Evolving Computation Graphs

Andreea Deac [1 2] **Jian Tang** [2 3]

Abstract

Graph neural networks (GNNs) have demonstrated success in modeling relational data, especially for data that exhibits homophily: when a connection between nodes tends to imply that they belong to the same class. However, while this assumption is true in many relevant situations, there are important real-world scenarios that violate this assumption. In this work, we propose Evolving Computation Graphs (ECGs), a novel method for enhancing GNNs on heterophilic datasets without requiring prior domain knowledge. Our approach builds on prior theoretical insights linking node degree, high homophily, and inter vs intra-class embedding similarity by rewiring the GNNs' computation graph towards adding edges that connect nodes that are likely to be in the same class. We utilise weaker classifiers to identify these edges and evaluate ECGs on a diverse set of recently-proposed heterophilous datasets, demonstrating improvements over 95% of the relevant baselines.

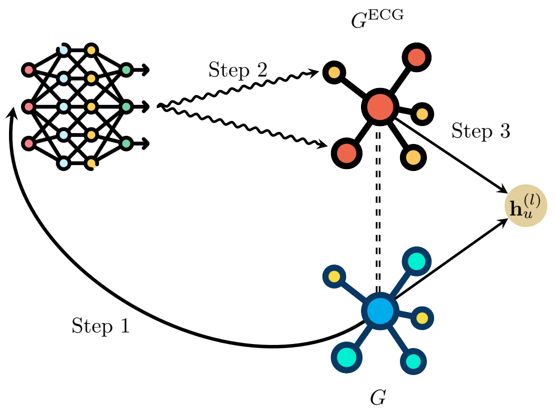

Figure 1. Illustration of ECG. **Step 1:** nodes in a graph, G, are embedded using a pre-trained weak classifier. **Step 2:** Based on these embeddings, a nearest-neighbour graph, G^{EGC}, is generated. This graph is likely to have improved propagation and homophily properties (illustrated by similar colours between neighbouring nodes). **Step 3:** Message passing is performed, both in the original and in the ECG graph, to update node representations.

1. Introduction

Neural networks applied to graph-structured data have demonstrated success across various domains, including practical applications like drug discovery transportation networks chip design and theoretical advancements . The fundamental concept behind Graph Neural Networks (GNNs) is that nodes communicate with their neighbouring nodes through messages in each layer. These messages, received from neighbours, are then aggregated in a permutation-invariant manner to contribute to a new node representation.

The performance of GNNs may thus rely on the underlying assumption of *homophily*, which suggests that nodes are connected by edges if they are similar based on their attributes or belonging to the same class, as commonly seen in social or citation networks. However, this assumption often fails to accurately describe real-world data when the graph contains *heterophilic* edges, connecting dissimilar nodes. This observation holds particular significance since GNNs tend to exhibit significantly poorer performance on heterophilic graphs compared to datasets known to be homophilic. Several studies (Zhu et al., 2020b;a; Wang & Zhang, 2022; He et al., 2022) highlighted this issue, using a mixture of strongly homophilous graphs—such as Cora, Citeseer and Pubmed —as well as a standard suite of six heterophilic datasets—Squirrel, Chameleon, Cornell, Texas, Wisconsin and Actor.

On this standard suite of heterophilic graphs, general GNN architectures tend to underperform unless there is high label informativeness (Ma et al., 2021). In prior work, this issue was tackled primarily by proposing modifications to the GNN architecture. These include changes to the aggregation function, such as separating self- and neighbour embeddings (Zhu et al., 2020b), mixing low- and high-frequency signals, and predicting and utilising the compatibility matrix. Other approaches involve using the Jacobi basis in spectral GNNs

[1]Université de Montréal [2]Mila Québec AI Institute [3]HEC Montréal. Correspondence to: Andreea Deac <andreeadeac22@gmail.com>.

Presented at the 2nd Annual Workshop on Topology, Algebra, and Geometry in Machine Learning (TAG-ML) at the 40th International Conference on Machine Learning, Honolulu, Hawaii, USA. 2023. Copyright 2023 by the author(s).

or learning cellular sheaves for neural sheaf diffusion.

However, Platonov et al. (2023) remaked that the standard heterophilous suite has significant drawbacks, such as data originating from only three sources, significant numbers of repeated nodes and improper evaluation regarding class imbalance. To address these shortcomings, a new benchmark was introduced incorporating improvements on all of the above issues. Once such corrections are accounted for, standard GNN models such as graph convolutional networks (GCN), GraphSAGE (SAGE), graph attention networks (GAT), and Graph Transformers (GT) demonstrated superior performance compared to architectures tailored specifically for heterophily, in spite of the heterophilic properties of the datasets. The notable exception is *-sep* (Zhu et al., 2020b) which consistently improved GAT and GT.

In light of this surprising discovery, we suggest that there should be alternate routes to making the most of heterophilic datasets. Rather than attempting to modify these standard GNNs, we propose modifying their *computation graph*: effectively, enforcing messages to be sent across additional pairs of nodes. These node pairs are chosen according to a particular measure of *similarity*. If the similarity metric is favourably chosen, such a computation graph will improve the overall homophily statistics, thereby creating more favourable conditions for GNNs to perform well.

We further propose that the *modification* of the computation graph should be separate from its *utilisation*. That is, we proceed in two phases: the first phase learns the representations that allow us to construct new computation graphs, and the second phase utilises those representations to construct new computation graphs, to be utilised by a GNN in each layer. This design choice makes our method elegant, performant and easy to evaluate: the two-phase nature means we are not susceptible to bilevel optimisation, the graphs we use need to be precomputed exactly once rather than updated on-the-fly in every layer, and because the same computation graph is used across all GNN layers, we can more rigidly evaluate how useful this graph is, all other things kept equal.

Hence, the essence of our method is *Evolving Computation Graphs* (**ECG**), which uses weak classifiers to generate node embeddings. These embeddings are then used to define a similarity metric between nodes (such as cosine similarity). We then select the edges corresponding to k-nearest neighbours, based on the similarity metric. The edges selected in this manner form a complementary graph, which we propose using in parallel with the input graph to update each node's representation. For this purpose, we use standard, off-the-shelf, GNNs. Our method is illustrated in Figure 1.

The nature of the weak classifier employed in ECG is flexible and, for the purpose of this paper, we used two representative options. The first option is a point-wise MLP classifier, attempting to cluster together nodes based on the given training labels, without any graph-based biases. For the second option, we attempt the converse: utilising the given graph structure and node features, but not relying on the training labels. This is a suitable setting for a self-supervised graph representation learning method, such as BGRL (Thakoor et al., 2021), which is designed to cluster together nodes with similar local neighbourhoods—both in terms of subgraphs and features.

To evaluate ECG, we conduct experiments on the benchmark suite proposed by Platonov et al. (2023). Our results demonstrate that ECG models outperform their GNN baselines in 19 out of 20 head-to-head comparisons. The most significant improvements can be noticed for GCNs—which are best suited to benefit from improved homophily—where improvements reach up to 10% in absolute terms.

2. Background

Graph representation learning setup We denote graphs by $G = (V, E)$, where V is the set of nodes and E is the set of edges, and we denote by $e_{uv} \in E$ the edge that connects nodes u and v. We can assume that the input graphs are provided to the GNNs via the *node feature matrix*, $\mathbf{X} \in \mathbb{R}^{|V|\times k}$ (such that $\mathbf{x}_u \in \mathbb{R}^k$ are the input features of node $u \in V$), and the *adjacency matrix*, $\mathbf{A} \in \{0,1\}^{|V|\times|V|}$, such that a_{uv} indicates whether nodes u and v are connected by an edge. We further assume the graph is *undirected*; that is, $\mathbf{A} = \mathbf{A}^\top$. We also use $d_u = \sum_{v\in V} a_{uv} \left(= \sum_{v\in V} a_{vu}\right)$ to denote the degree of node u.

We focus on node classification tasks with C representing the set of possible classes, where for node with input features $\mathbf{x}_u$, there is a label $y_u \in C$. Thus we aim to learn a function f that minimises $\mathbb{E}[\mathcal{L}(y_u, \hat{y}_u)]$, where $\hat{y}_u$ is the prediction of $f(\mathbf{x}_u) = \hat{y}_u$, and $\mathcal{L}$ is the cross-entropy loss.

Graph neural networks The one-step layer of a GNN can be summarised as follows:

$$\mathbf{h}_u^{(l)} = \phi^{(l)}\left(\mathbf{h}_u^{(l-1)}, \bigoplus_{(u,v)\in E} \psi^{(l)}\left(\mathbf{h}_u^{(l-1)}, \mathbf{h}_v^{(l-1)}\right)\right) \quad (1)$$

where, by definition, we set $\mathbf{h}_u^{(0)} = \mathbf{x}_u$. Leveraging different (potentially learnable) functions for $\phi^{(l)} : \mathbb{R}^k \times \mathbb{R}^m \to \mathbb{R}^{k'}$, $\bigoplus : \mathrm{bag}(\mathbb{R}^m) \to \mathbb{R}^m$ and $\psi^{(l)} : \mathbb{R}^k \times \mathbb{R}^k \to \mathbb{R}^m$ then recovers well-known GNN architectures.

Homophily has been repeatedly mentioned as an important measure of the graph, especially when it comes to GNN performance. It corresponds to an assumption that neighbouring nodes tend to share labels: $a_{uv} = 1 \implies y_u = y_v$, which is often the case for many industrially-relevant real world graphs (such as social networks). Intuitively, a graph with high homophily will make it easier to exploit neighbourhood structure to derive more accurate node labels. A discussion on relevant homophily metrics and how they apply to ECG can be found in Appendix B.

Weak classifier To derive novel computation graphs which are likely to result in higher test performance, we likely require "novel" homophilic connections to emerge—rather than amplifying the homophily already present in $\mathbf{A}$. Therefore, for the purposes of building a useful computation graph, our ECG method aims to first learn representations of nodes governed by a model which does *not* have access to inputs ($\mathbf{X}$), graph structure ($\mathbf{A}$) and training labels ($\mathbf{y}_{\text{tr}}$) simultaneously. We hence call such a model a "weak classifier", as it is not exposed to the same kind of inductive biases as a supervised GNN would (and hence it must obtain useful models which do not rely on some of these biases).

BGRL While using the embeddings from an MLP can offer a solid way to improve homophily metrics, their confidence will degrade for nodes where the model is less accurate outside of the training set—which are arguably the nodes we would like to improve predictions on the most. Accordingly, we may also withhold access to the training labels ($\mathbf{y}_{\text{tr}}$). Now the model is forced to arrange the node representations in a way that will be mindful of the input features and graph structure, but without knowing the task specifics upfront, and hence not vulnerable to overfitting on the training nodes. Such a weak classifier naturally lends itself to self-supervised learning on graphs.

Bootstrapped graph latents (BGRL) is a state-of-the-art self-supervised graph representation learning method. BGRL learns two GNN encoders with identical architecture; an *online* encoder, $\mathcal{E}_\theta$, and a *target* encoder, $\mathcal{E}_\phi$. BGRL also contains a *predictor* network p_θ. We offer a "bird's eye" view of BGRL, and defer to Thakoor et al. (2021) for details.

At each iteration, two data augmentations (e.g. random node/edge dropout) are applied to the input graph, obtaining augmented graphs $(\mathbf{X}_1, \mathbf{A}_1)$ and $(\mathbf{X}_2, \mathbf{A}_2)$. Then, the two encoders are applied to these augmentations, recovering a pair of latent node embeddings: $\mathbf{H}_1 = \mathcal{E}_\theta(\mathbf{X}_1, \mathbf{A}_1)$, $\mathbf{H}_2 = \mathcal{E}_\phi(\mathbf{X}_2, \mathbf{A}_2)$. $\mathbf{H}_1$ is additionally passed through the predictor network: $\mathbf{Z}_1 = p_\theta(\mathbf{H}_1)$. BGRL preserves cosine similarity between all corresponding nodes in $\mathbf{Z}_1$ and $\mathbf{H}_2$:

$$\mathcal{L}_{\text{BGRL}} = -\frac{\mathbf{Z}_1 \mathbf{H}_2^\top}{\|\mathbf{Z}_1\| \|\mathbf{H}_2\|} \tag{2}$$

Lastly, the parameters of the online encoder $\mathcal{E}_\theta$ and predictor p_θ are updated via SGD on $\mathcal{L}_{\text{BGRL}}$, and the parameters of the target encoder $\mathcal{E}_\phi$ are updated as the exponential moving average of the online encoder's parameters.

3. Evolving Computation Graphs

Next, we describe the ECG methodology. Please refer to Algorithm 1 in Appendix C for a pseudocode summary.

Step 1: Embedding extraction Firstly, we assume that an appropriate weak classifier has already been trained (as discussed in previous sections), and is capable of producing node embeddings. We start by invoking this classifier to obtain ECG embeddings $\mathbf{H}_{\text{ECG}} = \gamma(\mathbf{X}, \mathbf{A})$. We study two simple but potent variants of γ, as per the previous section:

MLP: We utilise a simple deep MLP[1]; that is, $\gamma(\mathbf{X}, \mathbf{A}) = \sigma(\sigma(\mathbf{X}\mathbf{W}_1)\mathbf{W}_2)$, where σ is the GELU activation.

BGRL: In this case, we set $\gamma = \mathcal{E}_\theta$, the online encoder of BGRL. We utilise a publicly available implementation of BGRL by DGL , which uses a two-layer GCN as γ. The parameters of γ are kept frozen after pre-training.

Step 2: Graph construction Having obtained $\mathbf{H}_{\text{ECG}}$, we can now use it to compute a similarity metric between the nodes, such as cosine similarity, as follows:

$$\mathbf{S} = \mathbf{H}_{\text{ECG}} \mathbf{H}_{\text{ECG}}^\top \qquad \hat{s}_{uv} = \frac{s_{uv}}{\|\mathbf{h}_{\text{ECG}_u}\| \|\mathbf{h}_{\text{ECG}_v}\|} \tag{3}$$

Based on this similarity metric, for each node $u \in V$ we select its neighbourhood $\mathcal{N}_u^{\text{ECG}}$ to be its k nearest neighbours in $\mathbf{S}$ (where k is a tunable hyperparameter): $\mathcal{N}_u^{\text{ECG}} = \text{top-}k_{v \in V}\, \hat{s}_{uv}$. Equivalently, we construct a new computation graph, $G^{\text{ECG}} = (V, E^{\text{ECG}})$, such that its edges are $E^{\text{ECG}} = \{(u, v) \mid u \in V \wedge v \in \mathcal{N}_u\}$. These edges are effectively determined by the weak classifier.

Step 3: Parallel message passing Finally, once the ECG graph, G^{ECG}, is available, we can run our GNN of choice over it. To retain the topological benefits contained in the input graph structure, we opt to run two GNN layers in parallel—one over the input graph (as in Equation 1), and one over the ECG graph, as follows:

$$\mathbf{h}_{\text{INP}_u}^{(l)} = \phi_{\text{INP}}^{(l)} \left(\mathbf{h}_u^{(l-1)}, \bigoplus_{(u,v) \in E} \psi_{\text{INP}}^{(l)} \left(\mathbf{h}_u^{(l-1)}, \mathbf{h}_v^{(l-1)} \right) \right) \tag{4}$$

$$\mathbf{h}_{\text{ECG}_u}^{(l)} = \phi_{\text{ECG}}^{(l)} \left(\mathbf{h}_u^{(l-1)}, \bigoplus_{(u,v) \in E^{\text{ECG}}} \psi_{\text{ECG}}^{(l)} \left(\mathbf{h}_u^{(l-1)}, \mathbf{h}_v^{(l-1)} \right) \right) \tag{5}$$

Then the representation after l layers is obtained by jointly transforming these two representations:

$$\mathbf{h}_u^{(l)} = \mathbf{W}^{(l)} \mathbf{h}_{\text{INP}_u}^{(l)} + \mathbf{U}^{(l)} \mathbf{h}_{\text{ECG}_u}^{(l)} \tag{6}$$

where $\mathbf{W}^{(l)}$ and $\mathbf{U}^{(l)}$ are learnable parameters.

[1]While training the MLP, logistic regression is attached to γ.

Table 1. ECG performance on datasets proposed in Platonov et al. (2023). We report accuracy for `roman-empire` and `amazon-ratings` and ROC AUC for `minesweeper`, `tolokers`, and `questions`.

Model	roman-empire	amazon-ratings	minesweeper	tolokers	questions
MLP	$65.88_{\pm 0.38}$	$45.90_{\pm 0.52}$	$50.89_{\pm 1.39}$	$72.95_{\pm 1.06}$	$70.34_{\pm 0.76}$
GCN	$73.69_{\pm 0.74}$	$48.70_{\pm 0.63}$	$89.75_{\pm 0.52}$	$83.64_{\pm 0.67}$	$76.09_{\pm 1.27}$
ECG-GCN	$84.53_{\pm 0.26}$ (↑)	$51.12_{\pm 0.38}$ (↑)	$92.63_{\pm 0.10}$ (↑)	$\mathbf{84.81}_{\pm 0.25}$ (↑)	$77.50_{\pm 0.35}$ (↑)
SAGE	$85.74_{\pm 0.67}$	$53.63_{\pm 0.39}$	$93.51_{\pm 0.57}$	$82.43_{\pm 0.44}$	$76.44_{\pm 0.62}$
ECG-SAGE	$87.88_{\pm 0.25}$ (↑)	$53.45_{\pm 0.27}$ (↓)	$94.11_{\pm 0.07}$ (↑)	$82.61_{\pm 0.29}$ (↑)	$77.23_{\pm 0.36}$ (↑)
GAT-sep	$88.75_{\pm 0.41}$	$52.70_{\pm 0.62}$	$93.91_{\pm 0.35}$	$83.78_{\pm 0.43}$	$76.79_{\pm 0.71}$
ECG-GAT-sep	$\mathbf{89.62}_{\pm 0.18}$ (↑)	$\mathbf{53.65}_{\pm 0.39}$ (↑)	$\mathbf{94.52}_{\pm 0.20}$ (↑)	$84.23_{\pm 0.25}$ (↑)	$77.38_{\pm 0.18}$ (↑)
GT-sep	$87.32_{\pm 0.39}$	$52.18_{\pm 0.80}$	$92.29_{\pm 0.47}$	$82.52_{\pm 0.92}$	$78.05_{\pm 0.93}$
ECG-GT-sep	$89.56_{\pm 0.16}$ (↑)	$53.25_{\pm 0.39}$ (↑)	$93.62_{\pm 0.27}$ (↑)	$84.00_{\pm 0.24}$ (↑)	$78.12_{\pm 0.32}$ (↑)
FSGNN	$79.92_{\pm 0.56}$	$52.74_{\pm 0.83}$	$90.08_{\pm 0.70}$	$82.76_{\pm 0.61}$	$\mathbf{78.86}_{\pm 0.92}$

Equations 4–6 can then be repeatedly iterated, like for any standard GNN layer. As G^{ECG} will contain noisy edges which do not contribute to useful propagation of messages, we additionally apply DropEdge when propagating over the ECG graph, with probability $p_{de} = 0.5$.

4. Experiments

We evaluate ECG on five diverse heterophilic node classification datasets, recently-proposed by Platonov et al. (2023): `roman-empire`, `amazon-ratings`, `minesweeper`, `tolokers` and `questions`.

We ran ECG as an extension on standard GNN models, choosing the "-sep" variant (Zhu et al., 2020b) for GAT and GT as it was noted to improve their performance consistently on these tasks (Platonov et al., 2023). Thus, our baselines are GCN, GraphSAGE, GAT-sep and GT-sep, which we extend by modifying their computation graph as in Section 3. For each ECG model, we ran three variants, depending on which weak classifier was used to select the complementary edges, E^{ECG}: the MLP, the BGRL, or a concatenation of the output of the two. In Appendix A, we present additional information on the experiments, together with the hyperparameters corresponding to the best validation results.

In Table 1, we show the test performance corresponding to the best ECG model for each of the five datasets, with hyperparameters selected using validation performance. There are 20 dataset-model combinations that ECG is tested on; on 19 of them (marked with arrow up in the table), ECG improves the performance of the corresponding GNN, the only exception being GraphSAGE on `amazon-ratings`.

Moreover, we observe the highest gains in performance are achieved by ECG-GCN, ranging from 1.17% to 10.84% (absolute values) in a manner that is correlated with the homophily of the dataset. This confirms that, due to its aggregation function, GCN is also the architecture most prone to performance changes based on the homophily.

References

He, D., Liang, C., Liu, H., Wen, M., Jiao, P., and Feng, Z. Block modeling-guided graph convolutional neural networks. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 36, pp. 4022–4029, 2022.

Ma, Y., Liu, X., Shah, N., and Tang, J. Is homophily a necessity for graph neural networks? *arXiv preprint arXiv:2106.06134*, 2021.

Platonov, O., Kuznedelev, D., Diskin, M., Babenko, A., and Prokhorenkova, L. A critical look at the evaluation of gnns under heterophily: are we really making progress? *arXiv preprint arXiv:2302.11640*, 2023.

Thakoor, S., Tallec, C., Azar, M. G., Azabou, M., Dyer, E. L., Munos, R., Veličković, P., and Valko, M. Large-scale representation learning on graphs via bootstrapping. In *International Conference on Learning Representations*, 2021.

Wang, X. and Zhang, M. How powerful are spectral graph neural networks. In *International Conference on Machine Learning*, pp. 23341–23362. PMLR, 2022.

Zhu, J., Rossi, R. A., Rao, A., Mai, T., Lipka, N., Ahmed, N. K., and Koutra, D. Graph neural networks with heterophily. *arXiv preprint arXiv:2009.13566*, 2020a.

Zhu, J., Yan, Y., Zhao, L., Heimann, M., Akoglu, L., and Koutra, D. Beyond homophily in graph neural networks: Current limitations and effective designs. *Advances in Neural Information Processing Systems*, 33, 2020b.