ECOLE
Experience-based Computation:
Learning to Optimise

Project Number: 766186
Project Acronym: ECOLE
Project title: Experienced-based Computation: Learning to Optimise

Deliverable D3.5

Integrated software environment and manual

Authors:
Stephen Friess, Gan Ruan, Leandro L. Minku, Xin Yao – University of Birmingham
Giuseppe Serra, Zhao Xu– NEC Laboratories Europe

Project Coordinator: Professor Xin Yao, University of Birmingham
Beneficiaries: Universiteit Leiden, Honda Research Institute Europe, NEC Laboratories Europe

H2020 MSCA-ITN
Date of the report: 30.09.2021

Contents

Executive summary

This deliverable presents the integrated software environments related to the research invested and the scientific contributions made regarding the work package WP3 of the ECOLE project about Big Data Analytics and Optimization using Novel Machine Learning Approaches. The report aims to collect and explain the functionalities, implementation and deployment details of the software developed in WP3. This includes, but is not limited to, programming languages, platforms and packages used to implement the software, hardware requirements, example data sets used to verify the performance, and any other particular settings to compile and run the software. To introduce the functions in depth, we also explain the major technical advantages and methodology. The example industrial applications in automotive and telecommunication verticals, and performance evaluation with benchmark datasets are reported as well to give potential users a complete picture of the software. All the software packages and example datasets have been published and uploaded to the public repository of the ECOLE project.

Major Achievements

The table below summarizes the major achievements regarding WP3.5. We explore some challenging research questions about learning to optimize, and emerging industrial problems related to the area. Novel solutions and insightful visions are contributed, together with published software packages.

Research Questions	**Discussion**
Computational models, such as reinforcement and meta-learning that can serve as basis for experience-based algorithms, require the definition of states and ways to represent meta-knowledge of different problem domains and environments. What are possible ways in the literature to do so for optimization algorithms?	To acquire state representations one can identify available literature in algorithm behavior studies. Promising existing approaches characterize exploration and exploitation behavior of optimization algorithms and obtain low-dimensional representations of search spaces based upon which different search behaviors can be characterized.
What are the weaknesses of existing approaches based upon low-dimensional representations of search spaces to characterize different search behavior in the literature and how can they be improved?	Existing approaches are problematic as they fail to properly separate the search behavior for a fixed algorithm over different optimization problems. Further, they rely upon two dimensional representations using self-organized maps. Significant improvements can be achieved considering additionally a channel to aggregate fitness values and using graph structures which more accurately model the neighborhood relationships of different search space regions.

What are possible application scenarios of interests for structured state representations in meta-heuristic and population-based optimization scenarios?	Within the literature the algorithm selection problem has been framed as meta-learning. Further, within our work we gave an example of how this problem being alternatively framed as an algorithm configuration problem. We provided an illustrative application of car shape optimization.
After considering the weaknesses of the existing problem formulation, how to appropriately formulate the computing resource allocation problem in Open Radio Access Network (O-RAN)?	The proposed problem formulation is designed with the attempt to make the total traffic data in each cluster close to the computing resource capacity at each hour of a day, so that an optimal solution would not lead to delays in certain periods of the day while other periods may have resources over-provisioning. In addition, it equally considers the effect of the delay and computing resource utilization rate on the fitness value of solutions.
As an optimization algorithm with capability of global search, evolutionary algorithm is expected to provide better solutions to the emerging problem in network industry. How to tailor an evolutionary algorithm to solve the new optimization problem?	The proposed evolutionary algorithm mainly includes initialization, mutation operator and random cluster splitting. Initialization aims to randomly generate an initialized population with feasible solutions on the problem representation. Mutation operator is designed to produce feasible offspring solutions. Random cluster splitting aims to randomly select a cluster from a solution in the optimized population of the previous day and randomly split it into two clusters, so as to produce an initial population for the problem of the next day.
Whether the proposed evolutionary method achieves better solutions than a baseline algorithm?	Empirical studies have been carried out on real-world and synthetic datasets and have demonstrated the significant superiority of the proposed evolutionary algorithm over the recent baseline method.
What are the weaknesses of existing approaches for representation learning in terms of interpretability?	Most of existing embedding techniques learn representation vectors in the latent space, but neither the single numbers nor their dimensions have an interpretable meaning. Consequently, the main limitation

	of these techniques laid in the fact that they can only capture relations among items by using vectors that are black box for end users. In contrast, our approach is able to encode the textual information within the estimated quantities, providing vectors that are self-explainable.
Does integration of additional textual information improve interpretability of representation learning?	Our findings indicate that learning human-understandable vector representations from textual data improves the explainability of the analyzed items, strengthening trust in AI systems. Additionally, as previously mentioned, we are able to directly encode the textual information within the learned vector representations. In fact, each dimension of the vector corresponds to a word in the vocabulary and each value represents the probability that the corresponding word would explain the entity.
What are potential ways of utilizing the developed software packages for interpreting node embedding to application scenarios?	The presented approach can elicit user opinion on product characteristics from user generated data (texts and behavior). Therefore, the developed software packages can be integrated with different learning tasks and domains, as long as some textual information is available.

1 Introduction

In the Experience-based Computation: Learning to Optimize (ECOLE) project, we aim to address several industrial optimization challenges, including Optimization using Natural Computation, Multi-objective Optimization and System Engineering and Big Data Analytics. The project has been further divided into several subprojects, where each Early Stage Researcher (ESR) is responsible for each subproject. The research targets of ECOLE include shortening the product design cycle, optimization of the resource consumption for services and products, and facilitating creation of more balanced and innovative products towards user preference. Instead of just developing technologies to solve a given problem, it takes a bold step forward and proposes to use knowledge automatically across different problem domains. Referring to knowledge, skill, and practice derived from problem solving processes in time, the goal is to automatically learn and transfer the experience of optimizing one product or process for solving other optimization problems.

The deliverable D3.5 Integrated Software Environment and Manual presents the developed software tools in terms of representation learning, deep probabilistic models with user generated data and modules for proactive dynamic optimization within the scope of the work package WP3 Big Data Analytics and Optimization using Novel Machine Learning Approaches. The achievements reported in the deliverable are mainly based on the research works of ESR6, ESR7 and ESR8.

The first part of the deliverable (Section 2) will specifically introduce developed technologies theoretically to provide the potential users of the software tools with a complete picture of the technical advantages and the methodologies. In particular, Sec. 2.1 will present a pipeline developed by ESR6 that can be used e.g. for car shape optimization. The proposed method extracts structured data formats from continuous evolutionary optimization algorithms which can be fed to learning algorithms for feature and representation learning. Sec. 2.2 will report the technologies developed for resource allocation in Open Radio Networks by ESR 7, which proposes a novel proactive dynamic optimization method based on natural computation. And at last Sec 2.3 will introduce deep probabilistic models developed by ESR8 for user preference learning with applications to product related data sets.

The second part (Section 3) will provide user manuals for each of the developed software modules and packages. We visualize the structure of the software tools, the dependences, as well as the pipeline to run the tools. In addition, some example data is provided to help the potential users quickly test the developed software.

We conclude the deliverable with a summary of the developed software tools and provide an outlook by giving insights into potential applications of our developed software modules and packages. We also give suggestions to practitioners on possible ways of adapting and improving them for a variety of different scenarios of interest.

2 Methodology

2.1 Methodology to Harness Procedural Optimization Data

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. And while the application of evolutionary optimization to modern machine learning methods [1] has regained notable interest within the recent years [2] [3] [4] [5], the reverse direction has surprisingly been ignored. Even though within the study of natural computing methods, research on neural and evolutionary computation have historically been strongly intertwined [6] [7] [8], as the processing of data generated by evolutionary approaches can benefit from incorporating flexible predictive methods. This situation 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 neural network methods comes particularly helpful to this regard, as they mimic the feature learning capabilities of biological systems [9] [10], thus can be flexibly trained for different tasks, to learn representations which help them to efficiently process and differentiate given input data.

We therefore developed methods within our work to enable practitioners to directly leverage data generated by the optimization algorithms themselves during the optimization process. Particularly, by developing methodology for post-processing it into a structured data format, which subsequently allows the employment of deep learning methods to learn characteristics capable of differentiating different continuous optimization problems and algorithms. We hope that these methods can pave the way towards new approaches which allow practitioners to learn problem-dependent algorithm components and recall these from predictions of inputs generated during the run-time of an optimization algorithm. The work is largely based on the conference paper published at IJCNN 2021 [11] and the paper submitted to SSCI 2021 [12] (under review).

We will introduce in Sec. 2.1.1 the methodology to obtain structured data formats from optimization data using search space partitions and discuss it in the context of previous work on algorithm behavior studies. Subsequently, we demonstrate in Sec. 2.1.2 how a pipeline using the previously obtained structured data format can be constructed for representation learning. By this means, features can be learned in a latent space such that different optimization problems are sufficiently disentangled according to their different properties. At last, we demonstrate in Sec. 2.1.3 how this approach in principle be used in shape optimization (e.g. car shape optimization) as a way of predicting problem-specific algorithm components.

2.1.1 Conversion to Structured Data Formats

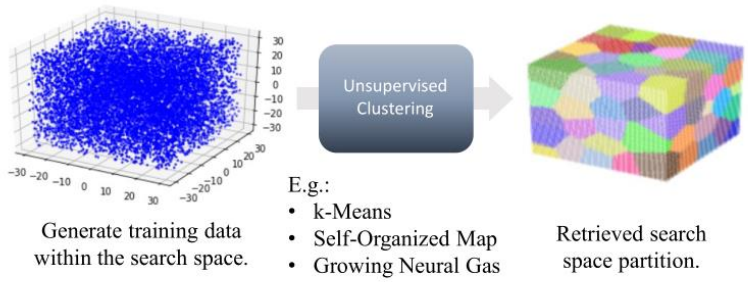


Figure 1 Visualization of the process for creating search space partitions.

In the following, we will elaborate the proposed methodology to obtain structured data formats from unstructured optimization data using methods to partition continuous search space. 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.

The principle approach is outlined in the top row of *Figure 1*. For a search space $\chi \subset R^d$, we first generate $D_T = \{x_1, \cdots, x_N\}$ samples uniformly random within the search space volume. This dataset D_T can then be subsequently used as training data for a 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 introduce in the following as clustering methods *k-means*, the *self-organized map* and the *growing neural gas*.

k-Means

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 $\{\mu_i\}_{i=1}^{N_C}$ 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 $\mu_i = (\Sigma_n r_{nk} \cdot x_n)/(\Sigma_n r_{nk})$, where $r_{nk} = 1$ if the closest μ_j for given x_n has j=k, otherwise it is *0*. The cluster centroids $\{\mu_i\}_{i=1}^{N_C}$ retrieved on the basis of the k-means algorithm can be further re-interpreted as nodes of a graph structure. To construct a graph, one simply considers for each cluster with centroid μ_i its associated decision volume *V(i)*, finds all neighboring volumes *V(j)* and subsequently builds an adjacency matrix $\boldsymbol{A}$, with a_{ij}=*1* for neighboring pairs and a_{ij}=*0* otherwise. This procedure is known as Delaunay triangulation.

Self-Organized Map

Structured clusters can be retrieved using the self-organized map (SOM) technique [13]. As mentioned, this approach has been used in prior work [14] **,** [15] as a way to partition the search space, originally motivated by work on modeling solution populations [16] 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 [13], 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 $\boldsymbol{x}$ by $\boldsymbol{\mu}_i(t+1) = \boldsymbol{\mu}_i(t) + h_{ci}(t)[\boldsymbol{x}(t) - \boldsymbol{\mu}_i(t)]$, where *c* is the index of the best matching unit (BMU), i.e. likewise to the k-means algorithm $c = arg\ min_i(||\boldsymbol{x}(t) - \boldsymbol{\mu}_i(t)||)$, and *i* being the index of its topological neighbors. Here, h_{ci} is a neighborhood function with $h_{ci}(t) = \alpha(t)\ \exp(-||\boldsymbol{\mu_c} - \boldsymbol{\mu_j}||^2/2\sigma^2(t))$, where $\sigma(t)$ and $\alpha(t)$ are monotonically decreasing functions of *t*. For the former, according to literature [13] 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 Figure 2) .

The Growing Neural Gas

The growing neural gas (GNG) [17] can be considered as a variation of the former SOM [13]. 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 μ_i being randomly initialized on the dataset D_T. Based upon a randomly drawn data point $\boldsymbol{x} \in D_T$, the nearest cluster $\boldsymbol{\mu}_1$ and second-nearest cluster $\boldsymbol{\mu}_2$ are determined. If the cluster μ_1 has edges, the ages of the edges are incremented and an error variable $\Delta error(1) = ||\boldsymbol{\mu_1} - \boldsymbol{x}||^2$ is calculated. The cluster $\boldsymbol{\mu_1}$ and its topological neighbors $\boldsymbol{\mu_n}$ are subsequently moved towards the drawn data point $\boldsymbol{x}$ by fractions ϵ_b and ϵ_n with $\Delta \boldsymbol{w}_{s_1} = \epsilon_b(\boldsymbol{x} - \boldsymbol{\mu}_1)$ and $\Delta \boldsymbol{w}_{s_n} = \epsilon_n(\boldsymbol{x} - \boldsymbol{\mu_n})$ 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 μ_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 μ_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.

Technical Requirements

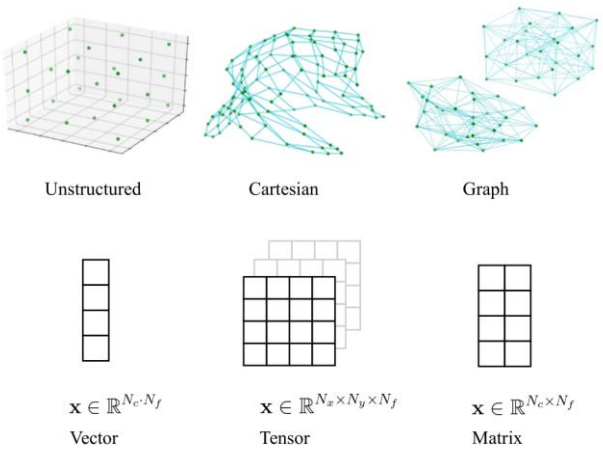


Figure 2 Different structured data formats that can be obtained using search space partitions.

To obtain structured data formats (c.f. Figure 2) using the previously search space partition methods, a variety of existing techniques can be used. As the k-means algorithm is usually fairly well known (e.g. [18]**)**, we used a standard library implementation based upon the Scikit-Learn software package [19]**.** Constructing Delaunay graphs of the retrieved partitions is less trivial. However, the SciPy software package can be used for this purpose [20]. At last, optimized implementations of the self-organized map and growing neural gas are contained within the NeuPy library [21]. Implementations of the bespoken library methods are well supplied within the code accompanying our report.

Table 1 List of major Python package used within this work.

Python Package	Major Purpose
Tensorflow	Library for constructing neural network architectures.
Keras	Python interface for the usage of Tensorflow.
DEAP	Provides different implementations of EAs.
SciPy	Efficient calculations of distances and of Voronoi graphs.
Scikit-Learn	Implements the k-Means clustering.
NeuPy	Implements SOM and GNG.

In the following, we provide additional insight into possible applications for structured data formats. With our first example being feature extraction and analysis using i.a. a custom Keras-based [22] implementations of graph convolutions based upon a TensorFlow [23] backend in combination with available implementations for graph coarsening operations [24]. And in the second example considering the scenario of predicting an operator in a shape optimization scenario using a freely available model from the ShapeNet library [25]. The optimization algorithms ES

and CMA-ES we use within our experiments are based upon readily available implementations from the DEAP library [26].

2.1.2 Feature Extraction and Analysis

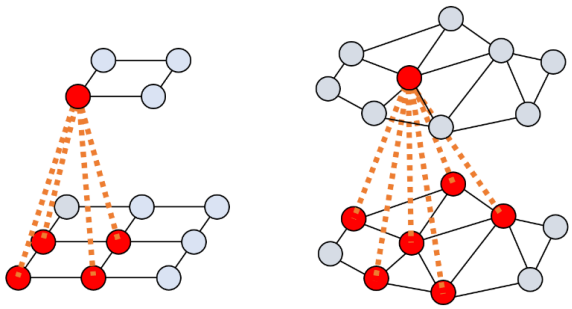


Figure 3 In traditional convolution layers (left side), filters have a smaller dimension than the given input domain, and can aggregate features from patches of pre-defined arbitrary size from the input. In Kipf & Welling's graph convolution [27], filters have the same dimension as the input graph, and only aggregate features from the direct neighborhood of a given node.

In the following, we follow-up an approach originally proposed in algorithm behavior studies [14], [15] for extracting features from procedural optimization data. Specifically, we demonstrate how a structured data format can aid to this regard. We specifically compare in the following the performances of different ways of processing different data formats in regards to neural network architectures which are chosen adequately to reflect the properties of the different data formats, as well as using different channels such as a solution channel (S), fitness channel (F) and combined channel (S+F). Note that a special emphasis within our study lies on investigating whether the specialized architectures for processing graph data **(**c.f. Figure 3**)** can demonstrate performance improvements. Note that we specifically rely upon the data post-processing pipeline illustrated in Figure 3 And as a structured data format we explicitly use finite differences between the initial and successor generation $\boldsymbol{\Delta z_g^r = z_0^r - z_1^r}$ to train the networks. Note that this finite difference interpretation may also establish a conceptual bridge to fitness landscape analytic techniques.

We train the networks upon data generated from a symmetric function set. For the combined solution and fitness channel (S+F), we find that all networks exhibit stable training performance and are capable of achieving high accuracies in the range of ~80-90%. Achieved values for the networks are listed in Table 2. Note that within prior available work [14], [15], different search spaces are used and results are only discussed qualitatively. Thus, these do not enable a direct quantitative comparison. 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 ~96%. Followed up by the CNN and MLP with about ~84%. Looking at the obtained feature spaces in Figure 5 in Column I & II, all compared methods exhibit clearly a high ability to separate 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.

*Figure 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 (**x**, f(**x**)) of candidate solutions and fitness values are converted by a search space partitioning method into a structured data format **z**, which subsequently can be fed to an adequate neural network architecture for feature extraction.*

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 Table 2. 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 ~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 ~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 ~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 in learning more accurate relationships between neighboring search space regions. However, it can still be suspected that the solution channel can demonstrate higher efficiency under shifts of the benchmark functions. At last, we test the behavior of our approach in regards to normalizing the benchmark functions, fitness values and its behavior on asymmetric functions (c.f. Column III & IV in Figure 5)**)**. Normalizing the fitness values such that for every benchmark function $0 \leq f(\boldsymbol{x}) \leq 1$, 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 column III of Figure 5). Particularly, clusters are separated into exponential$\sim 1 - exp(-|\boldsymbol{x}|)$ and quadratic ~x^2 funnel structure.

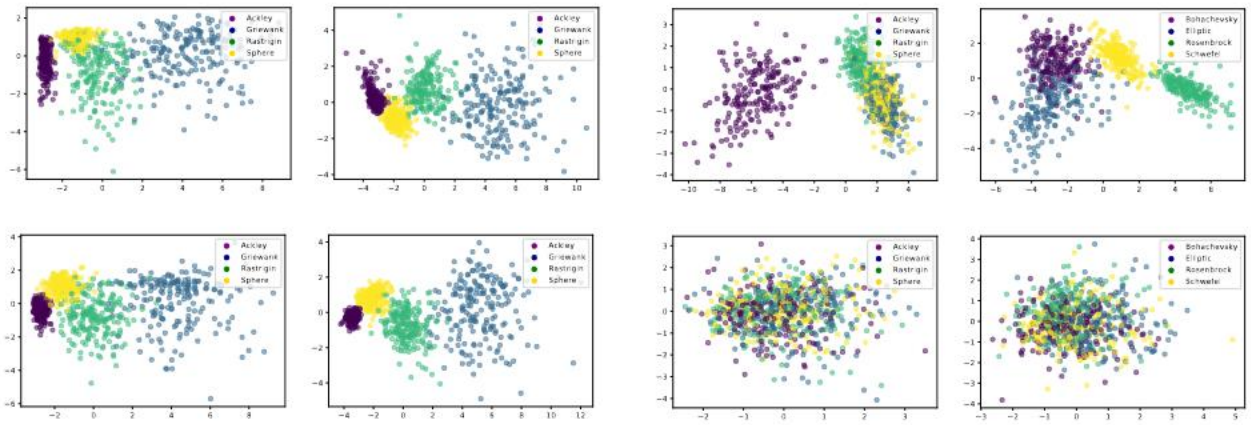


Figure 5 Column I and II: LDA-plots of the feature spaces obtained on a symmetric function set for the MLP, the CNN and GNN (GNG & Delaunay) (from left to right and top to bottom). Column III & IV: LDA-plots of the feature spaces for the GNG-based trained GNN using rescaled search space sizes and normalized fitness values on the symmetric (left) and asymmetric function set (right). Lower row. Obtained feature spaces for training on the solution channel (S).

Table 2 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

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 an asymmetric function set. 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. However, in comparison to the symmetric function set, 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.

2.1.3 Application for the Prediction of Operator Configurations

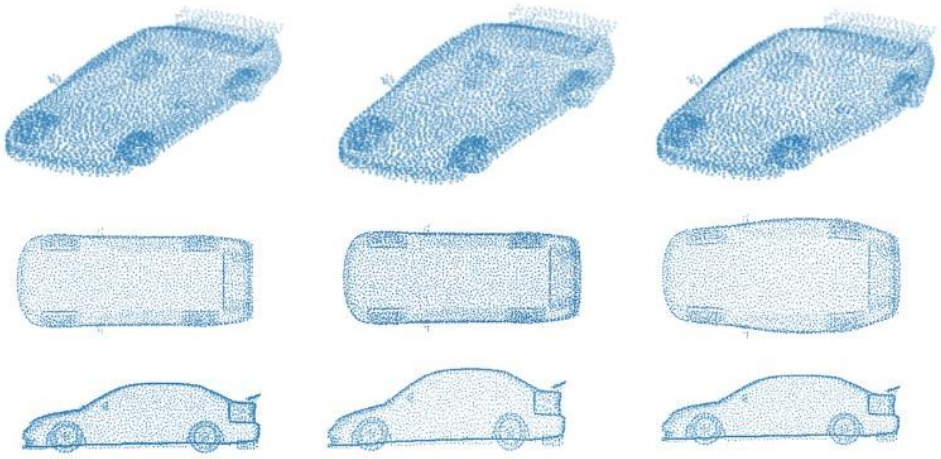


Figure 6 Column 1: Original source shape which is used within our studies to generate target shapes. Column 2-3: Two different target shapes used within our study generated by either increasing the height (column 2) or width of the cabin volume (column 3).

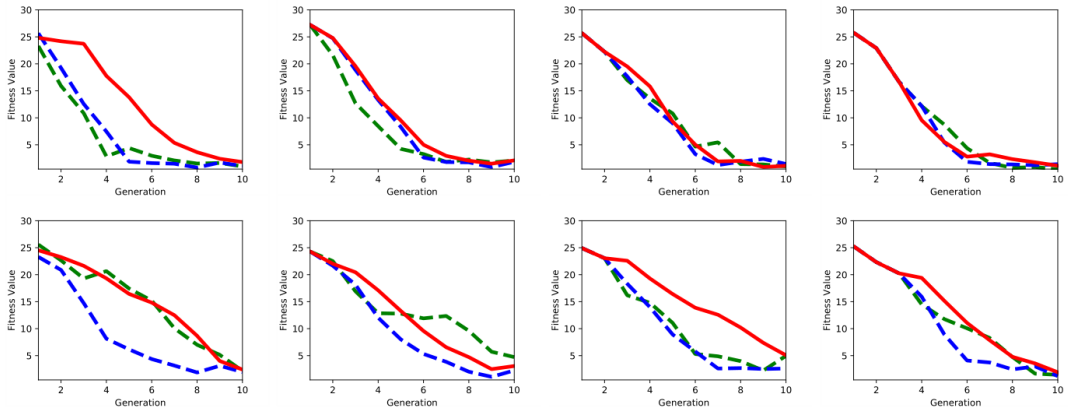


*Figure 7 Retrieved median fitness curves over 10 runs for the operator configuration prediction scenario. From left to right: With generational offset from -3 to 0. From top to bottom: For the first target shape from **Error! Reference source not found.**. in column 2 (top) and the second target shape in column 3 (bottom) from **Error! Reference source not found.**. Red continuous curves: Default runs. Dashed curves: For predicted configurations, with full configuration (σ, **C**) (green), and only covariance matrix **C** (blue).*

In the following, we additionally provide insight into how structured data formats can be harnessed for the prediction of inductive biases in the form of operator configurations in a shape optimization scenario. We specifically consider in the following a scenario where car shapes are optimized, by increasing the cabin volume along different dimensions. We construct the first one by constructing a target shape which requires the increase of the design variable for the height dimension. And the

second one, by constructing a target shape which requires an increase in the width dimension. The goal is then to optimize the source shape in column I such that it matches either the target shape in column II or III, using as objective function the modified Hausdorff distance [28]

$$d_H(X,Y) = \max\left\{ \sum_{x\in X} \min_{y\in Y} d(x,y), \sum_{y\in Y} \min_{x\in X} d(x,y) \right\}.$$

This measure calculates the similarity between the source shape *X* and a target shape *Y*. To model in the following the inductive bias we select the operator configuration which created the highest decrease in fitness. We collect in the following data from 200 runs where one half of it are from the first target shape problem and the other are from the second, and subsequently use it to train a network to directly predict these operator configurations. Explicitly, by converting procedural optimization data into a structured format, based upon the methodology described in our previous sections. We predict in the following the algorithm configuration as $(\sigma, \boldsymbol{C})$, meaning in terms of step-size and covariance matrix accordingly by means of training a multilayer perceptron to approximate the regression function $f(\Delta \boldsymbol{z}) = (\sigma, \boldsymbol{C})$, where we use the finite difference $\Delta \boldsymbol{z}^r = \boldsymbol{z}_0^r - \boldsymbol{z}_t^r$, with offset *t=2* for increased effectiveness. The input size of the multilayer perceptron is 100, and we use two hidden layers with 10 and 200 neurons with ReLU activation functions, such that to sufficiently perform well in the high-dimensional regression task. For the output layer, we choose $1 + d(d+1)/2$ neurons with a mix of linear and ReLU activation function to adequately model the output. However, we spare in the following a more detailed discussion and instead refer to our paper. After having jointly constructed predictor and bias, we can conduct our experiments in the following. We therefore predict operator configurations as inductive biases in a problem-specific manner during run-time.

Our results are plotted in Figure 7, where in the top row we display median results over 10 predictions on the first target shape problem, and the bottom row the predictions on the second target shape problem. We further consider scenarios in which we either predict the full configuration $(\sigma, \boldsymbol{C})$ (dashed green) or just the covariance matrix **C** (dashed blue), as well as reset the CMA-ES with the predicted configurations at different generational offset from -3 to 0 (left to right). Overall, modeling the bias component by predicting the full operator configuration, i.e. the covariance matrix **C** and step-size σ, leads to a result which is partly performing worse in comparison to the baseline (red continuous) due to overshooting. More consistent performance increases can be realized using just the predicted covariance **C** component. This effect can be reasonably explained, as a high step-size prediction can in principle amplify the effects of wrongly predicted covariance matrix **C**. Thus, only focusing on the latter allows the adaptive properties of the CMA procedure to correct any wrong prediction during run-time. The effect of using the predicted operator configuration is notably most pronounced when doing so in in the earlier generations. However, notably this additional performance gain realized through resetting the CMA-ES is partly negated by the expended function evaluations required to generate a prediction first.

2.2 Evolutionary Optimization for Proactive and Dynamic Computing Resource Allocation in Open Radio Access Network (O-RAN)

This section introduces the technologies developed for the proactive dynamic optimization with application to resource allocation in Open Radio Access Network (O-RAN) [29]. O-RAN is a recently proposed wireless network architecture to satisfy various demands in the wireless system of next generations [29]. Because of the openness of O-RAN [30], all kind of operators and vendors are able to switch their equipment into the network service, causing large and variable loads of traffic data. Therefore, intelligent techniques are needed to automate resource allocation in O-RAN, so as to efficiently exploit the finite computing resources and increase the service quality of the network operators as well as decrease the energy cost of the RAN domain. In order to solve this resource allocation problem, the literature has made some investigations. Some existing methods formulated the problem as a clustering task, e.g. [31]. However, the method is unsuitable as it defines the capacity utility of resource in an inappropriate way and it tends to cause much delay to the network [31]. Considering those limitations, a new objective function that better formulates the problem is proposed. Moreover, the existing method is mainly based on greedy search, which is not ideal as it could get stuck into local optima of the problem. Therefore, an evolutionary algorithm (EA) based optimization method is proposed to solve the new objective function, finding an optimal resource allocation scheme to proactively and dynamically deploy the computing resource for processing upcoming traffic data. Since the computing resource needs to be proactively deployed, a multivariate long short-term memory model is used in the proposed framework to predict future traffic data for a forward looking deployment scheme.

2.2.1 Proposed Formulation of the Computing Resource Allocation Problem

This section presents the proposed formulation of the computing resource allocation problem after considering the limitations of the existing problem formulation [31] that are stated as follows. The peak distribution in the formulation is redundant as optimizing the defined problem with the peak distribution does not directly optimizing the three objectives. Besides, the existing problem formulation is defined as making the averaged total traffic data in each cluster of 24 hours close to the computing resource capacity when it is optimized. Moreover, when optimizing the existing formulation, solutions with more delay in the network have more chance to survive than those with less computing unit capacity utility.

Suppose there are *N* points, $R = (r_1, ..., r_i, ..., r_N)$; each point r_i has a fixed position (r_i^1, r_i^2) (where r_i^1 and r_i^2 are the longitude and latitude, respectively) and data $\boldsymbol{f} = \left(f_0(r_i), ..., f_h(r_i), ..., f_{23}(r_i) \right)^T$, where $f_h(r_i)$ is sum of data of r_i from *h*-th to *(h+1)*-th hour at the current day.

At the end of each day, the clustering scheme of the next day needs to be found to deploy the BBUs to the RRHs in the network. Considering this background, the aim is to proactively cluster these N points to K clusters to achieve the optimization objective. Therefore, this problem needs to firstly predict the traffic data of all points and then dynamically find an optimal solution through an optimization algorithm based on the predicted traffic data, such that BBU capacity utility is

maximal, the delay in the network and the required number of BBUs is minimal. This problem can be also regarded as a time series clustering problem.

Let $\boldsymbol{X} = (x_1, ..., x_i, ..., x_N)$ be a vector such that $x_i = l$ means that the *i*-th point is in the *l*-th cluster and x_i is an integer ($1 \le x_i \le K$). At least one x_i is equal to any value of $[1, K]$. The objective function is presented as follows:

$$\begin{cases} \min\ F(X) = \omega * K + \dfrac{1}{K}\sum_{k=1}^{K} U(C_k) \\ s.t.\ dist(r_u, r_v) \le \tau\ \ (\forall r_u, r_v \in C_k) \end{cases} \tag{1}$$

where $K = \max(x_i)\ (i = 1, 2, ..., N)$ is the number of clusters; $\omega \in (0,1]$ is a parameter that controls the weight of K and $\dfrac{1}{K}\sum_{k=1}^{K} U(C_k)$; $dist(r_u, r_v)$ is the distance of any two points r_u and r_v in *k*-th cluster C_k; τ is a threshold controlling the distance of neighboring points; $U(C_k)$ is the difference between the sum data of points and 1 in the *k*-th cluster, which is defined as:

$$U(C_k) = \frac{1}{24} * \sum_{h=0}^{23} |\ f_h(C_k) - 1| \tag{2}$$

where $f_h(C_k)$ is the sum data of all points in *k*-th cluster, which is defined as:

$$f_h(C_k) = \sum_{r_{m \in C_k}} f_h(r_m) \tag{3}$$

where C_k is the set of all point in the *k*-th cluster, which is presented as follows

$$C_k = \{r_m \mid r_m \in C_k\} = \{r_m \mid x_m = k\} \tag{4}$$

2.2.2 Overview of the Proposed Evolutionary Algorithm for the Resource Allocation Problem

Due to brilliant capability of global search, evolutionary algorithms have been successfully applied to a variety of resource allocation problems [32] [33]. However, the existing methods are not applicable to the emerging optimization problem in the O-RAN area, since the problem formulation in this paper is different from that in the literature. More specifically, solution representation, genetic operators to generate offspring solutions and constraints handling mechanisms in existing evolutionary algorithms are not appropriate for the problem formulation in this paper. To this end, we propose an evolutionary approach, tailored for solving the new problem formulation.

This section introduces the framework of the proposed evolutionary algorithm, which is exhibited in Algorithm 1. Given a set of N points with their position coordinate in the network, the algorithm firstly calculates the distance of any two points to fill the adjacent matrix. Then, the initialization process of parameters is conducted. The initial values of the parameters *gencount* and d are set as 1 and *startdate*, respectively. After that, the initialization process of the algorithm randomly generates a population P with a set of feasible solutions. Then, allocate the computing resources

to all points day to day from the start date *startdate* to the end date *enddate*. Within the outer loop of the while, the first step is to input the traffic data of all points on the current day. Then, leverage a prediction model to forecast the traffic data of all points on the next day. The input of the prediction model is the traffic data of all points on the current day. Note that, in reality, the traffic data of all points is only completely collected and therefore available at the end of each day. Next step is to calculate the fitness value of all solutions in the initial population P for this day on the predicted traffic data using the equation (5).

Algorithm 1. **Framework of the proposed evolutionary algorithm**

Input: A set of *N* points $R = (r_1, ..., r_i, ..., r_N)$ with their position coordinate neighborhood threshold τ; population size *popsize*; maximal number of generations *maxgen*; start date *startdate* and end date *enddate*.

Output: The found best solutions x^* for days from *startdate* + 1 to *enddate* + 1. $\mathbf{P}_{init}$.

1. Calculate the distance of any two points $dist_{i,j}$ to fill the distance matrix $distM = dist_{i,j}$;
2. Initialize parameters: set the count of generation *gencount* = 1; set the count of days *d* = *startdate*;
3. **InitialPop()**: randomly generate a population P with a set of *popsize* feasible solutions;
4. **while** $d \le enddate$ **do**
5. Input the traffic data of each point $\boldsymbol{f}_i^d = \left(f_0^d(r_i), ..., f_h^d(r_i), ..., f_{23}^d(r_i) \right)^T$ on the current day, where $f_h^d(r_i)$ is sum of data of r_i from *h*-th to *(h + 1)*-th hour at the current day;
6. Utilize a forecasting model to predict the traffic data of each point on the next day ((d + 1)−th) $\boldsymbol{fp}_i^{d+1} = \left(f_0^{d+1}(r_i), ..., f_h^{d+1}(r_i), ..., f_{23}^{d+1}(r_i) \right)^T$ based on the current day's (*d*−th) traffic data f_i^d;
7. Calculate the fitness value of each solution in population **P** based on the predicted traffic data using the equation (1);
8. **while** $gencount \le maxgen$ **do**
9. Generate an offspring population P_{off} using the mutation operator based on the parent population **P**: **Mutation**(**P**);
10. Calculate the fitness value of each solution in population P_{off} based on the predicted traffic data $\boldsymbol{fp}_i^{d+1}$ using the equation (1);
11. Combine two populations P and P_{off} and select the top *popsize* solutions as the P for the next iteration;
12. *gencount* = *gencount* + 1
13. **end**
14. Select x_{d+1} with the smallest fitness value from P;
15. **RandomClusterSplitting**(P): conduct random cluster splitting on P to produce a new population as the initial population P for the next day;
16. d=d+1;
17. **end**
18. **Return** $x^* = (x_{startdatae+1}, ..., x_{enddatae+1})$

Afterwards, the algorithm iterates until the parameter *gencount* is no larger than the pre-setting maximal number of generations. At each generation, the proposed mutation operator is used to produce an offspring population P_{off} with feasible solutions based on the parent population P.

Solutions in the offspring population are also evaluated using the fitness function in equation (1). After that, the parent population P and the offspring population P_{off} are combined together. Then, the top *popsize* solutions with the smallest fitness values are selected from the combined population as the P for the next generation. After the iteration, a solution with the smallest fitness value is selected from P as the found best solution for the resource allocation problem at the d + 1-th day. Then, conduct the process of random cluster splitting on P to produce a new population as the initial population P for the next day. At the end of the iteration of all days, output the found best solutions for days from *startdate* + 1 to *enddate* + 1.

2.2.3 Description of Each Step of the Algorithm

We now introduce the details of three main steps in the proposed EA framework:

- Initialization process of the population: InitialPop(),
- Mutation operator to produce feasible offspring solutions: Mutation(P)
- Random cluster splitting to produce the initial population: RandomClusterSplitting(P).

These steps are specifically designed tailored for solving the resource allocation problem. Considering that the constraint of this problem is very special, to make the algorithm concise, the constrain handling strategy is not developed. Alternatively, those three procedures are just designed to produce feasible solutions considering the constraint.

Population initialization. The detailed procedures of the population initialization are stated in Algorithm 2, which aims to produce a population with *popsize* feasible solutions. The basic idea is to randomly group a set of random number of neighboring points together to form a feasible solution. The specific steps for producing each feasible solution are as follows. Firstly, a point r is randomly selected from the set of points. Then, find all neighboring points of r to form a neighboring set of this point *CloseSet*. After that, randomly pick several points with the number of no larger than Nc to cluster them to point r, where Nc is the size of *CloseSet*. Repeat these steps until all points in the set of all points are clustered.

Algorithm 2. **InitialPop(): procedures of the population initialization.**

Input: A set of *N* points $R = (r_1, ..., r_i, ..., r_N)$ with their position coordinate neighborhood threshold τ; population size *popsize*; distance matrix *disM*; neighborhood threshold τ ;

Output: The initialized population $\mathbf{P}$.

1. **for** $j := 1$ *to popsize* **do**
2. set the cluster count *clucount* as 0;
3. **while** all points are clustered **do**
4. Randomly select a point r from the set of points ;
5. Find all points from the rest points to get a set *CloseSet*, in which the distance of any point to r is smaller than τ, the size of the set *CloseSet* is Nc;
6. Randomly pick points from *CloseSet* with the number of less than or equal to Nc and group them with r as a cluster;
7. Set the value of those clustered points as *clucount*: x_k=*clucount*, where k is the index of those points grouped into the same cluster;
8. *clucount* = *clucount* + 1;
9. **end**

10. Get the j−th solution $P_j = (x_1^j, ..., x_N^j)$;
11. **end**
12. **Return P**

Mutation operator. As the only genetic operator in the proposed evolutionary algorithm, the proposed mutation operator combines the role of both crossover and mutation operator in normal evolutionary algorithms. How the deigned mutation operator achieves this goal is presented in Algorithm 3. Given the parent population P, the mutation operator produces an offspring population P_{off} with *popsize* feasible solutions. For each based on each solution of P, the operator firstly selects the isolated point as the set of *isoPoint*, in which each point is formed as one isolated cluster. Then, randomly generate a random number *randNum* between 0 and 1. If *randNum* is smaller than the pre-setting probability prob and the set *isoPoint* is not empty, randomly select an isolated point x with the index k from the set *isoPoint*; else, randomly select a point from all points. Here, prob controls the weight of decreasing the number of clusters and increasing the diversity of the clustering scheme. For example, if prob is large, more isolated points could be given more priority to be grouped to its adjacent clusters.

Afterwards, the mutation process is conducted on the selected point x. Firstly, find all adjacent clusters of x as *mutClusters*, in which all points have the distance to point x smaller than or equal to τ. If the size of the set *mutClusters* is larger than 1, just group x into one of the random cluster in the *mutClusters* to decrease the number of clusters; else, randomly select an adjacent cluster C of x and put those points whose distance to x is smaller than or equal to τ in the set *Cclose* with the number of Num. After that, randomly select several points from *Cclose* with the number of less than Num and group them and x into a new cluster. Following this way, feasible solutions in the offspring population P_{off} can be produced. In the second case, a new cluster is produced, which results in more cluster while increase the diversity of the clustering. Therefore, solutions with more resource capacity utility and less delay might be searched.

Algorithm 3. **Mutation(P): procedures of the mutation operator.**

Input: The parent population P distance matrix *disM*; neighborhood threshold τ; the probability of preferentially clustering isolated points prob.
Output: The offspring population P_{off}.

13. **for** $j := 1$ *to popsize* **do**
14. Select the isolated points as the set *isoPoint* from Pj, each of which solely forms a cluster;
15. Random generate a number *randNum* between 0 and 1: *randNum* = rand(0; 1) ;
16. **if** *randNum* < prob and *isoPoint* is not null **then**
17. Randomly select an isolated point x with the index k from the set *isoPoint*;
18. **else**
19. Randomly select a point from all points;
20. **end**
21. Find all adjacent clusters of x as *mutClusters*, in which all points have the distance to point x smaller than or equal to τ;
22. **if** size of the set *mutClusters* is larger than 1 **then**
23. Group x into a random cluster *randCluster* in *mutClusters* and set the value of x_k as the cluster number of *randCluster*: x_k = *randCluster* ;

24. **else**
25. Randomly select an adjacent cluster C and put those points whose distance to x is smaller than or equal to τ in the set *Cclose* with the number of *Num*;
26. Randomly select several points from *Cclose* with the number of less than *Num* and group them and x into a new cluster;
27. Set the value of those points in the new cluster as $\max(x_i)+1,(x=1,...,N)$
28. **end**
29. Get the j−th mutated solution $P_j=(x_1^j,...,x_N^j)$;
30. **end**
31. **Return P_{off}**

Random Cluster Splitting. Given that problems at different days have the same position information of all points, solutions found for the previous day might be useful for the problem with the next day. Bearing this in mind, we propose to conduct a random cluster splitting process on the found solutions of the problem at the previous day, so as to generate a novel population as the initial population for the optimization of the problem at the next day. The main idea is to split a randomly selected cluster into two clusters, in which each cluster has the random number of points. The specific procedures of the random cluster splitting is described in Algorithm 4. For each solution in the population P, conduct the following random cluster splitting on it. Firstly, randomly select a cluster C with more than one point as the cluster to be split. Then, calculate the number of points in the cluster C: NR. Next, randomly generate a number between 1 and $\lfloor NR/2 \rfloor$: $Nsplit=rand\left(1,\lfloor NR/2 \rfloor\right)$. After that, split the cluster C into two clusters with size of *Nsplit* and *NR − Nsplit,* respectively. Lastly, Set the value of those points in the cluster with the size of *NR − Nsplit* as $\max(x_i)+1,(x=1,...,N)$. Through this random cluster splitting process, the produced solutions can decrease the number of clusters by 1 and therefore maintain most clustering structure. At the same time, the diversity of the clustering structure can be increased through splitting the cluster, which might generate better solutions for the problem at the next day.

Algorithm 4. **Mutation(P): procedures of the mutation operator.**

Input: The parent population P distance matrix disM; neighborhood threshold τ; the probability of preferentially clustering isolated points prob.

Output: The offspring population P_{off}.

1. **for** $j:=1$ *to popsize* **do**
2. Randomly select a cluster C with more than one point;
3. Calculate the number of points in the cluster C: NR;
4. Randomly generate a number between 1 and $\lfloor NR/2 \rfloor$: $Nsplit=rand\left(1,\lfloor NR/2 \rfloor\right)$;
5. Split the cluster C into two clusters with size of Nsplit and NR − Nsplit, respectively;
6. Set the value of those points in the cluster with the size of NR − Nsplit as $\max(x_i)+1,(x=1,...,N)$
7. Get the j−th solution $P_j=(x_1^j,...,x_N^j)$
8. **end**
9. **Return P**

2.3 Representation Learning with User-Generated Data

User preference is critical in product design. To automatically detect customer opinions on aspects and features of products, we propose a novel method, which combines the strengths of statistical machine learning and deep neural networks, to analyze user generated data, including texts and behavior. This method will facilitate the manufacturers to perform user-centric product design, and thus increase satisfactory of their customers. The achieved results leads to the publication Interpreting Node Embedding with Text-labeled Graphs [34]. The following sections introduce the technical part and theoretical advantages of the proposed method.

2.3.1 Improving Node Embedding Interpretability

In recent years, with the advent of more powerful and capable machines, researchers have started focusing more and more on developing (deep) neural architectures for a wide range of applications. The success of neural-based approaches in different domains, as language modeling or computer vision, led these models to become the default choice for a wide range of applications. However, despite the impressive performances, some drawbacks are notably serious. First, many works have proved how easy is to fool a deep learning model by altering the input data with some perturbation. Experiments show that some changes in an image or a text would lead the model to completely miss the right label with high confidence. Second, deep learning approaches are generally considered as black-boxes. Given the obscure nature of neural-based methods, the latter are not able to deliver interpretable results and do not have a strong theoretical foundation. This is probably the best-known lack in this area and, depending on the domain, this problem can be crucial. For example, in a medical application, how can we trust a diagnosis suggested by an algorithm with high accuracy but without a complete understanding of the output? In automotive design, how can a manufactory invest in a new car design if we cannot investigate the reasons behind a given algorithmic decision? In business scenarios, how can a company make some possible profitable decision without understanding if the proposed move is the right one or not? Both limitations are particularly serious when dealing with real-world applications and decisions that could have an economic or social impact. Consequently, the wide use of machine learning methods in industrial, medical, and socio-economical applications is requiring the research community to provide explanations about the results. For this reason, interpretable AI has been increasingly receiving more and more attention across different scientific disciplines and industry sectors.

In this context, user-generated data (e.g. reviews, social posts, body measurements) could play a primary role to fill the gap between humans and AI. In addition, they may be used to discover market trends, user preferences and to improve market strategies and productivity. For example, for recommendation tasks, recent studies in the area tend to conclude that numerical rating data are not informative enough for discovering user preferences. Consequently, given the availability of large collections of textual data, such as product reviews and social media posts, many approaches have tried to extend and improve recommendation models by leveraging such textual information. In fact, corpora of textual documents contain a wealth of information. On the one hand, they may help users to make more conscious decisions. On the other hand, they may be used to improve the predictive performances of recommendation systems.
Given the complexity of modeling user-generated information, researchers have simultaneously started to develop new techniques to improve and speed up the learning phase of these

computationally demanding models. Due to the flexibility, effectiveness, and applicability for different tasks, representing items, words, and documents as vector representations, also known as embedding, has become a common procedure lately. The resulting vector representations are useful since they can reduce the dimensionality of the data and, at the same time, provide meaningful representations in the latent space. However, these embeddings are usually not explainable; if singularly evaluated, a 100D or 200D vector makes little sense from the human perspective and is not informative. The main limitation of these techniques laid in the fact that they can capture relations among items by using vectors that are only meaningful to each other. For instance, in a recommendation system scenario, if we try to evaluate user vectors without employing the review vectors, we would not be able to comprehend the latent textual information associated with them. To gain trust and to promote collaboration between AIs and humans, it would be better if those representations were intrinsically interpretable for humans.

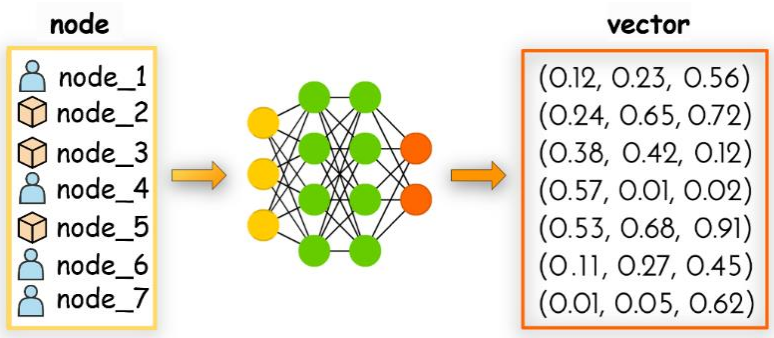


Figure 8 - Example of node embedding

In our work **[34]**, we focus our attention on developing an approach for improving the interpretability of node embedding. Many real-world user data are represented in forms of graphs, e.g. social networks, recommender systems and citation networks, since the graphs can capture relations and interactions of the involved entities. The performances, here too, are usually improved by using methodologies that represent the graph entities (e.g. nodes) through vector representations while preserving some structural information about the system. For instance, many graph neural network (GNN) approaches formulate graph learning with node embedding, and the downstream task, such as link prediction, node classification and (sub-) graph classification, is modeled with node embedding vectors. However, in terms of interpretability, most of the embedding approaches applied on graphs can provide meaningful representations in the latent space, but neither the single numbers contained in the vectors nor their dimensions have an interpretable meaning **[35]** (see Figure 8 for an example of node embedding). In this context, there are few previous works attempting to improve the interpretability of node embeddings. The existing works mainly aim to explain the embedding dimensions as clusters in an implicit manner, e.g. employing Canonical Polyadic decomposition **[36]**, and assigning a meaning to each vector dimension **[37]**. Unlike these approaches, our method focuses on improving the interpretability of node embeddings explicitly to get human understandable explanations by exploiting the extra textual information associated with the graphs. In fact, in many applications, nodes and edges in a graph are often associated with textual data, e.g. reviews in user-product graphs, social media posts in social media or medical narratives in doctor-patient graphs. Consequently, a question naturally

arises: could we integrate the textual information in the learning phase to improve the interpretability of node embedding? Our method maps the latent space of node embeddings into textual space through word-based vectors. Thus, the additional available textual information works as a human-understandable source to generate explanations of node embeddings. As reported in **[38]**, the most nuanced and sophisticated medium to express our feelings is our language. For this reason, we believe is important to understand and organize the textual information in a structured and intuitive way.

2.3.2 Generating Textual Explanations for Node Embeddings

We present an approach to represent the latent space of node embedding into a textual space by exploiting the available human-understandable information (i.e. the textual information) to interpret the embedding vectors. Starting from a text-labeled graph, we integrate the textual information into the model to learn interpretable node embeddings. For each node $\boldsymbol{i}$, we generate a textual explanation that is formulated as a node-specific word distribution conditioned on its embedding vector $\boldsymbol{x_i}$. Since the creation of the word-vectors is directly linked with the node embeddings (which are supposed to work well in downstream tasks), we use an objective function that combines the accuracy of a downstream task (e.g. rating prediction, sentiment analysis) with the likelihood of the textual corpus. We introduce an additional node clustering to model the patterns among the embedding vectors, the corresponding textual explanations, and the associated texts. The additional cluster assignment allows our model to learn the discrete structure of the graph data. In summary, our model attempts to combine two objectives: a) learn node embeddings that perform well in downstream tasks b) generate textual explanations of the learned vector representations.

After data preparation and subsequent vocabulary selection, our software comprises two major parts:

1. Generation of the textual explanations with our implemented model.
2. Evaluation of the results through both quantitative and qualitative experiments.

The detailed information about the above steps has been given in deliverable D3.3.

After preprocessing the raw review texts and performing the vocabulary selection, we are able to learn textual explanations for node embeddings by training our model. The schematic view of our architecture is presented in Figure 9.

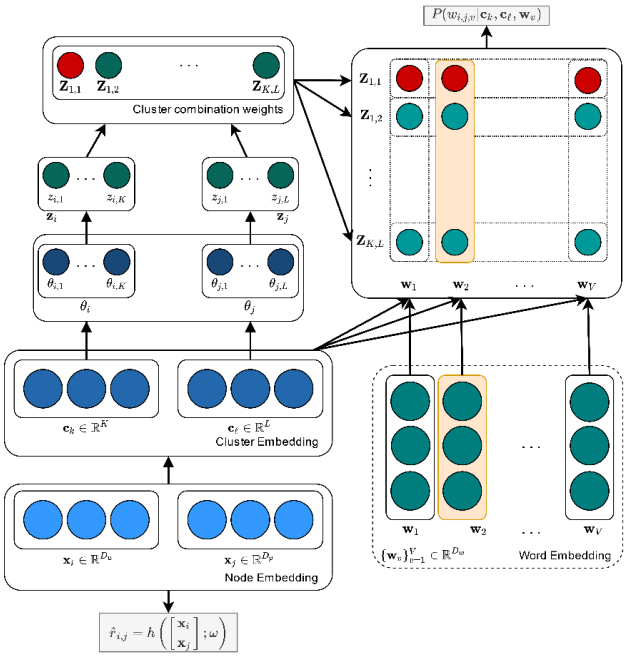


Figure 9 - Schematic view of our model. Dashed boxes represent input (non-trainable) data. The line connections depict the dependencies between the involved variables.

The learning of the node embeddings and the corresponding textual explanations is driven by two learning objectives. The first objective is the log-likelihood of the review corpus. The parameters to be learned include embedding vectors of clusters $\{\boldsymbol{c}_k\}_{k=1}^{K}$ and $\{\boldsymbol{c}_\ell\}_{\ell=1}^{L}$, and parameters ϕ, ξ, γ and ρ that define the neural networks f, g and ψ. Thus, the log-likelihood of an edge and the corresponding text is:

$$\mathcal{L}_1 = \log p(e_{i,j} \,|\boldsymbol{x}_i, \boldsymbol{x}_j, \gamma) + \\ + \sum_{v=1}^{S} \log \left(\sum_{k=1}^{K} \sum_{\ell=1}^{L} p(\boldsymbol{z}_i = k|\, \boldsymbol{x}_i, \boldsymbol{c}_k, \phi) \right. \\ \left. p(\boldsymbol{z}_j = \ell|\boldsymbol{x}_j, \boldsymbol{c}_\ell, \xi) p(\boldsymbol{w}_{i,j,v}|\boldsymbol{c}_k, \boldsymbol{c}_\ell, \boldsymbol{x}_v, \rho) \right)$$

where the first term represents the probability that an edge exists between two nodes. This is implicitly included in the computation of the textual information since we suppose that every edge, if exists, has some associated text.

The second objective is the error of the predictions. In our study case, this will be the rating prediction error.

$$\mathcal{L}_2 = \frac{1}{R} \left(\hat{r}_{i,j} - r_{i,j} \right)^2$$

with

$$\hat{r}_{i,j} = h\left(\begin{bmatrix} \boldsymbol{x}_i \\ \boldsymbol{x}_j \end{bmatrix}; \omega \right)$$

where $h(\cdot)$ can be any complex function. In our case, $h(\cdot)$ defines a deep neural network with the concatenation of the node embeddings $\boldsymbol{x}_i$ and $\boldsymbol{x}_j$ as input and ω as hyperparameters.

Finally, we can define the complete objective function as:

$$\min_{\Theta} \mathcal{L} = \min_{\Theta} (\mathcal{L}_1 + \mu \mathcal{L}_2)$$

where Θ represents the parametric space and μ is a hyperparameter to trade-off the importance of the prediction accuracy (i.e. mean squared error (MSE)) and the negative log-likelihood (NLL) of the corpus.
The hyper-parameters of the architecture were hard coded in the training script and, if necessary, need to be changed in the corresponding Python file.

Output

- β is a 3D tensor representing the probabilistic patterns among user clusters, product clusters and words. In particular, $\beta_{k,\ell,:}$ specifies a categorical word distribution conditioned on the user cluster k and the product cluster ℓ. It lies in a $(V-1)$-dimensional simplex Δ^{V-1}, i.e. $\sum_{v=1}^{V} \beta_{k,\ell,v} = 1$ and $\beta_{k,\ell,v} > 0$. V denotes the number of words. In Figure 9, this matrix is specified by the upper-right squared box. In few words, this matrix represents the probability that a given word $w \in V$ would explain a specific user-product cluster combination (k, ℓ).
- $\theta_{i,k}$ specifies the probability of the user i to belong to the user cluster k.
- $\theta_{j,\ell}$ specifies the probability of the product j to belong to the product cluster ℓ.

2.3.3 Evaluation of the Generated Textual Explanations

We can use β, $\theta_{i,k}$ and $\theta_{j,\ell}$, for the following purposes:

1. Generation of word-vector distributions for node embeddings. In this way, we are able to generate textual explanations for node vector representations that would be non-interpretable otherwise.
2. Quantitative evaluation of the generated explanations. Even though our case study can be evaluated by just relying on human perception of the highlighted relevant words, we introduce some metrics that could further help on assessing the quality of the results.
3. As explained before, β specifies the word distributions for each combination of user and product cluster (k, ℓ). We can plot them into a 2-dimensional space to understand whether there is any cluster organization. This would give additional information about the underlying structure of the analysed items, further enhancing the interpretability of the results.

To illustrate the results, we use a typical review network as a running example. Assume there is a bipartite graph $\boldsymbol{G}$ with $\boldsymbol{N}$ number of users and $\boldsymbol{M}$ number of products. Between a user $\boldsymbol{i}$ and a product $\boldsymbol{j}$, there is an edge $\boldsymbol{e_{ij}}$. Each edge is associated with a set of words (namely, a review) $\boldsymbol{s_{ij}} = \{\boldsymbol{w_{i,j,1}}, \dots, \boldsymbol{w_{i,j,S}}\}$ and a rating $\boldsymbol{r_{ij}}$. The size of the vocabulary, for each product category, is $\boldsymbol{V}$. The number of reviews is $\boldsymbol{R}$.

All the generated results and figures can be reproduced using the Jupyter notebook available in the repository.

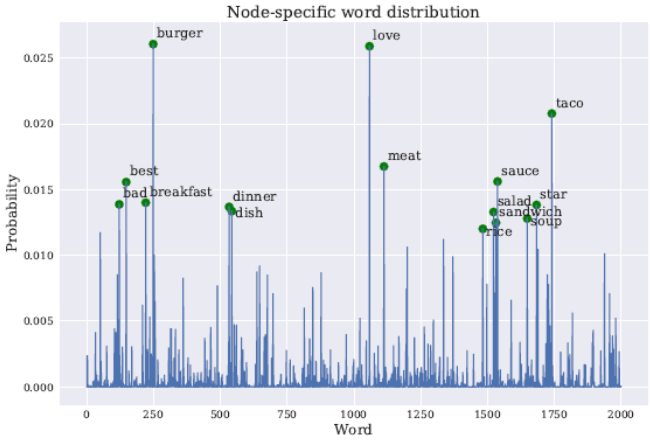
Generation of word-vector distributions

For a node, e.g. a user i, the textual explanation is formulated as a node-specific word distribution $p(\boldsymbol{w}_v|\boldsymbol{x}_i)$ conditioned on its embedding vector $\boldsymbol{x}_i$.

In particular, the probability of a word v to be used to explain the embedding $\boldsymbol{x}_i$ is computed as:

$$p(\boldsymbol{w}_v|\boldsymbol{x}_i) = \frac{1}{L}\sum_{k,\ell} \theta_{i,k}\, \beta_{k,\ell,v}$$

This is a marginal distribution over all possible user and product clusters. Since the target distribution is not related to any specific products, the product clusters are equally distributed, i.e. the term $1/L$ in the equation above. Textual explanations for product nodes can be generated in an equivalent manner. **Error! Reference source not found.** depicts the word-distribution for a given node.



TOP-15 WORDS	bad **best** **breakfast** burger dinner dish **love** **meat** rice salad sandwich **sauce** soup star **taco**
NODE-RELATED WORDS	ambiance authentic **best** **breakfast** choice cilantro decision guacamole **love** **meat** pas plain satisfied **sauce** spacious staple **taco**
	t-J(A,B) = 0.796 t-WDM(A,B) = 0.748

Figure 10 – Interpretability case study on a random node. The figure depicts the node-specific word distribution; the 15 highest probabilities are highlighted by green points. TOP-15 WORDS and NODE-RELATED WORDS refer to the sets A and B defined below. Black bold represents the overlapping words; blue bold highlights words that may explain further characteristics of the analyzed node.

In terms of interpretability, as already mentioned, the main limitation of the common techniques laid in the fact that they can capture relations among items by using vectors that are only meaningful to each other. Differently, our model can directly encode the textual information within the estimated quantities Indeed, through our representation, each dimension of the vector corresponds to a word in the vocabulary and each value represents the probability that the corresponding word would explain the selected node, providing vectors that are self-explainable.

Quantitative evaluation of the generated textual explanations

To visualize and evaluate the correlation between the generated word distributions and the node-related words in the data, we proceed as follows. Given a sampled node, we first extract the *top-15* words in the generated word distribution, i.e. the 15 words having the highest probability in the distribution. Second, we extract the set of words associated with this specific node in the data. Let denote with *A* and *B* respectively, these two sets.

The Jaccard similarity is used to measure similarity between two sets of words A and B, which is defined as follows:

$$J(A,B) = \frac{|A \cap B|}{|A \cup B|}$$

To further capture the semantic similarity of the two sets, we integrate the Word Mover's Distance (WDM), introduced by [39]; this metric takes into account the word similarities in the word embedding space and computes the minimum distance that the words in text A need to *travel* in the semantic space to reach the words in text B. Since Jaccard and WDM have different scales and behaviors, we apply MinMaxScaler to Jaccard, and transform WDM as follows:

$$t - WDM(A,B) = 1 - \left(\frac{d_i - \min(\boldsymbol{d})}{\max(\boldsymbol{d}) - \min(\boldsymbol{d})}\right)$$

where d_i is the distance between the sets A and B for a node i, and $\boldsymbol{d}$ represents all the distances between the two sets for each node in the graph. In this way, the transformed distance score is in the range $[0,1]$ and, opposite to the definition of semantic distance, the higher the value the closer are sets A and B.

We investigate the quality of the generated textual explanations by computing these scores for different data sets. **Error! Reference source not found.** illustrates the distribution of values of these measures across the nodes in the data sets.

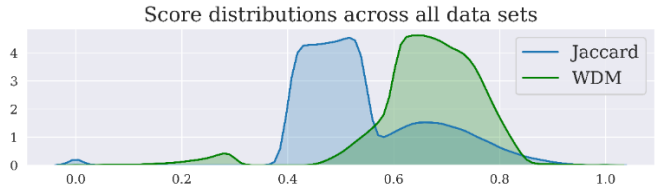


Figure 11 - Evaluation of the metric distributions across all data sets.

In this case, the Jaccard values mostly vary in the range $[0.4 - 0.7]$, while the WDM scores are highly concentrated in the range $[0.6 - 0.8]$. Note that the lower Jaccard scores do not affect the performances of the model, instead, confirms that it is able to generate textual explanations that are not redundant as would have been with too high similarity scores. Indeed, as written in [40], two sets of words can be semantically similar even with low lexical overlapping.

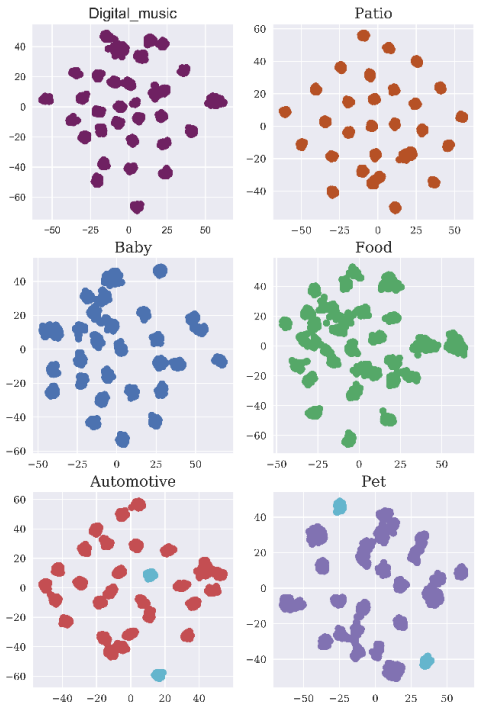
Analysis of the probabilistic patterns

For each category, to visualize the learned word-vector distributions, the t-SNE method [41] is employed for dimensionality reduction. The `sklearn` package can be used for this purpose.

Python package	Usage
`sklearn`	`sklearn.manifold.TSNE(n_components=2)`

Error! Reference source not found. illustrates the cluster organization for different categories taken from our experiments. One can find that the clusters are well structured and mapped into the 2-dimensional embedding space with different distributions. For further analysis, we select a pair of clusters from two different categories, and we compute the corresponding average word

distribution. In theory, one could infer the main *topic* of each cluster looking at the most probable words of the averaged cluster word distribution. The table reports the most probable words for each of the selected clusters. The results validate our hypothesis since, for each cluster, we can infer the sub-category of interest. For instance, the first cluster in the *Pet Supplies* category refers to the *grooming* sub-category, while the second one focuses on *aquariums*. Thus, the word distributions $\beta_{\cdot,\cdot,v}$ capture the latent structures of the data and help to find the patterns between user and product clusters, enhancing the interpretability of the results. Indeed, knowing the user and product cluster assignments one can find the *sub-categories* highly correlated with the given items.



Automotive		Pet Supplies	
Cluster 1	Cluster 2	Cluster 1	Cluster 2
battery	trailer	work	filter
charge	power	hair	fish
charger	gas	brush	water
power	away	look	gallon
plug	compressor	thing	plant
code	large	fur	heater
cord	pressure	smell	turtle
lead	jeep	quality	flow
transmission	price	wet	clean
phone	guy	stuff	pump
volt	hole	comb	algae
change	weight	shed	shrimp
adapter	fast	groom	work
smell	fuel	flea	gravel
connect	space	shampoo	tube
electronics	*performances*	*grooming*	*aquariums*

Figure 12 – Cluster organization of the word-vector distribution $\beta_{\cdot,\cdot,v}$ for different product categories. The cluster analyzed in the table are highlighted in cyan.

3 Manual of the Developed Software

3.1 Obtaining Structured Data Formats and Extracting Features from Procedural Optimization Data

The code we provide with our report is based upon our work *'Artificial Neural Networks as Feature Extractors in Continuous Evolutionary Optimization'* [11] and as elaborated in Section Feature Extraction and Analysis2.1.2 can be used to replicate the results for feature extraction and analysis on the given synthetic benchmark function set. The tool is implemented with Python 3.0 and the required packages are elaborated in the aforementioned technical requirements part of Section 2.1.1 .

We provide in summary within our code mainly two different functionalities:

- The first one being methods to generate training data for a given search space volume and subsequently setup the different search space partition methods.
- And the second one being methods to replicate the feature extraction and analysis studies on the synthetic benchmark function set as contained within our paper.

The software tool is organized as follows:

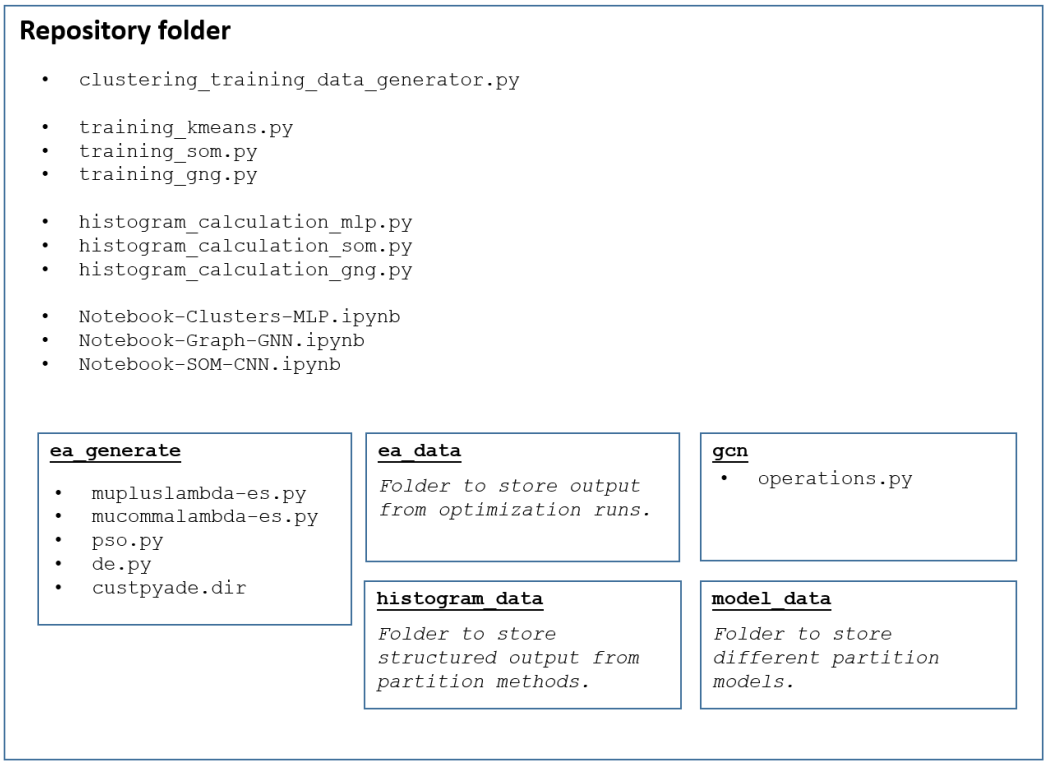


Figure 13 - Folder and file organization of our framework.

One can proceed with the provided code within our repository to replicate the results as follows, roughly according to the data post-processing pipeline as elaborated in Figure 4.

1. Selection / Setup of a Search Partition Method
2. Data Generation from Synthetic Experiments
3. Conversion of Unstructured Raw Data into a Structured Format
4. Loading and Preparation of the Structured Data for Training
5. Training of a Neural Network Architecture
6. Subsequent Feature Extraction and Analysis

The folder organization is elaborated as in *Figure 13***.** We already provide experimental data from step 1), 2) and 3) in the folders `model_data`, `ea_data` and `histogram_data` respectively. So in principle, depending of one's preferences, one can skip any step between 1) – 3) without requiring the execution of the previous ones. The folder `gcn` contains the necessary custom operations for graph convolution and pooling. Where the former is a custom Keras implementation and the latter is based upon MIT licensed code accompanying the original paper from Defferrard et al. [24]. Different implementations of evolutionary algorithms from the DEAP library licensed under LGPL-3.0 that can be used for experimentation are provided in the `ea_generate` folder. Otherwise, the scripts `training_\*.py` can be used to generate partition models, `histogram_calculation_\*.py` to obtain structured data formats and `Notebook-\*.pynb` to experiment with network architectures and analyze their feature extraction capabilities are contained within the main folder.

In the following, we will give a more in-depth description on how to use the scripts according to the previously elaborated step-by-step description.

Step 1 - <u>Setting up a Search Space Partition Method</u>

The scripts `training_som.py`, `training_kmeans.py` and `training_gng.py` implement the differently elaborated search space partitions. Upon being called from the command line or via an IDE, the script `clustering_training_data_generator.py` is imported and the `generateTrainingData` method is called to generate a training data set which exhaustively fills the search space and can subsequently be used for the setup of the aforementioned search space partition methods. Trained partition models are subsequently stored in the `model_data` folder.

Step 2 - <u>Data Generation from Synthetic Experiments</u>

Calling from the `ea_generate` folder e.g. the $(\mu+\lambda)$-ES via `m-plus-l_evolution-stategy.py` using the command line or within an IDE, starts experiments with preset experimental parameters contained in the file. Note that we rescale any generated solutions to a search space size of $[-30, 30]^d$ such that to ensure a uniform format for the subsequent application of a search space partition methods. Any subsequently generated files are stored within the `ea_data` folder for further processing.

Step 3 - <u>Converting Unstructured Raw Data into Structured Data</u>

To convert the unstructured draw data into a structured data format the scripts `histogram_calculation_som.py` for the self-organized map, int, `histogram_calculation_kmeans.py` for the k-means algorithm as well as as well as `histogram_calculation_gng.py` for the growing neural gas can be used. The obtained structured data format from running the scripts is stored in the folder `histogram_data`. Subsequently, it can be used for the training of neural network architectures.

Step 4 - <u>Replication of Experiments</u>
The Jupyter Notebooks `Notebook-SOM-MLP.ipynb`, `Notebook-SOM-CNN.ipynb` and `Notebook-SOM-GNN.ipynb` contain the necessary code to replicate the previously elaborated experiments on in regards feature extraction and analysis.

3.2 Evolutionary Optimization for Proactive and Dynamic Computing Resource Allocation in Open Radio Access Network

The repository contains code for the work Evolutionary Optimization for Proactive and Dynamic Computing Resource Allocation in Open Radio Access Network, which is prepared to submit to the journal of Information Sciences. As the paper has not been published, the code for this work will be released in the website of ECOLE later. The scripts are based on Python 3.7, and have been tested on Linux and Windows systems.

To get the results of comparing the proposed evolutionary algorithm (SplitEA) against the existing method (the greedy algorithm) and two variants of SplitEA (RandEA and CopyEA) on real-world and artificial datasets, the codes contain all the necessary steps to run the experiments:

1. Pre-process the raw network traffic data.
2. Generate artificial network traffic data with the location data and traffic data.
3. Apply the Long Short-Term Memory model to predict the traffic data (if the algorithms run on the datasets with the prediction);
4. Run all compared algorithms on all datasets.

The organization of our software tool is shown as below:

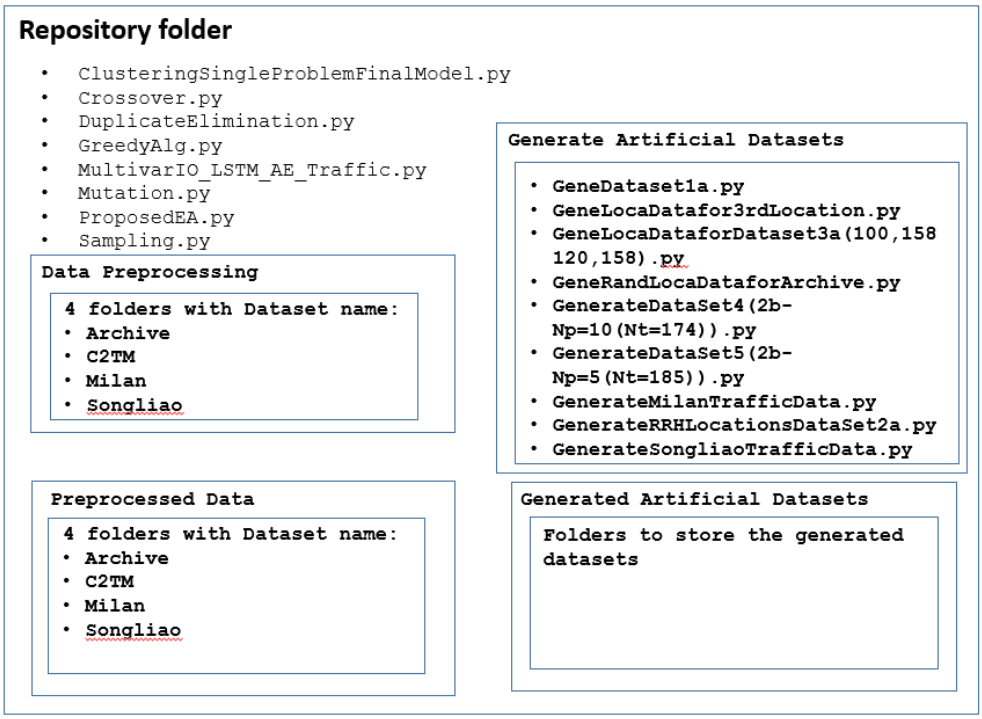


Figure 14 - Folder and file organization of our framework.

The primary idea of preprocessing the real-world data is to make each point in the location dataset has traffic data at each hour of a set of continuous days in the corresponding traffic dataset. Considering that four found datasets have different forms of types, the preprocessing is different.

Step 1 –<u>Data preprocessing</u>

All codes for the data preprocessing are put in the fold 'Data Preprocessing' for all four real-world datasets. All preprocessed datasets are put in the fold 'Preprocessed Data'.

Milan dataset [42]: we firstly collect the locations and coverage areas of active base stations observed in the two months at the website https://www.cellmapper.net/map from 11/01/2013 to 12/31/2013, to get the file `LocationCoverageBSMilan.xlsx`, from which the coverage information of those base stations is separately extracted as the file `CoverageRRHMilan.xlsx`. The row traffic data can be found in the link of paper [42]. The following `py.` files need to be run in order to preprocess the row traffic data:

1. `generate_kaggle_dataset.py`: generate the kaggle dataset from the original dataset;
2. `convert_traffic_volume_by_ID.py`: calculate the traffic of each grid by adding all activities (smsin, smsout, callin, callout and internet) for each grid regardless of the countrycode;
3. `calculate_RRH_traffic.py`: calculate the traffic of each RRH (Remote Radio Head) in each base station, according to the coverage information (CoverageRRHMilan.xlsx) of each RRH;
4. `normalize_AllDaysTraffic.py`: normalize all RRHs traffic data on all hours of all days' maximal and minimal traffic to the range of [0,1].

Songliao dataset [43]: `process_Mobility1.py` and `process_Mobility2.py` conduct the preprocessing. It first calculates the data of each point through adding all weights of coming to this point and originating from this point. After that, if there is one or more hours when some points do not have the data, delete those points from 'GPS.txt' and delete those rows with the data for those points to get the file 'GPS.txt'. Finally, normalize the data of each point to the [0, 1] range.

Archive dataset [44]: `sortHourCell.py` sort the traffic data on the hour in an ascending order. Then, delete those rows with the day if there is no data at one or more hours on those days. Next select the dataset with longest continuous days. Finally, the file `normalize_SplittedDataByDay.py` normalizes the data of each point to the [0, 1] range.

C2TM dataset [45]: `generate_kaggle_dataset.py` generate the dataset for each day from the row dataset file 'cellular_traffic.csv'. Given that there are too many base stations in the dataset, a subset of the total dataset is selected using the file `selectSubset.py`. Lastly, the file `SubsetData.py` normalizes the sub-set to the range [0,1].

Output files: `datasetname-date-traffic.csv` where "*datasetname*" is the name of four real-world datasets and "*date*" is the dates of the real-world dataset. One of the examples of Archive dataset on 2018.02.12 is `Archive-2018-02-12-traffic.csv`. Another output file is the location dataset: `datasetnameLocation.xlsx`.

Step 2 –<u>Generate Artificial Datasets</u>

Considering that each dataset has the location dataset and traffic dataset, several datasets are generated for both location dataset and traffic dataset, each of which has three different types generated datasets. All codes for the data generating the artificial datasets are put in the fold 'Generate Artificial Datasets'. All generated artificial datasets are put in the fold 'Generated Artificial Datasets'.

Three types of location datasets:

1. All points are totally randomly generated within a range;
2. High cohesion and low coupling w.r.t distance of points, which means that the distance of any two points in the same cluster is smaller than τ and the distance of any two points in different clusters is larger than τ, the maximal number of generated points in each cluster is Np.
3. Many points are gathered together while others are scattered away from those points. Ng and Nt are the number of gathered and total points, respectively

Three types of traffic datasets:

a) Totally randomly generated from (0,1) for each point at 24 hours of each day;
b) Generate the traffic data in a way that the optimal value of the objective function is known for the second case of the location dataset. More specifically, for each cluster in which all points have the distance close to each other smaller than τ, just split it into several sub-clusters and then generate the traffic data such that the total traffic data in each sub-cluster is equal to 1 at each hour of a day.
c) Follow the pattern of existing real dataset Milan and Songliao. For the pattern of Milan dataset, the traffic data for most points firstly decreases at the first five or six hours of each day and then increases until noon. Then, it remains stable at five or six hours and lastly it decreases.

As for the pattern of Songliao dataset, it firstly increases until eight or nine of each day and then remains stable for ten or eleven hour and lastly decreases. Those patterns are extracted from the traffic dataset of Milan and Songliao datasets.

There are 8 generated artificial datasets with seven days, which includes 1a, 2a, 3a 100/158, 3a 120/158, 1c-Milan, 1c-Songliao, 2b-Np=10 (Nt=174) and 2b-Np=5(Nt=185). Among those datasets, '1', '2' and '3' means the first, second and third location dataset, respectively; 'a', 'b' and 'c' means the first, second and third traffic dataset.

Output files: similar to the output files of the pre-processing in step 1, there are two types of output files: traffic and location dataset. `datasetname-date-traffic.csv` where "*datasetname"* is the name of four real-world datasets and "*date"* is the dates of the real-world dataset. One of the examples of Archive dataset on 2018.02.12 is `Archive-2018-02-12-traffic.csv`. Another output file is the location dataset: `datasetnameLocation.xlsx`.

Step 3 –<u>Conduct the Prediction on Archive and Milan Datasets</u>

Even though there are four real-world datasets, only two datasets (Archive and Milan) have enough days to do the prediction. Therefore, a two stacked multivariant Long Short-Term Memory model is used to do the prediction. The first 70% days of those two datasets is regarded as the training set with the remaining days as the testing set are split into training and test datasets. The file `MultivarIO_LSTM_AE_Traffic.py` is used to achieve this functionality.

Command to run the file: `python MultivarIO_LSTM_AE_Traffic.py`

Output files: `Pre-datasetname-date-traffic.csv` where "*datasetname"* is the name of four real-world datasets and "*date"* is the dates of the real-world dataset, e.g.: `Pre-Archive-2018-05-29-traffic.csv.`

Step 4 –<u>Run all compared algorithms on all datasets.</u>

Run all compared algorithms on all real-world datasets and artificial datasets. All related .py files are described as follows:

`ClusteringSingleProblemFinalModel.py` describes the proposed problem formulation in the paper mentioned before. `Crossover.py` and `Mutation.py` are the crossover and mutation operators used in the proposed evolutionary algorithm. `Sampling.py` is the initialization process for the proposed evolutionary algorithm. `GreedyAlg.py` is the file of the existing work greedy algorithm. `ProposedEA.py` is the file of the proposed evolutionary algorithm.

When running the experiment, just run the file of `GreedyAlg.py` and `ProposedEA.py` using the command `python GreedyAlg.py` and `python ProposedEA.py`. All problem- and algorithm-related parameters are set in those two files.

Open-source software like a library that was used in this software, i.e., python libraries used in pre-mentioned steps:

- Data pre-processing: numpy, pandas, datetime, dateutil, time;
- Artificial dataset generation: numpy, pandas, random, datetime, math,
- Traffic dataset prediction: numpy, pandas, datetime, time, math, tensorflow

- Running the algorithms: math, openpyxl, datetime, time, numpy, pandas, random, copy, sys, os, pymoo [47]. Pymoo is a library of multi-objective optimization in python. Detailed information can be found in [47] and the website: https://pymoo.org/.

Graphics processing unit (GPU) can be used to run the scrip `MultivarIO_LSTM_AE_Traffic.py` if GPU is applicable.

Output files: for evolutionary algorithms, there are two output xlsx files: `DynaGAStrategyF.xlsx` and `DynaGAStrategyX.xlsx`, where "*Strategy"* is the strategy to generate a new population for each day except for the first day. For the greedy algorithm, there are also two output xlsx files: `DynaGreedyAlgF.xlsx` and `DynaGreedyAlgX.xlsx`. The first file with the ending of "F.xlsx" stores the found four metrics and the fitness value at each day under 30 independent runs. The second file with the ending of "X.xlsx" stores the found optimal solution (clustering scheme specifying which point belongs to which cluster) at each day under 30 independent runs.

3.3 Interpreting Node Embedding with Text-labeled Graphs

The code for the work [34] is implemented with Python 2.7, and has been tested on Linux OS. The figure below visualizes the organization of our software package:

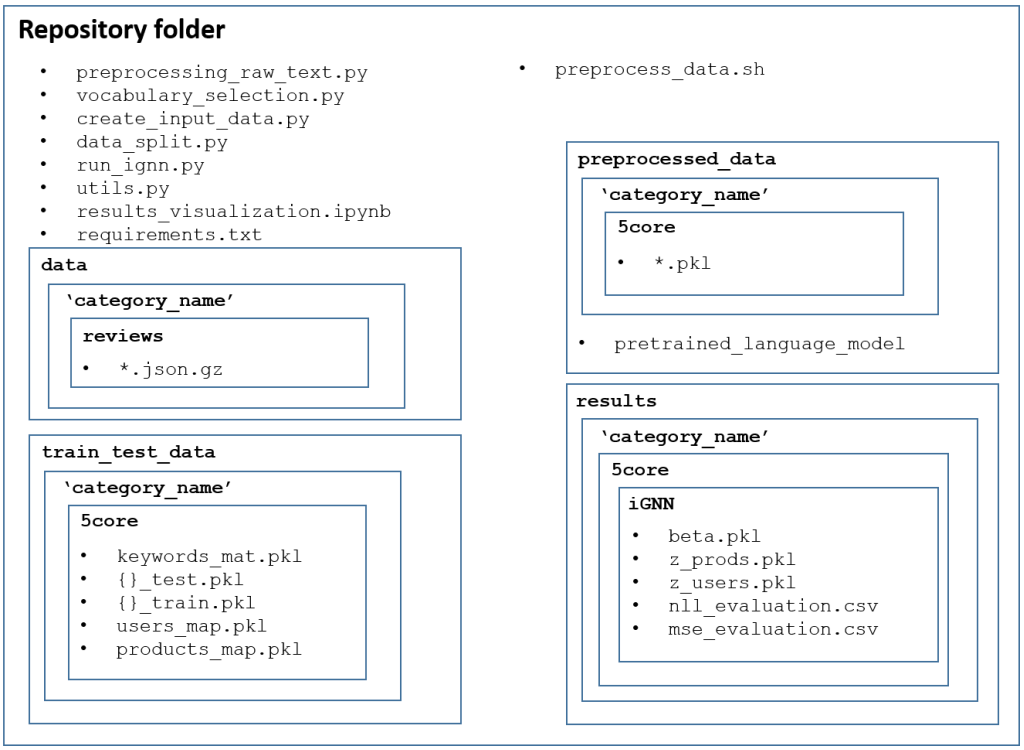


Figure 15 - Folder and file organization of our framework.

The python file `utils.py` contains paths, hyper-parameters and functions needed to run all the steps of the software. The list of product categories to evaluate can be changed in this file. Please, ensure to first download the corresponding raw review data from the link provided before and save them in the `data` folder.

We saved a pre-trained language model (`preprocessed_data/pretrained_language_model`); this will serve as a pre-trained language model for our model and will be fixed for the whole pipeline. Note that the pre-trained language model can be downloaded from other source.

Data sets

In the repository, we uploaded a small product category for experimentation, i.e. *Patio*. The raw reviews are contained in the directory `data/patio/reviews`. All the product categories are publicly available at: https://jmcauley.ucsd.edu/data/amazon/. Please note that we use the 5-core version of these data sets.

Dependencies

All the dependencies are installed if `pip install -r requirements.txt` is run.
If available, it is possible to use a GPU infrastructure to run the architecture faster. These scripts have been tested with `tensorflow_gpu 1.13.1`, `cuDNN 7.4` and `CUDA 10.0`.

Data preprocessing
Before training, some data preparation is needed to run the architecture. This includes data cleaning, vocabulary selection and data splitting. To preprocess the data, run the following bash command:

```
bash preprocess_data.sh
```

Training
Run the architecture - Once we prepared the data, we can run the architecture. To train the model, run the following command:

```
python run_ignn.py
```

Input files

- `train_test_data/{category_name}/5core/users_map.pkl`
  Dictionary of the form `{userID: index}`
- `train_test_data/{category_name}/5core/products_map.pkl`
  Dictionary of the form `{productID: index}`
- `train_test_data/{category_name}/5core/{}_train.pkl`
  `Replace {} with either 'users_ID', 'prods_ID', 'words', 'ratings'.`
  List of training data for users, products, ratings and reviews (i.e. biterm lists).
- `train_test_data/{category_name}/5core/{}_test.pkl`
  `Replace {} with either 'users_ID', 'prods_ID', 'words', 'ratings'.`
  List of testing data for users, products, ratings and reviews (i.e. biterm lists).
- `train_test_data/{category_name}/5core/keywords_mat.pkl`
  File containing the VxD vocabulary matrix, i.e. the vector representations of the words contained in the vocabulary (note that the vector representations are taken from the pretrained language model)

Output files

- `results/{category_name}/5core/iGNN/beta.pkl`: the file stores the β matrix as a `.pkl` file. This matrix will be used for the quantitative and qualitative evaluation of the generated explanations, as explained in the technology part.
- `results/{category_name}/5core/iGNN/z_users.pkl`: the file stores $\theta_{i,k}$, i.e. the probabilities of user i to belong to cluster k. For all users and user clusters.
- `results/{category_name}/5core/iGNN/z_prods.pkl`: the file stores $\theta_{j,\ell}$, i.e. the probabilities of product j to belong to cluster ℓ. For all products and product clusters.
- `results/{category_name}/5core/iGNN/mse_evaluation.csv`: the file stores the train and test MSE values for each evaluated epoch.
- `results/{category_name}/5core/iGNN/nll_evaluation.csv`: the file stores the train and test NLL values for each evaluated epoch.

Results evaluation

We can use the learned parameters β, $\theta_{i,k}$, and $\theta_{j,\ell}$ to generate textual explanations for nodes. To visualize and evaluate the results, we provided a Jupyter notebook file (`results_visualization.ipynb`). This file contains all the instructions to reproduce the results reported in the paper, and to create the output showed in Section 2.3.3. Apart from reproducing the reported results, the notebook can be used to manually explore the results.

4 Summary & Outlook

This deliverable reports the software modules and functionalities developed in WP3 of the ECOLE project. To facilitate the potential users to easily utilize the software for their research, we illustrate each functionality with respect to: technical contributions and advantages, implementation details, package dependencies, and pipelines of how to run the software tool properly. Some example data is provided as well for the users to test the software straightforwardly. In summary, the software contributes the following three functionalities with superior performance compared with recent baselines, and can be applied to difference domains.

The first function is designed to focus on improvement of population-based optimization algorithms themselves. The procedural metadata generated from such optimization algorithms is used to learn the structure of the optimization problems, and improve the performance of the optimization algorithms. A possible application scenario could be shape optimization. We have demonstrated the application in the deliverable. Generally, the practitioner is free to experiment with our pipeline and apply it to other similar scenarios e.g. algorithm selection and configuration.

Another function the software provides is resource optimization, in particular, allocating computing resource proactively to serve e.g. upcoming traffic load in ORAN. This function formulates the problem as a clustering problem with additional optimization objective of minimizing the number of the clusters. A novel evolutionary method is introduced to find better allocation scheme. The function can be used for other related resource optimization problems, for example, assigning resource for edge computing.

Last but not least, the software provides a function for product feature optimization towards user preference. The function can elicit user opinion on product characteristics from user generated data (texts and behavior). Instead of a black box method, the proposed software tool especially improves the interpretability of the results, and can learn human-understandable explanations for users/products. This function can also be used for different learning tasks and domains. The typical examples might include medical applications (doctor-patient graphs with medical narratives) and social media recommendations.

The developed software reported in the deliverable can be downloaded from the ECOLE public GitHub repository (https://github.com/ECOLE-ITN). The researchers are encouraged and appreciated to utilize our tools to compare their own methods for future research.

5 References

[1] Y. LeCun, Y. Bengio and G. Hinton, "Deep learning," *Nature,* vol. 521, p. 436–444, 2015.

[2] F. P. Such, V. Madhavan, E. Conti, J. Lehman, K. O. Stanley and J. Clune, "Deep neuroevolution: Genetic algorithms are a competitive alternative for training deep neural networks for reinforcement learning," *arXiv preprint arXiv:1712.06567,* 2017.

[3] T. Elsken, J. H. Metzen, F. Hutter and others, "Neural architecture search: A survey.," *J. Mach. Learn. Res.,* vol. 20, p. 1–21, 2019.

[4] R. Miikkulainen, J. Liang, E. Meyerson, A. Rawal, D. Fink, O. Francon, B. Raju, H. Shahrzad, A. Navruzyan, N. Duffy and others, "Evolving deep neural networks," in *Artificial Intelligence in the Age of Neural Networks and Brain Computing*, Elsevier, 2019, p. 293–312.

[5] K. O. Stanley, J. Clune, J. Lehman and R. Miikkulainen, "Designing neural networks through neuroevolution," *Nature Machine Intelligence*, vol. 1, p. 24–35, 2019.

[6] J. D. Schaffer, D. Whitley and L. 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*, 1992.

[7] X. Yao, "Evolving artificial neural networks," *Proceedings of the IEEE,* vol. 87, p. 1423–1447, 1999.

[8] X. Yao and M. M. Islam, "Evolving artificial neural network ensembles," *IEEE Computational Intelligence Magazine,* vol. 3, pp. 31-42, 2008.

[9] D. H. Hubel and T. N. Wiesel, "Receptive fields, binocular interaction and functional architecture in the cat's visual cortex," *The Journal of Physiology,* vol. 160, p. 106–154, 1962.

[10] C. Blakemore and G. F. Cooper, "Development of the brain depends on the visual environment," *Nature,* vol. 228, p. 477–478, 1970.

[11] S. Friess, P. Tiňo, Z. Xu, S. Menzel, B. Sendhoff and X. Yao, "Artificial Neural Networks as Feature Extractors in Continuous Evolutionary Optimization [accepted]," in *2021 IEEE International Joint Conference on Neural Networks (IJCNN)*, 2021.

[12] S. Friess, P. Tiňo, S. Menzel, B. Sendhoff and X. Yao, "Improving Evolutionary Optimization through Prediction of Inductive Biases with

Applications to Shape Optimization [In Review]," in *2021 IEEE Symposium Series on Computational Intelligence (SSCI)*.

[13] T. Kohonen, "Essentials of the self-organizing map," *Neural networks,* vol. 37, p. 52–65, 2013.

[14] C. Pang, M. Wang, W. Liu and B. 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*, 2016.

[15] L. Liu, C. Pang, W. Liu and B. Li, "Learning to Describe Collective Search Behavior of Evolutionary Algorithms in Solution Space," in *Asia-Pacific Conference on Simulated Evolution and Learning*, 2017.

[16] M. Turkey and R. Poli, "An empirical tool for analysing the collective behaviour of population-based algorithms," in *European Conference on the Applications of Evolutionary Computation*, 2012.

[17] B. Fritzke, "A growing neural gas network learns topologies," in *Advances in Neural Information Processing Systems*, 1995.

[18] C. M. Bishop, "Pattern recognition and machine learning," *Machine Learning,* vol. 128, 2006.

[19] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg and others, "Scikit-learn: Machine learning in Python," *Journal of Machine Learning Research,* vol. 12, p. 2825–2830, 2011.

[20] P. Virtanen, R. Gommers, T. E. Oliphant, M. Haberland, T. Reddy, D. Cournapeau, E. Burovski, P. Peterson, W. Weckesser, J. Bright and others, "SciPy 1.0: fundamental algorithms for scientific computing in Python," *Nature Methods*, vol. 17, p. 261–272, 2020.

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

[22] F. Chollet and others, "Keras: The python deep learning library," *ascl,* p. ascl–1806, 2018.

[23] M. Abadi, "TensorFlow: learning functions at scale," in *Proceedings of the 21st ACM SIGPLAN International Conference on Functional Programming*, 2016.

[24] M. Defferrard, X. Bresson and P. Vandergheynst, "Convolutional neural networks on graphs with fast localized spectral filtering," *Advances in Neural Information Processing Systems,* vol. 29, p. 3844–3852, 2016.

[25] A. X. Chang, T. Funkhouser, L. Guibas, P. Hanrahan, Q. Huang, Z. Li, S. Savarese, M. Savva, S. Song, H. Su, J. Xiao, L. Yi and F. Yu, *ShapeNet: An Information-Rich 3D Model Repository*, 2015.

[26] F.-A. Fortin, F.-M. D. Rainville, M.-A. Gardner, M. Parizeau and C. Gagné, "DEAP: Evolutionary Algorithms Made Easy," *Journal of Machine Learning Research*, vol. 13, p. 2171–2175, 7 2012.

[27] T. N. Kipf and M. Welling, "Semi-supervised classification with graph convolutional networks," *arXiv preprint arXiv:1609.02907*, 2016.

[28] P. Zhang, X. Yao, L. Jia, B. Sendhoff and T. Schnier, "Target shape design optimization by evolving splines," in *2007 IEEE Congress on Evolutionary Computation*, 2007.

[29] S. K. Singh, R. Singh and B. Kumbhani, "The evolution of radio access network towards open-ran: challenges and opportunities," in *2020 IEEE Wireless Communications and Networking Conference Workshops (WCNCW)*, 2020.

[30] L. Gavrilovska, V. Rakovic and D. Denkovski, "From Cloud RAN to Open RAN.," *Wirel. Pers. Commun.*, vol. 113, p. 1523–1539, 2020.

[31] L. Chen, D. Yang, D. Zhang, C. Wang, J. Li and others, "Deep mobile traffic forecast and complementary base station clustering for C-RAN optimization," *Journal of Network and Computer Applications*, vol. 121, p. 59–69, 2018.

[32] A. Perveen, R. Abozariba, M. Patwary and A. Aneiba, "Dynamic traffic forecasting and fuzzy-based optimized admission control in federated 5G-open RAN networks," *Neural Computing and Applications*, p. 1–19, 2021.

[33] P. R. Lewis, P. Marrow and X. Yao, "Evolutionary market agents and heterogeneous service providers: Achieving desired resource allocations," in *2009 IEEE Congress on Evolutionary Computation*, 2009.

[34] G. Serra, Z. Xu, M. Niepert, C. Lawrence, P. Tino and X. Yao, "Interpreting Node Embedding with Text-labeled Graphs," in *International Joint Conference on Neural Networks (IJCNN)*, 2021.

[35] A. Koç, I. Utlu, L. K. Senel and H. M. Ozaktas, "Imparting Interpretability to Word Embeddings," *arXiv preprint arXiv:1807.07279*, 2018.

[36] S. Al-Sayouri, E. Gujral, D. Koutra, E. E. Papalexakis and S. S. Lam, "t-pine: Tensor-based predictable and interpretable node embeddings," *Social Network Analysis and Mining*, vol. 10, p. 1–11, 2020.

[37] C. T. Duong, Q. V. H. Nguyen and K. Aberer, "Interpretable node embeddings with mincut loss," 2019.

[38] T. Hofmann, "Probmap–a probabilistic approach for mapping large document collections," *Intelligent Data Analysis*, vol. 4, p. 149–164, 2000.

[39] M. Kusner, Y. Sun, N. Kolkin and K. Weinberger, "From word embeddings to document distances," in *International conference on machine learning*, 2015.

[40] I. Lage, E. Chen, J. He, M. Narayanan, B. Kim, S. Gershman and F. Doshi-Velez, "An evaluation of the human-interpretability of explanation," *arXiv preprint arXiv:1902.00006*, 2019.

[41] L. v. d. Maaten and G. Hinton, "Visualizing data using t-SNE," *Journal of machine learning research*, vol. 9, p. 2579–2605, 2008.

[42] G. Barlacchi, M. De Nadai, R. Larcher, A. Casella, C. Chitic, G. Torrisi, F. Antonelli, A. Vespignani, A. Pentland and B. Lepri, "A multi-source dataset of urban life in the city of Milan and the Province of Trentino," *Scientific data*, vol. 2, p. 1–15, 2015.

[43] Z. Du, Y. Yang, Z. Ertem, C. Gao, L. Huang, Q. Huang and Y. Bai, "Inter-urban mobility via cellular position tracking in the southeast Songliao Basin, Northeast China," *Scientific data*, vol. 6, p. 1–6, 2019.

[44] R. T. Rodoshi, T. Kim and W. Choi, "Deep reinforcement learning based dynamic resource allocation in cloud radio access networks," in *2020 International Conference on Information and Communication Technology Convergence (ICTC)*, 2020.

[45] J. Blank and K. Deb, "pymoo: Multi-objective optimization in python," *IEEE Access*, vol. 8, p. 89497–89509, 2020.

[46] P. Schmidt and F. Biessmann, "Quantifying Interpretability and Trust in Machine Learning Systems," *arXiv preprint arXiv:1901.08558*, 2019.

[47] T. Mikolov, I. Sutskever, K. Chen, G. S. Corrado and J. Dean, "Distributed representations of words and phrases and their compositionality," in *Advances in neural information processing systems*, 2013.

[48] X. Yan, J. Guo, Y. Lan and X. Cheng, "A biterm topic model for short texts," in *Proceedings of the 22nd international conference on World Wide Web*, 2013.

[49] X. Chen, Y. Jin, S. Qiang, W. Hu and K. Jiang, "Analyzing and modeling spatio-temporal dependence of cellular traffic at city scale," in *2015 IEEE international conference on communications (ICC)*, 2015.