EA for Hyperparameter Optimization of Graph Neural Networks: a comparative study

Bio-Inspired Artificial Intelligence (Professor G. Iacca)

Stefano Pardini, Mattia Florio

***Abstract*—Deep Learning models have received increasing attention in recent years, due to their ability to perform end-to-end representation learning, with important generalization capabilities. As a natural consequence, academia and industry tried to find different ways to reduce the difficulties related to their adoption, giving birth to what is called *Automated Machine Learning (AutoML)*, that has demonstrated its powerful functions in hyperparameter optimization.**

The only limitation related to this kind of neural networks (e.g Convolutional Neural Networks) is that they can handle only Euclidean data, so data whose underlying structure is an Euclidean space (grid-like representation). On the other hand, Real World scenarios are better represented through Networks (non-Euclidean representations like graphs), and so a new kind of Artificial Neural Network model was invented, in order to process this kind of graph-data: *the Graph Neural Network (GNN)* model.

Combining these concepts, *Automated Graph Learning* tries to solve this problem of *Hyperparameter Optimization (HPO)* for Graph Neural Networks models. There is currently only one open-source framework for automated graph learning: *AutoGL*. For the AutoGL, traditional AutoML optimization algorithms such as grid search and random search are used for HPO.

Our work tries to introduce other kinds of optimization mechanisms, applying *Evolutionary Algorithms (EA)* to AutoGL, as a class of nature-inspired population-based stochastic search algorithms. The experimental results show that EAs could be an effective alternative to the hyperparameter optimization of GNN models.

***Index Terms*—Evolutionary Algorithm, Graph Neural Network, Graph Convolutional Network, Auto Graph Learning, Auto Machine Learning, Hyperparameter Optimization**

I. Introduction

Automated machine learning (AutoML) refers to the automation of the entire machine learning process from model construction to application. Most of the existing AutoML frameworks do not consider the particularity of graph data, therefore they cannot be directly applied to graph neural network models. Many real-world scenarios can be naturally modeled into graph machine learning, such as social media analysis and protein modeling. Although traditional deep learning methods have achieved great success, their performance in processing non-Euclidean data may be unsatisfactory. Therefore, **Graph Neural Networks (GNN)** have been proposed to specifically process graph data.

Automatic Graph Learning (AutoGL) refers to AutoML for graph data. AutoGL is the first open-source library of this kind [1]. **Hyperparameter Optimization (HPO)** algorithms mainly include grid search and random search. However, each HPO algorithm adopted in AutoGL has its own disadvantages. Therefore, we require more HPO options for the AutoGL. In this work, we attempted to apply **Evolutionary Algorithms (EA)** to AutoGL, as a class of nature-inspired population-based stochastic search algorithms. Because searching is based on a population of candidate solutions, instead of a single one, EAs have good adaptability and robustness. Therefore, EAs have been widely used to solve various optimization problems in practical applications. [5]

This study focuses on analyzing the performances of different EAs for HPO of GNN models. The EAs adopted are: *Genetic Algorithm (GA), Differential Evolution (DE), Evolution Strategies (ES) and Particle Swarm Optimization (PSO)*. Both the topology and learning parameters of GNNs were encoded into chromosomes for optimization; the problem solved by the GNN model considered, the **Graph Convolutional Network (GCN)**, is a Semi-Supervised Node Classification task, performed on the 2 benchmark datasets: CORA and CITESEER Datasets. Our aim is to show that EAs could be an effective alternative to the HPO of GNN models.

Before analysing our experiments, it is necessary to briefly introduce the main concepts related to Graph Neural Networks, and on the specific kind of learning that is performed.

A. Graph-Based Semi-Supervised Learning (Node Classification)

We consider the problem of classifying nodes (documents in our case) in a graph (such as a citation network, like CORA and CITESEER), where labels are only available for a small subset of nodes. This problem can be framed as **Graph-Based Semi-Supervised Learning**. In general, Semi-supervised learning aims to leverage unlabeled data to improve performance. A large number of semi-supervised learning algorithms jointly optimize two training objective functions: the supervised loss over labeled data and the unsupervised loss over both labeled and unlabeled data.

In graph-based semi-supervised learning the label information is smoothed over the graph via some form of explicit graph-based regularization [3], e.g. by using a graph Laplacian regularization term in the loss function. In fact, the loss function is typically defined as a weighted sum of the supervised loss over labeled instances and a graph Laplacian regularization term:

$$\mathcal{L} = \mathcal{L}_0 + \lambda \mathcal{L}_{reg} \tag{1}$$

$$\mathcal{L}_{reg} = \sum_{i,j} A_{i,j} ||f(X_i) - f(X_j)||^2 = f(X)^T \Delta f(X) \tag{2}$$

Here, $\mathcal{L}_0$ denotes the supervised loss w.r.t. the labeled part of the graph, $f(\cdot)$ can be a neural network differentiable function, λ is a weighting factor and X is a matrix of node feature vectors X_i. $\Delta = D - A$ denotes the unnormalized graph Laplacian of an undirected graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ with N nodes $v_i \in \mathcal{V}$, edges $(v_i, v_j) \in \mathcal{E}$, an adjacency matrix $A \in \mathbb{R}^{N \times N}$ (binary or weighted) and a degree matrix $D_{ii} = \sum_j A_{ij}$. [4]

The graph Laplacian regularization is based on the assumption that nearby nodes in a graph are likely to have the same labels. This assumption, however, might restrict modeling capacity, as graph edges need not necessarily encode node similarity, but could contain additional information. The GCN model solves a Semi-Supervised learning problem encoding the graph structure directly using a neural network model $f(X; A)$ and training on a supervised target $\mathcal{L}_0$ for all nodes with labels, thereby avoiding explicit graph-based regularization in the loss function (the Laplacian regularization term). Conditioning $f(\cdot)$ on the adjacency matrix of the graph will allow the model to distribute gradient information from the supervised loss L_0 and will enable it to learn representations of nodes both with and without labels. It is expected that this setting to be especially powerful in scenarios where the adjacency matrix contains information not present in the data X, such as citation links between documents in the citation network [4].

To conclude this brief introduction on the kind of learning performed, the Loss Function for this semi-supervised multiclass classification task, is a cross-entropy error over all labeled examples:

$$\mathcal{L} = - \sum_{l \in y_L} \sum_{f=1}^{F} Y_{lf} \ln Z_{lf}, \tag{3}$$

where F denotes the feature maps in the output layer, Z is the softmax activation function, defined as $softmax(x_i) = \frac{1}{\mathcal{Z}} exp(x_i)$ with $\mathcal{Z} = \sum_i exp(x_i)$, y_L is the set of node indices that have labels.

II. The Proposed Work

The open-source toolkit **AutoGL** shares various HPO algorithms for GNNs, such as grid search and random search. However, there are disadvantages:

- Grid Search: tries every possible combination of hyperparameters. *Result*: Combinatorial explosion.
- Random Search: tries random combination of hyperparameters. *Result*: Random performance.

EAs have been widely used to solve various optimization problems. In this work, EA is applied to HPO for GNNs to attempt to address the above disadvantages:

- Because some existing HPO algorithms use a single individual for optimization, the final solution has a certain dependence on the initial solution. EA is introduced into HPO algorithms because it searches the solution based on a population of candidate solutions, instead of a single one, owing to its good adaptability and robustness.
- HPO algorithms using EA obtain high-quality individuals through population evolution based on heuristic information.

A. Evolutionary Algorithms: a brief overview

1) Genetic Algorithms (GA): these are the first Evolutionary Algorithms that has been designed.

2) Evolution Strategies (ES): they are modern variants of Evolution Algorithms, where each individual encodes:

- a vector of the decision variables;
- a set of strategy parameters.

The novel contribution of this family of Evolutionary Algorithms is that they somehow try to co-evolve the decision variables (which characterise the specific problem) with the hyperparameters of the EA itself. The result is what can be considered as a form of *Meta-Evolution*: the algorithm tries to model the evolution of the evolution.

3) Differential Evolution (DE): it has intermediate features between Evolutionary Algorithms and Swarm Intelligence. Selection is implemented as "one-to-one spawing", i.e. 1 parent generates 1 offspring and competition occurs only between the parent and the offspring itself. It works because it has a double behaviour: Explorative at the beginning of the optimization process, Exploitative at the end.

4) Particle Swarm Optimization (PSO): it is an instance of Swarm Intelligence algorithms, where each candidate solution has basically 2 sources of information:

- the information coming from its neighbours;
- the information of its own past.

The result is the emergence of a complex behaviour, because the local interaction of the (simple) particles results in a global flow of the information through the entire swarm.

B. Optimized Hyper-Parameters and (Bounded) Search Space

In the tables below the optimized parameters, and their respective search spaces, are shown. It's important to mention that they were set following the indications of a work similar to this one [5].

TABLE I: Search space of continuous parameters.

	Parameter name	Type	Min	Max	Scaling type
P_1	Learning Rate	float	0.01	0.05	Log
P_2	Weight Decay	float	0.0001	0.001	Log
P_3	Dropout Rate	float	0.2	0.8	Linear

TABLE II: Search space of discrete parameters.

	Parameter name	Searching space
H_1	Number of Hidden Units	{4, 5,..., 16}
P_4	Activation Function	{leaky relu, relu, elu, tanh}
P_5	Max Epoch	{100, 101,..., 300}
P_6	Early Stopping	{10, 11,..., 30}

The number of layers in the convolution structure was fixed, and only the number of units in the *hidden layer* H_1 was optimized. It is because when the number of layers is above two, the effect is not greatly improved, and when the number of layers is too high, the training effect is significantly reduced [4].

C. Encoding

The encoding method of the hyperparameters is given in this subsection. The encoding process contains the following two parts:

- the encoding of the topological structure of the network
- the encoding of the parameters in the training process

where the former is embodied only in the coding of the number of hidden units (and, as we said, the number of layers is fixed).

To better participate in the search in the EA, the data types of each parameter were re-coded and the upper and lower limits were set (tables I and II):

- discrete parameters, such as categories and integers, were re-coded as floating-point numbers to facilitate the operation of the EA;
- some parameters were logged before they participated in the optimization of the EA.

At each evaluation, it was converted from the floating-point number to the original data type through a certain decoder and within the valid range.

The chromosome of the GCN model is represented as:

$$f = \{H_1^*, P_1^*, P_2^*, P_3, P_4, P_5, P_6\} \tag{4}$$

where H_1^* represents the number of hidden units H_1 transformed as $\ln H_1$. P_1^* represents the learning rate P_1 transformed as $\ln P_1$. P_2^* represents the weight decay P_2 been transformed as $\ln P_2$. f represents the individuals of this set of hyperparameters.

D. Fitness Evaluation

The objective of HPO algorithms is to optimize the individual fitness, where the individual represents the encoded chromosome regarding the hyperparameters. The following equations are used to evaluate the fitness of individual:

$$fitness(I_i) = loss(Model_{opt}(i)) \tag{5}$$

where: I_i denotes the individual; $fitness(I_i)$ denotes the fitness function of individual I_i; $loss()$ denotes the loss function, i.e. the cross-entropy loss according to Eq. 3. $Model_{opt}(i)$ is shown in following equation:

$$Model_{opt}(i) = model.train(decode(f_i)) \tag{6}$$

where $Model_{opt}(i)$ denotes the optimized model after the training process using the hyperparameter setting I_i. f_i denotes the encoded chromosome according to Eq. 4. $decode(f_i)$ denotes the reverse process of encoding, reversing all the values in the chromosome into the original format to facilitate the construction of the model.

III. Experiments

All the experiments have been performed using:

- *AutoGL*, as open-source library for Automated Graph Learning[1];
- *Inspyred*, as Python library for Bio-Inspired algorithms[2];
- *pycma*, as a specific Python implementation of the CMA-ES algorithm[3].

[1]https://github.com/THUMNLab/AutoGL

A. Dataset

As mentioned in the first section, our experiments were performed on the 2 citation networks: *CORA* and *CITESEER*. The statistics are shown in the table III:

TABLE III: Dataset Statistics

Dataset	Classes	Nodes	Edges	Features
CORA	7	2708	5429	1433
CITESEER	6	3327	4732	3703

We have kept the default train-val-test split computed internally by *AutoGL*: 20 training nodes per class (e.g.: 140 nodes in the case of CORA dataset, 20 randomly sampled from each class); 500 validation nodes; 1000 test samples.

B. EA Parameter Setting

Here we show the different parameter settings of each Evolutionary Algorithm:

TABLE IV: Genetic Algorithm (GA) parameters

	Population Size	Evaluations	Crossover	Mutation	Type
GA	30	450	0.95	0.05	Exploitative
GA	30	450	0.85	0.1	Neutral
GA	30	450	0.7	0.2	Explorative

Termination Criteria = Max number of Evaluations

TABLE V: Differential Evolution (DE) parameters

	Population Size	Generations	Crossover	Mutation	Type
DE	30	15	0.95	0.05	Exploitative
DE	30	15	0.85	0.1	Neutral
DE	30	15	0.7	0.2	Explorative

Termination Criteria = Max number of Generations

TABLE VI: Evolution Strategies (ES) parameters

	Population Size	Evaluations	Type
ES-plus	30	450	Exploitative
ES-comma	30	450	Explorative
CMA-ES	30	450	Neutral

Termination Criteria = Max number of Evaluations

TABLE VII: Particle Swarm Optimization (PSO) parameters

	Population Size	Evaluations	Inertia	Social Rate	Cognitive Rate	Type
PSO	30	450	0.7	3.0	0	Exploitative
PSO	30	450	0.7	0	3.0	Explorative
PSO	30	450	0.7	1.5	1.5	Neutral

Termination Criteria = Max number of Evaluations

In some cases the termination criteria are different as a consequence of the evolutionary algorithm implementation. As a final result, in each experiment, the **number of generations** is equal to **15**.

[2]https://github.com/aarongarrett/inspyred

[3]https://github.com/CMA-ES/pycma

In total we have 12 evolutionary algorithms. To reduce the stochastic effect on the results, we have run each of them 5 times, averaging the results element-wise. At the end, the total number of experiments is:

12 EA x *2 Datasets* x *5 runs = 120 experiments*

C. Performance Monitoring Metrics

1) Accuracy:

$$Accuracy = \frac{TP + TN}{TP + TN + FP + FN} \tag{7}$$

where TP, FP, TN, FN denote the numbers of true positives, false positives, true negatives and false negatives, respectively.

2) Diversity: Let point p have Cartesian coordinates (p_1, p_2) and let point q have coordinates (q_1, q_2). Then the distance between p and q is given by:

$$d(p, q) = \sqrt{(p_1 - q_1)^2 + (p_2 - q_2)^2 + \cdots + (p_n - q_n)^2} \tag{8}$$

The *diversity* is computed as an average over all the individuals in the population.

IV. Analysis

In this section, the performance of HPO with EAs is examined in the following aspects:

- the performance comparison between the HPO with various EAs is analyzed, using as metrics the Accuracy and the Diversity;
- the decoded GNN parameters of the best individual, for each EA, are compared, trying to find some common patterns regarding the optimal solution;
- the performance comparison between the HPO with various EAs is analyzed, using as a grouping factor the different algorithmic approach, categorized a-priori in one the following scenario: *Exploitative*, *Neutral* and *Explorative*.

To solve formatting issues, all the plots are inserted in the Appendix A and Appendix B with regard to Cora and Citeseer datasets respectively.

A. Accuracy vs Diversity

Please refer to 1 for CORA and to 4 for CITESEER. Here we have omitted the plots regarding PSO because we have encountered some problems related to the value ranges of the Diversity measure.

Genetic Algorithms and *Differential Evolution* algorithms, considering the 3 different approaches, behave almost in the same way: the Accuracy trend shows a sort of increasing monotonic behaviour, during generations, while the Diversity decreases with some fluctuations. Some considerations can be made looking to the 3 Evolution Strategies algorithms. In the *ES-comma* case, it seems that optimization doesn't work at all, because the Accuracy decreases over generations, and of course this is not desirable. While on the other hand, *ES-plus* works as expected.

The most interesting behaviour regards the *CMA-ES*: while the Accuracy function increases over time, the Diversity measure shows a similar trend as well. From the theory we know that it is a fundamental property which tells whether the population has potential for further evolution, helping to identify *Stagnation* or *Premature Convergence* situations occurred during the optimization process. In this case, it tells us that better results, in terms of Accuracy, would probably have been obtained with more Generations.

To sum up this first analysis, we consider *CMA-ES* as the best algorithm among the ones considered. Because the different EAs have comparable results in terms of Accuracy in the final population of candidate individuals, *CMA-ES* is the only one that preserves diversity among the individuals, so being for eventual improvements.

B. Architecture Comparison

Please refer to 2 for CORA and to 5 for CITESEER.

We've decided to compare the different sets of hyperparameters (the best candidate solution in their respective final populations) found by each EA. To perform this kind of analysis, we use a *Parallel Coordinates plot*, because this type of visualisation is used for plotting multivariate data. This plot is ideal for comparing many variables together and seeing the relationships between them.

The only clear pattern that emerges regards the *Early Stopping* parameter. In both cases (Cora and Citeseer), its optimal value ranges between 20 and 40, so avoiding high values.

The plot 2 related to Cora shows some weak patterns on: *learning rate*, *hidden nodes* and *dropout* parameters. The optimal *learning rate* found by the best EAs (in terms of Test Accuracy) approaches low values in the (bounded) search space; while the number of *hidden nodes* in the Convolutional layer of the GCN model has high optimal values, among the possible ones. Then *dropout rate* is optimized searching for high values (between 0.7 and 0.8) as well.

Let's now focus briefly on the plot 5 related to Citeseer. Also in this case, apart the already mentioned *Early Stopping* parameter, there are only weak patterns, referred especially to the *hidden nodes* and *weight decay*. The former is optimized searching for high values in the (bounded) search space. The latter should be the highest possible as well.

Although the results are not so significant, we can summarize this analysis section underlining the fact that different sets of GCN hyperparameters can lead to almost equal good results.

C. Which algorithmic approach is better?

Please refer to 3 for Cora and to 6 for Citeseer.

In both cases, the difference, in terms of Test Accuracy of the best individuals of the different algorithms, is negligible, showing a difference of the worst-best algorithms around 2%. Keeping in mind this, a more Explorative algorithmic approach seems to be preferable on the Cora dataset, while an Exploitative method should be indicated on Citeseer. Focusing on each subgroup, so considering the 4 algorithms in each "approach"

category, we aren't able to find consistent behaviour of the 4 considered evolutionary algorithms. For example, considering the Cora plot 3, the *DE-Explorative* performs better than the *PSO-Cognitive* while on the contrary the *PSO-Social* leads to a better result with respect to *DE-Exploitative*. The same conflicting results can be seen on the Citeseer plot 6.

V. Conclusion

A. Further Improvements

Many improvements can be made to this work:

- in order to check the differences in terms of Accuracy and computational resources required, the same experiments should have been conducted using the traditional HPO algorithms, already provided by the library *AutoGL*, so using *Grid Search* and *Random Search*.
- Higher *Population Size* and *number of Generations* could have been imposed on the EA optimization. Because of limited computational resources, it wasn't possible in our case.
- a different kind of optimization could have been made, considering the family of **Fitness-Free Evolutionary Algorithms**, in order to guide the search process towards Diversity and Novelty, rather than the optimality of the Fitness function. For example using a *Multidimensional Archive of Phenotypic Elites* (**Map-Elites**), choosing different behavioural features as projecting dimensions.

B. Final Considerations

In these experiments, we wanted to show the validity of Evolutionary Algorithms as alternative methods to perform hyperparameter optimization on Graph Neural Network models. We have conducted many experiments to test the performance of different EAs.

The experimental results show that the EAs could be an effective alternative to perform this kind of learning (optimizaton). As mentioned above, a lot of improvements can be done to obtain more consistent findings. Our ultimate goal was to combine these 2 powerful methods **EA** and **GNN**, in order to learn something new and very interesting.

References

[1] C. Guan et al.: *AutoGL: a library for automated Graph Learning*, pp. 1–8, 2021
[2] F. Scarselli et al: *The Graph Neural Network Model*, IEEE Transactions on Neural Networks, pp. 61–80, 2009
[3] Z. Yang et al: *Revisiting semi-supervised learning with graph embeddings*, International Conference on Machine Learning (ICML), 2016
[4] T. Kipf and M. Welling: *Semi-supervised classification with graph convolutional networks*, International Conference on Machine Learning (ICML), 2016
[5] C. Bu et al: *Automatic Graph Learning with Evolutionary Algorithms: An Experimental Study*, Pacific Rim International Conference on Artificial Intelligence: Trends in Artificial Intelligence, pp. 513–526, 2021

APPENDIX A
DATA ANALYSIS - CORA

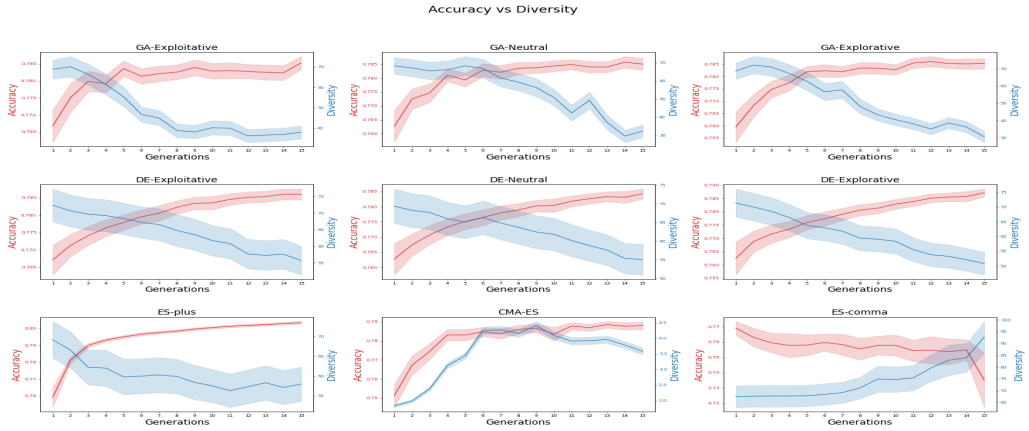

Fig. 1: Performance Monitoring on CORA dataset: Accuracy vs Diversity

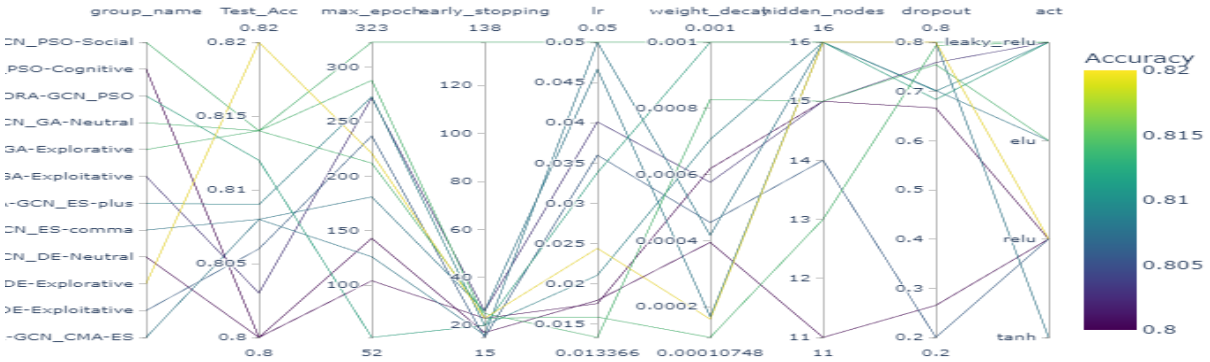

Fig. 2: Comparison on the final GCN architectures - CORA dataset

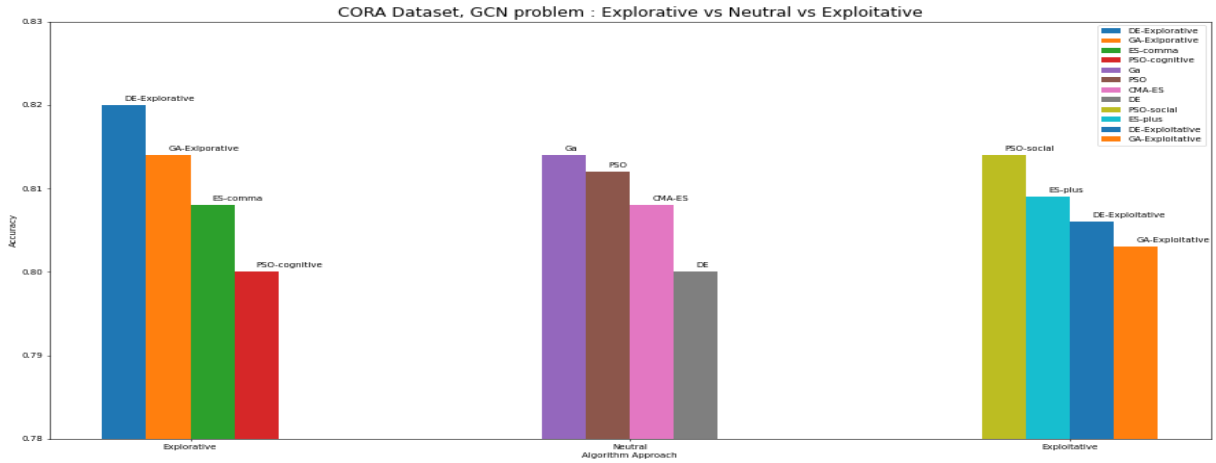

Fig. 3: Test Accuracy between different algorithmic approaches - CORA dataset

APPENDIX B
DATA ANALYSIS - CITESEER

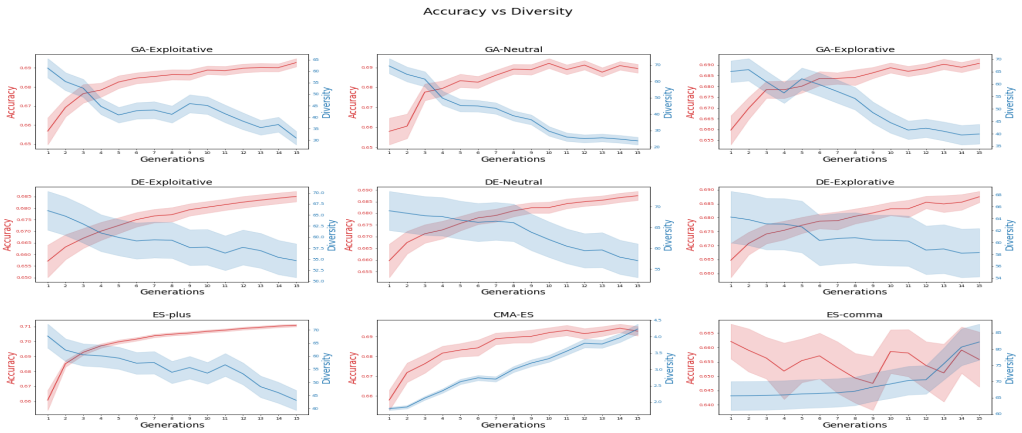

Fig. 4: Performance Monitoring on CITESEER dataset: Accuracy vs Diversity

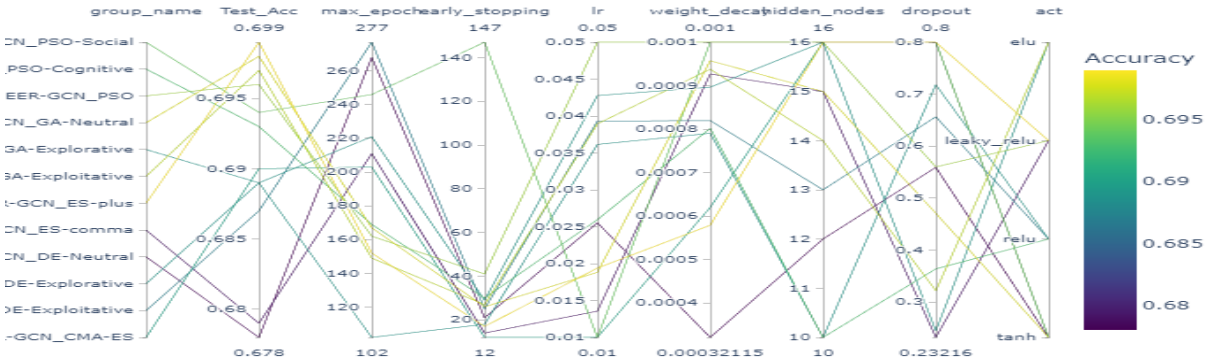

Fig. 5: Comparison on the final GCN architectures - CITESEER dataset

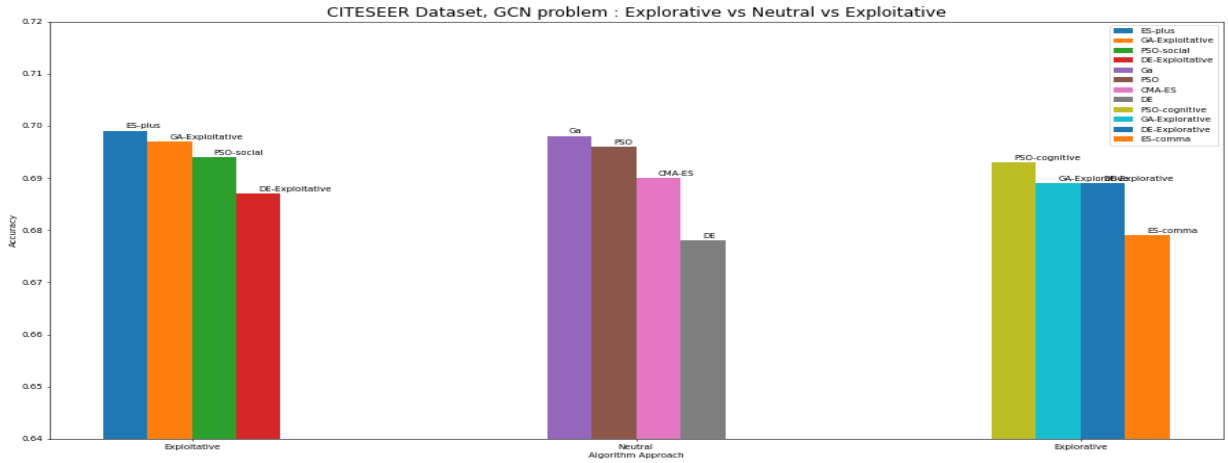

Fig. 6: Test Accuracy between different algorithmic approaches - CITESEER dataset

APPENDIX C
CONTRIBUTION

- **Stefano Pardini**: *Architecture Design*, *AutoGL-Inspyred code integration*, *experiments execution*
- **Mattia Florio**: *selection of literature resources*, *WandB Data Extraction*, *Data Analysis*