BALANCING PERFORMANCE AND COMPLEXITY WITH ADAPTIVE GRAPH COARSENING

Marek Dědič
Czech Technical University in Prague

Lukáš Bajer & Pavel Procházka
Cisco Systems, Inc.

Martin Holeňa
Institute of Computer Science, Czech Academy of Sciences

ABSTRACT

We present a method for graph node classification that allows a user to precisely select the resolution at which the graph in question should be simplified and through this provides a way of choosing a suitable point in the performance-complexity trade-off. The method is based on refining a reduced graph in a targeted way following the node classification confidence for particular nodes.

1 INTRODUCTION

Machine learning on graphs has, in recent years, seen an explosion in popularity, the underlying graph topology has, however, received much less attention. The main aim of this work is to explore the performance-complexity characteristics in the context of graph learning, as introduced in Procházka et al. (2022), mainly to enable learning on graphs that would otherwise be prohibitively large. Consider an undirected graph G. The result of a repeated application of graph coarsening (such as a contraction of a subset of its edges) is a sequence of graphs $G_0, G_1, G_2, \dots, G_L$ where $G_0 = G$. Given a model M that operates on graphs, a performance metric, and a complexity metric, the sequence $G_0, G_1, \dots, G_L$ corresponds to points in the performance-complexity plane, where advancing along the sequence generally hurts performance and decreases complexity. This performance-complexity characteristic allows for a choice of a **working point** tailored to the specific scenario. The method proposed in the rest of this work evaluates the graphs in reverse order, i.e. starting with the simplest one. As such, the algorithm only trains the model on a subset of simpler graphs, lowering the complexity of selecting the working point.

2 A METHOD FOR FLEXIBLE PERFORMANCE-COMPLEXITY BALANCING

Our work builds on the HARP method Chen et al. (2018a) for pretraining methods such as node2vec Grover & Leskovec (2016) on coarsened graphs. The sequence $G_0, G_1, G_2, \dots, G_L$ is generated in HARP consecutively. In an overview, the HARP algorithm first ahead-of-time consecutively coarsens the graph. The method itself can then be executed by repeating the following steps on the graphs from the coarsest to the finest (i.e., from G_L to G_0):

1. **Training on an intermediary graph**. The graph embedding model is trained on G_i, producing its embedding Φ_{G_i}.
2. **Embedding prolongation**. The embedding Φ_{G_i} is *prolonged* into $\Phi_{G_{i-1}}$ by copying embeddings of merged nodes. $\Phi_{G_{i-1}}$ is then used as the starting point for training on G_{i-1}.

While the prolongation used by HARP is sufficient when used as a means of pre-training, the approach is far too crude when studying the relationship between graph complexity and the quality of graph embedding. In order to overcome this limitation, we present the adaptive prolongation approach. This algorithm works with the pre-coarsened graphs produced by HARP, however, the embedding is learned in a different manner. There are K prolongation steps (where generally $K \neq L$) and each of them uses all graphs $G_L, \dots, G_0$. The prolongation steps are driven by local properties of the graph with relation to the downstream task, allowing for different levels of granularity

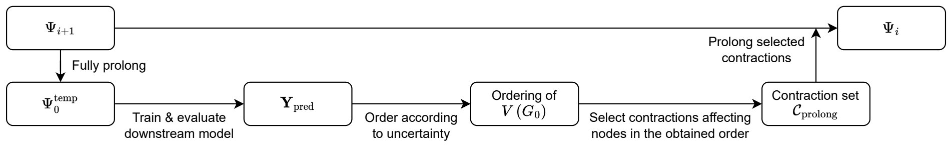
Ψ_{i+1}
Fully prolong
Ψ_0^{temp}
Train & evaluate downstream model
$\mathbf{Y}_{\text{pred}}$
Order according to uncertainty
Ordering of $V(G_0)$
Select contractions affecting nodes in the obtained order
Contraction set $\mathcal{C}_{\text{prolong}}$
Prolong selected contractions
Ψ_i

Figure 1: A schematic explanation of the adaptive prolongation algorithm for obtaining the embedding Ψ_i from Ψ_{i+1}.

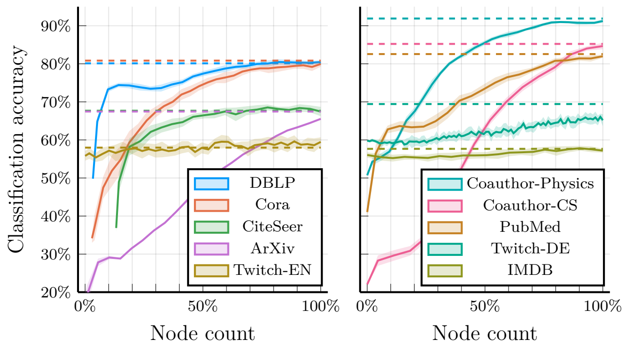

Figure 2: Downstream classifier test-set accuracies at different steps of adaptive prolongation. Dashed line shows the baseline node2vec model accuracy. The node count is taken relative to the total node count in each dataset. The results are averaged over ten runs, with the solid line representing the mean and the shaded area denoting one standard deviation.

in different parts of the graph. Let us denote $\Psi_K, \dots, \Psi_0$ the resulting embedding sequence. The algorithm starts with the coarsest graph G_L, trains a graph model to compute its embedding Ψ_K and gradually refines it until reaching the embedding Ψ_0. These prolongation steps are interlaid with continued training of the graph model, as in standard HARP. A description of a single prolongation step from Ψ_{i+1} to Ψ_i is schematically outlined in Figure 1 and described in detail in Appendix A.

3 Experimental evaluation

The proposed methods were experimentally verified on 10 publicly available datasets. The node2vec algorithm was used for generating the node embeddings, with an MLP classifier providing the predictions for node classification. A detailed description of the used datasets and hyperparameter setup is available in Appendix B. In order to study the effect of adaptive prolongation, for each prolongation step, the intermediary embedding was fully prolonged to obtain an embedding of the original graph G. A classifier was then trained with this embedding as input. This setup allows us to compare classification accuracy at each step of the adaptive prolongation, as shown in Figure 2.

The results were statistically validated by comparing the adaptive prolongation model to the baseline model at k-th deciles of node count using the Bayesian Wilcoxon signed-rank test Benavoli et al. (2014). The results of note are that at 60% complexity, the models have over a 99% probability of being within 10 percentage points of performance on the full graph and at 80% complexity, they have over 99% probability of being withing 5 percentage points of performance.

4 Conclusion

In this work, a novel approach to prolonging graphs in the HARP setting was presented that selectively prolongs the graph in a way that maximizes performance of the considered downstream task under limited graph size. All of the proposed methods were experimentally verified, with the headline result being that at about 40% reduction in node count, the accuracy was still reasonably close to the accuracy on a full graph for most datasets. In future work, a direct way of tackling the outlined problem may be studied as an alternative to the proposed approach.

URM Statement

The authors acknowledge that at least one key author of this work meets the URM criteria of ICLR 2024 Tiny Papers Track.

References

Alessio Benavoli, Giorgio Corani, Francesca Mangili, Marco Zaffalon, and Fabrizio Ruggeri. A Bayesian Wilcoxon signed-rank test based on the Dirichlet process. In *Proceedings of the 31st International Conference on Machine Learning*, pp. 1026–1034, Beijing, China, June 2014. PMLR. URL `https://proceedings.mlr.press/v32/benavoli14.html`. ISSN: 1938-7228.

Aleksandar Bojchevski and Stephan Günnemann. Deep Gaussian Embedding of Graphs: Unsupervised Inductive Learning via Ranking. In *6th International Conference on Learning Representations*, February 2018. URL `https://openreview.net/forum?id=r1ZdKJ-0W`.

Haochen Chen, Bryan Perozzi, Yifan Hu, and Steven Skiena. HARP: Hierarchical Representation Learning for Networks. *Proceedings of the AAAI Conference on Artificial Intelligence*, 32 (1), April 2018a. ISSN 2374-3468. URL `https://ojs.aaai.org/index.php/AAAI/article/view/11849`. Number: 1.

Jie Chen, Tengfei Ma, and Cao Xiao. FastGCN: Fast Learning with Graph Convolutional Networks via Importance Sampling. In *6th International Conference on Learning Representations*, February 2018b. URL `https://openreview.net/forum?id=rytstxWAW`.

Matthias Fey and Jan Eric Lenssen. Fast Graph Representation Learning with PyTorch Geometric, April 2019. arXiv: 1903.02428.

Xinyu Fu, Jiani Zhang, Ziqiao Meng, and Irwin King. MAGNN: Metapath Aggregated Graph Neural Network for Heterogeneous Graph Embedding. In *Proceedings of The Web Conference 2020*, pp. 2331–2341, April 2020. doi: 10.1145/3366423.3380297. URL `http://arxiv.org/abs/2002.01680`. arXiv:2002.01680 [cs].

Aditya Grover and Jure Leskovec. node2vec: Scalable feature learning for networks. In *Proceedings of the 22nd ACM SIGKDD international conference on Knowledge discovery and data mining*, pp. 855–864, 2016.

Weihua Hu, Matthias Fey, Marinka Zitnik, Yuxiao Dong, Hongyu Ren, Bowen Liu, Michele Catasta, and Jure Leskovec. Open Graph Benchmark: Datasets for Machine Learning on Graphs, February 2021. URL `http://arxiv.org/abs/2005.00687`. arXiv:2005.00687 [cs, stat].

Diederik P. Kingma and Jimmy Ba. Adam: A Method for Stochastic Optimization, January 2017. arXiv:1412.6980.

Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Kopf, Edward Yang, Zachary DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, and Soumith Chintala. PyTorch: An Imperative Style, High-Performance Deep Learning Library. In H. Wallach, H. Larochelle, A. Beygelzimer, F. d\textquotesingle Alché-Buc, E. Fox, and R. Garnett (eds.), *Advances in Neural Information Processing Systems 32*, pp. 8024–8035, Vancouver, Canada, 2019. Curran Associates, Inc. URL `http://papers.neurips.cc/paper/9015-pytorch-an-imperative-style-high-performance-deep-learning-library.pdf`.

Pavel Procházka, Michal Mareš, and Marek Dědič. Downstream Task Aware Scalable Graph Size Reduction for Efficient GNN Application on Big Data. In *Information Technologies - Applications and Theory (ITAT 2022)*, Zuberec, Slovakia, 2022.

Benedek Rozemberczki, Carl Allen, and Rik Sarkar. Multi-Scale attributed node embedding. *Journal of Complex Networks*, 9(2):cnab014, April 2021. ISSN 2051-1329. doi: 10.1093/comnet/cnab014. URL `https://doi.org/10.1093/comnet/cnab014`.

Till Hendrik Schulz, Tamás Horváth, Pascal Welke, and Stefan Wrobel. Mining Tree Patterns with Partially Injective Homomorphisms. In Michele Berlingerio, Francesco Bonchi, Thomas Gärtner, Neil Hurley, and Georgiana Ifrim (eds.), *Machine Learning and Knowledge Discovery in Databases*, Lecture Notes in Computer Science, pp. 585–601, Cham, 2019. Springer International Publishing. ISBN 978-3-030-10928-8. doi: 10.1007/978-3-030-10928-8_35.

Oleksandr Shchur, Maximilian Mumme, Aleksandar Bojchevski, and Stephan Günnemann. Pitfalls of Graph Neural Network Evaluation, June 2019. URL `http://arxiv.org/abs/1811.05868`. arXiv:1811.05868 [cs, stat].

Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Dropout: A Simple Way to Prevent Neural Networks from Overfitting. *Journal of Machine Learning Research*, 15(56):1929–1958, 2014. ISSN 1533-7928. URL `http://jmlr.org/papers/v15/srivastava14a.html`.

Zhilin Yang, William Cohen, and Ruslan Salakhudinov. Revisiting Semi-Supervised Learning with Graph Embeddings. In *Proceedings of The 33rd International Conference on Machine Learning*, pp. 40–48, New York, NY, USA, June 2016. PMLR. URL `https://proceedings.mlr.press/v48/yanga16.html`.

A One step of adaptive prolongation

The sequence $G_0, G_1, G_2, \ldots, G_L$ as defined in Section 1 is generated in HARP consecutively. Let φ_i denote the mapping $G_i = \varphi_i\left(G_{i-1}\right)$. Following Schulz et al. (2019), we restrict the definition of such a coarsening φ_i to only consist of a series of edge contractions $\mathcal{C} \subseteq E\left(G\right)$. Let us denote $\Psi_K, \ldots, \Psi_0$ the embedding sequence produced by the adaptive prologation schema. A description of a single prolongation step from Ψ_{i+1} to Ψ_i follows and is described in further detail in Algorithm 1.

The procedure keeps track of all the edge contractions that were made in the dataset augmentation part of the algorithm and gradually reverses them. To this end, apart from the embedding Ψ_i, the set of all contractions yet to be reversed as of step i is kept as $\mathcal{C}_L^{(i)}, \ldots, \mathcal{C}_0^{(i)}$, with the initial values $\mathcal{C}_j^{(K)}$ corresponding to the underlying coarsening φ_j.

In each prolongation step, the embedding Ψ_{i+1} is prolonged to Ψ_i by selecting a set of n_p contractions $\mathcal{C}_{\mathrm{prolong}}$ and undoing them by copying and reusing the embedding of the node resulting from the contraction to both of the contracted nodes. To obtain $\mathcal{C}_{\mathrm{prolong}}$, nodes of G_0 are first ordered in such a way that corresponds to the usefulness of prolonging them. Subsequently, the set $\mathcal{C}_L^{(i+1)}, \ldots, \mathcal{C}_0^{(i+1)}$ is ordered to match this node ordering by considering the nodes that the individual contractions affect. $\mathcal{C}_{\mathrm{prolong}}$ is then selected by taking the first n_p contractions. If multiple contractions affecting the same node are available in the sequence $\mathcal{C}_L^{(i+1)}, \ldots, \mathcal{C}_0^{(i+1)}$, one is selected from $\mathcal{C}_j^{(i+1)}$ corresponding to the coarsest-level coarsening. The sequence $\mathcal{C}_L^{(i)}, \ldots, \mathcal{C}_0^{(i)}$ is produced from $\mathcal{C}_L^{(i+1)}, \ldots, \mathcal{C}_0^{(i+1)}$ by removing all of the edges contained in $\mathcal{C}_{\mathrm{prolong}}$.

To obtain an ordering of nodes of G_0 based on the usefulness of their prolongation, the embedding Ψ_{i+1} is fully prolonged to a temporary embedding of the full graph, Ψ_0^{temp}. The downstream model is then trained using this temporary embedding to obtain $\boldsymbol{Y}_{\mathrm{pred}}$, the predicted posterior distribution of classes for each node in G_0. The nodes are ordered by the entropy of the posterior, making the algorithm prolong the nodes where the classifier is the most uncertain.

B Detailed experiment settings and results

The proposed methods were experimentally verified on 10 publicly available datasets. The datasets Cora and CiteSeer Yang et al. (2016) were used with the "full" train-test split as in Chen et al. (2018b). In addition, 2 variants of the Twitch dataset Rozemberczki et al. (2021) with the hignest node count (DE and EN) were used. Five medium sized datasets were also used, the PubMed dataset Yang et al. (2016), the DBLP dataset Bojchevski & Günnemann (2018), the IMDB dataset Fu et al.

Algorithm 1 Adaptive prolongation

```
Require: $G_0$                                          ▷ The original graph
Require: $\boldsymbol{y}_{\text{train}}$                ▷ Training labels
Require: $n_p$                                          ▷ The number of nodes to prolong
Require: $\Psi_{i+1}$                                   ▷ The previous embedding
Require: $\mathcal{C}_L^{(i+1)}, \dots, \mathcal{C}_0^{(i+1)}$   ▷ A list of all the contraction sets yet to be reversed
Ensure: $\Psi_i$                                        ▷ The next embedding
Ensure: $\mathcal{C}_L^{(i)}, \dots, \mathcal{C}_0^{(i)}$   ▷ Updated contraction list without the prolonged contractions

  $node\_order \leftarrow$ GET_NODE_ORDER$(G_0, \Psi_{i+1}, \boldsymbol{y}_{\text{train}}, \mathcal{C}_L^{(i+1)}, \dots, \mathcal{C}_0^{(i+1)})$
  $\mathcal{C}_{\text{prolong}} \leftarrow$ SELECT_CONTRACTIONS$(node\_order, n_p, \mathcal{C}_L^{(i+1)}, \dots, \mathcal{C}_0^{(i+1)})$
  $\Psi_i \leftarrow$ use $\mathcal{C}_{\text{prolong}}$ to prolong the embedding $\Psi_{i+1}$
  $\mathcal{C}_L^{(i)}, \dots, \mathcal{C}_0^{(i)} \leftarrow$ remove contractions in $\mathcal{C}_{\text{prolong}}$ from $\mathcal{C}_L^{(i+1)}, \dots, \mathcal{C}_0^{(i+1)}$

  function GET_NODE_ORDER$(G_0, \Psi_{i+1}, \boldsymbol{y}_{\text{train}}, \mathcal{C}_L^{(i+1)}, \dots, \mathcal{C}_0^{(i+1)})$
      $\Psi_0^{\text{temp}} \leftarrow$ use $\mathcal{C}_L^{(i+1)}, \dots, \mathcal{C}_0^{(i+1)}$ to fully prolong the current embedding $\Psi_{i+1}$ to $G_0$
      $model \leftarrow$ TRAIN_DOWNSTREAM_MODEL$(\Psi_0^{\text{temp}}, \boldsymbol{y}_{\text{train}})$
      $\boldsymbol{Y}_{\text{pred}} \leftarrow$ PREDICT$(model, node)$ for each $node \in V(G_0)$
      $entropy\_per\_node \leftarrow H(\boldsymbol{Y}_{\text{pred}})$
      return $V(G_0)$, sorted in descending order by $entropy\_per\_node$
  end function

  function SELECT_CONTRACTIONS$(ordered\_nodes, n_p, \mathcal{C}_L^{(i+1)}, \dots, \mathcal{C}_0^{(i+1)})$
      $\mathcal{C}_{\text{prolong}} \leftarrow \{\}$
      for $node \in ordered\_nodes$, until $|\mathcal{C}_{\text{prolong}}| = n_p$ do
          $contraction \leftarrow$ RESOLVE_CONTRACTION$(node, \mathcal{C}_{\text{prolong}}, \mathcal{C}_L^{(i+1)}, \dots, \mathcal{C}_0^{(i+1)})$
          If $contraction \neq$ null, add $contraction$ to $\mathcal{C}_{\text{prolong}}$
      end for
      return $\mathcal{C}_{\text{prolong}}$
  end function

  function RESOLVE_CONTRACTION$(node, \mathcal{C}_{\text{prolong}}, \mathcal{C}_L^{(i+1)}, \dots, \mathcal{C}_0^{(i+1)})$
      $contraction \leftarrow$ null
      for $j \in \{0, \dots, L\}$ do          ▷ I.e. all steps of the original coarsening from finest to coarsest
          $contraction\_candidate \leftarrow$ find in $\mathcal{C}_j^{(i+1)}$ a contraction that affects $node$, if not found,
  continue with $j + 1$
          if $contraction\_candidate \in \mathcal{C}_{\text{prolong}}$ then
              return $contraction$
          end if
          $contraction \leftarrow contraction\_candidate$
          $node \leftarrow$ apply $contraction$ to $node$, so that in the next loop, a subsequent contraction
  may be selected
      end for
      return $contraction$
  end function
```

Table 1: Hyper-parameter values used for different datasets

Hyper-parameter	**Cora**	**CiteSeer**	**PubMed**	**DBLP**	**Twitch**	**IMDB**	**ArXiv**	**Coauthor**
Embedding dimension	128	32	64	32	128	128	128	128
# of random walks	4	5	3	2	10	40	10	40
Random walk length	20	20	40	20	80	100	80	10
Context window size	5	5	20	5	3	5	20	5
Node2vec learning rate	0.01	0.01	0.01	0.01	0.025	0.01	0.01	0.01
Node2vec batch size	128	128	128	128	128	256	128	256
Node2vec epochs	5	7	1	1	5	1	1	1
# of MLP layers	3	3	1	3	2	2	3	2
MLP hidden layer width	128	256	128	256	64	64	256	16
Dropout rate	0.5	0.5	0.5	0.5	0.5	0.5	0.5	0.5
MLP learning rate	0.01	0.01	0.01	0.01	0.01	0.01	0.01	0.01
MLP epochs	30	80	300	300	500	100	300	100

(2020) and both variants of the Coauthor dataset Shchur et al. (2019). Finally, one large dataset was used, the OGB ArXiv dataset Hu et al. (2021).

The hyper-parameters for both the node2vec model used for the embedding training and the multi-layer perceptron used for downstream classification were initially set to values used in prior art (see Hu et al. (2021); Fey & Lenssen (2019)) and then manually fine-tuned for each dataset.

The achitecture of the algorithm was identical accross all datasets, with the only difference being in the values of the hyper-parameters, as listed in Table 1. For the Cora dataset, the node2vec model generated an embedding in $\mathbb{R}^{128}$ from 4 random walks of length 20 for each node with a context window of size 5. The optimizer ADAM Kingma & Ba (2017) was used with a learning rate of 0.01 and batches of 128 samples. The model was trained for 5 epochs and in each step of the adaptive prolongation, 100 nodes were prolonged, until reaching the original graph (the value of n_p was calculated so that the total number of training epochs would match baseline model training). The MLP classifier using the embeddings featured 3 linear layers of 128 neurons with batch normalization after each layer. Each layer was normalized using dropout Srivastava et al. (2014) with the rate of 0.5. Finally, a linear layer was used for the class prediction. For the classifier, ADAM with a learning rate of 0.01 was used for 30 epochs of training with the cross-entropy loss function. Dataset features weren't used for the classifier training as the aim of this work is to compare the embeddings. The experiment was run 10 times end-to-end and results averaged. The experiments were implemented using PyTorch Paszke et al. (2019) and PyTorch Geometric Fey & Lenssen (2019).

B.1 Results

To study the distribution of model properties, the results were evaluated at k-th deciles of the node count of the full graph, for all possible values of k. At each decile, the performance of the model was compared to the baseline node2vec model using the Wilcoxon signed-rank test with the Holm-Bonferroni correction for multiple hypothesis testing. The hypotheses that the models are equivalent with the baseline were rejected by the test at the 5% level of significance for $k \in \{1, 2, 3, 4, 5\}$, suggesting that the adaptive prolongation approach is valid and practically useful in situations where at least half of the nodes is available.

The results were also studied from the point of view of Bayesian estimation by comparing the performance of the proposed models to that of the baseline model using the Bayesian Wilcoxon signed-rank test Benavoli et al. (2014) for 3 different widths of the region of practical equivalence (ROPE), 1%, 5% and 10%. The probabilities that the two models are practically equivalent are listed in Table 2.

Table 2: The probabilities that the adaptive approach will be practically equivalent to node2vec when compared on different fractions of the full graph and with different widths of the region of practical equivalence.

Nodes	**1% ROPE**	**5% ROPE**	**10% ROPE**
10%	0%	0.3%	2.5%
20%	0%	0.8%	14.1%
30%	0%	1.7%	35.3%
40%	0%	5.3%	72.0%
50%	0.1%	35.3%	85.7%
60%	0.6%	62.2%	99.7%
70%	32.0%	84.7%	100.0%
80%	30.0%	99.9%	100.0%
90%	48.9%	100.0%	100.0%
100%	87.7%	100.0%	100.0%