Trajectory-based Graph Neural Architecture Search

Abstract

Graph Neural Networks (GNN) have recently come to the forefront displaying superior performance on graph-structured data, and their potential has been exhibited in many real world applications. In spite of this, and similarly to other Deep Learning (DL) architectures, GNNs also come with a tedious but essential tuning work required to achieve decent performance on a given dataset. This process is heavily dependant on the specific scenario and requires domain expert knowledge. To overcome these dependencies, Neural Architecture Search (NAS) is the process of automatically finding the optimal neural architecture. In this work we propose a trajectory-based metaheuristic NAS method for GNNs, named Fuzzy Simulated Annealing NAS (FSA-NAS), which can find competitive solutions significantly faster than competitor approaches. Specifically, we design a novel probabilistic search space and tackle NAS as a bi-objective optimization problem. Additionally, we design a fuzzy rule based system to guide the search process. These two pieces are then combined with a variation of the well-known simulated annealing algorithm. Our experiments show that our method can find competitive results, similar to the current state-of-the-art in terms of performance, whilst achieving up to a 60x speedup even when running on low-end hardware.

1 Introduction

Nowadays, the growing availability of data on a large scale, along with the unstoppable development of computational resources, is making machine learning techniques for pattern recognition and data mining the main characters of both academic and industrial landscapes [Murphy, 2022]. In this context, some models based on neural networks excel when it comes to Euclidean data such as images, where, for instance, Convolutional Neural Networks (CNN) have been shown to be the go-to solution for this task [Li *et al.*, 2022]. However, these approaches fail to properly extract patterns from non-Euclidean data, and in particular when represented in the form of graphs.

To address such scenarios, GNNs [Gori *et al.*, 2005; Scarselli *et al.*, 2009] have emerged as a leading solution, demonstrating outstanding performance on graph-structured data. Generally speaking, and regardless of the addressed task, GNNs assume that node features are dependant on both themselves and other nodes belonging to its neighborhood. In other words, there exists a constant information exchange between the nodes in a graph defined by its topology [Veličković, 2023; Wu *et al.*, 2021]. GNNs potential has been demonstrated in many real world applications such as precision agriculture [Vyas and Bandyopadhyay, 2022], traffic flow forecasting [Chen *et al.*, 2021] and product recommendation in Amazon [Virinchi *et al.*, 2023], among many others. In spite of this, and similarly to other DL architectures, GNNs also come with a tedious but essential tuning work required to achieve the optimal performance on a given dataset. This process is heavily dependant on the specific scenario and usually requires domain expert knowledge to achieve reasonably good results.

Neural Architecture Search (NAS) has recently arisen to automate the procedure of finding the optimal neural architecture for a given task [Ren *et al.*, 2021]. A wide range of NAS frameworks have been proposed with great success, allowing for the discovery of new architectures outperforming classical handcrafted models [Elsken *et al.*, 2019]. Most of the effort in NAS has been focused on CNN architectures, where works like ENAS [Pham *et al.*, 2018] achieved state-of-the-art results while significantly reducing runtimes by combining Reinforcement Learning (RL) with parameter sharing. Other strategies such as evolutionary algorithms or Bayesian Optimization (BO) have also shown great success. However, little (but promising) work has been done from the GNN perspective, where most of the proposals make use of RL [Chen *et al.*, 2022; Zhou *et al.*, 2022]. Unlike those, [Shi *et al.*, 2022] chooses an evolutionary approach, namely a genetic algorithm (i.e., a population-based metaheuristic), achieving favorable results. Additionally, it is worth mentioning that NAS techniques developed for other architectures such as CNNs cannot be directly applied to GNNs given the differences between search spaces [Zhou *et al.*, 2022]. As a consequence of this, an explicit effort has been made in order to adapt them to GNNs.

However, the aforementioned approaches, albeit promising, commonly require a huge amount of computational re-

sources. From the best-case scenario of using a few high-end GPUs [Pham *et al.*, 2018; Chen *et al.*, 2022], to nearly a thousand [Zoph and Le, 2017]. Thus, making it nearly impractical to apply such techniques to any real-life scenario.

In this work we propose a novel algorithm based on trajectory-based metaheuristics which can find competitive GNN architectures significantly faster than state-of-the-art competitors. Unlike population-based metaheuristics, such as evolutionary algorithms, trajectory-based strategies handle a single solution per iteration in the search process. This makes them ideal for scenarios where the objective function is expensive to compute, and NAS is one such case. Specifically, we design a novel probabilistic search space and tackle NAS as a bi-objective optimization problem, attending not only to the performance of the architectures but also to their size. Additionally, we design a fuzzy rule based system to guide the search. These two components are subsequently combined with the widely recognized simulated annealing algorithm. Our experiments show that our method can find competitive results whilst requiring significantly less time, even when running on low-end hardware.

The contributions of this paper are summarized as follows:

- To the best of our knowledge, the very first attempt to tackle Graph Neural Architecture Search (GNAS) using a multi-objective trajectory-based framework.
- A fuzzy rule based system which introduces expert knowledge to guide our search process.
- The proposed trajectory-based approach is empirically evaluated, showing a solid potential in the context of GNAS.

2 Related work

2.1 Graph Neural Networks

GNNs were first proposed in [Gori *et al.*, 2005] and essentially work by performing an exchange of information between nodes in the graph. According to [Bronstein *et al.*, 2021], GNNs can be classified in regard of how the information belonging to the neighborhood of a node is combined. More precisely, they define three flavours. Let ϕ, ψ, and $\bigoplus$ be two neural networks and a permutation-invariant aggregator, respectively. Also, let u and v be nodes from a graph, $\mathcal{N}(u)$ the set of neighbors for a given node u, and x_v the feature vector for node v.

Convolutional GNNs [Defferrard *et al.*, 2016; Kipf and Welling, 2017] aggregate neighboring features with fixed weights following Equation 1.

$$h_u = \phi\left(x_u, \bigoplus_{v \in N(u)} c_{uv}\psi(x_v)\right) \tag{1}$$

where c_{uv} is a constant specifying the importance of node u to node's v representation.

Attentional GNNs [Veličković *et al.*, 2018; Brody *et al.*, 2022], on the other hand, do not assume fixed weights between nodes and make use of attention mechanisms to identify the importance a node has for its neighbors, as per Equation 2.

$$h_u = \phi\left(x_u, \bigoplus_{v \in N(u)} a(x_u, x_v)\psi(x_v)\right) \tag{2}$$

where a is a learnable self-attention mechanism that computes the importance coefficients implicitly.

Lastly, message-passing GNNs [Battaglia *et al.*, 2016; Gilmer *et al.*, 2017; Zambaldi *et al.*, 2019] are the most general and powerful flavour in terms of expressive power, encompassing both convolutional and attentional networks, as per Equation 3.

$$h_u = \phi\left(x_u, \bigoplus_{v \in N(u)} \psi(x_u, x_v)\right) \tag{3}$$

here ψ is a learnable message-passing function that computes u's vector sent to v, and the aggregation can be seen as the aforementioned information exchange in the graph.

2.2 Neural Architecture Search

NAS can be defined as the process of automating DL architecture engineering [Elsken *et al.*, 2019]. Interestingly enough, it has shown to be capable of outperforming manually designed architectures in a variety of tasks such as image classification [Zoph *et al.*, 2018; Real *et al.*, 2019] or semantic segmentation [Chen *et al.*, 2018].

NAS frameworks are formally categorized based on three main elements: search space, search strategy and validation strategy. However, contributions are normally grouped according to their search strategy [Elsken *et al.*, 2019]. The three main groups where most of the contributions are concentrated are approaches based on RL, evolutionary algorithms and BO.

RL based approaches [Zoph *et al.*, 2018; Baker *et al.*, 2017; Cai *et al.*, 2018] are the most common ones. They make use of a controller, generally a Recurrent Neural Network (RNN), whose rewards come from the estimated performance of the explored architectures.

Strategies based on evolutionary algorithms maintain a population of architectures which evolves for a certain number of generations through a crossover and mutation operators. These approaches have been used to design and train neural networks for more than 30 years [Miller *et al.*, 1989; Montana and Davis, 1989; Whitley *et al.*, 1990]. [Shi *et al.*, 2022] is an example of this strategy applied to GNNs.

BO approaches compare architectures based on similarity functions. The idea behind this is that well performing architectures will be similar to each other. Contributions like [Domhan *et al.*, 2015] show that these techniques can achieve outstanding results for CNNs.

3 Proposed methods

In this section we introduce a NAS technique for GNNs based on trajectory-based metaheuristic algorithms. First, we present a novel probabilistic search space which evolves during the search process, allowing the search strategy to obtain better architectures. Four neighbor operators are defined as the tools through which the search strategy interacts with

the search space. Secondly, we introduce an evaluation strategy which relies on low-fidelity estimations combined with early stopping. This is then complemented with a fuzzy rule based system which guides the search process by defining a comparison strategy which attends to both the performance of the architectures on the validation set and the size of the obtained networks. Lastly, we propose FSA-NAS, combining the aforementioned elements.

3.1 Search space

Combining ideas from macro [Zhong *et al.*, 2018] and micro-architectures [Zoph and Le, 2017], we propose a hybrid search space in which architectures are represented as a set of consecutive blocks composed by a set of minimal elements. More precisely, we consider the following three elements: a GNN layer from the literature, an activation function and a dropout layer. Additionally, hyperparameters for each one of those elements are also part of the search space. The set of available values for each case is not fixed and can be defined upon the needs of each specific scenario.

Among the hyperparameters for the layers, input and output dimensions work slightly different and will define, similarly to [Zoph and Le, 2017], three different kinds of blocks in regards to the ratio between them: augmenting blocks if the output dimension is bigger than the input and reducing blocks if the input is bigger than the output, and equal blocks if the dimensions are preserved. For the augmenting blocks, the number of output channels will be selected from the interval $[D_{in} + 1,\ 2D_{in}]$ with uniform probability, where D_{in} stands for the number of input channels. It is worth noting that for the input block D_{in} will be the number of features in the data. For the remaining blocks, it will be the number of output channels of the previous block. Regarding reduction blocks, the number output channels will also be selected from the interval $[N,\ max(\lfloor \frac{D_{in}}{2} \rfloor,\ N)]$ with uniform probability, where N is the number of output classes in the data. Needless to say, the output dimension of the last block in the architecture will be forcefully set to N.

Elements building the blocks will have a probability distribution associated with their corresponding set of available values. This distribution will evolve during the search process depending on how promising specific values are; that is, values achieving good results will have their probabilities increased and values achieving bad results will have their probabilities decreased otherwise. Formally, let $\mathcal{E}_{values}$ be the set of available values for element $\mathcal{E}$. For each available value $e_i \in \mathcal{E}_{values}$, we keep track of the number of times it was present in an improving movement, ν_{e_i}. Then, the probability associated with a specific value will be computed as per Equation 4.

$$p_{e_i} = \frac{\nu_{e_i}}{\sum_{e_j \in \mathcal{E}_{values}} \nu_{e_j}} \tag{4}$$

This makes the search space an element to be optimized. It is worth mentioning that the probability update process through which probabilities are increased will take into account the whole structure of the network, meaning that every block in the architecture will have its configuration updated. The aim of this process is to optimize the blocks globally rather than individually, in order to avoid ending up with pieces that work well individually but fail to achieve their objective when working together as a complete structure.

Finally, the mechanism through which the search strategy will interact with the search space will be by sampling or *querying* for new architectures given the learned distributions.

Neighbor operators

The objective of these operators is to introduce local changes to the architectures in order to obtain a similar one which could potentially be better. Thus, the neighbor operator is the tool used by the search strategy to obtain new architectures from the search space.

We propose four neighbor operators which produce a neighboring architecture by mutating the current one. The first increases the depth of the current architecture. If it is the first time the new depth is reached, a default block is generated. The second one reduces depth removing the last block of the current. The third generates a neighboring architecture by substituting a randomly selected block from the current architecture with a newly queried one form the search space, hyperparameters included. The fourth produces a neighboring architecture by randomly selecting a block from the current architecture and quering the search space for a new set of hyperparameters.

Similarly to the rest of the elements of the search space, these operators will be applied with a probability that will change depending on the quality of the architectures they produce.

3.2 Comparison methodology

We tackle NAS as a multi-objective optimization problem [Lu *et al.*, 2020] in which we aim to maximize performance on a validation set while minimizing the size of the network. Introducing the size of the network as an objective to minimize is key in order to avoid oversized networks which provide little performance improvements.

The most common approaches in order to optimize towards several objectives are either defining a mono-objective function combining every objective, or adapting the algorithm to handle several objectives independently by adopting a Pareto-based ranking schema, such as NSGA-II [Deb *et al.*, 2002]. Defining a combined mono-objective function can be a complex task, as the objectives might take values within different intervals, or can be a source of bias if a weighted combination of them is used, thus leading to undesired behavior. Furthermore, combined mono-objective functions normally lead to a loss in explainability. On the other hand, the biggest majority of Pareto-based approaches rely on evolutionary algorithms since they operate over a population, thus enabling them to find a set of optimal solutions, facilitating the solution of multi-objective optimization problems [Deb, 2001]. Evolutionary algorithms suffer, however, from heavy computational costs since a big population of individuals needs to be evaluated. In light of this, and with the objective of defining a mechanism to compare architectures imitating an expert's reasoning, we propose a fuzzy rule based system to achieve this goal. This mechanism not only enables us to tackle NAS

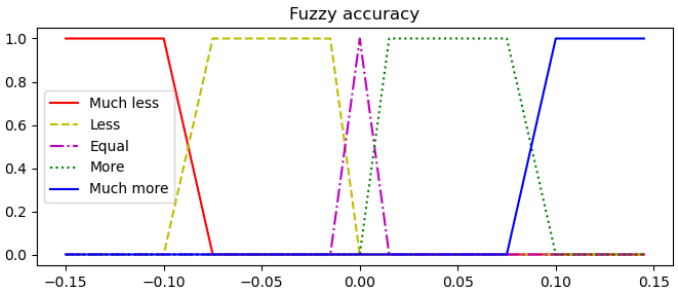

Figure 1: Fuzzy linguistic variable for accuracy

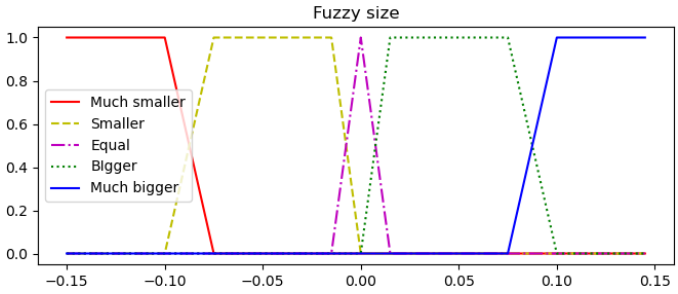

Figure 2: Fuzzy linguistic variable for size

as a bi-objective optimization problem, but also does so in a more explainable manner.

Two fuzzy linguistic variables have been defined in order to characterize the relationship between a candidate architecture and a reference one in terms of quality. The first one relates the accuracy on the validation set, and the second one the size of the models in number of parameters.

The term set for the accuracy is defined in Equation 5 and each term has a fuzzy set associated as depicted in Figure 1.

$$T(Accuracy) = \{ML,\ \text{Much less};\ L,\ \text{Less};\ E,\ \text{Equal};\ M,\ \text{More};\ MM,\ \text{Much more}\} \tag{5}$$

Similarly, the term set for the size is defined according to Equation 6, and its corresponding fuzzy sets are shown in Figure 2.

$$T(Size) = \{MS,\ \text{Much smaller};\ S,\ \text{Smaller};\ E,\ \text{Equal};\ B,\ \text{Bigger};\ MB,\ \text{Much bigger}\} \tag{6}$$

Membership functions in Figures 1 and 2 have been obtained by homogeneously partitioning the input space, a common methodology to define this kind of functions. Although, a more accurate definition could possibly be found by following a fine-tuning approach, this common definition of symmetric functions provides the user explainable results since these functions are highly interpretable.

Also, a variable corresponding to the set of possible decisions given two architectures is defined in Equation 7.

$$T(Decision) = \{RJ,\ \text{Reject};\ RD,\ \text{Redemption};\ NB,\ \text{New best}\} \tag{7}$$

where reject implies that the candidate architecture is considered worse than the reference and thus is rejected, redemption implies that the candidate is slightly worse than the reference but good enough to be considered in order to escape from local optima, and new best implies that the candidate is better than the reference and will be accepted as the new best found solution.

Finally, a set of rules of the form depicted in Equation 8 is defined in Table 1 in order to make a decision given two architectures.

$$r_a \text{ is } t_a \ \wedge\ r_s \text{ is } t_s \implies t_d \tag{8}$$

where t_a, t_s and t_d are fuzzy terms from the variables *Accuracy*, *Size* and *Decision*, respectively, and r_a, r_s measure the differences in terms of quality for both objectives and are computed as follows:

$$r_a = Accuracy_{cand} - Accuracy_{ref} \tag{9}$$

$$r_s = \frac{\#Params_{cand}}{\#Params_{ref}} - 1 \tag{10}$$

Additionally, r_i is t_i indicates that the relation r_i between the architectures is labeled with term t_i. A relation will be assigned the term with maximum membership according to Equation 11.

$$r_i \text{ is } t_i \iff t_i = \arg\max_t \mu_t(r_i) \tag{11}$$

To further clarify this idea, if the relation between candidate and reference for accuracy is labeled as *Equal*, and from the perspective of the size the assigned term is *Much Smaller*, we have the situation of having a candidate architecture which is equal in terms of accuracy but much smaller in size and will be consequently accepted as the corresponding rule indicates.

3.3 Search strategy

The search strategy dictates how the search space is explored. A wide variety of techniques have been studied, from RL using RNN controllers to evolutionary algorithms, going through BO, among the most relevant ones.

Despite achieving outstanding results, we point out several issues for reconsideration: RL based approaches usually rely on a RNN which also needs to be designed, thus converging to initial problem of finding an appropriate neural architecture. BO approaches also commonly rely on an surrogate model which aims to estimate the performance of the explored architectures, which implies that not only a model needs to be designed but also there must exist data to train it. Evolutionary methods are heavily dependant on the size of the population as diversity is essential to achieve good results. Consequently, a high number of architectures need to be evaluated thus dramatically increasing the computational complexity.

In light of the above, we propose a trajectory-based metaheuristic method. Namely, an adaptation of the classic simulated annealing algorithm. The following advantages are highlighted: it is no longer needed neither to design a surrogate model nor to collect data for its training and the amount of evaluations required is significantly reduced as only one architecture is explored at each step.

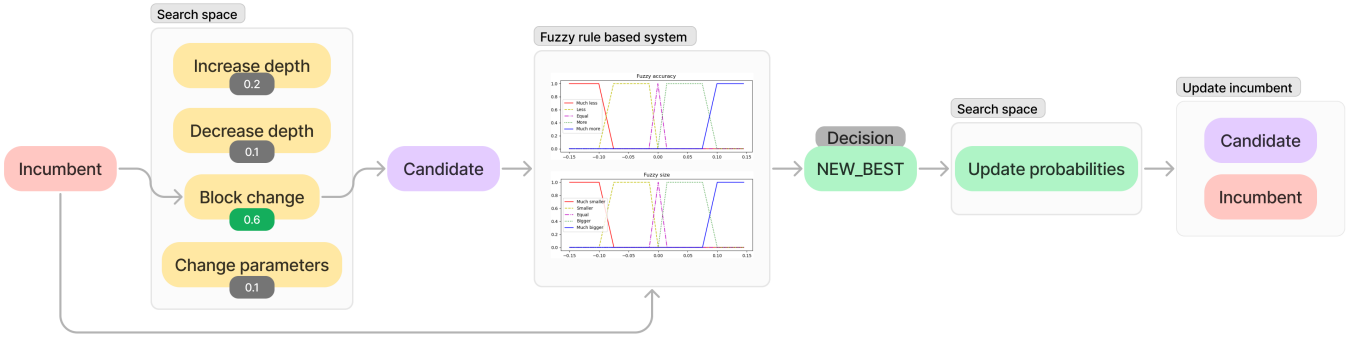

Figure 3: Schematic representation of an FSA-NAS flow in which a new improving candidate is generated and accepted. More precisely, lines 7 to 12 from Algorithm 1 are represented. The process of comparing the new incumbent with the current optimum is identical and thus not included.

IF		THEN
t_a	t_s	t_d
Much Less	Much Smaller	Reject
	Smaller	
	Equal	
	Bigger	
	Much Bigger	
Less	Much Smaller	Redemption
	Smaller	Redemption
	Equal	Redemption
	Bigger	Reject
	Much Bigger	Reject
Equal	Much Smaller	New Best
	Smaller	New Best
	Equal	Redemption
	Bigger	Redemption
	Much Bigger	Redemption
More	Much Smaller	New Best
	Smaller	
	Equal	
	Bigger	
	Much Bigger	
Much More	Much Smaller	New Best
	Smaller	
	Equal	
	Bigger	
	Much Bigger	

Table 1: Rule base

Fuzzy Simulated Annealing

Algorithm 1 shows pseudocode for Fuzzy Simulated Annealing NAS (FSA-NAS), Figure 3 shows a potential flow for the algorithm. It starts by generating and evaluation an initial architecture which is predefined. Then, at each step, a candidate architecture is obtained by mutating the incumbent solution through one of the neighbor operators introduced in Section 3.1 (line 7). Once the accuracy on the validation set is known for the candidate, it is compared to the incumbent by the fuzzy rule system defined in Section 3.2 (line 9). If the decision obtained indicates that the candidate improves the incumbent, then it is accepted and the probabilities of the search space are updated accordingly (lines 11 and 12). Next, the newly obtained incumbent is compared to the best found solution, and if it sets a new optimum, it is updated and probabilities are updated accordingly again (lines 15 and 16). On the other hand, a non-improving architecture can be accepted with a certain probability if the obtained decision is *Redeption*; that is, if the new solution is close enough in terms of quality to the incumbent. The probability of acceptance is reduced as the search reaches its end according to Equation 12.

$$p(i,\ \mathcal{N}) = \frac{\mathcal{N} - i}{\mathcal{N}} \tag{12}$$

where i is the current iteration and $\mathcal{N}$ is the maximum number of iterations. This leads to a search whose behaviour converges from randomness to greediness.

4 Experimental evaluation

In this section we describe the set of experiments conducted to evaluate the quality of our method.

4.1 Methodology

We evaluate our method on a transductive learning setting, namely the widespread citation datasets: Cora [McCallum *et al.*, 2000], Citeseer [Sen *et al.*, 2008] and PubMed [Sen *et al.*, 2008]. The selected data splits are public fixed splits also extracted from [Yang *et al.*, 2016], provided by PyG library.

We conduct a total of 32 runs, exploring 150 architectures in each run. The results reported will be a trimmed mean of the 32 runs, removing the best and the worst cases, to reduce the effect of possible biased executions.

Tables 2 and 3 summarize the selected values composing the search space. The former contains values for the hyperparameters of the architecture while the latter shows the available layers, activation functions and regularization techniques. Additionally, the maximum depth is limited to two blocks.

Algorithm 1 Fuzzy Simulated Annealing

Input: $\mathcal{D}$ (dataset)
Parameter: $\mathcal{N}$ (number of architectures to explore)

$SS \leftarrow$ initializeSearchSpace($\mathcal{D}$)
$\mathcal{A}^* \leftarrow$ generateInitialArchitecture($\mathcal{SS}$)
$\mathcal{A}^*_{acc},\ \mathcal{A}^*_{size} \leftarrow$ evaluateArchitecture($\mathcal{A}^*,\ \mathcal{D}$)
$\mathcal{A}^i \leftarrow \mathcal{A}^*$
$i \leftarrow 0$
while $i < \mathcal{N}$ **do**
  $\mathcal{A}^c \leftarrow$ neighboringOperator($\mathcal{A}^i,\ \mathcal{SS}$)
  $\mathcal{A}^c_{acc},\ \mathcal{A}^c_{size} \leftarrow$ evaluateArchitecture($\mathcal{A}^c,\ \mathcal{D}$)
  $t^i_d \leftarrow$ obtainDecision($\mathcal{A}^c_{acc},\ \mathcal{A}^c_{size},\ \mathcal{A}^i_{acc},\ \mathcal{A}^i_{size}$)
  if $t^i_d ==$ New Best **then**
    $\mathcal{A}^i \leftarrow \mathcal{A}^c$
    updateProbabilities($\mathcal{A}^i,\ \mathcal{SS}$)
    $t^*_d \leftarrow$ obtainDecision($\mathcal{A}^c_{acc},\ \mathcal{A}^c_{size},\ \mathcal{A}^*_{acc},\ \mathcal{A}^*_{size}$)
    if $t^*_d ==$ New Best **then**
      $\mathcal{A}^* \leftarrow \mathcal{A}^c$
      updateProbabilities($\mathcal{A}^*,\ \mathcal{SS}$)
    end if
  else if $t^i_d ==$ Redemption $\wedge$ acceptProbability($i,\ \mathcal{N}$) **then**
    $\mathcal{A}^i \leftarrow \mathcal{A}^c$
  end if
  $i \leftarrow i + 1$
end while
return $\mathcal{A}^*$

Regarding the training setup, every architecture is trained using Adam optimizer with a learning rate of 0.01 and a weight decay of 0.0005. Furthermore, we opt for a low fidelity estimation evaluation strategy, which consists of reducing the number of epochs used during the training of the architectures to evaluate. In that line, the selected values are the following: at most 25 epochs for architectures evaluated during the search process and at most 100 epochs for the training of the best found architecture. In all cases, early stopping will be used in order to prevent models from overfitting and also to contribute to reducing runtimes.

Lastly, we run our experiments in a regular PC with an Intel Core i7-7700K CPU, 16GB of DDR4 RAM and a NVIDIA GTX 1060 GPU with 6GB of GDDR5 VRAM.

Hyperparameters	Search space
Attention heads	1, 2, 3, 4, 5, 6, 7, 8
Filter size (only for ChebConv)	1, 2, 3, 4, 5
Aggregator	*sum*, *mean*, *min*, *max*, *mul*
Readout	*avg*, *concat*
Dropout (nodes neighborhood)	0, 0.05, 0.1, 0.15, 0.2, 0.25, 0.3, 0.4, 0.5, 0.6

Table 2: Search space for hyperparameters

4.2 Results

To validate our method, we compare it with the state-of-the-art in NAS applied to GNNs. Namely, we consider: AGNN [Zhou *et al.*, 2022], GraphNAS [Gao *et al.*, 2020], AGNAS [Chen *et al.*, 2022], Genetic-GNN [Shi *et al.*, 2022], and DSS [Li *et al.*, 2021]. [1]

Results for the node classification task are summarized in Table 4. The focus of the analysis will not be limited to the performance of the architectures on the test set, but also on the efficiency of the search. In order to provide a better understanding of how efficient our proposal is compared to others, we will include runtimes in our comparisons as well as hardware, when available. All data is directly reported from their corresponding works. The best results on each column for both accuracies and runtimes is highlighted in bold.

In terms of accuracy on the test set, we observe that our method is slightly behind the state-of-the-art for all datasets. More precisely, at least 5.7% less for Cora, 6.3% for Citeseer and 2.6% in PubMed. On the other hand, our method significantly outperform the rest in terms of runtime even when running in less powerful hardware. We once again compare, for each dataset, our slowest execution, with DSS which is the quickest one of the rest. Our method achieves a speedup of 67.23x on Cora, 23.09x on Citeseer and 23.12x on PubMed.

Further analysing our results, we point out the relatively high standard deviation for both accuracy and runtimes. We attribute this behavior to the high randomization within the search space.

4.3 Ablation study

To evaluate the influence of the proposed probabilistic search space on the efficiency and effectiveness of our method, we conduct an ablation study in which the process of updating probabilities is disabled; that is, values for the different elements in the search space will have uniform probabilities during the whole search process.

Results for the experiments on all three citation datasets are summarized in Table 5, where accuracy, size (in millions of parameters), and runtimes have been gathered for the best found architectures for each dataset. Interestingly enough, it can be seen that updating the probabilities of the search space does not seem to highly influence the accuracy obtained by the best found architectures (although some minor improvements are observed), but it mainly influences the size of the architecture. More precisely, updating probabilities leads to smaller architectures with similar, slightly better, performance. Also, as a consequence of exploring smaller architectures, the runtimes are slightly reduced. Optimizing the search space by updating the probabilities of the components allows our method to learn that bigger architectures do not necessarily lead to better performance. Consequently, we

[1] Our GTX 1060 6GB has 1280 CUDA cores and a clock speed of 1506 Mhz, whilst the GTX 1080 Ti has 3584, 2.8 times more, a clock speed of 1582MHz and 11GB of GDDR5X VRAM. Similarly, the RTX 2080Ti has 4352 CUDA cores, 3.4 times more, a clock speed of 1640 Mhz and 11GB of GDDR6 VRAM. It is also worth mentioning that [Chen *et al.*, 2022] use 4 of such GPUs. For further details, please refer to NVIDIA's webpage

Component	Search space
Layers	GAT [Veličković *et al.*, 2018], GATv2 [Brody *et al.*, 2022], ChebConv [Defferrard *et al.*, 2016], GCN [Kipf and Welling, 2017], TransformerConv [Shi *et al.*, 2022], GraphConv [Morris *et al.*, 2019]
Activation functions	ReLU, ELU, Sigmoid, Tanh, Softplus
Dropout	0, 0.25, 0.35, 0.45, 0.5, 0.6

Table 3: Search space components

NAS Methods		**Cora**	**Citeseer**	**PubMed**	**GPU**
AGNN	Accuracy	83.6 ± 0.3%	73.8 ± 0.7%	79.7 ± 0.4%	GTX 1080Ti
	Runtime	0.5 days	0.5 days	0.5 days	
GraphNAS	Accuracy	83.7 ± 0.4%	73.5 ± 0.3%	80.5 ± 0.3%	GTX 1080Ti
	Runtime	2 hours	2 hours	9 hours	
AGNAS	Accuracy	83.9 ± 0.4%	**73.9 ± 0.3%**	**80.6 ± 0.3%**	4x RTX 2080Ti
	Runtime	1 hour	1 hour	1 hour	
Genetic-GNN	Accuracy	83.8 ± 0.5%	73.5 ± 0.8%	79.2 ± 0.6%	N/A
	Runtime	N/A	N/A	N/A	
DSS	Accuracy	**83.9 ± 0.3%**	73.3 ± 0.3%	80.3 ± 0.2%	N/A
	Runtime	0.9 hours	0.8 hours	0.9 hours	
FSA-NAS	Accuracy	77.9 ± 2.2%	67.0 ± 2.1%	76.6 ± 1.1%	GTX 1060 6GB
	Runtime	**48.19 ± 26.25 seconds**	**124.70 ± 48.43 seconds**	**140.13 ± 79.92 seconds**	

Table 4: Runtimes for Genetic-GNN are not reported in their work and thus marked as N/A. Similarly, no hardware was specified for Genetic-GNN and DSS either.

		Search space probability update	
		Disabled	Enabled
Cora	Accuracy	77.1 ± 2.70%	**77.9 ± 2.20%**
	Size	0.83 ± 1.21 MM	**0.50 ± 0.59 MM**
	Runtime	68.65 ± 30.06 s	**48.19 ± 26.25 s**
Citeseer	Accuracy	66.7 ± 2.90%	**67.0 ± 2.10%**
	Size	5.39 ± 7.20 MM	**1.91 ± 1.84 MM**
	Runtime	153.40 ± 57.49 s	**124.70 ± 48.43 s**
PubMed	Accuracy	76.4 ± 1.50%	**76.6 ± 1.10%**
	Size	0.18 ± 0.23 MM	**0.12 ± 0.16 MM**
	Runtime	161.74 ± 64.12 s	**140.13 ± 79.92 s**

Table 5: Ablation study comparison. Best results highlighted in bold.

conclude that the proposed probabilistic search space positively contributes to the efficiency of our method.

5 Conclusions and future work

In this work, we introduce a novel NAS method for GNNs named FSA-NAS, based on trajectory-based metaheuristic algorithms. FSA-NAS extends the simulated annealing algorithm by immitating expert behavior through a fuzzy rule-based system. Lastly, we have proposed a novel probabilistic search space, which has been shown to enable our method to discover smaller and more performant architectures.

Our results show that, despite being slightly behind in terms of accuracy on the test set, our method is capable of producing competitive architectures significantly faster than other state-of-the-art GNAS methods, even in low-end hardware. In light of this, we conclude that it offers an outstanding balance between quality and time which can be valuable in a wide range of scenarios.

For future work, our aim is to improve our search space in order to reduce randomness and increase expressiveness. Additionally, we intend to adapt some of the ideas of the proposed method to explore models with different inductive bias, such as CNNs, to further validate the effectiveness and efficiency of trajectory-based metaheuristics for NAS. Finally, we would also like to verify whether our method can be used to quickly obtain a baseline architecture which can be fine tuned using other techniques, similarly to [Tan and Le, 2019].

References

[Baker *et al.*, 2017] Bowen Baker, Otkrist Gupta, Nikhil Naik, and Ramesh Raskar. Designing neural network architectures using reinforcement learning. In *Proceedings of the 10th ICLR*, 2017.

[Battaglia *et al.*, 2016] Peter Battaglia, Razvan Pascanu, Matthew Lai, Danilo Jimenez Rezende, and Koray kavukcuoglu. Interaction Networks for Learning about Objects, Relations and Physics. In *Proceedings of the 30th NeurIPS*, pages 4509–4517, 2016.

[Brody *et al.*, 2022] Shaked Brody, Uri Alon, and Eran Yahav. How attentive are graph attention networks? In *Proceedings of the 10th ICLR*, 2022.

[Bronstein *et al.*, 2021] Michael M. Bronstein, Joan Bruna, Taco Cohen, and Petar Veličković. Geometric deep learning: Grids, groups, graphs, geodesics, and gauges. *ArXiv*, abs/2104.13478, 2021.

[Cai *et al.*, 2018] Han Cai, Tianyao Chen, Weinan Zhang, Yong Yu, and Jun Wang. Efficient architecture search by network transformation. In *Proceedings of the 32th AAAI*, pages 2787–2794, 2018.

[Chen *et al.*, 2018] Liang-Chieh Chen, Maxwell D. Collins, Yukun Zhu, George Papandreou, Barret Zoph, Florian Schroff, Hartwig Adam, and Jonathon Shlens. Searching for efficient multi-scale architectures for dense image prediction. In *Proceedings of the 32nd NeurIPS*, pages 8713–8724, 2018.

[Chen *et al.*, 2021] Xu Chen, Junshan Wang, and Kunqing Xie. TrafficStream: A Streaming Traffic Flow Forecasting Framework Based on Graph Neural Networks and Continual Learning. In *Proceedings of the 30th IJCAI*, pages 3620–3626, 2021.

[Chen *et al.*, 2022] Jiamin Chen, Jianliang Gao, Yibo Chen, Babatounde Moctard Oloulade, Tengfei Lyu, and Zhao Li. Auto-gnas: A parallel graph neural architecture search framework. *IEEE Transactions on Parallel and Distributed Systems*, 33(11):3117–3128, 2022.

[Deb *et al.*, 2002] Kalyanmoy Deb, Amrit Pratap, Sameer Agarwal, and T. Meyarivan. A fast and elitist multiobjective genetic algorithm: NSGA-II. *IEEE Transactions on Evolutionary Computation*, 6(2):182–197, 2002.

[Deb, 2001] Kalyanmoy Deb. Multi-objective optimization using evolutionary algorithms. In *Wiley-Interscience series in systems and optimization*, 2001.

[Defferrard *et al.*, 2016] Michaël Defferrard, Xavier Bresson, and Pierre Vandergheynst. Convolutional neural networks on graphs with fast localized spectral filtering. In *Proceedings of the 30th NeurIPS*, pages 3844–3852, 2016.

[Domhan *et al.*, 2015] Tobias Domhan, Jost Tobias Springenberg, and Frank Hutter. Speeding up automatic hyperparameter optimization of deep neural networks by extrapolation of learning curves. In *Proceedings of the 24th IJCAI*, pages 3460–3468, 2015.

[Elsken *et al.*, 2019] Thomas Elsken, Jan Hendrik Metzen, and Frank Hutter. Neural architecture search: A survey. *Journal of Machine Learning Research*, 20:55:1–55:21, 2019.

[Gao *et al.*, 2020] Yang Gao, Hong Yang, Peng Zhang, Chuan Zhou, and Yue Hu. Graph neural architecture search. In *Proceedings of the 29th IJCAI*, pages 1403–1409, 7 2020.

[Gilmer *et al.*, 2017] Justin Gilmer, Samuel S. Schoenholz, Patrick F. Riley, Oriol Vinyals, and George E. Dahl. Neural message passing for quantum chemistry. In *Proceedings of the 34th ICML*, pages 1263–1272, 2017.

[Gori *et al.*, 2005] Marco Gori, Gabriele Monfardini, and Franco Scarselli. A new model for learning in graph domains. In *Proceedings. 2005 IEEE International Joint Conference on Neural Networks*, volume 2, pages 729–734, 2005.

[Kipf and Welling, 2017] Thomas N. Kipf and Max Welling. Semi-supervised classification with graph convolutional networks. In *Proceedings of the 5th ICLR*, 2017.

[Li *et al.*, 2021] Yanxi Li, Zean Wen, Yunhe Wang, and Chang Xu. One-shot graph neural architecture search with dynamic search space. In *Proceedings of the 35th AAAI*, pages 8510–8517, 2021.

[Li *et al.*, 2022] Zewen Li, Fan Liu, Wenjie Yang, Shouheng Peng, and Jun Zhou. A survey of convolutional neural networks: Analysis, applications, and prospects. *IEEE Transactions on Neural Networks and Learning Systems*, 33(12):6999–7019, 2022.

[Lu *et al.*, 2020] Zhichao Lu, Ian Whalen, Yashesh Dhebar, Kalyanmoy Deb, Erik Goodman, Wolfgang Banzhaf, and Vishnu Naresh Boddeti. NSGA-Net: Neural architecture search using multi-objective genetic algorithm. In *Proceedings of the 29th IJCAI*, pages 4750–4754, 2020.

[McCallum *et al.*, 2000] Andrew Kachites McCallum, Kamal Nigam, Jason Rennie, and Kristie Seymore. Automating the construction of internet portals with machine learning. *Information Retrieval*, pages 127–163, 2000.

[Miller *et al.*, 1989] Geoffrey F. Miller, Peter M. Todd, and Shailesh U. Hegde. Designing neural networks using genetic algorithms. In *Proceedings of the 3th International Conference on Genetic Algorithms*, pages 379–384, 1989.

[Montana and Davis, 1989] David J. Montana and Lawrence Davis. Training feedforward neural networks using genetic algorithms. In *Proceedings of the 11th IJCAI*, pages 762–767, 1989.

[Morris *et al.*, 2019] Christopher Morris, Martin Ritzert, Matthias Fey, William L. Hamilton, Jan Eric Lenssen, Gaurav Rattan, and Martin Grohe. Weisfeiler and Leman Go Neural: Higher-Order Graph Neural Networks. In *Proceedings of the 33th AAAI*, pages 4602–4609, 2019.

[Murphy, 2022] Kevin P. Murphy. *Probabilistic Machine Learning: An introduction*. MIT Press, 2022.

[Pham *et al.*, 2018] Hieu Pham, Melody Guan, Barret Zoph, Quoc Le, and Jeff Dean. Efficient neural architecture

search via parameters sharing. In *Proceedings of the 35th ICML*, pages 4095–4104, 2018.

[Real *et al.*, 2019] Esteban Real, Alok Aggarwal, Yanping Huang, and Quoc V. Le. Regularized evolution for image classifier architecture search. In *Proceedings of the 33th AAAI*, pages 4780–4789, 2019.

[Ren *et al.*, 2021] Pengzhen Ren, Yun Xiao, Xiaojun Chang, Po-yao Huang, Zhihui Li, Xiaojiang Chen, and Xin Wang. A comprehensive survey of neural architecture search: Challenges and solutions. *ACM Comput. Surv.*, 2021.

[Scarselli *et al.*, 2009] Franco Scarselli, Marco Gori, Ah Chung Tsoi, Markus Hagenbuchner, and Gabriele Monfardini. The graph neural network model. *IEEE Transactions on Neural Networks*, 20(1):61–80, 2009.

[Sen *et al.*, 2008] Prithviraj Sen, Galileo Namata, Mustafa Bilgic, Lise Getoor, Brian Galligher, and Tina Eliassi-Rad. Collective classification in network data. *AI Magazine*, 29(3):93, Sep. 2008.

[Shi *et al.*, 2022] Min Shi, Yufei Tang, Xingquan Zhu, Yu Huang, David Wilson, Yuan Zhuang, and Jianxun Liu. Genetic-gnn: Evolutionary architecture search for graph neural networks. *Knowledge-Based Systems*, 247:108752, 2022.

[Tan and Le, 2019] Mingxing Tan and Quoc Le. EfficientNet: Rethinking model scaling for convolutional neural networks. In *Proceedings of the 36th ICML*, pages 6105–6114, 2019.

[Veličković *et al.*, 2018] Petar Veličković, Guillem Cucurull, Arantxa Casanova, Adriana Romero, Pietro Liò, and Yoshua Bengio. Graph Attention Networks. In *Proceedings of the 6th ICLR*, 2018.

[Veličković, 2023] Petar Veličković. Everything is connected: Graph neural networks. *Current Opinion in Structural Biology*, 79:102538, 2023.

[Virinchi *et al.*, 2023] Srinivas Virinchi, Anoop Saladi, and Abhirup Mondal. Recommending related products using graph neural networks in directed graphs. In *Machine Learning and Knowledge Discovery in Databases*, pages 541–557, 2023.

[Vyas and Bandyopadhyay, 2022] Anoushka Vyas and Sambaran Bandyopadhyay. Dynamic structure learning through graph neural network for forecasting soil moisture in precision agriculture. In *Proceedings of the 31st IJCAI*, pages 5185–5191, 2022.

[Whitley *et al.*, 1990] Darrell Whitley, Timothy Starkweather, and Christopher Bogart. Genetic algorithms and neural networks: optimizing connections and connectivity. *Parallel Computing*, 14(3):347–361, 1990.

[Wu *et al.*, 2021] Zonghan Wu, Shirui Pan, Fengwen Chen, Guodong Long, Chengqi Zhang, and Philip S. Yu. A comprehensive survey on graph neural networks. *IEEE Transactions on Neural Networks and Learning Systems*, 32(1):4–24, 2021.

[Yang *et al.*, 2016] Zhilin Yang, William W. Cohen, and Ruslan Salakhutdinov. Revisiting semi-supervised learning with graph embeddings. In *Proceedings of the 33rd ICML*, pages 40–48, 2016.

[Zambaldi *et al.*, 2019] Vinícius Flores Zambaldi, David Raposo, Adam Santoro, Victor Bapst, Yujia Li, Igor Babuschkin, Karl Tuyls, David P. Reichert, Timothy P. Lillicrap, Edward Lockhart, Murray Shanahan, Victoria Langston, Razvan Pascanu, Matthew M. Botvinick, Oriol Vinyals, and Peter W. Battaglia. Deep reinforcement learning with relational inductive biases. In *Proceedings of the 7th ICLR*, 2019.

[Zhong *et al.*, 2018] Zhaobai Zhong, Zichen Yang, Boyang Deng, Junjie Yan, Wei Wu, Jing Shao, and Cheng-Lin Liu. BlockQNN: Efficient Block-Wise Neural Network Architecture Generation. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, pages 2314–2328, 2018.

[Zhou *et al.*, 2022] Kaixiong Zhou, Xiao Huang, Qingquan Song, Rui Chen, and Xia Hu. Auto-GNN: Neural architecture search of graph neural networks. *Frontiers in Big Data*, 2022.

[Zoph and Le, 2017] Barret Zoph and Quoc V. Le. Neural Architecture Search with Reinforcement Learning. In *Proceedings of the 5th ICLR*, 2017.

[Zoph *et al.*, 2018] B. Zoph, V. Vasudevan, J. Shlens, and Q. V. Le. Learning transferable architectures for scalable image recognition. In *Proceedings of 2018 IEEE/CVF CVPR*, pages 8697–8710, 2018.