EFFICIENT REPRESENTATION LEARNING OF SUBGRAPHS BY SUBGRAPH-TO-NODE TRANSLATION

Anonymous authors
Paper under double-blind review

ABSTRACT

A subgraph is a data structure that can represent various real-world problems. We propose Subgraph-To-Node (S2N) translation, which is a novel formulation to efficiently learn representations of subgraphs. Specifically, given a set of subgraphs in the global graph, we construct a new graph by coarsely transforming subgraphs into nodes. We perform subgraph-level tasks as node-level tasks through this translation. By doing so, we can significantly reduce the memory and computational costs in both training and inference. We conduct experiments on four real-world datasets to evaluate performance and efficiency. Our experiments demonstrate that models with S2N translation are more efficient than state-of-the-art models without substantial performance decrease.

1 INTRODUCTION

Graph neural networks (GNNs) have been developed to learn representations of nodes, edges, and graphs (Bronstein et al., 2017; Battaglia et al., 2018; Zhou et al., 2020). Recently, Alsentzer et al. (2020) has proposed SubGNN, a specialized architecture for learning representations of subgraphs. This architecture outperforms prior models; however, it requires a lot of memory and computations to learn the non-trivial structure and various attributes in subgraphs.

In this paper, we propose 'Subgraph-To-Node (S2N)' translation, a novel method to create data structures to solve subgraph-level prediction tasks efficiently. The S2N translation constructs a new graph where its nodes are original subgraphs, and its edges are relations between subgraphs. The GNN models can encode the node representations in the translated graph. Then, we can get the results of the subgraph-level tasks by performing node-level tasks from these node representations.

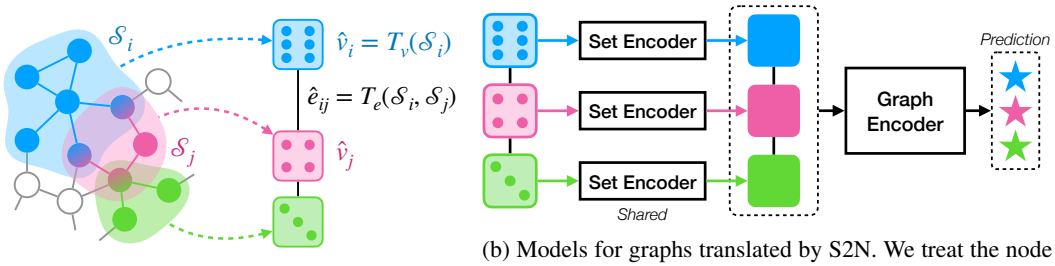
For example, in a knowledge graph where subgraphs are diseases, nodes are symptoms, and edges are relations between symptoms based on knowledge in the medical domain, the goal of the diagnosis task is to predict the type of a disease (i.e., the class of a subgraph). Using S2N translation, we can make a new graph of diseases, nodes of which are diseases and edges of which are relations between them (e.g., whether two diseases share any symptoms).

The S2N translation enables efficient subgraph representation learning for the following two reasons. First, it provides a small and coarse graph in which the number of nodes is reduced to the number of original subgraphs. We can load large batches of subgraphs on the GPU and parallelize the training and inference. Second, there is a wider range of models to choose from in encoding translated graphs. We confirm that even a simple pipeline of DeepSets (Zaheer et al., 2017) and GCN (Kipf & Welling, 2017) can outperform state-of-the-art models.

We conduct experiments with four real-world datasets to evaluate the performance and efficiency of S2N translation. We measure the number of parameters, throughput (samples per second), and latency (seconds per forward pass) for efficiency (Dehghani et al., 2021). We demonstrate that models with S2N translation are more efficient than the existing approach without a significant performance drop. Even some models perform better than baselines in three of the four datasets.

2 SUBGRAPH-TO-NODE TRANSLATION

We introduce the Subgraph-To-Node (S2N) translation and our specific design choices. We also suggest model families for the subgraph prediction task using S2N translated graphs.



(a) The S2N translation. Subgraphs $\mathcal{S}_i$ and $\mathcal{S}_j$ are transformed into nodes $\hat{v}_i$ and $\hat{v}_j$ by T_v, and an edge $\hat{e}_{ij}$ between them is formed by T_e.

(b) Models for graphs translated by S2N. We treat the node $\hat{v}$ in the translated graph as a set of nodes in the original subgraph $\mathcal{S}$. Thus, we apply a set encoder first, then a graph encoder (GNN) to their outputs for the prediction.

Figure 1: Overview of the Subgraph-To-Node translation and the models for translated graphs.

Notations We first summarize the notations in the subgraph representation learning, particularly in the subgraph classification task. Let $\mathcal{G} = (\mathbb{V}, \boldsymbol{A}, \boldsymbol{X})$ be a global graph where $\mathbb{V}$ is a set of nodes ($|\mathbb{V}| = N$), $\boldsymbol{A} \in \{0,1\}^{N \times N}$ is an adjacency matrix, and $\boldsymbol{X} \in \mathbb{R}^{N \times F_0}$ is a node feature matrix. A subgraph $\mathcal{S} = (\mathbb{V}^{\text{sub}}, \boldsymbol{A}^{\text{sub}})$ is a graph formed by subsets of nodes and edges in the global graph $\mathcal{G}$. For the subgraph classification task, there is a set of M subgraphs $\mathbb{S} = \{\mathcal{S}_1, \mathcal{S}_2, ..., \mathcal{S}_M\}$, and for $\mathcal{S}_i = (\mathbb{V}_i^{\text{sub}}, \boldsymbol{A}_i^{\text{sub}})$, the goal is to learn its representation and the logit vector $\boldsymbol{y}_i \in \mathbb{R}^C$ where C is the number of classes.

Overview of S2N Translation The S2N translation reduces the memory and computational costs in the model training and inference by constructing a new coarse graph that summarizes the original subgraph into a node. As illustrated in Figure 1a, for each subgraph $\mathcal{S}_i \in \mathbb{S}$ in the global graph $\mathcal{G}$, we create a node $\hat{v}_i = T_v(\mathcal{S}_i)$ in the translated graph $\hat{\mathcal{G}}$; for all pairs $(\mathcal{S}_i, \mathcal{S}_j)$ of two close subgraphs in $\mathcal{G}$, we make an edge $\hat{e}_{ij} = T_e(\mathcal{S}_i, \mathcal{S}_j)$ between corresponding nodes in $\hat{\mathcal{G}}$. Here, T_v and T_e are translation functions for nodes and edges in $\hat{\mathcal{G}}$, respectively. Formally, the S2N translated graph $\hat{\mathcal{G}} = (\hat{\mathbb{V}}, \hat{\boldsymbol{A}})$ where $|\hat{\mathbb{V}}| = M$ and $\hat{\boldsymbol{A}} \in \{0,1\}^{M \times M}$ is defined by

$$\hat{\mathbb{V}} = \{\hat{v}_i | \hat{v}_i = T_v(\mathcal{S}_i),\ \mathcal{S}_i \in \mathbb{S}\}, \quad \hat{\boldsymbol{A}}[i,j] = \hat{e}_{ij} = T_e(\mathcal{S}_i, \mathcal{S}_j). \tag{1}$$

We can choose any function for T_v and T_e. They can be simple heuristics or modeled with neural networks to learn the graph structure (Franceschi et al., 2019; Kim & Oh, 2021; Fatemi et al., 2021).

Detailed Design of S2N Translation In this paper, we choose straightforward designs of T_v and T_e with negligible translation costs. For T_v, we use a function that ignores the internal structure $\boldsymbol{A}_i^{\text{sub}}$ of the subgraph $\mathcal{S}_i = (\mathbb{V}_i^{\text{sub}}, \boldsymbol{A}_i^{\text{sub}})$ and treats the node as a set (i.e., $\mathbb{V}_i^{\text{sub}}$). For T_e, we make an edge if at least one common node between two subgraphs $\mathcal{S}_i$ and $\mathcal{S}_j$. They are defined as follows:

$$\hat{v}_i = T_v(\mathcal{S}_i) = \mathbb{V}_i^{\text{sub}}, \quad \hat{e}_{ij} = T_e(\mathcal{S}_i, \mathcal{S}_j) = \begin{cases} 1 & \text{if } |\mathbb{V}_i^{\text{sub}} \cap \mathbb{V}_j^{\text{sub}}| \neq 0 \\ 0 & \text{otherwise} \end{cases}. \tag{2}$$

In some cases, this particular translation provides a more intuitive description for real-world problems than a form of subgraphs. For a fitness social network (EM-User) from Alsentzer et al. (2020) (subgraphs: users, nodes: workouts, edges: whether multiple users complete workouts), it will be translated into a network of users connected if they complete the same workouts. This graph directly expresses the relation between users and follows the conventional approach to express social networks where nodes are users.

Models for S2N Translated Graphs We propose simple but strong model pipelines for S2N translated graphs. Since the node $\hat{v}_i$ is a set of original nodes in $\mathcal{S}_i$, we first use a set encoder $\mathcal{E}_{\text{set}} : \hat{\mathbb{V}} \to \mathbb{R}^F$ (Wagstaff et al., 2021) where F is a dimension of the representation. It takes a set of node features in $\hat{v}_i$ as an input and generates the representation $\hat{\boldsymbol{h}}_i \in \mathbb{R}^F$ of $\hat{v}_i$, that is,

$$\hat{\boldsymbol{h}}_i = \mathcal{E}_{\text{set}}(\hat{v}_i) = \mathcal{E}_{\text{set}}(\mathbb{V}_i^{\text{sub}}) = \mathcal{E}_{\text{set}}(\{\boldsymbol{x}_u | \boldsymbol{x}_u = \boldsymbol{X}[u,:], u \in \mathbb{V}_i^{\text{sub}}\}). \tag{3}$$

Then, given the node representation $\hat{\boldsymbol{h}}_i$, we apply a graph encoder $\mathcal{E}_{\text{graph}} : \mathbb{R}^{M \times F} \times \{0,1\}^{M \times M} \to \mathbb{R}^{M \times C}$ to get the logit vector $\hat{\boldsymbol{y}}_i \in \mathbb{R}^C$. For the input and output of $\mathcal{E}_{\text{graph}}$, we use matrices $\hat{\boldsymbol{H}} \in \mathbb{R}^{M \times F}$ and $\hat{\boldsymbol{Y}} \in \mathbb{R}^{M \times C}$ where the ith rows are $\hat{\boldsymbol{h}}_i$ and $\hat{\boldsymbol{y}}_i$, respectively.

$$\hat{\boldsymbol{Y}} = \mathcal{E}_{\text{graph}}(\hat{\boldsymbol{H}}, \hat{\boldsymbol{A}}). \tag{4}$$

Table 1: Statistics of real-world datasets before and after S2N translation.

	PPI-BP	HPO-Neuro	HPO-Metab	EM-User
# nodes (before $\rightarrow$ after)	17.1K $\rightarrow$ 1.6K	14.6K $\rightarrow$ 4.0K	14.6K $\rightarrow$ 2.4K	57.3K $\rightarrow$ 324
# edges (before $\rightarrow$ after)	317.0K $\rightarrow$ 55.7K	3.2M $\rightarrow$ 6.6M	3.2M $\rightarrow$ 2.5M	4.6M $\rightarrow$ 87.2K
Density (before $\rightarrow$ after)	0.002 $\rightarrow$ 0.021	0.030 $\rightarrow$ 0.413	0.030 $\rightarrow$ 0.439	0.003 $\rightarrow$ 0.830
# classes	6	10	6	2
Node / Edge homophily	0.449 / 0.391	0.176 / 0.175	0.195 / 0.189	0.514 / 0.511

For $\mathcal{E}_{\text{graph}}$, we can take any GNNs that perform message-passing between nodes. This node-level message-passing on translated graphs is analogous to message-passing at the subgraph level in SubGNN (Alsentzer et al., 2020).

3 Experiments

This section describes the experimental setup, including datasets, training, evaluation, and models.

Datasets We use four real-world datasets, PPI-BP, HPO-Neuro, HPO-Metab, and EM-User, introduced in Alsentzer et al. (2020). The task is subgraph classification where nodes $\mathbb{V}$, edges $\boldsymbol{A}$, and subgraphs $\mathcal{S} \in \mathbb{S}$ are given in datasets. There are two input node features $\boldsymbol{X}$ pretrained with GIN or GraphSAINT from the same paper. Detailed description and statistics are in Appendix B.

Training and Evaluation In the original setting from the SubGNN paper, evaluation (i.e., validation and test) samples cannot be seen during the training stage. Following this protocol, we create different S2N graphs for each stage using train and evaluation sets of subgraphs ($\mathbb{S}_{\text{train}}$ and $\mathbb{S}_{\text{eval}}$). For the S2N translation, we use $\mathbb{S}_{\text{train}}$ only in the training stage, and use both $\mathbb{S}_{\text{train}} \cup \mathbb{S}_{\text{eval}}$ in the evaluation stage. That is, we predict unseen nodes based on structures translated from $\mathbb{S}_{\text{train}} \cup \mathbb{S}_{\text{eval}}$ in the evaluation stage. In this respect, node classification on S2N translated graphs is inductive.

Models for S2N Translated Graphs We use two- or four-layer DeepSets (Zaheer et al., 2017) with sum or max operations as $\mathcal{E}_{\text{set}}$ for all S2N models. For $\mathcal{E}_{\text{graph}}$, we use well-known graph neural networks: GCN (Kipf & Welling, 2017) and GAT (Veličković et al., 2018). In addition, LINKX (Lim et al., 2021) and FAGCN (Bo et al., 2021), models that perform well in non-homophilous graphs are employed. All GNNs are one- or two-layer models. See Appendix C.1 for their hyperparameters. Since LINKX is designed for the transductive setting, we make a small change in LINKX to work in the inductive setting. We call this variant LINKX-I. See Appendix C.2 for this modification.

Baselines We use current state-of-the-art models for subgraph classification as baselines: Sub2Vec (Adhikari et al., 2018), Graph-level GIN (Xu et al., 2019), and SubGNN (Alsentzer et al., 2020). We report the best performance among three variants for Sub2Vec (N, S, and NS) and two results by different pretrained embeddings for SubGNN. All baselines results are reprinted from Alsentzer et al. (2020).

4 Results

In this section, we analyze the characteristics of S2N translated graphs and compare our models and baselines on classification performance and efficiency.

Analysis of S2N Translated Graphs Table 1 summarizes dataset statistics before and after S2N translation, including node (Pei et al., 2020) and edge homophily (Zhu et al., 2020). Except for HPO-Neuro, translated graphs have a smaller number of nodes ($\times 0.006$ – $\times 0.03$) and edges ($\times 0.17$ – $\times 0.78$) than original graphs. For HPO-Neuro, it has twice as many edges as the original graph, but has $\times 0.27$ fewer nodes. Since the number of edges decreased less than nodes, translated graphs are denser than originals ($\times 9.7$ – $\times 297$). We also find that they are non-homophilous (low homophily), which means there are many connected nodes of different classes.

Note that we propose multi-label node and edge homophily for multi-label datasets (HPO-Neuro):

$$h^{\text{node, ml}} = \frac{1}{|\mathbb{V}|} \sum_{v \in \mathbb{V}} \left(\frac{1}{|\mathcal{N}(v)|} \sum_{u \in \mathcal{N}(v)} \frac{|\mathbb{L}_u \cap \mathbb{L}_v|}{|\mathbb{L}_u \cup \mathbb{L}_v|} \right), \; h^{\text{edge, ml}} = \frac{1}{|\mathbb{A}|} \sum_{(u,v) \in \mathbb{A}} \frac{|\mathbb{L}_u \cap \mathbb{L}_v|}{|\mathbb{L}_u \cup \mathbb{L}_v|}, \tag{5}$$

where $\mathbb{L}_v$ is a set of labels of v, $\mathcal{N}(v)$ is a set of neighbors of v, and $\mathbb{A} = \{(u,v) | \boldsymbol{A}[u,v] = 1\}$. They generalize the existing multi-class homophily and we discuss more in Appendix D.

Table 2: Summary of classification performance in mean micro-F1 score over 10 random seeds for real-world datasets. Results of the unpaired t-test with the best baseline are denoted by superscripts ($\star$: outperformed with p-value $< .05$, $\sim$: p-value $> .05$). We mark with daggers (†) the reprinted results from Alsentzer et al. (2020).

Model	Embedding	PPI-BP	HPO-Neuro	HPO-Metab	EM-User
Sub2Vec Best†	-	$30.9_{\pm 2.3}$	$22.3_{\pm 6.5}$	$13.2_{\pm 4.7}$	$85.9_{\pm 1.4}$
Graph-level GIN†	-	$39.8_{\pm 5.8}$	$53.5_{\pm 3.2}$	$45.2_{\pm 2.5}$	$56.1_{\pm 5.9}$
SubGNN†	GIN	$59.9_{\pm 2.4}$	$63.2_{\pm 1.0}$	$53.7_{\pm 2.3}$	$81.4_{\pm 4.6}$
SubGNN†	GraphSAINT	$58.3_{\pm 1.7}$	$64.4_{\pm 1.9}$	$42.8_{\pm 3.5}$	$81.6_{\pm 4.0}$
S2N + GCN	GIN	$61.4^{\sim}_{\pm 1.6}$	$59.0_{\pm 0.7}$	$51.6_{\pm 1.8}$	$70.2_{\pm 2.3}$
S2N + GCN	GraphSAINT	$60.6^{\sim}_{\pm 1.2}$	$59.9_{\pm 0.7}$	$50.6_{\pm 1.9}$	$69.0_{\pm 4.5}$
S2N + GAT	GIN	$60.8^{\sim}_{\pm 2.7}$	$53.1_{\pm 1.9}$	$47.9_{\pm 3.4}$	$71.4_{\pm 6.3}$
S2N + GAT	GraphSAINT	$60.4^{\sim}_{\pm 1.4}$	$54.6_{\pm 2.0}$	$49.4_{\pm 4.5}$	$80.2_{\pm 4.8}$
S2N + LINKX-I	GIN	$60.9^{\sim}_{\pm 1.8}$	$62.9_{\pm 1.1}$	$55.9^{\sim}_{\pm 2.6}$	$83.3_{\pm 3.6}$
S2N + LINKX-I	GraphSAINT	$61.3^{\sim}_{\pm 1.5}$	$62.9^{\sim}_{\pm 1.3}$	$57.9^{\star}_{\pm 2.1}$	$84.7^{\sim}_{\pm 2.9}$
S2N + FAGCN	GIN	$62.8^{\star}_{\pm 1.2}$	$64.5^{\sim}_{\pm 1.3}$	$58.2^{\star}_{\pm 2.7}$	$80.0_{\pm 4.0}$
S2N + FAGCN	GraphSAINT	$60.7^{\sim}_{\pm 3.1}$	$63.3^{\sim}_{\pm 1.1}$	$57.5^{\star}_{\pm 3.3}$	$82.9_{\pm 3.7}$

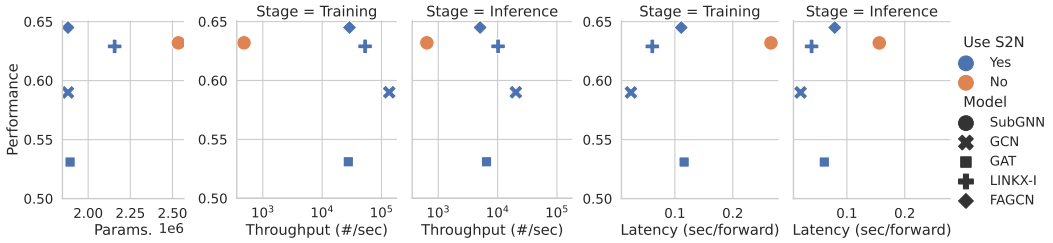


Figure 2: The # of parameters, throughput, and latency of S2N models and SubGNN on HPO-Neuro.

Performance In Table 2, we report the mean and standard deviation of micro-F1 score over ten runs of our models and baselines. LINKX-I and FAGCN, which are known to work well in non-homophilous graphs, perform on par with or better than the best baseline in 12 of 16 cases. For PPI-BP and HPO-Metab, some models even outperform SubGNN with statistical significance (p-value $< .05$). Notably, all S2N models outperform SubGNN in the PPI-BP, which has relatively high homophily. GCN and GAT underperform LINKX-I and FAGCN for most experiments.

Efficiency In Figure 2, we show the number of parameters, throughput (subgraphs per second), and latency (seconds per forward pass) of S2N models and SubGNN on HPO-Neuro. We make two observations in this figure. First, S2N models use fewer parameters and process many samples faster (i.e., higher throughput and lower latency) than SubGNN. In particular, for throughput, S2N models can process 8 to 280 times more samples than SubGNN for the same amount of time. Second, the training throughput is higher than the inference throughput in S2N models. Generally, as in SubGNN, throughput increases in the inference step, which does not require gradient calculation. This is because the S2N models use message-passing between training and inference samples (See §3). Thus, they compute both training and inference samples, requiring more computation for the inference stage. See Appendix E for the experimental setup and results on other datasets.

5 Conclusion and Future Research

We propose Subgraph-To-Node (S2N) translation, a novel way to learn representations of subgraphs efficiently. Using S2N, we create a new graph where nodes are original subgraphs, edges are relations between subgraphs, and perform subgraph-level tasks as node-level tasks. S2N translation significantly reduces memory and computation costs without performance degradation.

There are limitations in this research. First, we used simple translate functions and did not explore them deeply. *How do we define aggregated features and structures in translated graphs?* Second, we do not yet know the properties of subgraphs that affect the performance of the S2N translation. *What properties of subgraphs can be learned after translation?* We leave these as future directions.

REFERENCES

Bijaya Adhikari, Yao Zhang, Naren Ramakrishnan, and B Aditya Prakash. Sub2vec: Feature learning for subgraphs. In *Pacific-Asia Conference on Knowledge Discovery and Data Mining*, pp. 170–182. Springer, 2018.

Takuya Akiba, Shotaro Sano, Toshihiko Yanase, Takeru Ohta, and Masanori Koyama. Optuna: A next-generation hyperparameter optimization framework. In *Proceedings of the 25th ACM SIGKDD international conference on knowledge discovery & data mining*, pp. 2623–2631, 2019.

Emily Alsentzer, Samuel G Finlayson, Michelle M Li, and Marinka Zitnik. Subgraph neural networks. *Proceedings of Neural Information Processing Systems, NeurIPS*, 2020.

Michael Ashburner, Catherine A Ball, Judith A Blake, David Botstein, Heather Butler, J Michael Cherry, Allan P Davis, Kara Dolinski, Selina S Dwight, Janan T Eppig, et al. Gene ontology: tool for the unification of biology. *Nature genetics*, 25(1):25–29, 2000.

Peter W Battaglia, Jessica B Hamrick, Victor Bapst, Alvaro Sanchez-Gonzalez, Vinicius Zambaldi, Mateusz Malinowski, Andrea Tacchetti, David Raposo, Adam Santoro, Ryan Faulkner, et al. Relational inductive biases, deep learning, and graph networks. *arXiv preprint arXiv:1806.01261*, 2018.

Deyu Bo, Xiao Wang, Chuan Shi, and Huawei Shen. Beyond low-frequency information in graph convolutional networks. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 35, pp. 3950–3957, 2021.

Giorgos Bouritsas, Fabrizio Frasca, Stefanos Zafeiriou, and Michael M Bronstein. Improving graph neural network expressivity via subgraph isomorphism counting. *arXiv preprint arXiv:2006.09252*, 2020.

Gecia Bravo Hermsdorff and Lee Gunderson. A unifying framework for spectrum-preserving graph sparsification and coarsening. *Advances in Neural Information Processing Systems*, 32, 2019.

Michael M Bronstein, Joan Bruna, Yann LeCun, Arthur Szlam, and Pierre Vandergheynst. Geometric deep learning: going beyond euclidean data. *IEEE Signal Processing Magazine*, 34(4):18–42, 2017.

Chen Cai, Dingkang Wang, and Yusu Wang. Graph coarsening with neural networks. In *International Conference on Learning Representations*, 2021. URL `https://openreview.net/forum?id=uxpzitPEooJ`.

Wei-Lin Chiang, Xuanqing Liu, Si Si, Yang Li, Samy Bengio, and Cho-Jui Hsieh. Cluster-gcn: An efficient algorithm for training deep and large graph convolutional networks. In *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, pp. 257–266, 2019.

Gene Ontology Consortium. The gene ontology resource: 20 years and still going strong. *Nucleic acids research*, 47(D1):D330–D338, 2019.

Mostafa Dehghani, Anurag Arnab, Lucas Beyer, Ashish Vaswani, and Yi Tay. The efficiency misnomer. *arXiv preprint arXiv:2110.12894*, 2021.

Chenhui Deng, Zhiqiang Zhao, Yongyu Wang, Zhiru Zhang, and Zhuo Feng. Graphzoom: A multi-level spectral approach for accurate and scalable graph embedding. In *International Conference on Learning Representations*, 2020. URL `https://openreview.net/forum?id=r1lGO0EKDH`.

William Falcon and The PyTorch Lightning team. PyTorch Lightning, 3 2019. URL `https://github.com/PyTorchLightning/pytorch-lightning`.

Bahare Fatemi, Layla El Asri, and Seyed Mehran Kazemi. Slaps: Self-supervision improves structure learning for graph neural networks. *Advances in Neural Information Processing Systems*, 34, 2021.

Matthias Fey and Jan E. Lenssen. Fast graph representation learning with PyTorch Geometric. In *International Conference on Learning Representations Workshop on Representation Learning on Graphs and Manifolds*, 2019.

Luca Franceschi, Mathias Niepert, Massimiliano Pontil, and Xiao He. Learning discrete structures for graph neural networks. In *International conference on machine learning*, pp. 1972–1982. PMLR, 2019.

William L Hamilton, Rex Ying, and Jure Leskovec. Inductive representation learning on large graphs. In *Proceedings of the 31st International Conference on Neural Information Processing Systems*, pp. 1025–1035, 2017.

Taila Hartley, Gabrielle Lemire, Kristin D Kernohan, Heather E Howley, David R Adams, and Kym M Boycott. New diagnostic approaches for undiagnosed rare genetic diseases. *Annual review of genomics and human genetics*, 21:351–372, 2020.

Zengfeng Huang, Shengzhong Zhang, Chong Xi, Tang Liu, and Min Zhou. Scaling up graph neural networks via graph coarsening. In *Proceedings of the 27th ACM SIGKDD Conference on Knowledge Discovery & Data Mining*, pp. 675–684, 2021.

Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. In *International conference on machine learning*, pp. 448–456. PMLR, 2015.

Yu Jin, Andreas Loukas, and Joseph JaJa. Graph coarsening with preserved spectral properties. In *International Conference on Artificial Intelligence and Statistics*, pp. 4452–4462. PMLR, 2020.

Dongkwan Kim and Alice Oh. How to find your friendly neighborhood: Graph attention design with self-supervision. In *International Conference on Learning Representations*, 2021. URL `https://openreview.net/forum?id=Wi5KUNlqWty`.

Thomas N. Kipf and Max Welling. Semi-supervised classification with graph convolutional networks. In *International Conference on Learning Representations*, 2017.

Sebastian Köhler, Leigh Carmody, Nicole Vasilevsky, Julius O B Jacobsen, Daniel Danis, Jean-Philippe Gourdine, Michael Gargano, Nomi L Harris, Nicolas Matentzoglu, Julie A McMurry, et al. Expansion of the human phenotype ontology (hpo) knowledge base and resources. *Nucleic acids research*, 47(D1):D1018–D1027, 2019.

Derek Lim, Felix Hohne, Xiuyu Li, Sijia Linda Huang, Vaishnavi Gupta, Omkar Bhalerao, and Ser Nam Lim. Large scale learning on non-homophilous graphs: New benchmarks and strong simple methods. *Advances in Neural Information Processing Systems*, 34, 2021.

Andreas Loukas. Graph reduction with spectral and cut guarantees. *Journal of Machine Learning Research*, 20:1–42, 2019.

Andreas Loukas and Pierre Vandergheynst. Spectrally approximating large graphs with smaller graphs. In *International Conference on Machine Learning*, pp. 3237–3246. PMLR, 2018.

Changping Meng, S Chandra Mouli, Bruno Ribeiro, and Jennifer Neville. Subgraph pattern neural networks for high-order graph evolution prediction. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 32, 2018.

Dylan Mordaunt, David Cox, and Maria Fuller. Metabolomics to improve the diagnostic efficiency of inborn errors of metabolism. *International journal of molecular sciences*, 21(4):1195, 2020.

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 AAAI Conference on Artificial Intelligence*, volume 33, pp. 4602–4609, 2019.

Jianmo Ni, Larry Muhlstein, and Julian McAuley. Modeling heart rate and activity data for personalized fitness recommendation. In *The World Wide Web Conference*, pp. 1343–1353, 2019.

Mathias Niepert, Mohamed Ahmed, and Konstantin Kutzkov. Learning convolutional neural networks for graphs. In *International conference on machine learning*, pp. 2014–2023. PMLR, 2016.

Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, et al. Pytorch: An imperative style, high-performance deep learning library. In *Advances in neural information processing systems*, pp. 8026–8037, 2019.

Hongbin Pei, Bingzhe Wei, Kevin Chen-Chuan Chang, Yu Lei, and Bo Yang. Geom-gcn: Geometric graph convolutional networks. In *International Conference on Learning Representations*, 2020. URL https://openreview.net/forum?id=S1e2agrFvS.

Jiezhong Qiu, Qibin Chen, Yuxiao Dong, Jing Zhang, Hongxia Yang, Ming Ding, Kuansan Wang, and Jie Tang. Gcc: Graph contrastive coding for graph neural network pre-training. In *Proceedings of the 26th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, pp. 1150–1160, 2020.

Aravind Subramanian, Pablo Tamayo, Vamsi K Mootha, Sayan Mukherjee, Benjamin L Ebert, Michael A Gillette, Amanda Paulovich, Scott L Pomeroy, Todd R Golub, Eric S Lander, et al. Gene set enrichment analysis: a knowledge-based approach for interpreting genome-wide expression profiles. *Proceedings of the National Academy of Sciences*, 102(43):15545–15550, 2005.

Petar Veličković, Guillem Cucurull, Arantxa Casanova, Adriana Romero, Pietro Liò, and Yoshua Bengio. Graph Attention Networks. In *International Conference on Learning Representations*, 2018. URL https://openreview.net/forum?id=rJXMpikCZ.

Edward Wagstaff, Fabian B Fuchs, Martin Engelcke, Michael A Osborne, and Ingmar Posner. Universal approximation of functions on sets. *arXiv preprint arXiv:2107.01959*, 2021.

Keyulu Xu, Weihua Hu, Jure Leskovec, and Stefanie Jegelka. How powerful are graph neural networks? In *International Conference on Learning Representations*, 2019. URL https://openreview.net/forum?id=ryGs6iA5Km.

Yuning You, Tianlong Chen, Yongduo Sui, Ting Chen, Zhangyang Wang, and Yang Shen. Graph contrastive learning with augmentations. *Advances in Neural Information Processing Systems*, 33, 2020.

Manzil Zaheer, Satwik Kottur, Siamak Ravanbakhsh, Barnabas Poczos, Russ R Salakhutdinov, and Alexander J Smola. Deep sets. *Advances in neural information processing systems*, 30, 2017.

Hanqing Zeng, Hongkuan Zhou, Ajitesh Srivastava, Rajgopal Kannan, and Viktor Prasanna. Graph-saint: Graph sampling based inductive learning method. In *International Conference on Learning Representations*, 2020. URL https://openreview.net/forum?id=BJe8pkHFwS.

Jie Zhou, Ganqu Cui, Shengding Hu, Zhengyan Zhang, Cheng Yang, Zhiyuan Liu, Lifeng Wang, Changcheng Li, and Maosong Sun. Graph neural networks: A review of methods and applications. *AI Open*, 1:57–81, 2020.

Jiong Zhu, Yujun Yan, Lingxiao Zhao, Mark Heimann, Leman Akoglu, and Danai Koutra. Beyond homophily in graph neural networks: Current limitations and effective designs. *Advances in Neural Information Processing Systems*, 33:7793–7804, 2020.

Marinka Zitnik, Rok Sosic, and Jure Leskovec. Biosnap datasets: Stanford biomedical network dataset collection. *Note: http://snap. stanford. edu/biodata Cited by*, 5(1), 2018.

Table 3: Statistics of real-world datasets in original forms (before S2N translation).

	PPI-BP	HPO-Neuro	HPO-Metab	EM-User
# nodes	17,080	14,587	14,587	57,333
# edges	316,951	3,238,174	3,238,174	4,573,417
# subgraphs	1,591	4,000	2,400	324
Train/Valid/Test splits	80/10/10	80/10/10	80/10/10	75/15/15
Density of the global graph	0.0022	0.0304	0.0304	0.0028
Average # nodes / subgraph	$10.2_{\pm 10.5}$	$14.8_{\pm 6.5}$	$14.4_{\pm 6.2}$	$155.4_{\pm 100.2}$
Average density of subgraphs	$0.216_{\pm 0.188}$	$0.767_{\pm 0.141}$	$0.757_{\pm 0.149}$	$0.010_{\pm 0.006}$
Average # components / subgraph	$7.0_{\pm 5.5}$	$1.5_{\pm 0.7}$	$1.6_{\pm 0.7}$	$52.1_{\pm 15.3}$
# classes	6	10	6	2
Single- or multi-label	Single-label	Multi-label	Single-label	Single-label

A Related Work

Subgraph Representation Learning There have been various approaches to use subgraphs for expressiveness (Niepert et al., 2016; Morris et al., 2019; Bouritsas et al., 2020), scalability (Hamilton et al., 2017; Chiang et al., 2019; Zeng et al., 2020), and augmentation (Qiu et al., 2020; You et al., 2020). However, only a few studies deal with learning representations of subgraphs. The Subgraph Pattern Neural Network (Meng et al., 2018) learns subgraph evolution patterns but does not generalize to subgraphs with varying sizes. The Subgraph Neural Network (SubGNN) (Alsentzer et al., 2020) is the first approach of subgraph representation learning using topology, positions, and connectivity. However, SubGNN requires large memory and computation costs to encode the mentioned information for prediction. Our method allows efficient learning of subgraph representations without a complex model design.

Graph Coarsening Our S2N translation summarizes subgraphs into nodes, and in that sense, it is related to graph coarsening methods (Loukas & Vandergheynst, 2018; Loukas, 2019; Bravo Hermsdorff & Gunderson, 2019; Jin et al., 2020; Deng et al., 2020; Cai et al., 2021; Huang et al., 2021). They focus on creating coarse graphs while preserving specific properties in a given graph, such as spectral similarity or distance. The biggest challenge in this line of research is to learn the boundary of nodes in coarse graphs (or super-nodes) where they are unknown. Since we designate super-nodes in the form of subgraphs, coarse graphs can be created with simple heuristics.

B Datasets

Subgraph datasets PPI-BP, HPO-Neuro, HPO-Metab, and EM-User are proposed in Alsentzer et al. (2020), and can be downloaded from the GitHub repository[1]. In Table 3, we summarize statistics of datasets in original forms without S2N translation. We describe their nodes, edges, subgraphs, tasks, and references in the following paragraphs.

PPI-BP The global graph of PPI-BP (Zitnik et al., 2018; Subramanian et al., 2005; Consortium, 2019; Ashburner et al., 2000) is a human protein-protein interaction (PPI) network; nodes are proteins, and edges are whether there is a physical interaction between proteins. Subgraphs are sets of proteins in the same biological process (e.g., alcohol bio-synthetic process). The task is to classify processes into six categories.

HPO-Neuro and HPO-Metab These two HPO (Human Phenotype Ontology) datasets (Hartley et al., 2020; Köhler et al., 2019; Mordaunt et al., 2020) are knowledge graphs of phenotypes (i.e., symptoms) of rare neurological and metabolic diseases. Each subgraph is a collection of symptoms associated with a monogenic disorder. The task is to diagnose the rare disease: classifying the disease type among subcategories (ten for HPO-Neuro and six for HPO-Metab).

[1] `https://github.com/mims-harvard/SubGNN`

EM-User EM-User (Users in EndoMondo) dataset is a social fitness network from Endomondo (Ni et al., 2019). Here, subgraphs are users, nodes are workouts, and edges exist between workouts completed by multiple users. Each subgraph represents the workout history of a user. The task is to profile a user's gender.

C MODELS

This section describes the model details we used: hyperparameter tuning and LINKX-I design. All models are implemented with PyTorch (Paszke et al., 2019), PyTorch Geometric (Fey & Lenssen, 2019), and PyTorch Lightning (Falcon & The PyTorch Lightning team, 2019).

C.1 HYPERPARAMETERS

We tune seven hyperparameters using TPE (Tree-structured Parzen Estimator) algorithm in Optuna (Akiba et al., 2019) by 30 trials: weight decay (10^{-9} – 10^{-6}), the number of layers in $\mathcal{E}_{\text{set}}$ (2 or 4), the number of layers in $\mathcal{E}_{\text{graph}}$ (1 or 2), the pooling operating in $\mathcal{E}_{\text{set}}$ (sum or max), dropout of channels and edges ($\{0.0, 0.1, ..., 0.5\}$), and gradient clipping ($\{0.0, 0.1, ..., 0.5\}$). We use batch normalization (Ioffe & Szegedy, 2015) for all S2N models except LINKX-I.

C.2 INDUCTIVE LINKX (LINKX-I)

Given node features $\boldsymbol{X} \in \mathbb{R}^{N \times F_0}$ and an adjacent matrix $\boldsymbol{A} \in \mathbb{R}^{N \times N}$, LINKX (Lim et al., 2021) model computes the logit matrix $\boldsymbol{Y} \in \mathbb{R}^{N \times C}$ by following equations,

$$\boldsymbol{H}_A = \text{MLP}_A(\boldsymbol{A}) \in \mathbb{R}^{N \times F},\ \boldsymbol{H}_X = \text{MLP}_X(\boldsymbol{X}) \in \mathbb{R}^{N \times F}, \tag{6}$$

$$\boldsymbol{Y} = \text{MLP}_f\left(\text{ReLU}(\boldsymbol{W}_f[\boldsymbol{H}_A \| \boldsymbol{H}_X] + \boldsymbol{H}_A + \boldsymbol{H}_X)\right) \text{ where } \boldsymbol{W}_f \in \mathbb{R}^{F \times 2F} \tag{7}$$

The computation of the first single layer in $\text{MLP}_A = \text{Linear}_A \circ \text{ReLU} \circ \text{Linear}_A \circ ...$ is as follows

$$\text{Linear}_A(\boldsymbol{A}) = \boldsymbol{A}\boldsymbol{W}_A, \quad (\boldsymbol{A}\boldsymbol{W}_A)[i,k] = \sum_{j \in \mathcal{N}(i)} \boldsymbol{W}_A[j,k], \quad \boldsymbol{W}_A \in \mathbb{R}^{N \times F}. \tag{8}$$

In our inductive setting, we have $\hat{\boldsymbol{A}}_{\text{train}}$ and $\hat{\boldsymbol{A}}_{\text{train+eval}}$, the shapes of which are

$$\hat{\boldsymbol{A}}_{\text{train}} \in \{0,1\}^{M_{\text{train}} \times M_{\text{train}}},\ \hat{\boldsymbol{A}}_{\text{train+eval}} \in \{0,1\}^{(M_{\text{train}}+M_{\text{eval}}) \times (M_{\text{train}}+M_{\text{eval}})}. \tag{9}$$

If we train MLP_A on $\hat{\boldsymbol{A}}_{\text{train}}$, we cannot process $\hat{\boldsymbol{A}}_{\text{train+eval}}$, because shapes of matrix multiplication do not match (i.e., $M_{\text{train}} + M_{\text{eval}} \neq M_{\text{train}}$). Thus, in LINKX-I, we use the modified matrix multiplication $\circledast$ in MLP_A to aggregate parameters corresponding training nodes only. Formally, for the matrix $\hat{\boldsymbol{A}}_{M_*} \in \mathbb{R}^{M_* \times M_*}$ of the arbitrary shape,

$$(\hat{\boldsymbol{A}}_{M_*} \circledast \boldsymbol{W}_A)[i,k] = \sum_{j \in \mathcal{N}(i) \wedge j \in \mathbb{V}_{\text{train}}} \boldsymbol{W}_A[j,k], \quad (\hat{\boldsymbol{A}}_{M_*} \circledast \boldsymbol{W}_A) \in \mathbb{R}^{M_* \times F} \tag{10}$$

The remaining parts are the same as LINKX.

D GENERALIZATION OF HOMOPHILY TO MULTI-LABEL CLASSIFICATION

Node (Pei et al., 2020) and edge homophily (Zhu et al., 2020) are defined by,

$$h^{\text{edge}} = \frac{|\{(u,v) | (u,v) \in \mathbb{A} \wedge y_u = y_v\}|}{|\mathbb{A}|},\ h^{\text{node}} = \frac{1}{|\mathbb{V}|} \sum_{v \in \mathbb{V}} \frac{|\{(u,v) | u \in \mathcal{N}(v) \wedge y_u = y_v\}|}{|\mathcal{N}(v)|}, \tag{11}$$

where y_v is the label of the node v. In the main paper, we define multi-label node and edge homophily by,

$$h^{\text{edge, ml}} = \frac{1}{|\mathbb{A}|} \sum_{(u,v) \in \mathbb{A}} \frac{|\mathbb{L}_u \cap \mathbb{L}_v|}{|\mathbb{L}_u \cup \mathbb{L}_v|},\ h^{\text{node, ml}} = \frac{1}{|\mathbb{V}|} \sum_{v \in \mathbb{V}} \left(\frac{1}{|\mathcal{N}(v)|} \sum_{u \in \mathcal{N}(v)} \frac{|\mathbb{L}_u \cap \mathbb{L}_v|}{|\mathbb{L}_u \cup \mathbb{L}_v|} \right). \tag{12}$$

If we compute $r = \frac{|\mathbb{L}_u \cap \mathbb{L}_v|}{|\mathbb{L}_u \cup \mathbb{L}_v|}$ for single-label multi-class graphs, $r = \frac{1}{1} = 1$ for nodes of same classes, and $r = \frac{0}{2} = 0$ for nodes of different classes. That makes $h^{\text{edge, ml}} = h^{\text{edge}}$ and $h^{\text{node, ml}} = h^{\text{node}}$ for single-label graphs.

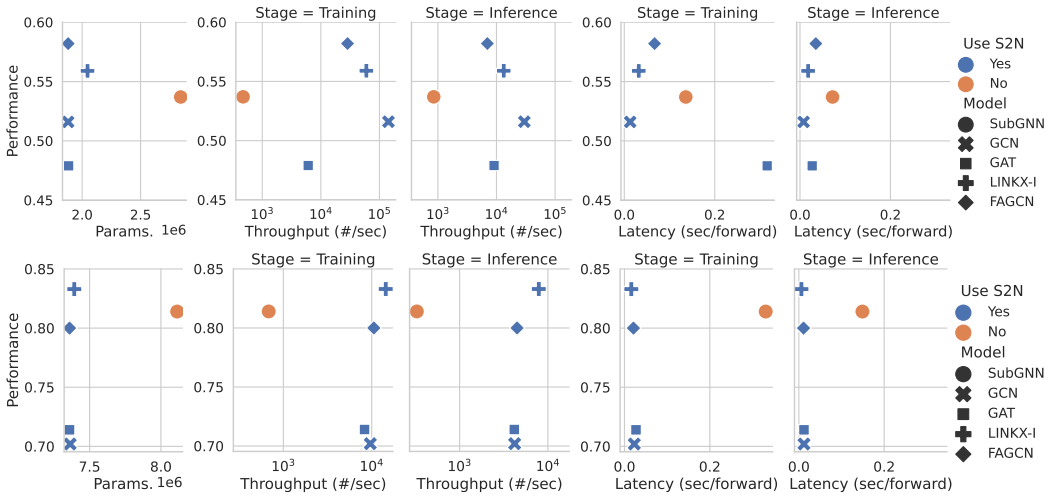


Figure 3: The number of parameters, throughput, and latency of S2N models and SubGNN on HPO-Metab (Top) and EM-User (Bottom).

E Details on Efficiency Experiments

E.1 Experimental Setup

We compute throughput (subgraphs per second) and latency (seconds per forward pass) by following equations.

$$\text{Training throughput} = \frac{\text{\# of training subgraphs}}{\text{training wall-clock time (seconds) / \# of epochs}}, \tag{13}$$

$$\text{Inference throughput} = \frac{\text{\# of validation subgraphs}}{\text{validation wall-clock time (seconds) / \# of epochs}}, \tag{14}$$

$$\text{Training latency} = \frac{\text{training wall-clock time (seconds)}}{\text{\# of training batches}}, \tag{15}$$

$$\text{Inference latency} = \frac{\text{validation wall-clock time (seconds)}}{\text{\# of validation batches}}. \tag{16}$$

We use the best hyperparameters (including batch sizes) for each model and take the mean wall-clock time over 50 epochs. Our computation device is Intel(R) Xeon(R) CPU E5-2640 v4 and single GeForce GTX 1080 Ti.

E.2 Results

In Figure 3, we report the number of parameters, throughput, and latency on HPO-Metab and EM-User. We cannot experiment with PPI-BP since it takes more than 48 hours in pre-computation. The result generally follows analyses of HPO-Neuro case reported in the main paper (Figure 2). One exception is that the training latency of GAT in HPO-Metab is higher than that of SubGNN. Note that latency ignores the parallelism from large batch sizes (Dehghani et al., 2021). Our model can show relatively high latency since it requires full batch computation.