Towards Synthetic Data Pretraining for Hyperparameter Optimization in Graph Learning

Marek Dědič[1,*], Martin Holeňa[2]

[1]Czech Technical University in Prague, Trojanova 13, Prague, Czech Republic
[2]Czech Academy of Sciences, Pod Vodárenskou věží 271/2, Prague, Czech Republic

Abstract

Hyperparameter optimization (HPO) for Graph Neural Networks (GNNs) remains computationally expensive, and the Cross-RF meta-learning framework has shown promise in accelerating the search by transferring knowledge from prior runs across datasets. However, a key limitation restricts its practical applicability: the requirement for an existing library of completed HPO runs as pre-training data. This work-in-progress paper investigates pre-training the metamodel on synthetic graph datasets – generated via the stochastic block model – as a substitute for real prior runs, enabling transfer learning without any prior experimentation on real graphs.

Keywords

hyperparameter optimization, metalearning, transfer learning, graph neural networks

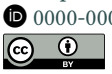
1. Introduction

Graph neural networks (GNNs) have become the tool of choice for learning on the structured data that pervades modern applications – network flows, system call trees, privilege graphs, molecules, and social networks are all most faithfully represented as graphs. Yet the practical performance of a GNN hinges on hyperparameter optimization (HPO), an often overlooked but decisive step: the choice of learning rate, regularization strength, hidden dimensions, or sampling strategy routinely swings model quality by wide margins. Because each candidate configuration requires training a GNN to convergence, HPO is also expensive, and this cost is precisely what motivates methods that can transfer knowledge from previously tuned datasets to new ones. Such transfer, however, has so far relied on a library of previously completed HPO runs on real datasets, which is expensive to assemble and narrow in the range of graph properties it covers. In this work, we propose Synthetic-RF, a metalearning HPO method in which the surrogate predicting the performance of a hyperparameter configuration is pre-trained not on real datasets but on generated graphs, freeing the pre-training phase from the datasets that happen to be available. We describe how such graphs are generated so as to span the properties of interest, and report a preliminary evaluation of the resulting tuner against random search on the Cora dataset.

1.1. Related Work

HPO for graph learning remains a young field, driven by the high cost of GNN training and the complicated interactions between their hyperparameters. A large share of the existing effort falls under Neural Architecture Search (NAS) for GNNs, where GraphNAS [1] and Auto-GNN [2] use reinforcement learning to search the space of GNN architectures. These methods search each dataset from scratch, however, and neither reuses the results of prior searches nor exploits the structure shared across datasets—the very information a pre-trained surrogate is designed to capture.

Studies that tune GNN hyperparameters directly tend to be confined to a single application domain, such as molecular property prediction [3, 4, 5, 6] or mRNA degradation prediction [7]. Because they

ITAT'26: Information Technologies – Applications and Theory, September 25–29, 2026, Biele Karpaty, Slovakia
*Corresponding author.
marek@dedic.eu (M. Dědič); martin@cs.cas.cz (M. Holeňa)
https://dedic.eu/ (M. Dědič); https://www.cs.cas.cz/~martin/ (M. Holeňa)
0000-0003-1021-8428 (M. Dědič); 0000-0002-2536-9328 (M. Holeňa)


draw on only a handful of datasets from one domain, they offer little basis for learning how tuning outcomes transfer across the diverse range of graph structures encountered in practice—the setting our synthetic pre-training approach is meant to address.

Pretraining on synthetic data is not a novel idea – it is one of the core ideas behind recent works such as TabPFN [8]. However, this idea has not been explored in the context of hyperparameter optimization.

1.2. Contributions

Our previous work [9] studies HPO for general GNNs, benchmarks the most commonly used HPO methods for GraphSAGE [10] across a diverse set of graph datasets, and introduces a meta-learning approach for hyperparameter transfer. This work extends that by looking at the hyperparameter transfer problem as a pre-training problem, where we learn a surrogate model that can predict the performance of hyperparameter configurations on unseen datasets based on their meta-features. This allows us to warm-start the HPO process on new datasets, potentially reducing the number of costly evaluations needed to find optimal hyperparameters. In [9], this was done on other real-world datasets, limiting its breadth. In this work, we instead use synthetic datasets in the pre-training phase, which allows us to be more flexible in the pre-training and not be limited by the available real-world datasets.

Section 1.1 reviews related work on HPO methods, particularly in the context of graph learning. Section 2 provides the background of HPO methods. Section 3.1 recalls the Cross-RF HPO method for graph learning from our previous work and describes its limitations. Section 3.2 introduces the Synthetic-RF method, which addresses these limitations by pre-training a surrogate model on synthetic datasets. Section 4 outlines our methodology, hyperparameter spaces, and evaluated methods and reports on the performance of the Synthetic-RF method, comparing its performance to random search. Finally, Section 5 summarizes our findings and discusses future research directions.

2. Problem Definition

Nearly every machine learning model, graph neural networks (GNNs) included, exposes a set of hyperparameters that must be fixed prior to training. Selecting good values for them is the task of hyperparameter optimization (HPO). Although such tuning was long carried out by hand, the field has come to accept that principled, automated HPO yields models that both perform and generalize better. From an optimization standpoint, HPO is often a black-box problem, and an unusually costly one: assessing a single configuration entails training a model and measuring its validation performance.

Our focus throughout is HPO for supervised learning, and we adopt the notation and formalism introduced by [11]. Let $\mathscr{D} \in \mathrm{D}$ be a labelled dataset made up of pairs $(\boldsymbol{x}_i, y_i)$ with features $\boldsymbol{x}_i \in \mathscr{X}$ and labels $y_i \in \mathscr{Y}$, and take its elements to be i.i.d. draws from an unknown joint distribution $\mathrm{P}_{\mathscr{X}\mathscr{Y}}$. When it is convenient to treat the dataset as a whole, we stack its features into the matrix $\boldsymbol{X}_{\mathscr{D}}$ and its labels into the vector $\boldsymbol{y}_{\mathscr{D}}$.

Supervised learning seeks a function $f : \mathscr{X} \to \mathbb{R}^g$ that turns features into predictions and behaves well on data it has not seen, where g equals the number of classes in classification and $g = 1$ in regression. The collection of admissible such functions is the *hypothesis space* $\mathscr{H}$. Producing a concrete model depends on a hyperparameter configuration (i. e. a vector of all hyperparameters) $\lambda \in \mathbf{\Lambda}$, where $\mathbf{\Lambda}$ is the *search space*, and we capture the training procedure as an *inducer* $\mathscr{F} : \mathrm{D} \times \mathbf{\Lambda} \to \mathscr{H}$. Given a dataset $\mathscr{D} \in \mathrm{D}$ and a configuration $\lambda \in \mathbf{\Lambda}$, the inducer returns a fitted model $\hat{f} = \mathscr{F}(\mathscr{D}, \lambda)$, which we occasionally write as $\hat{f}_\lambda$ to stress its dependence on λ; here $\mathrm{D} = \bigcup_{n \in \mathbb{N}} (\mathscr{X} \times \mathscr{Y})^n$ denotes the space of all datasets.

Once a model $\hat{f}$ has been fitted, its quality is judged on fresh data through a performance measure $\rho : \bigcup_{m \in \mathbb{N}} (\mathscr{Y}^m \times \mathbb{R}^{g \times m}) \to \mathbb{R}$. Given the true labels and the corresponding predictions over m samples, ρ returns a scalar score – such as accuracy in a classification setting or mean squared error in a regression one, to name two common choices.

Since the inducer $\mathscr{F}$ is governed by its configuration $\lambda \in \mathbf{\Lambda}$, HPO amounts to locating the configuration λ^* that maximizes ρ on the held-out part of $\mathscr{D}$. In practice, computational limits restrict the search

to a finite subset $\tilde{\Lambda} \subset \mathbf{\Lambda}$, the *search space*. A typical HPO algorithm proceeds iteratively: it picks configurations from $\tilde{\Lambda}$, trains the corresponding models, scores them with ρ, and uses those scores to decide which configurations to try next. We formalize such an algorithm as a tuner $\tau : \left(\mathscr{D}, \mathscr{F}, \tilde{\Lambda}, \rho\right) \to \hat{\lambda}$, which, given a dataset $\mathscr{D}$, an inducer $\mathscr{F}$, a search space $\tilde{\Lambda}$, and a performance measure ρ, outputs an estimate $\hat{\lambda} \in \tilde{\Lambda}$ of the optimum λ^*.

3. RF-based Methods for Graph HPO

In this section, we introduce the metalearning Synthetic-RF method, a novel approach for HPO specifically designed for graph learning, building on our previous metalearning works [9, 12]. This section first introduces the Cross-RF (*cross-dataset random forest*) method from those previous works, followed by the newly-proposed Synthetic-RF method.

3.1. The Cross-RF Method

Assume access to a *graph property descriptor* $\delta : \mathrm{D} \to \mathbb{R}^d$, which encodes a graph dataset $\mathscr{D}$ as a vector of numerical characteristics $\boldsymbol{d}$ such as its node count, edge count, or average degree, thereby summarizing each dataset in a common, structured form. Assume further a *metamodel* $\mathscr{M}_\rho : \mathbb{R}^d \times \mathbf{\Lambda} \to \mathbb{R}$ that, from a dataset's descriptor $\delta(\mathscr{D})$ (its metafeatures) together with a configuration λ, predicts the performance ρ (e.g. accuracy or F1-score) that a model $\hat{f}_\lambda$ trained on $\mathscr{D}$ under λ would attain. With these two ingredients, the Cross-RF tuner is defined as

$$\tau\left(\mathscr{D}, \mathscr{F}, \tilde{\Lambda}, \rho\right) = \arg\max_{\lambda \in \tilde{\Lambda}} \mathscr{M}_\rho\left(\delta(\mathscr{D}), \lambda\right)$$

Whether this is practical hinges on the metamodel $\mathscr{M}_\rho$: besides being accurate, it must be cheap enough to query that scoring every configuration in $\tilde{\Lambda}$ at each optimization step remains affordable.

We realize $\mathscr{M}_\rho$ with a standard regression model – here a random forest (RF) regressor, though any regressor would serve – fitted to a meta-dataset $\mathscr{A}$. Starting from $\mathscr{A} = \emptyset$, we grow $\mathscr{A}$ by recording, as training proceeds, triples of dataset descriptors, the configurations tried, and the performance they achieved:

$$\mathscr{A} = \Big\{\Big(\delta\left(\mathscr{D}^{(1)}\right), \lambda^{(1)}, \rho(\boldsymbol{y}_{\mathscr{D}^{(1)}}, \hat{f}_{\lambda^{(1)}}\left(\boldsymbol{X}_{\mathscr{D}^{(1)}}\right))\Big), \\ \Big(\delta\left(\mathscr{D}^{(2)}\right), \lambda^{(2)}, \rho(\boldsymbol{y}_{\mathscr{D}^{(2)}}, \hat{f}_{\lambda^{(2)}}\left(\boldsymbol{X}_{\mathscr{D}^{(2)}}\right))\Big), \ldots\Big\}$$

Note that if the method is applied to a single learning task, and hence a single dataset, every descriptor $\delta\left(\mathscr{D}^{(i)}\right)$ in $\mathscr{A}$ coincides and contributes nothing for the metamodel to learn from. This degenerate case corresponds to the RF metalearning method of [12]. The Cross-RF method of [9] escapes it by drawing the entries of $\mathscr{A}$ from several datasets, letting $\mathscr{M}_\rho$ learn how performance varies with dataset properties across graphs. This approach also has serious limitations, as the meta-dataset $\mathscr{A}$ is constructed from a limited number of datasets, which may not be representative of the diversity of graph datasets encountered in practice. This can lead to poor generalization performance of the metamodel $\mathscr{M}_\rho$ when applied to new datasets that differ significantly from those seen during training – essentially, the datasets used in the metamodel pretraining might be out-of-distribution for the problem we want to solve.

3.2. Synthetic-RF: A Synthetic Dataset Generation Approach

The Synthetic-RF method addresses the limitations of the Cross-RF method by generating synthetic datasets to populate the meta-dataset $\mathscr{A}$. This approach allows us to create a diverse set of graph datasets that can better represent the variety of possible graph datasets, thus improving the generalization performance of the metamodel $\mathscr{M}_\rho$. In general, we would want the meta-dataset to be broad and

encompass a wide range of graph properties, so that the metamodel can learn to generalize across different types of graphs. However, due to the limited scope of this preliminary work, we focus on generating synthetic datasets that are similar to the particular target dataset $\mathscr{D}$ for which we are optimizing the hyperparameters in terms of their properties. This is a limitation of the current work, but we believe that it is a reasonable starting point for exploring the potential of synthetic dataset generation for graph HPO. In future work, we plan to explore more sophisticated methods for generating synthetic datasets that can better capture the diversity of real-world graph datasets.

In order to generate the synthetic datasets for the metamodel pretraining, we use the stochastic block model (SBM) [13], which is a generative model for random graphs that can produce graphs with community structure. The SBM allows us to control the number of communities, the size of each community, and the probability of edges between nodes within and between communities. By varying these parameters, we can generate a wide range of synthetic graph datasets with different properties.

Concretely, each synthetic dataset is generated so as to mimic the target dataset $\mathscr{D}$ while introducing controlled variability. We first summarize the target dataset by a vector of five statistics

$$\boldsymbol{s} = (n,\ k,\ \bar{c},\ h,\ m)\,,$$

where n is the number of nodes, k the number of classes, $\bar{c}$ the average node degree, $h \in [0,1]$ the edge homophily (the fraction of edges connecting nodes of the same class), and m the number of node features. This vector serves as the baseline around which the synthetic datasets are generated.

To obtain a set of $n_{\mathscr{D}}$ distinct synthetic datasets rather than a single copy of the target, we perturb $\boldsymbol{s}$ independently for each synthetic dataset $i \in \{1, \dots, n_{\mathscr{D}}\}$ (we omit the dataset index to keep the notation simpler). Each of the first four statistics is scaled by an independent multiplicative factor drawn uniformly,

$$\tilde{s}_j = \mathrm{clip}\left(u_j\, s_j\right), \quad u_j \sim \mathscr{U}\left(1-\varepsilon, 1+\varepsilon\right),$$

where ε controls the perturbation strength and clip restricts each statistic to a sensible range: the node count is kept within prescribed bounds $[n_{\min}, n_{\max}]$ chosen to bracket the target, the number of classes lies in $[2, k_{\max}]$, the average degree is at least one, and the edge homophily is confined to $[0.05, 0.95]$ so that neither the homophilous nor the heterophilous regime degenerates. The bounds on the node count matter for a reason particular to our metamodel: a random forest extrapolates poorly outside the range of its training data, so the synthetic datasets must bracket the target in descriptor space rather than lie far from it. The number of node features is not perturbed, being instead set to $\tilde{m} = \max\left(m, \left\lceil \log_2 \tilde{k} \right\rceil + 1\right)$, that is, to the target's feature count, raised only if too low to accommodate one feature cluster per class. This yields a family of perturbed statistics $\tilde{\boldsymbol{s}} = \left(\tilde{n}, \tilde{k}, \tilde{c}, \tilde{h}, \tilde{m}\right)$ whose properties are scattered around those of the target dataset.

The $\tilde{n}$ nodes are partitioned into $\tilde{k}$ blocks, one per class, with block sizes $\boldsymbol{b} = \left(b_1, \dots, b_{\tilde{k}}\right)$ drawn from a symmetric Dirichlet distribution with concentration α and rescaled so that $\sum_c b_c = \tilde{n}$, yielding slightly unequal community sizes. We denote $\bar{b} = \tilde{n}/\tilde{k}$ the average block size. The SBM is parametrized by an intra-block edge probability p_{in} (between nodes of the same community) and an inter-block probability p_{out} (between different communities). We choose

$$p_{\text{in}} = \frac{\tilde{h}\,\tilde{c}}{\bar{b}}, \qquad p_{\text{out}} = \frac{\left(1-\tilde{h}\right)\tilde{c}}{\tilde{n}-\bar{b}},$$

so that two properties of the target are preserved in expectation: the average node degree matches $\tilde{c}$, and the fraction of within-class (homophilous) edges matches $\tilde{h}$. To see this, consider a node in a block of the average size $\bar{b}$; treating the blocks as approximately equal in size, it has $\bar{b} - 1 \approx \bar{b}$ potential neighbours inside its own community and $\tilde{n} - \bar{b}$ potential neighbours outside it, each realized independently with probability p_{in} and p_{out} respectively. Its expected degree is therefore

$$\bar{b}\, p_{\text{in}} + \left(\tilde{n} - \bar{b}\right) p_{\text{out}} = \tilde{h}\,\tilde{c} + \left(1-\tilde{h}\right)\tilde{c} = \tilde{c},$$

recovering the target average degree. Of these edges, the within-community ones contribute $\bar{b}\, p_{\text{in}} = \tilde{h}\,\tilde{c}$ in expectation, a fraction $\tilde{h}\,\tilde{c}/\tilde{c} = \tilde{h}$ of the total, so the expected edge homophily equals $\tilde{h}$ as well. The

edges are then sampled from the SBM as an undirected graph, and each node is assigned the label of its block.

Finally, a feature matrix $\boldsymbol{X} \in \mathbb{R}^{\tilde{n} \times \tilde{m}}$ is generated by drawing each node's feature vector conditionally on its block, so that the features carry class-relevant information analogous to a real attributed graph. We follow the standard construction of [14] for generating clustered classification problems: each class is assigned one isotropic Gaussian cluster whose centre is a vertex of a $\tilde{m}$-dimensional hypercube, and the nodes of the corresponding block are drawn from it. All $\tilde{m}$ features are informative – none are redundant linear combinations, repeated, or pure noise – the class proportions match the block sizes $\boldsymbol{b}$, and no label noise is introduced, so a node's features and its block label are consistent by construction. The features are consequently real-valued and, unlike the sparse binary indicators of many citation networks, dense; the metamodel nevertheless observes both the synthetic and the target datasets only through the descriptor δ, whose attribute-related entries are similarity measures rather than raw feature values. The feature matrix $\boldsymbol{X}$, the sampled graph structure, and the block labels together form a complete synthetic node-classification dataset on which a GNN can be trained to populate the meta-dataset $\mathscr{A}$.

3.3. Dataset Property Descriptor

The descriptor δ is how the metamodel represents a dataset: two datasets with identical descriptors are, as far as $\mathscr{M}_\rho$ is concerned, the same learning problem. Its entries must therefore be quantities along which the response of a GNN to its hyperparameters actually varies, and they must be cheap to evaluate, since computing $\delta(\mathscr{D})$ is the part of the tuning that is meant to replace training runs. Both requirements point towards summary statistics of the graph.

Which statistics those are follows from how a GNN operates. A message-passing model propagates information along edges, so the value of doing this at all – and the depth over which it pays off – is governed by how the labels are distributed over the topology. This is what the various homophily coefficients and the node label informativeness measure. These measures have an effect on the number of layers and the aggregation function: on a strongly homophilous graph, aggregating over a wide neighbourhood reinforces the signal, whereas on a heterophilous one it dilutes it. The degree statistics play a similar role for the width and the sampling of neighbourhoods, since they determine how much information each aggregation step actually pools, and the component count indicates how far information can travel at all. The attribute-related properties capture the complementary question of how much the node features already reveal about the class before any propagation, which is what regularization strength and dropout must be balanced against. For the concrete choice of entries we reuse the feature set of [9].

- **Structure only:** label-agnostic topological descriptors – the node, edge, and component counts, the average node degree, and the global assortativity.
- **Structure and attributes:** the alignment between topology and features, quantified as the mean cosine similarity of the node features across connected pairs.
- **Structure and task:** the bulk of the descriptor, characterizing how topology relates to the node labels. It gathers a range of homophily coefficients (edge, node, class, attribute, and adjusted), node label informativeness, and label-aware connectivity scores (balanced and adjusted accuracy), together with positive-class-specific quantities such as the mean degree of positive nodes, the fraction of positive nodes with degree above one or two, and the prevalence of positive edges.
- **Attributes and task:** how cleanly the features distinguish the classes, captured by the mean within-positive-class attribute similarity and the two cross-class similarities (positive-to-negative and negative-to-positive).

4. Preliminary experimental evaluation of the Synthetic-RF method

4.1. Experimental Setup

We conduct our experiments on the Cora dataset [15], a citation network with 2 708 nodes, 5 429 edges and 1 433 binary features for each node. The task was inductive node classification. We compare our proposed Synthetic-RF method against random search. The methods were both implemented using the Optuna hyperparameter optimization framework [16].

Each HPO method combination was evaluated independently 5 times to account for variability in performance due to random initialization and stochastic training processes. For both methods, we set a patience of 10 – if no improvement in validation performance was observed for 10 consecutive trials, the optimization process was stopped early.

4.2. Synthetic Dataset Generation

The synthetic datasets used to pretrain the metamodel were generated as described in Section 3.2, taking Cora as the target dataset, so that the baseline statistics were $\boldsymbol{s} = (2708, 7, 3.90, 0.81, 1433)$. We generated $n_{\mathscr{D}} = 5$ synthetic datasets, evaluating 30 hyperparameter configurations on each, which yields a meta-dataset $\mathscr{A}$ of 150 observations. The configurations were drawn uniformly at random from the search space described below, without replacement, and the same 30 configurations were used for every synthetic dataset, so that the metamodel observes each of them across the whole range of generated dataset properties.

The perturbation strength was set to $\varepsilon = 0.3$, that is, each mimicked statistic was scaled by a factor drawn uniformly from $[0.7, 1.3]$. The node count was clipped to $[n_{\min}, n_{\max}] = [1500, 4000]$, a range chosen to bracket Cora's 2 708 nodes, and the number of classes to at most $k_{\max} = 20$. The block sizes were drawn from a symmetric Dirichlet distribution with concentration $\alpha = 5$, which keeps the communities of comparable but not identical size. Since Cora's feature count is not perturbed, all synthetic datasets have 1 433 node features, matching the target; note, however, that these are real-valued, whereas Cora's are binary bag-of-words indicators.

4.3. Search Spaces

In our experiments, we used the well-studied and general framework of GraphSAGE [10] as the base GNN architecture for node classification tasks. We defined a search space encompassing 8 hyperparameters that are critical to the performance of GraphSAGE models:

- Number of layers: 1 to 3
- Hidden layer width: 16 to 64
- Aggregation function: mean, max
- Activation function: ReLU, PReLU
- Learning rate: 10^{-2}, 5×10^{-3}, 10^{-3}
- Optimizer: Adam, AdamW
- L_2 regularization strength: 0, 5×10^{-4}, 5×10^{-3}
- Dropout rate: 0, 0.5

The ranges were chosen based on the prior art [10]. All experiments were trained for a maximum of 200 epochs, which was seldomly reached due to the fact that early stopping was employed to prevent overfitting.

4.4. Metamodel Architecture

Fot the metamodel, we used a scikit-learn Random Forest regressor with all parameter values taken as defaults from scikit-learn – most importantly, 100 trees and 30% of all features considered for each split. Several small implementation details are worth mentioning. First, in each step of the optimisation, all

of the already evaluated configurations from $\mathscr{A}$ are excluded from the set of candidate configurations $\tilde{\Lambda}$, so that the same configuration is never needlessly re-evaluated. Second, all categorical features are one-hot encoded before being passed to the Random Forest model. Finally, in all our experiments, we used the F1-score as the performance metric ρ to be optimised.

4.5. Results and Analysis

The preliminary experimental evaluation is limited in nature. Since only one dataset and two methods were evaluated, the results are not sufficient to draw any general conclusions. However, the results do provide some initial insights into the performance of the Synthetic-RF method. On average (across 5 independent runs), Synthetic-RF achieved a final F1 score of 0.8628, while random search achieved an F1 score of 0.8535. This indicates that Synthetic-RF is able to find better hyperparameter configurations than random search, at least on the Cora dataset. However, the margin of improvement is modest at best, and further experiments on additional datasets and with more HPO methods are needed to validate the effectiveness of the Synthetic-RF method.

4.6. Computational Cost

The computational cost of Synthetic-RF splits into two parts: the trials on synthetic data that populate the meta-dataset $\mathscr{A}$, and the trials on the target dataset performed during the optimisation itself. In the setup reported here, the synthetic datasets are generated to model the target dataset and are therefore a synthetic trial costs roughly as much as a trial on the target dataset. Including the pretraining, the method therefore consumes more computation than random search does.

The two parts differ in character, however. The pretraining trials are incurred once, ahead of time and independently of any particular tuning task, and they are mutually independent, so they may be spread across whatever hardware is available; a pretrained metamodel can then be reused for every subsequent tuning task, dividing its cost across all of them. The trials on the target dataset, in contrast, are what the practitioners care about: they are performed at tuning time and, for any sequential HPO method, they are inherently serial, each configuration being chosen based on those before it. It is this budget that pretraining on synthetic data is meant to reduce.

5. Conclusion

In this work, we build upon our recently introduced Cross-RF method for hyperparameter optimization (HPO) of Graph Neural Networks (GNNs). We introduced Synthetic-RF, a HPO metamodel pretrained on synthetically generated graphs similar to the target graph. We described a stochastic-block-model-based strategy for generating synthetic graphs that closely resemble the target graph, yielding a family of graphs that are similar in structure and properties. We demonstrated that Synthetic-RF can be used to warm-start the HPO process.

Our experimental results are preliminary and limited in scope, but they show that Synthetic-RF outperforms random search on the well-known Cora dataset. However, the margin is very small, and we acknowledge that further experiments are needed to validate the effectiveness of Synthetic-RF across a wider range of datasets.

5.1. Future Work

Given the results obtained, the future work needs to focus on a more comprehensive study of the graph generation process, including the exploration of different graph generation techniques and their impact on the performance of Synthetic-RF, and the study of the properties of the generated graphs. Additionally, the limitation of the Synthetic-RF model may be the random forest used as the metamodel. Using stronger metamodels such as the TabPFN may lead to potentially significant improvements.

Acknowledgments

Martin Holeňa's research was partially supported by the German Research Foundation (DFG) project funding no. 441926934.

Declaration on Generative AI

During the preparation of this work, the authors used Claude Opus 4.8 in order to: Drafting content, Paraphrase and reword. After using these tools/services, the authors reviewed and edited the content as needed and take full responsibility for the publication's content.

References

[1] Y. Gao, H. Yang, P. Zhang, C. Zhou, Y. Hu, Graph Neural Architecture Search, in: Proceedings of the Twenty-Ninth International Joint Conference on Artificial Intelligence, volume 2, 2020, pp. 1403–1409. URL: https://www.ijcai.org/proceedings/2020/195. doi:10.24963/ijcai.2020/195.

[2] K. Zhou, X. Huang, Q. Song, R. Chen, X. Hu, Auto-GNN: Neural architecture search of graph neural networks, Frontiers in Big Data 5 (2022). URL: https://www.frontiersin.org/journals/big-data/articles/10.3389/fdata.2022.1029307/full. doi:10.3389/fdata.2022.1029307, publisher: Frontiers.

[3] Y. Yuan, W. Wang, W. Pang, A Genetic Algorithm with Tree-structured Mutation for Hyperparameter Optimisation of Graph Neural Networks, in: 2021 IEEE Congress on Evolutionary Computation (CEC), 2021, pp. 482–489. URL: https://ieeexplore.ieee.org/abstract/document/9504717. doi:10.1109/CEC45853.2021.9504717.

[4] Y. Yuan, W. Wang, W. Pang, A systematic comparison study on hyperparameter optimisation of graph neural networks for molecular property prediction, in: Proceedings of the Genetic and Evolutionary Computation Conference, Association for Computing Machinery, New York, NY, USA, 2021, pp. 386–394. URL: https://dl.acm.org/doi/10.1145/3449639.3459370. doi:10.1145/3449639.3459370.

[5] Y. Yuan, W. Wang, W. Pang, Which hyperparameters to optimise? an investigation of evolutionary hyperparameter optimisation in graph neural network for molecular property prediction, in: Proceedings of the Genetic and Evolutionary Computation Conference Companion, 2021, pp. 1403–1404. URL: https://doi.org/10.1145/3449726.3463192. doi:10.1145/3449726.3463192.

[6] A. Ebadi, M. Kaur, Q. Liu, Hyperparameter optimization and neural architecture search algorithms for graph Neural Networks in cheminformatics, Computational Materials Science 254 (2025). URL: https://www.sciencedirect.com/science/article/pii/S0927025625002472. doi:10.1016/j.commatsci.2025.113904.

[7] V. Vodilovska, S. Gievska, I. Ivanoska, Hyperparameter Optimization of Graph Neural Networks for mRNA Degradation Prediction, in: 2023 46th MIPRO ICT and Electronics Convention (MIPRO), 2023, pp. 423–428. URL: https://ieeexplore.ieee.org/abstract/document/10159737. doi:10.23919/MIPRO57284.2023.10159737.

[8] N. Hollmann, S. Müller, L. Purucker, A. Krishnakumar, M. Körfer, S. B. Hoo, R. T. Schirrmeister, F. Hutter, Accurate predictions on small data with a tabular foundation model, Nature (2025). URL: https://www.nature.com/articles/s41586-024-08328-6. doi:10.1038/s41586-024-08328-6, publisher: Springer Nature.

[9] M. Dědič, M. Bělohlávek, Benchmarking and Transfer Learning for Hyperparameter Optimization of Graph Neural Networks, in: Proceedings of the 18th International Conference on Agents and Artificial Intelligence, volume 4, Marbella, Spain, 2026, pp. 3079–3086.

[10] W. Hamilton, Z. Ying, J. Leskovec, Inductive representation learning on large graphs, in: Advances in neural information processing systems, 2017, pp. 1024–1034.

[11] B. Bischl, M. Binder, M. Lang, T. Pielok, J. Richter, S. Coors, J. Thomas, T. Ullmann, M. Becker, A.-L. Boulesteix, D. Deng, M. Lindauer, Hyperparameter optimization: Foundations, algorithms,

best practices, and open challenges, WIREs Data Mining and Knowledge Discovery 13 (2023). URL: https://onlinelibrary.wiley.com/doi/abs/10.1002/widm.1484. doi:10.1002/widm.1484.

[12] P. Procházka, M. Mareš, M. Dědič, Which Graph Properties Affect GNN Performance for a Given Downstream Task?, in: Proceedings of the 23nd Conference Information Technologies – Applications and Theory (ITAT 2023), volume 3498 of *CEUR Workshop Proceedings*, CEUR-WS.org, Tatranské Matliare, Slovakia, 2023, pp. 58–66. URL: https://ceur-ws.org/Vol-3498/#paper7.

[13] P. W. Holland, K. B. Laskey, S. Leinhardt, Stochastic blockmodels: First steps, Social Networks 5 (1983) 109–137. URL: https://www.sciencedirect.com/science/article/pii/0378873383900217. doi:10.1016/0378-8733(83)90021-7.

[14] I. Guyon, Design of experiments of the NIPS 2003 variable selection benchmark, in: NIPS 2003 workshop on feature extraction and feature selection, volume 253, 2003, p. 40.

[15] Z. Yang, W. Cohen, R. Salakhudinov, Revisiting Semi-Supervised Learning with Graph Embeddings, in: Proceedings of The 33rd International Conference on Machine Learning, 2016, pp. 40–48. URL: https://proceedings.mlr.press/v48/yanga16.html.

[16] T. Akiba, S. Sano, T. Yanase, T. Ohta, M. Koyama, Optuna: A next-generation hyperparameter optimization framework, in: Proceedings of the 25th ACM SIGKDD international conference on knowledge discovery & data mining, 2019, pp. 2623–2631. doi:10.1145/3292500.3330701.