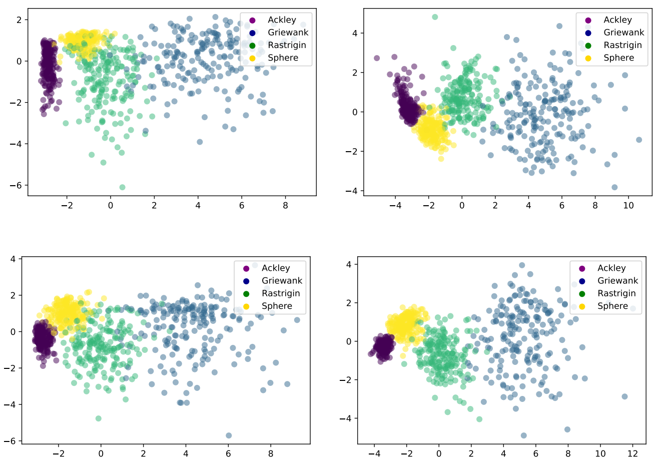

Fig. 7. LDA-plots of the feature spaces obtained on the symmetric function set from Tbl. II for the MLP, the CNN and GNN (GNG & Delaunay) (from left to right and top to bottom).

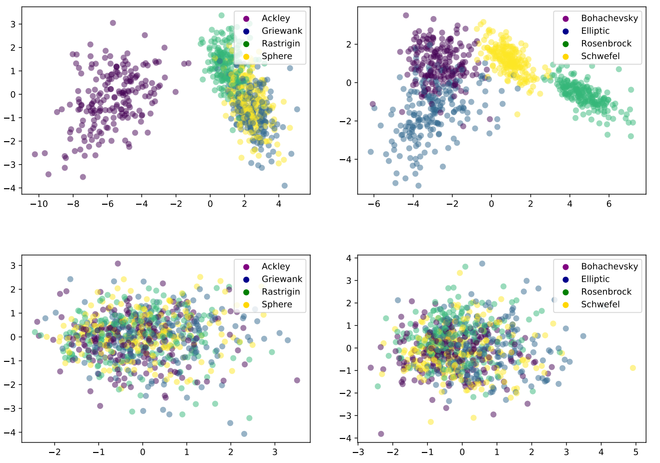

Fig. 8. Upper row: LDA-plots of the feature spaces for the GNG-based trained GNN using rescaled search space sizes and normalizes fitness values on the symmetric (left) and asymmetric function set (right). Lower row. Obtained feature spaces for training on the solution channel (S).

in learning more accurate relationships between neighboring search space regions.

B. Rescaling of Benchmark Functions, Fitness Values and the Set of Asymmetric Functions

At last, we test the behavior of our approach in regards to rescaling the benchmark functions, fitness values and its behavior on asymmetric functions. For the symmetric function set, we find that rescaling the benchmarks to a uniform search space size of $[-5.12, 5.12]^d$ while keeping the algorithm configuration fixed has only a negligible effect. We thus neglect a further discussion of it, however keep it within the further parts and suggest the practitioner to generally consider such an approach, as hyperparameters are mostly tuned to characteristic length scales of a given set of optimization problems. Normalizing the fitness values such that for every benchmark function $0 \leq f(x) \leq 1$, we find that on the symmetric function set the clusters within the feature space order themselves according to the different funnel structures of their benchmark functions (c.f. upper left panel in Fig. 8). Particularly, clusters are separated into exponential $\sim 1 - \exp(-|x|)$ and quadratic $\sim x^2$ funnel structure. But notably, we find that an intra-cluster separation is still evident. Particularly, between functions with low (Sphere & Griewank) and strong periodic modulation (Rastrigin) superimposed on them in relation to their search space sizes.
At last, we consider our asymmetric function set as given in the lower half of Tbl. II. Training our graph neural network upon data generated from these benchmarks, we find initially, that the training does not properly converge. Therefore, we apply the previously elaborated fitness normalization step. Subsequently, we find that the network training properly converges and we likewise find within the feature space, that the clusters separate according to the different funnel structures (c.f. upper right panel in Fig. 8). However, in comparison to the symmetric function set (c.f. lower left panel in Fig. 8), we find that training the network solely on the solution channel likewise does not retrieve a feature space in which the optimization problem can be separated (lower right panel).

VII. Conclusions & Outlook

In conclusion, we find that we can use our framework to learn features in a latent space which are able to sufficiently separate different continuous optimization problems. And particularly we have shown that a graph-based approach demonstrates highest performance. Thus, enabling us to short-cut the calculation of problem characteristics in the traditional algorithm selection framework, by encapsulating this step into a training procedure of neural network architectures. Notably, this also reflects notions of problem features existing in cognitive research [12], [13]. Further, we have shown that depending on the problem structure and meaning of fitness values, our approach can be adapted to a variety of different scenarios. And particularly, the combined solution and fitness channel showed highest performance. We hope our investigation to be a further stepping stone towards a more cohesive framework which could allow us to predict and learn problem-tailored algorithm components [45] for continuous evolutionary search algorithms. With the most practical scenarios at hand being the prediction of improved initializations for Estimation of Distribution and CMA-ES algorithms. In the sense of inductive biases within current research on metalearning frameworks.

VIII. Acknowledgments

This research has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement number 766186 (ECOLE). It was also supported by the Program for Guangdong Introducing Innovative and Enterpreneurial Teams (Grant No. 2017ZT07X386), Shenzhen Science and Technology Program (Grant No. KQTD2016112514355531), and the Program for University Key Laboratory of Guangdong Province (Grant No. 2017KSYS008).

References

[1] J David Schaffer, Darrell Whitley, and Larry J Eshelman. Combinations of genetic algorithms and neural networks: A survey of the state of the art. In *Proceedings of COGANN-92: International Workshop on Combinations of Genetic Algorithms and Neural Networks*, pages 1–37. IEEE, 1992.

[2] Xin Yao. Evolving artificial neural networks. *Proceedings of the IEEE*, 87(9):1423–1447, 1999.

[3] Xin Yao and Md. Monirul Islam. Evolving artificial neural network ensembles. *IEEE Computational Intelligence Magazine*, 3(1):31–42, 2008.

[4] Geoffrey E Hinton and Steven J Nowlan. How learning can guide evolution. *Adaptive Individuals in Evolving Populations: Models and Algorithms*, 26:447–454, 1996.

[5] Stefano Nolfi and Dario Floreano. Learning and evolution. *Autonomous Robots*, 7(1):89–113, 1999.

[6] Yann LeCun, Yoshua Bengio, and Geoffrey Hinton. Deep learning. *Nature*, 521(7553):436–444, 2015.

[7] Omid E David and Iddo Greental. Genetic algorithms for evolving deep neural networks. In *Proceedings of the Companion Publication of the 2014 Annual Conference on Genetic and Evolutionary Computation*, pages 1451–1452, 2014.

[8] Felipe Petroski Such, Vashisht Madhavan, Edoardo Conti, Joel Lehman, Kenneth O Stanley, and Jeff Clune. Deep neuroevolution: Genetic algorithms are a competitive alternative for training deep neural networks for reinforcement learning. *arXiv preprint arXiv:1712.06567*, 2017.

[9] Thomas Elsken, Jan Hendrik Metzen, Frank Hutter, et al. Neural architecture search: A survey. *J. Mach. Learn. Res.*, 20(55):1–21, 2019.

[10] Risto Miikkulainen, Jason Liang, Elliot Meyerson, Aditya Rawal, Daniel Fink, Olivier Francon, Bala Raju, Hormoz Shahrzad, Arshak Navruzyan, Nigel Duffy, et al. Evolving deep neural networks. In *Artificial Intelligence in the Age of Neural Networks and Brain Computing*, pages 293–312. Elsevier, 2019.

[11] Kenneth O Stanley, Jeff Clune, Joel Lehman, and Risto Miikkulainen. Designing neural networks through neuroevolution. *Nature Machine Intelligence*, 1(1):24–35, 2019.

[12] David H Hubel and Torsten N Wiesel. Receptive fields, binocular interaction and functional architecture in the cat's visual cortex. *The Journal of Physiology*, 160(1):106–154, 1962.

[13] Colin Blakemore and Grahame F Cooper. Development of the brain depends on the visual environment. *Nature*, 228(5270):477–478, 1970.

[14] Mikdam Turkey and Riccardo Poli. An empirical tool for analysing the collective behaviour of population-based algorithms. In *European Conference on the Applications of Evolutionary Computation*, pages 103–113. Springer, 2012.

[15] Terry Jones and Stephanie Forrest. Fitness distance correlation as a measure of problem didculty for genetic algorithms. In *Proc. 6th Internat. Conf. on Genetic Algorithms*, 1995.

[16] Teuvo Kohonen. Essentials of the self-organizing map. *Neural networks*, 37:52–65, 2013.

[17] Chengshan Pang, Mang Wang, Weiming Liu, and Bin Li. Learning features for discriminative behavior analysis of evolutionary algorithms via slow feature analysis. In *Proceedings of the 2016 on Genetic and Evolutionary Computation Conference Companion*, pages 1437–1444, 2016.

[18] Lei Liu, Chengshan Pang, Weiming Liu, and Bin Li. Learning to describe collective search behavior of evolutionary algorithms in solution space. In *Asia-Pacific Conference on Simulated Evolution and Learning*, pages 196–207. Springer, 2017.

[19] Yann Lecun and Yoshua Bengio. *Convolutional networks for images, speech, and time-series*. MIT Press, 1995.

[20] Edmund K Burke, Michel Gendreau, Matthew Hyde, Graham Kendall, Gabriela Ochoa, Ender Özcan, and Rong Qu. Hyper-heuristics: A survey of the state of the art. *Journal of the Operational Research Society*, 64(12):1695–1724, 2013.

[21] Mario A Muñoz, Yuan Sun, Michael Kirley, and Saman K Halgamuge. Algorithm selection for black-box continuous optimization problems: A survey on methods and challenges. *Information Sciences*, 317:224–245, 2015.

[22] Pascal Kerschke, Holger H Hoos, Frank Neumann, and Heike Trautmann. Automated algorithm selection: Survey and perspectives. *Evolutionary Computation*, 27(1):3–45, 2019.

[23] Olaf Mersmann, Bernd Bischl, Heike Trautmann, Mike Preuss, Claus Weihs, and Günter Rudolph. Exploratory landscape analysis. In *Proceedings of the 13th Annual Conference on Genetic and Evolutionary Computation*, pages 829–836, 2011.

[24] Moritz Seiler, Janina Pohl, Jakob Bossek, Pascal Kerschke, and Heike Trautmann. Deep learning as a competitive feature-free approach for automated algorithm selection on the traveling salesperson problem. In *International Conference on Parallel Problem Solving from Nature*, pages 48–64. Springer, 2020.

[25] Mohamad Alissa, Kevin Sim, and Emma Hart. Algorithm selection using deep learning without feature extraction. In *Proceedings of the Genetic and Evolutionary Computation Conference*, pages 198–206, 2019.

[26] Mohamad Alissa, Kevin Sim, and Emma Hart. A deep learning approach to predicting solutions in streaming optimisation domains. In *Proceedings of the 2020 Genetic and Evolutionary Computation Conference*, pages 157–165, 2020.

[27] Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural Computation*, 9(8):1735–1780, 1997.

[28] Christopher M Bishop. *Pattern Recognition and Machine Learning*. Springer, 2006.

[29] Fabian Pedregosa, Gaël Varoquaux, Alexandre Gramfort, Vincent Michel, Bertrand Thirion, Olivier Grisel, Mathieu Blondel, Peter Prettenhofer, Ron Weiss, Vincent Dubourg, et al. Scikit-learn: Machine learning in python. *Journal of Machine Learning Research*, 12(Oct):2825–2830, 2011.

[30] Pauli Virtanen, Ralf Gommers, Travis E Oliphant, Matt Haberland, Tyler Reddy, David Cournapeau, Evgeni Burovski, Pearu Peterson, Warren Weckesser, Jonathan Bright, et al. SciPy 1.0: fundamental algorithms for scientific computing in Python. *Nature Methods*, 17(3):261–272, 2020.

[31] Bernd Fritzke. A growing neural gas network learns topologies. In *Advances in Neural Information Processing Systems*, pages 625–632, 1995.

[32] Ian Goodfellow, Yoshua Bengio, Aaron Courville, and Yoshua Bengio. *Deep learning*, volume 1. MIT Press Cambridge, 2016.

[33] Thomas N Kipf and Max Welling. Semi-supervised classification with graph convolutional networks. *arXiv preprint arXiv:1609.02907*, 2016.

[34] Michael M Bronstein, Joan Bruna, Yann LeCun, Arthur Szlam, and Pierre Vandergheynst. Geometric deep learning: Going beyond Euclidean data. *IEEE Signal Processing Magazine*, 34(4):18–42, 2017.

[35] Zonghan Wu, Shirui Pan, Fengwen Chen, Guodong Long, Chengqi Zhang, and S Yu Philip. A comprehensive survey on graph neural networks. *IEEE Transactions on Neural Networks and Learning Systems*, 2020.

[36] Mathias Niepert, Mohamed Ahmed, and Konstantin Kutzkov. Learning convolutional neural networks for graphs. In *International Conference on Machine Learning*, pages 2014–2023. PMLR, 2016.

[37] Inderjit S Dhillon, Yuqiang Guan, and Brian Kulis. Weighted graph cuts without eigenvectors a multilevel approach. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 29(11):1944–1957, 2007.

[38] Michaël Defferrard, Xavier Bresson, and Pierre Vandergheynst. Convolutional neural networks on graphs with fast localized spectral filtering. *Advances in Neural Information Processing Systems*, 29:3844–3852, 2016.

[39] Diego Mesquita, Amauri Souza, and Samuel Kaski. Rethinking pooling in graph neural networks. *Advances in Neural Information Processing Systems*, 33:2220–2231, 2020.

[40] François Chollet et al. Keras: The python deep learning library. *ascl*, pages ascl–1806, 2018.

[41] Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *3rd International Conference on Learning Representations*, 2015.

[42] Y Shevchuk. NeuPy: Neural Networks in Python, 2019.

[43] Thomas Bäck. *Evolutionary Algorithms in Theory and Practice: Evolution Strategies, Evolutionary Programming, Genetic Algorithms*. Oxford University Press, 1996.

[44] Félix-Antoine Fortin, François-Michel De Rainville, Marc-André Gardner, Marc Parizeau, and Christian Gagné. DEAP: Evolutionary algorithms made easy. *Journal of Machine Learning Research*, 13:2171–2175, jul 2012.

[45] Stephen Friess, Peter Tiňo, Stefan Menzel, Bernhard Sendhoff, and Xin Yao. Representing experience in continuous evolutionary optimisation through problem-tailored search operators. In *2020 IEEE Congress on Evolutionary Computation (CEC)*, pages 1–7. IEEE, 2020.