Spatio-Temporal Activity Recognition for Evolutionary Search Behavior Prediction

Stephen Friess[§], Peter Tiňo[§], Stefan Menzel[‡], Zhao Xu[†], Bernhard Sendhoff[‡] and Xin Yao[§*]
[§] CERCIA, School of Computer Science, University of Birmingham, UK
[‡] Honda Research Institute Europe GmbH, 63073 Offenbach a.M., Germany
[†] NEC Laboratories Europe GmbH, 69115 Heidelberg, 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*—Traditional methods for solving problems within computer science rely mostly upon the application of handcrafted algorithms. As however manual engineering of them can be considered to be a tedious process, it is interesting to consider how far internal mechanisms can be directly learned in an end-to-end manner instead. This is especially tempting to consider for metaheuristic and evolutionary optimization routines which inherently rely upon creating abundant amounts of data during run-time. To implement such an approach for these types of algorithms, it effectively requires a pipeline to first acquire derandomized algorithm components in a domain-dependent manner and secondly a mapping to select them based upon characteristic features which unveil the black box character of an optimization problem. While in principle, within our prior work we proposed methods for extracting spatial features from metadata, these unfortunately neglect the time-dependent nature of it. Thus, underperform in scenarios when the inputs generated from initial iterations are not expressive enough. For this reason we specifically develop within this work architectures for spatio-temporal data processing. Particularly, we find that our proposed GCN-GRU and LSTM architectures, which take inspiration from CNN-LSTMs originally proposed for activity recognition in multimedia data-streams, demonstrate high efficiency and most consistent performance on time series of variable length. Further, we can also demonstrate that the class activation map (CAM) for interpretable learning with time series data helps to understand and reflects problem-dependent properties of the search behavior of an optimization algorithm.**

***Index Terms*—Representation Learning, Algorithm Selection, Graph Neural Networks, Activity Recognition, Time Series Classification.**

I. Introduction

Derandomization has been introduced with great success within recent decades to metaheuristic and evolutionary algorithms. While traditional classic variants rely mostly upon uncorrelated stochastic noise for solving optimization problems, the concept of derandomization abandons this in favor of allowing the dynamic adaption of an algorithm's internal mechanisms to a problem-structure during run-time [1]–[3]. Residing to fundamental research on the processes in developmental biology these algorithms are often based upon, it has been well justified, that the variational mechanisms underlying natural evolution can exhibit properties such that to produce phenotypic variety in a focused and directed manner through developmental biases [4]. However, these may not only foster adaption to a specific environment, but may exhibit a more broader learning capability. Such that they are able to generalize performance gains to future unseen environments [5]. Notably, this notion mirrors well the concept of inductive biases within learning theory [6]. While one might be therefore inclined to pragmatically build a single monolithic algorithm, which simply learns the structures of different problems it solves during its lifetime, the no-free-lunch theorems [7] effectively set hard constraints to it.

Thus, the best one can do, is to learn algorithm components in a domain-dependent manner, and recall them based upon accessible problem characteristics. Essentially, this suggests an end-to-end learning approach. But while this line of thinking is being only recently popularized within traditional research [8], it has been previously already established for the domain of optimization. Particularly, algorithm selection and configuration problems constitute metalearning problems [9]. Which imply a functional separation between high-level model-free and low-level model-based components [10], [11], where the former are required to explicitly process any generated metadata such that to unveil the black box character optimization problems often possess.

Our paper is therefore dedicated towards advancing the state of methodology by putting an emphasis on processing the time-dependent nature of the generated metadata. Therefore, we review in Sec. II available literature towards end-to-end learning approaches of optimization algorithms. Subsequently, in Sec. III we elaborate on methods originally proposed for activity recognition and time series classification and suggest extensions to enable these for processing of spatio-temporal metadata generated by population-based optimization algorithms. Following this up, we investigate these proposed extensions within in an experimental study in Sec. IV, specifically, with a focus on evaluating their computational efficiency. We end our paper with a discussion in Sec. V and give an outlook on future potential directions to explore with the proposed methodology.

II. Related Work

Within traditional algorithm research, end-to-end learning approaches are being currently popularized, which offer possible new applications for e.g. minimum cut [8] or graph

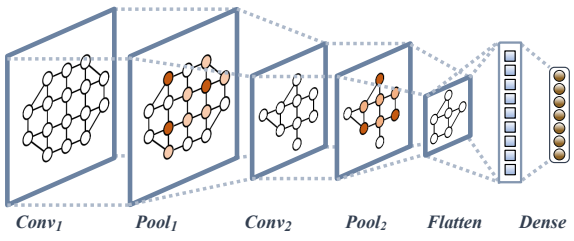

Fig. 1. The graph-based spatial feature extraction as proposed within our prior work [12]. Based upon two graph convolutions [13] with 25 and 16 filters, as well as graph pooling operations [14], structured data formats descriptive of a solution distribution are processed such that to extract low-dimensional features descriptive of search behavior.

recovery problems [15]. However, within metaheuristic and evolutionary optimization, this notion has been already established previously. Particularly, so called algorithm selection and configuration have been identified to constitute metalearning problems [9].

Though, traditional algorithm selection frameworks mostly rely upon tedious pipelines to map a given problem instance to a best known solver [16], [17], which may involve a combination of handcrafted feature extraction methods [18] and trainable predictive components. The advantage of modern day pattern recognition methods is that it allows in principle to directly short-cut any intermediate steps, such that predictive models can be directly learned in an end-to-end manner. Notable advances to this regard have been particularly made for combinatorial optimization. For instance, concerning the usage of long short-term memory networks (LSTMs) for 1d bin-packing problems [19], [20], as well as the use of convolutional neural networks (CNNs) [21] and transformers [22] to predict optimal solvers for traveling salesperson problems (TSP). Though, while a learned predictive model might not always be able to fully substitute handcrafted solvers, it may at times be able to generate higher performing solutions [20], thus can be considered to be a useful complementary [23]. The comfort of combinatorial problems is that in principle the search spaces thereof are well-structured and problem characteristics may often be accessible in advance.

However, the situation is more elusive for continuous optimization problems which often possess a black box character. Thus, problem characteristics have to be either explicitly calculated or require processing of metadata generated during run-time. In principle, the latter constitutes in most population-based algorithms the solution population P_g over successive iterations g, i.e. a set $P_g = \{\mathbf{p}_i^{(g)} | i = 1, \cdots, \mu\}$ with $\mathbf{p}_i^{(g)} = (\mathbf{x}_i^{(g)}, \mathbf{F}(\mathbf{x}_i^{(g)}))$, for $\mathbf{x}_i \in \chi$ with search space $\chi \subset \mathbb{R}^d$, $\mu = P_g$ and objective functions $\mathbf{F} : \chi \rightarrow \mathbb{R}^N$. Initial work on harnessing this metadata explored the use of self-organized maps (SOM) to model the structure of a solution population [24]. However, in principle it is impractical for application scenarios, as it requires the retraining of the map at each iteration. More loosely based upon this work [25] harness the SOM instead as a way to obtain a structured representation of high-dimensional search spaces χ. Which they use to convert solution populations P_g into a tensor data format, from which subsequently characteristics can be extracted using slow-feature analysis (SFA). It has been alternatively demonstrated that a CNN can achieve comparable results [26] while avoiding drawbacks of the SFA. However, without taking explicitly further interest into distinguishing different problems from metadata. Addressing some of the short-comings of these previous works, we previously introduced [12] graph-based representations of search spaces in combination with novel graph neural networks (GNN) as an alternative for processing metadata. Particularly, showing that under the additional inclusion of a channel for fitness values, these can effectively distinguish different optimization problems and outperform more conventional methods. However, these still neglect the time-dependent nature of the retrieved metadata and thus perform only poorly in scenarios where the inputs in the fitness channel are not informative.

Therefore, to further advance the state of methodology and address this open issue, we specifically look in the following into end-to-end architectures for spatio-temporal data processing. While using the previously introduced graph-based methodology for spatial feature extraction, we take inspiration from methodology originally proposed for activity recognition and time series classification, for temporal data processing.

III. Spatio-Temporal Data Processing

A. Spatial Feature Extraction

In the following, we will use as mentioned for spatial feature extraction graph-based approaches as introduced by [12]. Besides graph-based data formats in the form of tuples of adjacency matrices and feature matrices $(\mathbf{A}, \mathbf{X})$ with $\mathbf{A} \in \mathbb{R}^{N \times N}$ and $\mathbf{X} \in \mathbb{R}^{N \times F}$, the use of specialized graph neural network [27], [28], particularly so called graph convolutions (GCNs) [13] as defined by

$$\mathbf{H}^{(n)} = \sigma^{(n)}(\hat{\mathbf{A}}\ \mathbf{H}^{(n-1)}\mathbf{W}^{(n)}), \tag{1}$$

is central to their method. Where $\mathbf{W}^{(n)} \in \mathbb{R}^{F \times F'}$ is a weight matrix, $\mathbf{H}^{(0)} = \mathbf{X}$, as well as $\hat{\mathbf{A}}$ being an effective adjacency matrix descriptive of the structure of the graph. While in principle, different methods to perform convolution operations on graphs exist (e.g. [28], [29]), the approach based upon the operation by Kipf & Welling [13] is particularly elegant, because in principle it is based upon a low-order heat diffusion model. Thus, the first two multiplicative terms in Eq. (1) can be interpreted as computing 'heat' propagation within the graph structure. Note, that due to this physical interpretation, these convolutions do not take in account the importance which difference node features might have.

To address this issue, we will consider within our work additionally graph attention operations (cf. Fig. 2) as originally introduced by [30] in which the elements of the effective adjacency matrix are explicitly learned through a regression

function $\hat{A}_{ij} := f(\mathbf{h}_i, \mathbf{h}_j)$ during training. Where the regression function is given by

$$f(\mathbf{h}_i, \mathbf{h}_j) = \frac{\exp\left(\text{LeakyReLU}\left([\mathbf{h}_i^T\mathbf{W}||\mathbf{h}_j^T\mathbf{W}]\mathbf{a}\right)\right)}{\exp\left(\sum_{k\in\mathcal{N}_i}\text{LeakyReLU}([\mathbf{h}_i^T\mathbf{W}||\mathbf{h}_k^T\mathbf{W}]\mathbf{a})\right)}, \tag{2}$$

where $\mathbf{a} \in \mathbb{R}^{2F'}$ is a vector with trainable weights, $||$ a concatenation operation, $\mathcal{N}_i$ the neighborhood of a reference node i inclusive of itself and $\mathbf{h}_i, \mathbf{h}_j, \mathbf{h}_k$ are row vectors of the feature matrix $\mathbf{X}$. To stabilize the training of a graph attention network, [30] additionally suggest to use instead an average of K individual attention operations, so called 'attention heads', such that Eq. (1) is modified to

$$\mathbf{H}^{(n)} = \sigma^{(n)}\left(\frac{1}{K}\sum_{k=1}^{K}\hat{\mathbf{A}}_k\ \mathbf{H}^{(n-1)}\mathbf{W}_k^{(n)}\right). \tag{3}$$

In principle, given the aforementioned graph convolution and attention operations, we have all ingredients together to build a variety of different graph neural networks for spatial feature extraction.

B. Temporal Data Processing

1) CNN-based Time Series Classification: While a great variety of different approaches have been proposed within the literature [31], we focus in the following particularly on discriminative deep network methods that can be trained in an end-to-end manner [32]. Particularly popular to this regard are architectures which are centered upon simple 1d convolution operations. Given a multi-variate time series signal $\mathbf{X} \in \mathbb{R}^{T\times d}$ with d dimensions of length T, with $\mathbf{H}^{(0)} = \mathbf{X}$ convolution operations are then iteratively applied such that

$$\mathbf{H}_{ij} = \sigma(\Sigma_{l,p}\mathbf{H}_{(i-1)\times s+l,p}\mathbf{W}_{l,p}) \tag{4}$$

with non-linearity $\sigma(\cdot)$, s being a stride and the filter matrix being given by $\mathbf{W} \in \mathbb{R}^{T\times d}$. Note that, within this framework the multi-variate nature of the time series is accounted for by means of treating these as multi-channeled signals, while the time dimension is kept being treated as 1-dimensional. For our purposes, two particular CNN-based architectures are of most interest. Where the first one has been dubbed within the literature as *Fully Connected Network* (FCN) [33] and the second one simply as *Encoder* (ENC) [34]. Both architectures are high-performing for either multivariate or short-time series, while at the same time lightweight in terms of trainable parameters [32]. We refer the interested reader for more in-depth details to available literature [33], [34].

2) ANN-RNN-based Spatio-Temporal Activity Recognition: Combinations of ANNs for feature extraction and RNNs for sequential processing have been established for applications of activity recognition in multimedia data-streams [35], [36], vehicle behavior prediction [37] and pre-miRNA classification [38]. The most common instantiation is in the form of a CNN-LSTM which is directly trained in an end-to-end manner. Within our work, we specifically propose GNN-RNN architectures for processing metadata generated by optimization algorithms. Particularly, using the aforementioned

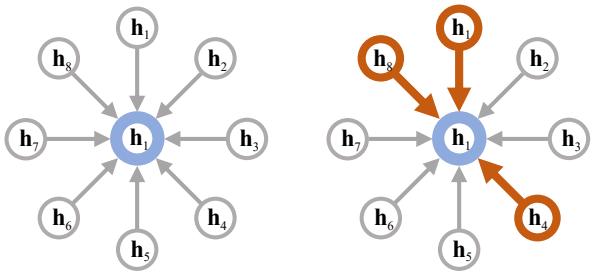

Fig. 2. Illustration of the difference between graph convolution [13] (left) and graph attention operations [30] (right). While the graph convolution only propagates node features based upon the number of neighborhood connections, graph attention operations explicitly allow nodes with more important features to be valued higher when features are propagated in a neighborhood.

GCN and GAT operations from Sec. III-A for spatial feature extraction. The particular RNN we use is either a simple recurrent neural network (sRNN), a gated recurrent unit (GRU) or long short-term memory (LSTM). Considering an input sequence $\mathbf{x}_1, \cdots, \mathbf{x}_T$, for T time-steps, a simple recurrence can be formulated through [39], [40]

$$\begin{aligned}\mathbf{h}_t &= \tanh(\mathbf{b} + \mathbf{W}\mathbf{h}_{t-1} + \mathbf{U}\mathbf{x}_t),\\ \text{and } \hat{\mathbf{y}}_t &= \sigma(\mathbf{c} + \mathbf{V}\mathbf{h}_t),\end{aligned} \tag{5}$$

where $\mathbf{h}_t$ is the hidden output and $\hat{\mathbf{y}}$ the predicted label for input element $\mathbf{x}_t$, as well as $\mathbf{b}$, $\mathbf{c}$, $\mathbf{W}$, $\mathbf{V}$ being bias vectors and weight matrices and $\sigma(\cdot)$ an activation function. This architecture transforming the entire input sequence into an output sequence of equal length T is known as being many-to-many. Neglecting the second equation and only calculating the last hidden output $\mathbf{h}_T$, we retrieve an architecture which is known as many-to-one. Within our implementation [41], we will neglect the second equation and only use $\mathbf{h}_t$ as output. In comparison to this simple RNN (sRNN), the state-of-the-art can be considered to be posed by the long short-term memory (LSTM) network. The LSTM can be described in total six update equations

$$\begin{aligned}\mathbf{f}_t &= \sigma(\mathbf{W}_f \cdot [\mathbf{h}_{t-1}, \mathbf{x}_t] + \mathbf{b}_f),\\ \mathbf{i}_t &= \sigma(\mathbf{W}_i \cdot [\mathbf{h}_{t-1}, \mathbf{x}_t] + \mathbf{b}_i),\\ \tilde{\mathbf{C}}_t &= \tanh(\mathbf{W}_C \cdot [\mathbf{h}_{t-1}, \mathbf{x}_t] + \mathbf{b}_C]),\\ \mathbf{C}_t &= \mathbf{f}_t * \mathbf{C}_{t-1} + \mathbf{i}_t * \tilde{\mathbf{C}}_t,\\ \mathbf{o}_t &= \sigma(\mathbf{W}_o \cdot [\mathbf{h}_{t-1}, \mathbf{x}_t] + \mathbf{b}_o)\end{aligned} \tag{6}$$

and at last the computation of the hidden state

$$\mathbf{h}_t = \mathbf{o}_t * \tanh(\mathbf{C}_t), \tag{7}$$

which can be used analogous to the second equation in (5) to calculate the step-wise output. Note, that $*$ denotes a component-wise product. A key difference of the LSTM to the sRNN is the explicit introduction of gating mechanisms in the first three lines of Eq. (6) through memory cells $\mathbf{C}_t$. These not only dynamically control the influence of past hidden states on future ones, but are also an essential milestone in enabling and mitigating problems with the training of RNNs

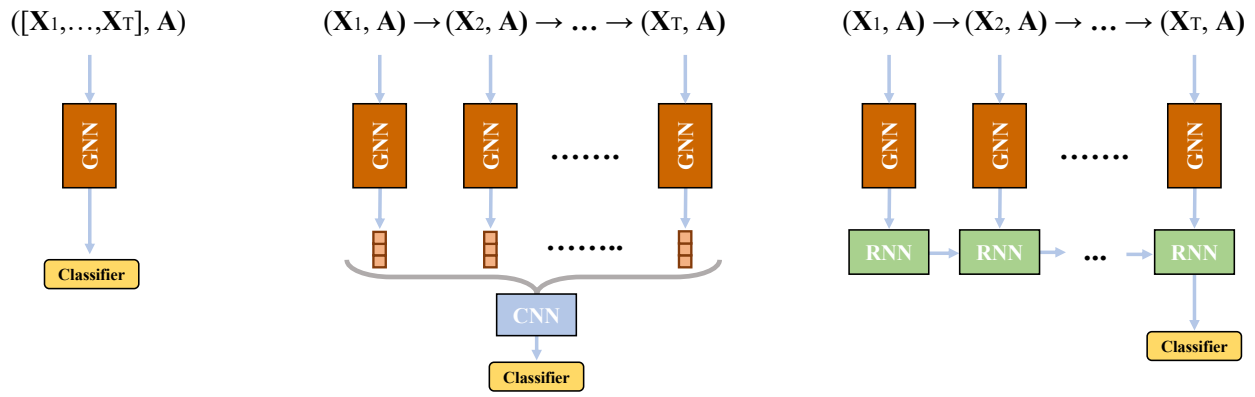

Fig. 3. Simplified schematics of the end-to-end architectures used within our work. From left to right: Graph neural network under simple concatenation of input graphs (GNN), graph neural network as spatial feature extractor with CNN-based time series classification architecture on top (GNN-FCN & GNN-ENC) and at last GNN-based spatial feature extraction with recurrent network for temporal processing on top (GNN-RNN).

on datasets with long-term dependencies. However, due to the large amount of parameters introduced, LSTMs cannot always be considered to be a reasonable choice. To keep the novelties introduced by LSTMs, but at the same time prune their complexities, the so called Gated Recurrent Unit (GRU) has been alternatively introduced. Defined by in total four equations

$$\begin{aligned} \mathbf{z}_t &= \sigma(\mathbf{W}_z \cdot [\mathbf{h}_{t-1}, \mathbf{x}_t] + \mathbf{b}_z), \\ \mathbf{r}_t &= \sigma(\mathbf{W}_r \cdot [\mathbf{h}_{t-1}, \mathbf{x}_t] + \mathbf{b}_r), \\ \tilde{\mathbf{h}}_t &= \tanh(\mathbf{W}_h \cdot [\mathbf{r}_t * \mathbf{h}_{t-1}, \mathbf{x}_t] + \mathbf{b}_h), \\ \mathbf{h}_t &= (1 - \mathbf{z}_t) * \mathbf{h}_{t-1} + \mathbf{z}_t + \tilde{\mathbf{h}}_t, \end{aligned} \tag{8}$$

the gated recurrent unit shares obvious similarities to Eq.(6), however skips intermediate steps introduced by the calculation of the memory cell $\mathbf{C}_t$.

IV. Experimental Studies

A. Generation of Synthetic Metadata

As no agreed upon reference datasets exist of yet, we have to reside in the following to explicitly create one by ourselves. For this reason, we use in the following the symmetric optimization function set from our previous study [12], which consists out of the commonly used Ackley, Griewank, Rastrigin and Sphere function (e.g. see for reference [42]) as illustrated in Fig. 6, which either have exponential $\sim 1-\exp(-|x|)$ or quadratic $\sim x^2$ global symmetric funnel structure, different periodicities superimposed on them, as well as a single global optimum contained at the origin. As evolutionary search algorithm (EA) we use the $(\mu+\lambda)$ Evolution Strategy [42]. We initialize the start population only in the corner of the search space defined by $[s_b/2, s_b]^d$, where s_b is the upper boundary and $d = 3$. As otherwise, the crossover operator will lead to rapid convergence due to the central position of the global optimum.

We configure the EA with $\mu = \lambda = 10$, i.e. a population size of 10, strategy parameter $\sigma \in [0.1, 2.0]$ for a reference size of $s_b = 5.12$ and crossover and mutation probability of 0.5. To convert metadata into a structured data format, we explicitly use a graph-structure evolved using a growing neural gas to map out the search space. Using the nodes of this graph structure, a solution population P_g can then be converted into a structured data format $\mathbf{z}_g \in \mathbb{R}^{N \times F}$ by finding the closest graph node to each solution in P_g, and summing up solutions associated with them. Note, that we do not consider a fitness channel within our study. Similar to [24], we distinguish between growth and non-growth regions explicitly by calculating for successive generations $g \rightarrow g+1$ the vectors $\Delta\mathbf{z}_g$ and $\mathbf{z}_g^*$, where the first one encodes changes in the solution distribution under application of the EA, and the second one invariant parts. We explicitly use within our experiments time-series generated from running the EA for 20 iterations, and generate 1000 samples for each benchmark. To accommodate for the varying search space size, we rescale σ by a factor $s_b/5.12$ accordingly.

B. Network Configuration and Training

Because we will evaluate and compare in the following in total 12 different neural networks architectures, we have to ensure that they are configured and trained in a way such that to enable a fair comparison. Though, admittingly our emphasis is not on finding out which architecture is in theory the best, but instead which one can be considered to be reasonably efficient and performant under a fixed budget.

Therefore, at best we only minimally modify the previously elaborated architectures. Within all experiments, we use a GNN either based upon GCN or GAT operations for spatial feature extraction. The architecture of the spatial feature extractor is based upon the originally proposed one by [26], however with the adjustments for processing graph data as suggested within our previous study [12] (cf. Fig. 1) using graph convolution and coarsening layers [14], [43], as well as 10-d output features. For the GAT-based feature extraction, we simply replace the convolution layers. Note, that GAT architectures explicitly introduce the number of attention heads K as a hyperparameter.

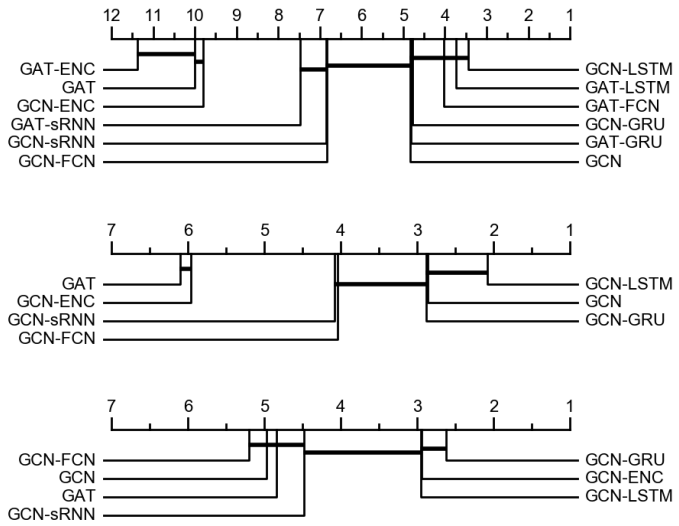

Fig. 4. Critical difference diagrams for the comparison of all GCN and GAT architectures from Tbl. II and III on the 20 time-steps long dataset (top), as well as the subset of GCN architectures on the 20 time-steps long dataset (center) and 10 time-steps long dataset (bottom). Note, that on the horizontal axis the total number of achievable ranks is denoted, while the bent lines springing from the labels indicate the rank of the architecture. Architectures connected by a thick black line are statistically indistinguishable in performance.

TABLE I
HYPERPARAMETERS AND TESTING RANGES FOR THE DIFFERENT ARCHITECTURES. NOTE THAT FOR SIMPLICITY RNN* DENOTES SRNN, GRU AND LSTM, AS WELL AS ARGS HIDDEN NODES AND ATTENTION HEADS.

	Epochs	**Batches**	**Learning Rate**	**Args***	
Range	-	2^n	$q \times 10^z$	2^n	n
GCN	300	256	0.001	-	-
GAT	300	256	0.001	-	3
GCN-FCN	300	16	0.001	-	-
GCN-ENC	300	12	0.00001	-	-
GCN-RNN*	300	256	0.001	8	-
GAT-FCN	300	16	0.001	-	3
GAT-ENC	300	12	0.00001	-	3
GAT-RNN*	300	256	0.001	8	3

For temporal data processing, we either use a CNN or RNN-based component as illustrated in Fig. 3. For training the architectures, we set an upper fixed budget of 300 epochs and use for all architectures the Adam optimizer [44]. We vary the batch size according to 2^n between 8 to 256. In principle, the usage of large batch sizes of 256 can be justified for most architectures. However, the FCN and ENC architectures exhibit unstable training behavior when doing so. We therefore can verify as previously suggested [32], that smaller batches of size 16 and 12 exhibit significantly better training performance for each. For the learning rate, we likewise can confirm the suggestion for the ENC architecture by [32]. For the number of attention heads in the GAT architectures, we consider K in the interval $[3, 8]$. We find that $K = 3$ proves to be a good choice. As the performance gain with more attention heads is at best only minimal and mostly accompanied with significant computational overhead. Noteworthy, the GAT architectures may also benefit from smaller learning rates. However, in the investigated range of $q \times 10^z$ for $q \in \{0.5, 1.0\}$ and $z = -5, \cdots, -3$, this gain was at best only minimal. At last, we tune the hyperparameter for the number of hidden nodes in the RNN architectures in the range of 4 to 128. We find that the usage of 8 hidden neurons exhibits the best performance. Note, that this seems to somewhat reflect the product of number of classes and node features $\#C \cdot F$. All final hyperparameters are listed in Tbl. I.

C. Statistical Performance Comparison

For performance comparison we reside in the following to the validation accuracy after 300 epochs under consideration of a 80-20 training to test data split and the estimated wall clock time in reference to a NVIDIA Tesla V100-SXM2-16GB provided as a cloud instance [45]. The latter is a necessity, as even though if a particular configuration may exhibit high training performance, it can be potentially rendered infeasible for practical applications by the produced computational overhead.

Because retrieved validation accuracies are subject to stochastic variation, we therefore reside in the following towards taking the median from in total 30 training runs. As however, most of the the performance values are distributed non-Gaussian, we further need to conduct additional statistical testing. Particularly, we reside in the following to what one may dub in analogy to [32] as a Kruskal-Wallis-Holm test, meaning that we first use the Kruskal-Wallis test to reject the null hypothesis, and subsequently perform tuplewise unpaired Wilcoxon rank-sum tests, for which we additionally use the Holm-Bonferroni correction.

To visualize the performance of the different architectures we

TABLE II
COMPARISON OF ALL TESTED ARCHITECTURES IN TERMS OF RANK (R) OF THE PERFORMANCE DATA SETS, MEAN VALIDATION ACCURACY AND ESTIMATED WALL CLOCK TIME (WCT) IN REFERENCE TO A CLOUD GPU INSTANCE CALCULATED FROM IN TOTAL 30 TRIALS EACH.

Architecture	WCT (20)	Accuracy (20)	$R_\Sigma(20)$	$R_{GCN}(20)$	Accuracy (10)	$R_{GCN}(10)$
GCN	≈ 0.78 min	68.84%	4.84	2.86	62.53%	4.97
GAT	≈ 130.00 min	63.56%	10.01	6.10	62.51%	4.84
GCN-FCN	≈ 20.00 min	65.67%	6.83	4.04	62.22%	5.20
GCN-ENC	≈ 21.36 min	63.95%	9.81	5.97	64.05%	2.95
GCN-sRNN	≈ 8.75 min	65.82%	6.85	4.08	62.86%	4.48
GCN-GRU	≈ 6.25 min	68.42%	4.77	2.88	64.33%	2.62
GCN-LSTM	≈ 6.25 min	67.96%	3.44	2.08	64.03%	2.95

TABLE III
PERFORMANCE VALUES FOR THE GAT ARCHITECTURES IN IN TERMS OF RANK (R), MEAN VALIDATION ACCURACY AND ESTIMATED WALL CLOCK TIME (WCT) FOR IN TOTAL 30 TRIALS EACH.

Architecture	WCT (20)	Accuracy (20)	$R_\Sigma(20)$	$R_{GAT}(20)$
GAT-FCN	≈ 2600 min	67.50%	4.03	2.56
GAT-ENC	≈ 2400 min	62.13%	11.38	6.66
GAT-sRNN	≈ 2775 min	65.38%	7.48	4.47
GAT-GRU	≈ 2750 min	67.00%	4.82	3.02
GAT-LSTM	≈ 2460 min	67.68%	3.74	2.40

use critical difference diagrams. Meaning that, architectures are ranked from 1 to M, where M is the total number of compared architectures, and statistically similar architectures are indicated by a connected line. To calculate the ranking $r(\mathcal{D})$ of a performance dataset $\mathcal{D}$ for each architecture, we derived

$$r(\mathcal{D}) = M - \frac{M-1}{|\mathcal{D}|(N-|\mathcal{D}|)|}\sum_{i=1}^{|\mathcal{D}|} W_i, \tag{9}$$

in which we perform the sum over the total number of observations of a performance dataset $|\mathcal{D}|$ with N being the total number of measurements from all performance data sets, further W_i being the number of observations a particular observation i in $\mathcal{D}$ dominates in the remaining $M-1$ datasets.

D. Discussion of Results

Recorded performance values and critical difference diagrams are given in Tbl. II & III and Fig. 4. First of all, the most obvious observation is that the GAT architectures create a large computational overhead with an increase in training time of up to $\approx$ 2800 mins in reference to the GPU instance. Note however, that due practical reasons we use for computations of the GAT architectures TPU instances instead. Thus reducing the overhead by at least 50-70%. Comparing the architectures with GCN and GAT as domain-level feature extractors altogether, we find that the performance increase in regards to mean validation accuracies is at best only minimal. Ranging for the GAT-FCN up to 1.83%. Overall, considering the large computational overhead and the minimal performance improvement, the usage of GAT-based architectures cannot to be justified. Thus, due to to their high practicality, we therefore take in the following our GCN architectures under closer scrutiny. We find that the GCN-GRU and GCN-LSTM exhibit highest performance in the 10 and 20 time-steps long datasets, followed up by either the GCN architecture for the long time-series or ENC architecture for the short-time series. Interestingly, the ENC architecture performs significantly worse on the long time-series dataset, while the FCN architecture performs slightly better. This is somewhat surprising, as we are dealing in both scenarios with comparably short time-series. Though, somewhat reflecting results from [32]. An interesting observation is also, that the GCN architecture under input concatenation significantly strives on longer time-series, while the input concatenation with GAT layer performs in comparison in both scenarios suboptimal.

E. Interpretation of the Learned Metadata Characteristics

A natural interest arising when dealing with learning tasks involving complex datasets is to interpret what a predictive method has actually learned. We therefore analyze obtained feature spaces and time-dependent characteristics. A representative feature space obtained using LDA and the GCN architecture under input concatenation is plotted in Fig. 5. Notably, the network learned to separate benchmark functions according to whether they have globally a strong quadratic funnel structure with $\sim x^2$ or not. While for instance, the Rastrigin function in Fig. 6 has the same funnel structure, it also features a strong periodicity superimposed on top of it. Thus, significantly distorting the behavior of the EA on a global level. Note, that this is unlike to the Griewank function, which has a less pronounced periodicity.

At last, we use the so called class activation map (CAM) for the GCN-FCN to peek into what parts of the time-series are making the most important contribution for predicting a certain class c. The CAM value for a class c is calculated with

$$\text{CAM}_c(t) = \sum_m w_m^c A_m(t), \tag{10}$$

where $A_m(t)$ is the output of the m-th filter at time-step t of the FCN architecture and w_m^c is the weight connecting the filter output with the output neuron z_c of the SoftMax classification layer. Note, that for this analysis we explicitly remove intermediate hidden layers of the classification head used in the prior experiments.

Overall, we find that the median CAM values in Fig. 5 reflect

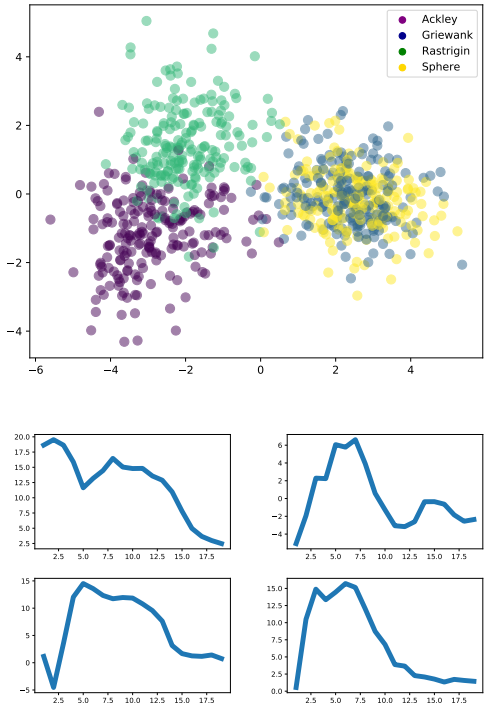

Fig. 5. Top panel: Visualization of learned features from the concatenated GCN architecture for the 20 time-steps data set. Bottom panel: Median CAM values learned for the different classes of benchmark functions for the 20 time-steps data set (Ackley, Griewank, Rastrigin & Sphere) from top left to bottom right.

slope characteristics of the different benchmarks. Notably, for the three functions with quadratic funnel structure $\sim x^2$, the CAM assigns a high importance to the interval between the 2nd and 8th time-step which reflects fast convergence. This phase is elongated to the 15th generation for the Rastrigin function, which features significant distortions on a global level. While for Griewank, which is globally similar to Sphere, the local distortions become only important around the 12th step due to the algorithm becoming stuck. Note, that for the Ackley function with a flat and concave funnel structure of $\sim 1-\exp(-|x|)$, the early regions of the time-series have higher importance.

V. Conclusions & Outlook

In conclusion, we found that among our proposed architectures GCN-GRUs and LSTMs demonstrate most consistent performance on variable time series lengths and high efficiency in terms of accuracy and training time. Further, we also demonstrated that the CAM for interpretable learning with time series data can help to understand as well as reflect global properties of the different problem-dependent search behaviors of the optimization algorithm. Note, that while we could have used different algorithms to this regard, we would expect them to behave vaguely similar on the chosen benchmarks.

For future studies, we suggest to consider alternatives to GAT for learning spatial anisotropies, as well as the development of a CAM for RNNs. The goal in a future study would also be to

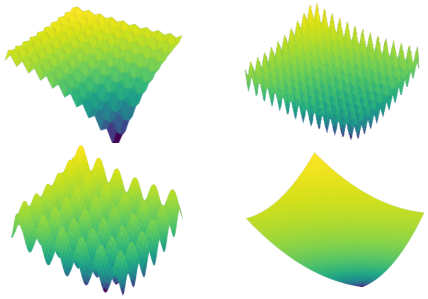

Fig. 6. Slope topologies of the different benchmarks with global optimum in the bottom corner of which we analyze the learned metadata characteristics by the FCN architecture of the problem-dependent search behavior with the CAM. From top left to bottom right: Ackley, Griewank, Rastrigin & Sphere.

explicitly predict an algorithm or configuration within an early classification scenario [46]. Though, noteworthy the flexibility of RNN architectures could also foster the integration of agent-environment models from reinforcement learning. To conclude our paper, we emphasize that essential convergence theorems of evolutionary optimization algorithms are formulated in terms of spatio-temporal relationships arising within the algorithm-problem interaction [47]. Thus, the learning of spatio-temporal relationships not only fosters insight into the problem-dependent behavior of optimization algorithms in a goal and reward-oriented manner [48], but can also be considered to be based upon theoretical first-principles. It is interesting to consider what it means to extrapolate this to scenarios concerning open-endedness [49] and dynamically changing objectives.

VI. 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 Research Institute of Trustworthy Autonomous Systems (RITAS), the Guangdong Provincial Key Laboratory (Grant No. 2020B121201001), the Program for Guangdong Introducing Innovative and Enterpreneurial Teams (Grant No. 2017ZT07X386), the Shenzhen Science and Technology Program (Grant No. KQTD2016112514355531).

References

[1] Nikolaus Hansen and Andreas Ostermeier. Completely derandomized self-adaptation in evolution strategies. *Evolutionary Computation*, 9(2):159–195, 2001.

[2] Nikolaus Hansen. The CMA Evolution Strategy: A Comparing Review. In I. Inza E. Bengoetxea J.A. Lozano, P. Larrañaga, editor, *Towards a New Evolutionary Computation*, pages 75–102. Springer, 2006.

[3] Nikolaus Hansen. The CMA Evolution Strategy: A Tutorial. *arXiv preprint arXiv:1604.00772*, 2016.

[4] Tobias Uller, Armin P Moczek, Richard A Watson, Paul M Brakefield, and Kevin N Laland. Developmental bias and evolution: A regulatory network perspective. *Genetics*, 209(4):949–966, 2018.

[5] Loizos Kounios, Jeff Clune, Kostas Kouvaris, Günter P Wagner, Mihaela Pavlicev, Daniel M Weinreich, and Richard A Watson. Resolving the paradox of evolvability with learning theory: How evolution learns to improve evolvability on rugged fitness landscapes. *arXiv preprint arXiv:1612.05955*, 2016.

[6] Tom M Mitchell. The need for biases in learning generalizations. Technical Report CBM-TR-117, Laboratory for Computer Science Research, Rutgers University, 1980.

[7] David H Wolpert and William G Macready. No free lunch theorems for optimization. *IEEE Transactions on Evolutionary Computation*, 1(1):67–82, 1997.

[8] Petar Veličković and Charles Blundell. Neural algorithmic reasoning. *Patterns*, 2(7):100273, 2021.

[9] Kate A Smith-Miles. Cross-disciplinary perspectives on meta-learning for algorithm selection. *ACM Computing Surveys (CSUR)*, 41(1):6, 2009.

[10] Sang Wan Lee, Shinsuke Shimojo, and John P O'Doherty. Neural computations underlying arbitration between model-based and model-free learning. *Neuron*, 81(3):687–699, 2014.

[11] Jane X Wang. Meta-learning in natural and artificial intelligence. *Current Opinion in Behavioral Sciences*, 38:90–95, 2021.

[12] Stephen Friess, Peter Tiňo, Zhao Xu, Stefan Menzel, Bernhard Sendhoff, and Xin Yao. Artificial neural networks as feature extractors in continuous evolutionary optimization. In *2021 International Joint Conference on Neural Networks (IJCNN)*, pages 1–9. IEEE, 2021.

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

[14] 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.

[15] Harsh Shrivastava, Xinshi Chen, Binghong Chen, Guanghui Lan, Srinvas Aluru, Han Liu, and Le Song. Glad: Learning sparse graph recovery. *arXiv preprint arXiv:1906.00271*, 2019.

[16] 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.

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

[18] 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.

[19] 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.

[20] 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.

[21] 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.

[22] Xavier Bresson and Thomas Laurent. The transformer network for the traveling salesman problem. *arXiv preprint arXiv:2103.03012*, 2021.

[23] Mohamad Alissa, Kevin Sim, and Emma Hart. A neural approach to generation of constructive heuristics. In *2021 IEEE Congress on Evolutionary Computation (CEC)*, pages 1147–1154. IEEE, 2021.

[24] Mikdam Turkey and Riccardo Poli. A model for analysing the collective dynamic behaviour and characterising the exploitation of population-based algorithms. *Evolutionary Computation*, 22(1):159–188, 2014.

[25] 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.

[26] 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.

[27] 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.

[28] 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.

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

[30] Petar Veličković, Guillem Cucurull, Arantxa Casanova, Adriana Romero, Pietro Lio, and Yoshua Bengio. Graph attention networks. *arXiv preprint arXiv:1710.10903*, 2017.

[31] Zhengzheng Xing, Jian Pei, and Eamonn Keogh. A brief survey on sequence classification. *ACM SIGKDD Explorations Newsletter*, 12(1):40–48, 2010.

[32] Hassan Ismail Fawaz, Germain Forestier, Jonathan Weber, Lhassane Idoumghar, and Pierre-Alain Muller. Deep learning for time series classification: a review. *Data Mining and Knowledge Discovery*, 33(4):917–963, 2019.

[33] Zhiguang Wang, Weizhong Yan, and Tim Oates. Time series classification from scratch with deep neural networks: A strong baseline. In *2017 International Joint Conference on Neural Networks (IJCNN)*, pages 1578–1585. IEEE, 2017.

[34] Joan Serrà, Santiago Pascual, and Alexandros Karatzoglou. Towards a universal neural network encoder for time series. In *CCIA*, pages 120–129, 2018.

[35] Tara N Sainath, Oriol Vinyals, Andrew Senior, and Haşim Sak. Convolutional, long short-term memory, fully connected deep neural networks. In *2015 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 4580–4584. IEEE, 2015.

[36] Jeffrey Donahue, Lisa Anne Hendricks, Sergio Guadarrama, Marcus Rohrbach, Subhashini Venugopalan, Kate Saenko, and Trevor Darrell. Long-term recurrent convolutional networks for visual recognition and description. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 2625–2634, 2015.

[37] Haipeng Xiao, Miguel Angel Sotelo, Yulin Ma, Bo Cao, Yuncheng Zhou, Youchun Xu, Rendong Wang, and Zhixiong Li. An improved LSTM model for behavior recognition of intelligent vehicles. *IEEE Access*, 8:101514–101527, 2020.

[38] Abdulkadir Tasdelen and Baha Sen. A hybrid CNN-LSTM model for pre-miRNA classification. *Scientific Reports*, 11(1):1–9, 2021.

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

[40] JD Bermúdez, P Achanccaray, ID Sanches, L Cue, P Happ, and RQ Feitosa. Evaluation of recurrent neural networks for crop recognition from multitemporal remote sensing images. In *Anais do XXVII Congresso Brasileiro de Cartografia*, pages 800–804, 2017.

[41] François Chollet et al. Keras. https://keras.io, 2015.

[42] Dan Simon. *Evolutionary optimization algorithms*. John Wiley & Sons, 2013.

[43] 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.

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

[45] Google Research. Colaboratory. https://colab.research.google.com/, 2022.

[46] Marc Rußwurm, Sébastien Lefèvre, Nicolas Courty, Rémi Emonet, Marco Körner, and Romain Tavenard. End-to-end learning for early classification of time series. *arXiv preprint arXiv:1901.10681*, 2019.

[47] Günter Rudolph. Finite markov chain results in evolutionary computation: A tour d'horizon. *Fundamenta Informaticae*, 35(1-4):67–89, 1998.

[48] David Silver, Satinder Singh, Doina Precup, and Richard S Sutton. Reward is enough. *Artificial Intelligence*, page 103535, 2021.

[49] Risto Miikkulainen and Stephanie Forrest. A biological perspective on evolutionary computation. *Nature Machine Intelligence*, 3(1):9–15, 2021.