Scalable Graph Size Reduction for Efficient GNN Application

Pavel Procházka[1], Michal Mareš[1,2] and Marek Dědič[1,3]

[1]Cisco Systems, Inc., Karlovo náměstí 10, Prague, Czech Republic
[2]Czech Technical University in Prague, Technická 2, Prague, Czech Republic
[3]Czech Technical University in Prague, Trojanova 13, Prague, Czech Republic

Abstract

Graph neural networks (GNN) present a dominant framework for representation learning on graphs for the past several years. The main strength of GNNs lies in the fact that they can simultaneously learn from both node related attributes and relations between nodes, represented by edges. In tasks leading to large graphs, GNN often requires significant computational resources to achieve its superior performance. In order to reduce the computational cost, methods allowing for a flexible balance between complexity and performance could be useful. In this work, we propose a simple scalable task-aware graph preprocessing procedure allowing us to obtain a reduced graph such as GNN achieves a given desired performance on the downstream task. In addition, the proposed preprocessing allows for fitting the reduced graph and GNN into a given memory/computational resources. The proposed preprocessing is evaluated and compared with several reference scenarios on conventional GNN benchmark datasets.

1. Introduction

Graph neural networks (GNNs) have proven to achieve superior performance on a number of graph datasets and are adopted in industrial applications across many fields. Superior GNN performance is, however, often paid for by a numerically intensive training procedure with a significant memory footprint. In addition, fine tuning parameters of a complex GNN can prove challenging as well. The issue is emphasised whenever the problem modeled by the graph is evolving in time and retraining is required to keep the model accurate.

In many industrial applications, high computational cost caused by training complex models might be an issue. Moreover, some real-world problems lead to high amount of data that does not fit the memory of a single machine. In order to apply GNNs to such big data, either some data reduction or their processing in a distributive way is required.

Traditional simple structure-agnostic baselines like the Naive Bayes classifier usually scale very well at a reasonable cost, making them useful in scenarios where such a simple model performs well enough.

In this paper, we address the problem of complexity-performance trade-off and propose a method operating on an arbitrary performance level between a simple base model and the fine-tuned GNN. The method consists in a systematic reduction of the graph for GNN, implying complexity reduction at the expense of performance.

ITAT'22: Information technologies – Applications and Theory, September 23–27, 2022, Zuberec, Slovakia
paprocha@cisco.com (P. Procházka); mimares@cisco.com (M. Mareš); madedic@cisco.com (M. Dědič)

As a side product, the graph reduction procedure produces a hierarchical clustering of nodes in the graph that is optimised for a given task. We explore these clusters and demonstrate how to use them to better understand the coarsening procedure.

1.1. Related work

Graph neural networks arose independently in several works. The authors derived graph convolutional network (GCN) as a spectral generalisation of image convolution in [1]. Node2vec as a node embedding that is trained to minimise the distance of similar nodes within a graph is proposed in [2]. A "bridge" between Bayesian message passing and GNNs is presented in [3], where the authors derive a GNN from probabilistic assumptions.

Several high-level concepts of GNN complexity reduction appear in literature. To mention some of them: Graph batching proposed in [4] introduces graph sampling (batching) to enable stochastic gradient descent. Next, feature propagation through the network layers is avoided in [5]. Instead, the product of the feature matrix and powers of the adjacency matrix are concatenated and passed to the multi-layer perceptron without any negative impact on performance. Another approach is to identify and remove redundant operations as proposed in [6]. One can also consider decomposing the graph as in [7] in order to enable distributive training of GNNs, since each cluster represents an independent graph component – this idea was adopted in [8]. Connection of batching with distributive processing is considered in [9]. In contrast with these approaches, the proposed complexity reduction is achieved by graph coarsening.

A systematic way of graph coarsening associated with

node2vec is considered in the HARP framework [10]. The idea is to reduce the graph step-by-step by means of edge and star collapsing, training a node2vec model on the coarsened graph and then to step-by-step refine the embedding to the original graph. The motivation behind HARP is to provide an embedding incorporating both local and global graph properties. Using HARP on studying the complexity-performance trade-off is proposed in [11], where the graph coarsening is optimised with respect to graph properties. The prolonging procedure is then driven by the downstream task. In contrast to this work, our coarsening is driven in a different way, nonetheless respecting the downstream task.

Graph clustering / partitioning [7] can be used for distributive processing as described above. Another use-case could be organising the data (typically in an unsupervised way) in order to better understand its structure. On top of traditional clustering, hierarchical clustering enables a tree-structured organisation of the clusters, in contrast to conventional flat clusters. In [12], the authors proposed hierarchical clustering of words in a corpus that is based on a bi-gram language model and the hierarchy is driven by mutual information between clusters. In [13], the authors propose a hierarchical clustering of nodes in a graph that is driven by a pair sampling ratio derived from weights of the adjacency matrix. To the best of our knowledge, there is no available hierarchical clustering of nodes in a graph that is aware of the downstream task as we propose in this work.

1.2. Goals and Contribution

The work reported in this paper presents the following contributions:

1. We motivate and formalise the problem of a complexity performance trade-off and propose a general way of systematically solving and evaluating this problem. The core of this concept is GNN graph size reduction based on edge contraction.
2. We adopt the edge-contraction-based coarsening procedure to a particular task by means of a proper order of the edges for contraction. This order of edges is given by a similarity measure of predictive posterior distributions of adjacent nodes, where a simple reference model on the given task is used to obtain the posterior distributions for all nodes.
3. We experimentally compare various similarity measures of probability distributions driving the coarsening procedure with respect to the target complexity performance trade-off. In particular, we show that the computational graph size can be almost halved with only a small impact on the performance.
4. As a side product, we observed that the graph coarsening based on edge contraction naturally produces a hierarchical clustering. This clustering combines information about predictions from the base model with the graph structure, hence the clustering is driven by the downstream task.

1.3. Structure of the Paper

The paper is organised as follows: A formal description of the complexity-performance trade-off and a general framework for solving this problem is presented in Section 2. A coarsening procedure based on edge contraction as a particular instance of the proposed framework is presented in Section 3. Section 4 then discusses a critical part of the edge contraction procedure, which is the order in which the edges are contracted. Hierarchical clustering view of the considered graph coarsening procedure is the topic of Section 5. Experiments are presented in Section 6 and the work is concluded in Section 7, where we also outline future research directions.

2. Graph Complexity Reduction

We consider a collection of nodes $\mathcal{V}$, edges $\mathcal{E} \subseteq \mathcal{V} \times \mathcal{V}$, real valued node feature vectors $\mathcal{X}$ and node categorical labels $\mathcal{Y}$ constituting an undirected graph $G = (\mathcal{V}, \mathcal{E}, \mathcal{X}, \mathcal{Y})$. We denote $e = (v_A, v_B)$ the edge connecting nodes v_A and v_B. The neighborhood of the node v is denoted $\mathcal{N}(v)$ and the tuple $(\star, v)$ stands for a set of all edges incident on v, that is $(\star, v) = \{(\tilde{v}, v)\}_{\tilde{v} \in \mathcal{N}(v)}$.

The downstream task is transductive node classification, where labels are available for nodes in the training set. A model M is assumed to provide a prediction $\hat{y}$ for each node. The performance of the model $P(\mathcal{Y}, \hat{\mathcal{Y}})$ is then given by a statistic evaluated on the test nodes (e.g. classification accuracy), where the pair $(\mathcal{Y}, \hat{\mathcal{Y}})$ denotes true and predicted node labels. We also consider a complexity of the model M applied on graph G given by $C(G, M)$.

2.1. Problem Formulation

Given a (GNN) model M and a graph G, the goal of this paper is to find a graph $G^\star = (\mathcal{V}^\star, \mathcal{E}^\star, \mathcal{X}^\star, \mathcal{Y}^\star)$ jointly with a refinement mapping $L : \hat{\mathcal{Y}}^\star \to \hat{\mathcal{Y}}$ in order to either

1. maximize performance $P(\mathcal{Y}, L(\hat{\mathcal{Y}}^\star))$ subject to complexity restriction $C(G^\star, M) \leq C_m$ for a given complexity constrain C_m or
2. minimize complexity $C(G^\star, M)$ subject given minimal required performance $P(\mathcal{Y}, L(\hat{\mathcal{Y}}^\star)) \geq P_m$.

2.2. Graph Algorithm Complexity

There are two main aspects of GNN complexity. First, the underlying graph G needs to be stored in memory. Second, the training procedure typically runs through the graph and needs to access it in order to drive the learning process. The authors report both memory and computational complexity of various training techniques of GCN in [4] – Table 1. In principle, both complexities or even any function of them can be covered by the complexity term $C(G, M)$.

2.3. The Performance Complexity Trade-Off

Given a size-decreasing sequence of graphs

$$S = [G_1, \dots, G_i, \dots, G_N], \tag{1}$$

where $G_i = (\mathcal{V}_i, \mathcal{E}_i, \mathcal{X}_i, \mathcal{Y}_i)$ and $|\mathcal{V}_i| > |\mathcal{V}_j| \iff i < j$, we expect a model (GNN) performance to be proportional to the graph size. The main focus of this paper is on the construction of such size-decreasing graph sequence that would enable the fulfillment of our ultimate optimisation problem described in Section 2.1. Evaluating the performance of the model M on each graph in the sequence, one can plot dependency of performance on the complexity (graph size). Denoting P_i performance of model M achieved using G_i with refinement into the original graph G, our goal can be achieved according to illustration[1] in Figure 1:

- Given maximum available complexity C_m, we select such a graph that the complexity given by the number of nodes in the graph is smaller than C_m that is $|\mathcal{V}_i| \leq C_m$ and achieve performance P_3 – blue arrows.
- Given minimal required performance complexity P_m, we select such a graph that $|\mathcal{P}_i| \geq P_m$ with complexity $C_2 = |\mathcal{V}_2|$ – green arrows.

2.4. Finding Working Point Using Performance Complexity Trade-Off in Practice

In practice, it could be very expensive to calculate the performance for all graphs in the sequence S according to Figure 1, hence no resources would be saved as claimed in our main motivation. In order to overcome this limitation, we can consider one or more of the following options:

[1]The illustration captures an intuition that the performance would grow with the graph complexity. As it is shown in Section 6, this is not always strictly true given a particular model. Nevertheless, if we focus on practical usage (Section 2.4) the non-decreasing property is not required in real application.

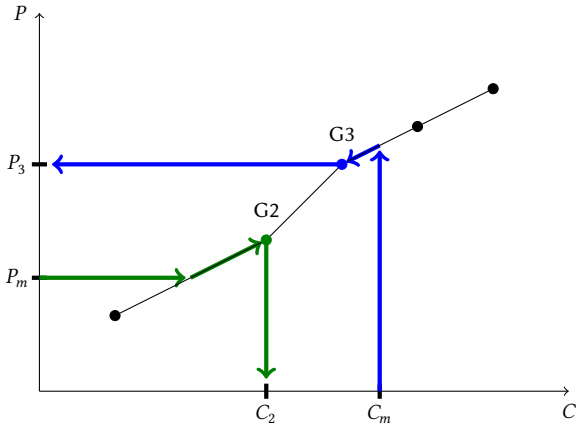


Figure 1: Achieving the target operating point by means of complexity performance trade-off. In case of minimal required performance (green arrows), we find the graph G_2, where the model achieves at least performance P_m. In case of maximal available resources (blue arrows), we select the model achieving performance P_3.

- In order to find the proper working point, we do not need to have the complete curve as in Figure 1. One possible strategy could be to start with a small graph, where the complexity would be low, and to continue until the desired performance is achieved.
- Another option that can be applied in some cases could be to rely on a generalization of the provided information that is a transfer of the proper graph size among tasks, time, models, etc.
- Finally, if we provide some restrictions according to Section 3.4, we can use an upper-bound (or its estimation) as described in Section 4.1 instead of true GNN performance.

3. Graph Size Reduction Strategy Based on Edge Collapsing

This section describes a way of obtaining the graph sequence from Equation (1), which is then used for the target complexity-performance trade-off evaluation. This approach is based on the edge contraction procedure that is described within this Section. The order of edges for contraction driving the size-decreasing graph sequence from Equation (1) is described in Section 4 in greater detail. Finally, a hierarchical clustering induced by the edge contracting procedure can be found in Section 5.

We can write the sequence S from Equation (1) as

$$S = [(\mathcal{V}_1, \mathcal{E}_1, \mathcal{X}_1, \mathcal{Y}_1), \dots, (\mathcal{V}_N, \mathcal{E}_N, \mathcal{X}_N, \mathcal{Y}_N)], \tag{2}$$

where $G_1 = (\mathcal{V}_1, \mathcal{E}_1, \mathcal{X}_1, \mathcal{Y}_1)$ stands for the original graph. The considered graph-size-reduction process is driven step by step. We first describe a single graph coarsening step, that is, how to get from G_i to G_{i+1}, in Equation (1) and then define an overall graph reduction procedure.

3.1. Single Edge Contraction Step

Given an edge to contract, its incident nodes are merged (removed) and a new node is established – see Figure 2 for an illustration. More precisely, given a graph $G_i = (\mathcal{V}_i, \mathcal{E}_i, \mathcal{X}_i, \mathcal{Y}_i)$ and an edge $e = (v_X, v_Y), e \in \mathcal{E}_i, v_X \in \mathcal{V}_i, v_Y \in \mathcal{V}_i$ to remove, the new nodes and edges describing G_{i+1} are given by

$$\mathcal{V}_{i+1} = \mathcal{V}_i \setminus \{v_X, v_Y\} \cup \{v_{XY}\}, \tag{3}$$

where v_{XY} is a new (merged) node and

$$\mathcal{E}_{i+1} = \mathcal{E}_i \setminus ([v_X, v_Y], \star) \cup \{(v_{XY}, v)\}_{v \in \mathcal{N}_i(v_X, v_Y)}, \tag{4}$$

where

$$([v_X, v_Y], \star) = (v_X, \star) \cup (v_Y, \star)$$

$$\mathcal{N}_i(v_X, v_Y) = \mathcal{N}_i(v_X) \cup \mathcal{N}_i(v_Y) \setminus \{v_X, v_Y\},$$

where the neighborhood is taken according to G_i.

In the context of our setup, that is, in the situation where we want to apply a GNN on the new graph G_{i+1} and evaluate its performance on the original one (G_1), the following issues arise and must be resolved:

1. Based on the nodes v_X and v_Y, we need to aggregate training labels (if available) to the new node v_{XY} and decide if this label should be used for training on the coarsened graph, that is to define $\mathcal{Y}_{i+1}$ based on $\mathcal{Y}_i$. This problem is discussed in Section 3.3.
2. Apart from labels, node features must be aggregated as well (transforming $\mathcal{X}_i$ into $\mathcal{X}_{i+1}$) – see Section 3.2 for details[2].
3. Finally, once we have a prediction for the node v_{XY}, we need to establish predictions for all the nodes from $\mathcal{V}_1$ that were merged into v_{XY}. We introduced the refinement mapping L for this purpose in Section 2.1. More details and a simple refinement mapping is presented Section 3.4.

Within this paper, we only show a very simple possible solution to address each of the mentioned issue. A deeper investigation of each of them is not within the scope of this paper and is reserved for future work.

[2]Handling the edge features also needs to be considered if they are available. Since this is not the case in our assumptions, we leave this problem for future work.

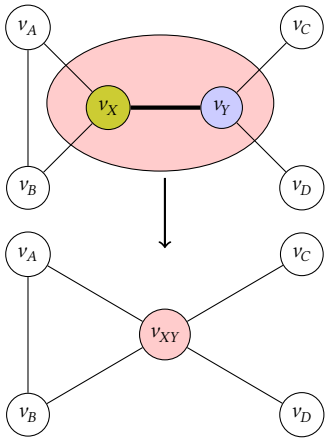


Figure 2: Coarsening given by contracting the edge (v_X, v_Y), where the nodes v_X and v_Y are replaced by a new node v_{XY} and the edges connecting v_X and v_Y are rewired to v_{XY}. As discussed in Section 5, this atomic merge can also be seen as one step in a hierarchical clustering, where clusters v_X and v_Y become child of the cluster v_{XY}.

3.2. Feature Aggregation

The GNN model assumes all feature vectors to have the same dimension for each node, hence some aggregation is required when merging two nodes. In particular, we have two nodes v_X, v_Y with feature vectors x_X and x_Y of length F that must be aggregated by some aggregation a into one vector $x_{XY} = a(x_X, x_Y)$ of length F corresponding to the newly merged node v_{XY}.

Although a number of solutions can be considered, we assume a simple weighted mean aggregation within this work, that is

$$x_{XY} = a(x_X, x_Y) = \frac{w_X x_X + w_Y x_Y}{w_X + w_Y}, \tag{5}$$

where the weight w_A is given by the number of nodes from the original graph G_1 that were merged to v_A in the preceding steps.

3.3. Label Aggregation

Similarly as in the case of features, we also need to aggregate labels associated with the nodes v_X, v_Y that are supposed to be merged. We (ad-hoc) consider a similar approach as in Equation (5), where we convert the labels to a prior distribution p_y on labels and we obtain

$$p_{y_{XY}} = \frac{\omega_X p_{y_X} + \omega_Y p_{y_Y}}{\omega_X + \omega_Y}, \tag{6}$$

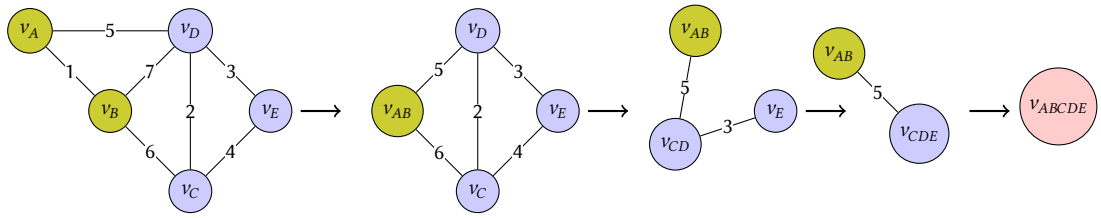


Figure 3: Graph sequence determined by a given sequence of edges to contract, where the numbers on edges stand for the edge order for contraction. We use colors to denote labels and to indicate the problem of label aggregation – their aggregation is clear until the final node in the last step.

where ω_A denotes[3] the number of training labels merged into v_A.

In multi-class classification, the prior distribution p_y is in the form of a probability vector with a single non-zero (unit) element placed on the position of true label, if available. Since the weight ω_A is non-zero only for at least one training sample, only probability vectors related to the training samples are reflected in Equation 6.

3.4. Label Refinement

Label refinement is related to the node embedding prolonging in [10], where we need to transform a prediction about a cluster into predictions about its constituent nodes. Within this work, we assume a trivial copy-paste method, where all merged nodes share the same prediction.

More precisely, running a (GNN) model on G_i, we receive a prediction for each node from $\mathcal{V}_i$ that needs to be prolonged to the nodes from $\mathcal{V}_1$ for the target accuracy evaluation. A relation of nodes in $\mathcal{V}_i$ and $\mathcal{V}_1$ is hierarchical so that each node $v \in \mathcal{V}_i$ can be seen as a hierarchical composition of nodes $\bar{\mathcal{V}} \subseteq \mathcal{V}_1$ such that the node v arose by merging nodes from $\bar{\mathcal{V}}$ according to Equation (3) in preceding steps. The considered simple prolonging mapping L then takes the prediction related to v and applies it for all nodes from $\bar{\mathcal{V}}$.

3.5. Graph Coarsening Sequence

We have already described a single edge contraction step from G_i to G_{i+1}, which assumed a given edge to remove. Given an ordered list of edges to remove, we can step-by-step obtain the sequence from Equation (1) in order to evaluate performance-complexity trade-off for a given graph and model. An example of the graph decomposition is illustrated in Figure 3, while the final missing piece of this procedure, that is ordering the edges to be contracted, is described in Section 4.

[3]Recall that ω_A denotes the number of training labels merged into v_A, while w_A from Equation (5) stands for the number of all nodes merged into v_A.

4. Ordered Sequence of Edges for Removal

This Section focuses on the last missing piece of the complexity-performance trade-off evaluation, which is obtaining a sequence of edges driving the edge contraction procedure. First, a simple performance upper-bound of the graph that was reduced according to Section 3 is presented. The suggested edge sorting algorithm aims to provide an ordering of edges such that the upper bound would be maximised at each step when the edge contraction algorithm is applied in this order.

4.1. Label Refinement Induced Performance Upper Bound

Considering all nodes in the testing set, in Figure 3 we can see that the nodes of the same color (label) are merged together in the first three steps. In that case, 100% accuracy can theoretically be achieved if the merged nodes are predicted correctly. However, the final edge contraction merges together nodes with different colors (labels) and the performance from this step forward cannot be greater than 60%, which is achieved if the node v_{ABCDE} is predicted to be a blue one. If it is predicted as a green one, only 40% accuracy is achieved. In this case, 60% presents a performance upper-bound since no model can achieve better performance in the final single node graph. Note that the upper bound is 100% for the first three steps.

More formally, according to Section 3.4, we assume all the merged nodes[4] $\bar{\mathcal{V}} \subseteq \mathcal{V}_1$ to share the prediction of the corresponding node $v \in \mathcal{V}_i$. Obviously, if true test labels differ within $\bar{\mathcal{V}}$, some prediction errors are unavoidable. Given the node $v \in \mathcal{V}_i$, we calculate a histogram of the test labels from $\bar{\mathcal{V}}$. Denoting y_0 the most common testing label within $\bar{\mathcal{V}}$, a minimal number of errors on testing nodes from $\bar{\mathcal{V}}$ is given by the number of test labels that differs to y_0. Denoting this number of errors as $U(v)$ for each node $v \in \mathcal{V}_i$, the performance upper-bound can be

[4]Recall $\bar{\mathcal{V}}$ as a set of nodes from the original graph that are merged into node v for a given $v \in \mathcal{V}_i$.

written as:

$$\frac{1}{|T|}\sum_{v\in\mathcal{V}_i} U(v) \tag{7}$$

where $|T|$ denotes the number of nodes from V_1 in test set.

We also note that considering this upper-bound as performance in the performance-complexity trade-off (Section 2.1), we achieve a non-increasing sequence of performances for the sequence S in Equation (1).

4.2. Edge Ordering Maximizing the Performance Upper Bound

We can achieve the upper-bound in Equation (7) equal to 1 while merging the nodes with the same label. Once we start to merge nodes with varying labels, the upper-bound starts to decrease. This would be a trivial operation if all labels would be available. Nevertheless, if only training labels are available, we need to somehow estimate labels for nodes for which the ground truth is not available.

Instead of comparing true labels, we propose to compare a predictive posterior distribution related to the downstream task, where the prediction is provided by an auxiliary (base) model. We propose then to calculate a similarity measure upon pair of posterior distributions for all edges (corresponding node pairs) and to sort the edges according to the similarity to establish the order of edges to be contracted.

4.3. Base Model

The base model serves to evaluate a posterior probability for each node in graph determining the pairwise node similarity for sorting the edges. Since the model is assumed to process the entire graph, it is expected to be a simple one.

4.4. Similarity Measures

Within this paper, we experimentally evaluate several similarity measures of the probability density functions. In particular, we consider cross-entropy and the KL divergence as similarity measures of posterior probability distributions. The asymmetry of these measures does not have any effect, since we assume an undirected graph, that is the similarity is calculated for both directions and the smaller value is used.

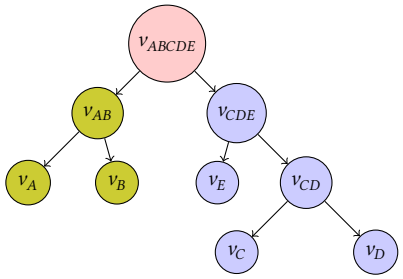


Figure 4: Hierarchical clustering tree corresponding to edge contraction in Figure 3

.

5. Hierarchical Clustering of Nodes Induced by Edge Contraction

As demonstrated in Figure 2, the edge-contraction-based coarsening leads to a hierarchical clustering represented by a tree, where the nodes in the original graph $v \in \mathcal{V}_1$ form leaves of the tree and each edge contraction defines a relation between a parent and its children according to Equation (3). As an example, a tree for the procedure from Figure 3 is shown in Figure 4.

Although this clustering is only a side-effect of the edge contraction procedure, it brings[5] a useful insight into the quality of the edge ordering. The label refinement 3.4 basically operates within each hierarchical cluster, since the (GNN) model is assumed to make a prediction about this cluster that needs to be refined into a prediction for the nodes in the original graph G.

5.1. Multi-Step Edge Contraction

The edge contraction described in Section 3.1 can be easily extended by considering multiple edges to be contracted simultaneously. In that case, a sub-graph defined by a collection of edges to contract is considered. Each component of this graph then forms a new merged node.

The motivation for this multiple-step consideration is to decrease the depth of the hierarchical tree on one hand or reduce the number of graphs in Sequence (1) on the other. The depth of the tree referring to the hierarchical cluster can be as high as the number of nodes in the graph. The multi-edge contraction procedure and its relationship with the edge contraction step described in Section 3.1 is illustrated in Figure 5 in a particular example.

[5] We discuss other potential applications in Section 7.

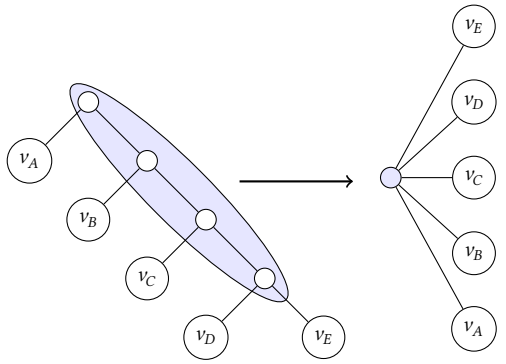


Figure 5: Illustration of a multi-edge coarsening provided in one step and its relation to the edge one-by-one edge coarsening, where we use hierarchical clustering as described in Section 5. A graph of five nodes given by edges $\mathcal{E} = [(v_D, v_E), (v_C, v_D), (v_B, v_C), (v_A, v_B)]$ is coarsened in four steps (left). If these steps are provided simultaneously, a flat tree is obtained (right).

6. Experiments

This section provides experimental validation of the proposed complexity-performance trade-off on traditional real-world graph datasets.

6.1. Experiments Objectives

Since our claims are not as straightforward as stating that one method overcomes another one, we first claim, what we try to achieve within the experiments.

First of all, we would like to find a sweet spot of the complexity-performance trade-off for a given task enabling us to outperform the baseline with reduced complexity. We found this point in all cases shown in Figure 6a.

The next objective is to compare the performance upper-bound from Section 4.1 with true performance – Figure 6a. Since the performance upper-bound can be calculated simply compared to the true performance, it can be considered as an approximation of the true performance in the complexity-performance trade-off. This would save significant computational resources needed for true performance evaluation for all graphs in the sequence S, but making conclusions on upper-bound should be validated.

We further investigate impact of various steps driving the edge order for removal described in Section 4 on the complexity performance trade-off and we also attempt to validate using a posterior distribution for edge sorting (4.2) instead of the prior (labels) used in the upper-bound (Figure 6b). Finally, we demonstrate the node clustering as a vital visualization of a particular graph coarsening.

6.2. Experiment Setup

We consider a logistic regression model as the base model in all experiments. Based on the posterior distribution obtained from the logistic regression, we calculate the ordering of edges for removal using the KL-divergence and cross-entropy similarity measures on predictive node distribution. We also consider random ordering as a reference.

We first calculate the graph sequence according to Section 3.5. This sequence inherently provides the performance upper-bound (Section 4.1). A GCN [1] is then trained on each graph in the sequence, where the labels and features are aggregated according to Sections 3.2 and 3.3 and performance (accuracy) is obtained on the test set refined to the original graph (Section 3.4). We consider the cross entropy loss function within training, where the cross entropy is calculated against the prior distribution calculated in Equation (6). The node is assumed to be a training one if it contains at least one aggregated label from G_1.

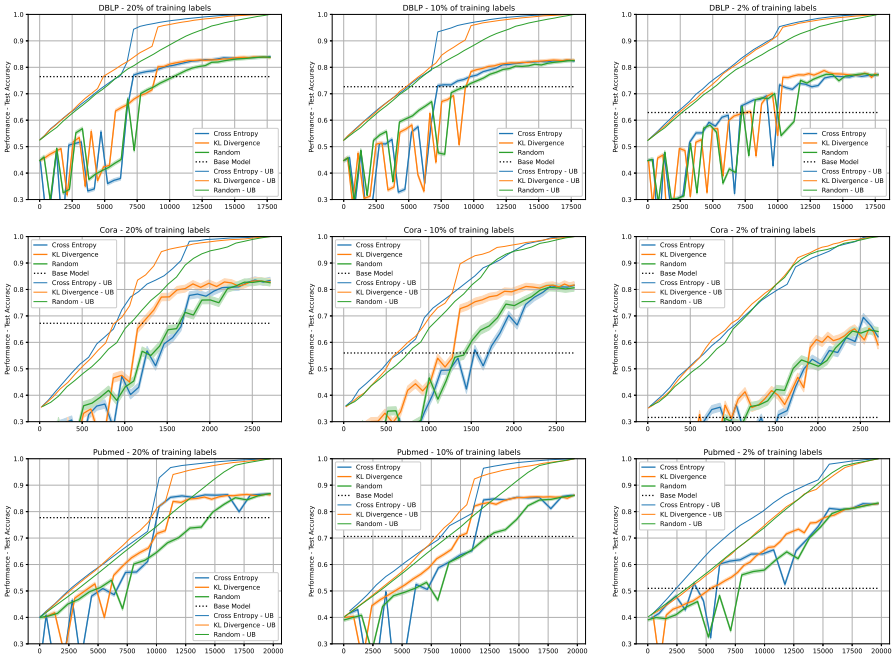
We consider complexity given by the number of nodes in the graph within our experiments. This simplified complexity measure takes only graph into account, it is not suitable for comparison of multiple models, since their complexity is not taken into account.

We evaluated the performance-complexity trade-off for Cora, PubMed [14] and DBLP [15] datasets, where we sub-sampled training labels – see results in Figure 6a. In order to validate the use of the posterior instead of prior distribution, we used a distribution given by the base model if the label was not available and if it was, a distribution given by $rp_y + (1-r)p_{\hat{y}}$, where p_y denotes the prior distribution of data determined by the training labels, $p_{\hat{y}}$ the posterior distribution (base model output) and r is a coefficient enabling mixing between the two distributions. The results are shown in Figure 6b.

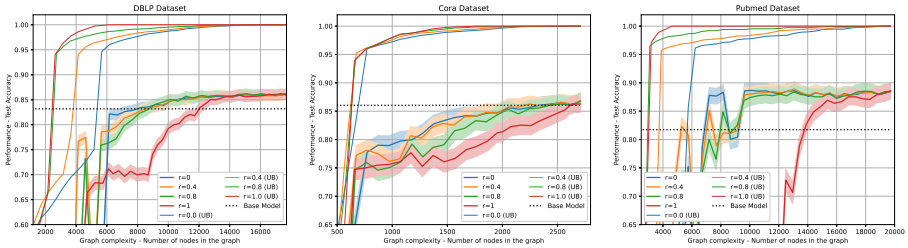
The complexity performance trade-off curves in Figures 6a and 6b are equipped by 90% symmetric confidence intervals, where the interval is given by the corresponding quantiles of Beta distribution constructed as the conjugated prior to binomial distribution given by x errors out of N_t testing examples (note that x/N_t stands for accuracy).

6.3. Performance Upper-Bound versus GNN Performance

The comparison of the upper-bound and true performance can be found in Figures 6a and 6b. The upper-bound does not suffer from the fluctuations as the true performance. In case when the base model is strong enough, the working point selection according to the upper-bound could bring reasonable results.



(a) Performance comparison of various distributions used for sorting the edges for contracting. The dependencies are plotted for different number of training labels.



(b) Performance comparison of various distribution used for sorting the edges for contracting.

Figure 6: Evaluation of performance-complexity trade-off.

6.4. Impact of Edges Ordering on Performance Complexity Trade-Off

Impact of the chosen distribution similarity measure on performance is shown in Figure 6a. Surprisingly, even the random baseline gives good results in some cases. While no metric is the best choice in case of Pubmed and DBLP datasets, KL divergence seems to be a superior choice in case of Cora dataset. In case of 2% training labels, the base model is so weak that qualitative difference between the edge sorting strategies disappears for all datasets. Interestingly, the provided observations made on true performance can be also deduced from the upper-bounds.

As can be seen in Figure 6b, adding prior information makes the results consistently worse. This is probably caused by the fact that neighbouring nodes with the same label in the training set are merged first and the lack in training data then causes the observed performance drop. In contrast with evaluating the similarity measures, the upper-bound provides an inverse observation.

6.5. Hierarchical Clustering

The hierarchical clustering described in Section 5 presents a neat way of manually validating the graph coarsening procedure. We evaluated the base model on the Karate Club dataset and used the proposed coarsening procedure according to Section 5.1. Two edges were contracted in each step. The resulting tree is shown in Figure 7.

Observing the tree, we can recognise splits (or coars-

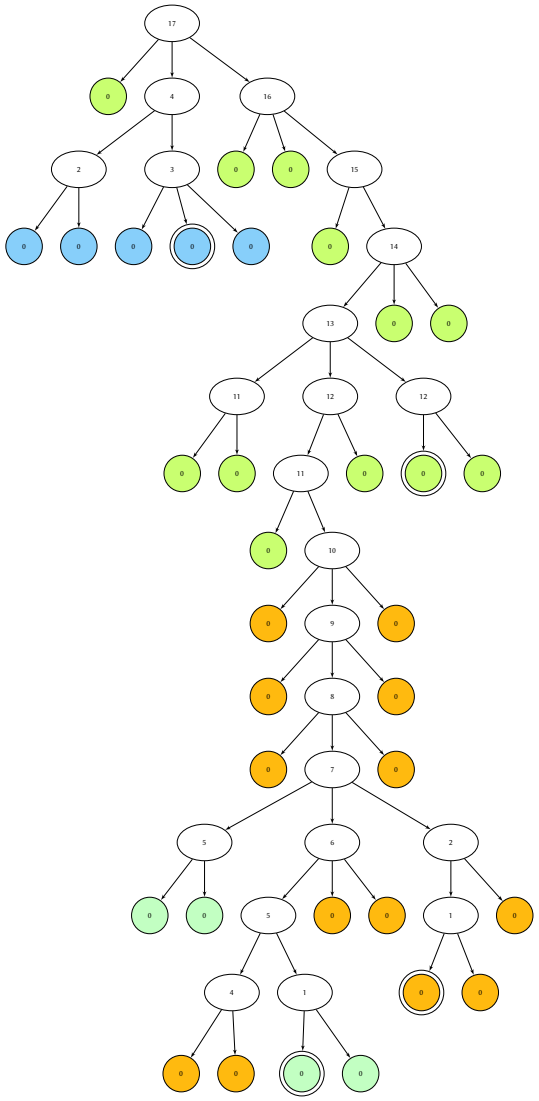


Figure 7: The hierarchical tree corresponding to the edge contracting procedure driven by the base model on the Karate Club dataset. The true labels of the leaf nodes are color-coded and training labels are denoted by a double circle. The number in the circle denotes the step in the edge contraction order – in each step, two edges are simultaneously contracted according to Section 5.1.

ening steps) resulting from merging nodes with the same label, which is the desired behavior, because the graph size is reduced without any performance loss in such a case. However, merging differently labelled nodes unavoidably leads to a performance loss, because our label refinement procedure guarantees the same prediction for each node within the cluster (see Section 4.1).

In case of Karate Club dataset tree (Figure 7), the first four steps of the procedure merge only nodes with a matching label. However, from the fifth step onward, at least two errors occur and affect the total accuracy (one error in case of test accuracy).

7. Conclusion and Discussion

We formalized the performance-complexity trade-off and described a framework for smoothly adjusting either desired model complexity for a given performance requirements or performance for available computational resources. The framework consists of a methodical step-by-step graph coarsening by way of edge contraction, where we addressed practical aspects of feature and label aggregation within node merging and also prediction refinement to the original nodes from the merged node. We have also shown that this edge coarsening naturally produces a hierarchical clustering of the nodes in the graph.

The ordering of edges for the contracting procedure is then studied in greater detail, where we suggest to employ a base model to produce a predictive posterior distribution for each node and then sort the edges according to the distance of probability distributions of adjacent nodes.

We applied the proposed framework to widely used publicly available graph datasets. In particular, we explored the coarsening procedure on the Karate Club dataset by means of a hierarchical tree. Finally we compared KL-divergence and cross-entropy similarity measures for driving the edge ordering.

Although our primary focus was the complexity performance trade-off, we believe that we explored several research directions for future works.

The considered label refinement is extremely simple, however, we can alternatively consider an arbitrary model within each cluster. This will incur some additional cost, but a distributive processing may be utilized since the clusters do not overlap. Moreover, the information from this ”local” model can be combined with the global one, which could lead to superior performance since both local and global graph contexts would be considered. In addition, we would still avoid running the GNN on the entire graph.

If we focus on the hierarchical clustering, we can relax our assumption of applying a simple model on the entire graph to produce a predictive posterior distribution. Instead, we could consider as strong model as possible to achieve the best possible clustering for a given use case.

Within this work, we used the hierarchical clustering as a complementary/inspection tool for the graph coarsening. Nevertheless, it would be interesting to compare its suitability for other purposes – e.g. distributive processing as an alternative to [7]. In addition, the clustering

is built on a supervised setup. An unsupervised approach based on e.g. the cosine distance of the node2vec embedding could be an interesting alternative to [13].

References

[1] T. N. Kipf, M. Welling, Semi-supervised classification with graph convolutional networks, arXiv preprint arXiv:1609.02907 (2016).

[2] A. Grover, J. Leskovec, node2vec: Scalable feature learning for networks, in: Proceedings of the 22nd ACM SIGKDD international conference on Knowledge discovery and data mining, 2016, pp. 855–864.

[3] H. Dai, B. Dai, L. Song, Discriminative embeddings of latent variable models for structured data, in: International conference on machine learning, PMLR, 2016, pp. 2702–2711.

[4] W.-L. Chiang, X. Liu, S. Si, Y. Li, S. Bengio, C.-J. 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, 2019, pp. 257–266.

[5] E. Rossi, F. Frasca, B. Chamberlain, D. Eynard, M. Bronstein, F. Monti, Sign: Scalable inception graph neural networks, arXiv preprint arXiv:2004.11198 7 (2020) 15.

[6] Z. Jia, S. Lin, R. Ying, J. You, J. Leskovec, A. Aiken, Redundancy-free computation for graph neural networks, in: Proceedings of the 26th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining, 2020, pp. 997–1005.

[7] G. Karypis, V. Kumar, Metis: A software package for partitioning unstructured graphs, partitioning meshes, and computing fill-reducing orderings of sparse matrices (1997).

[8] D. Zheng, C. Ma, M. Wang, J. Zhou, Q. Su, X. Song, Q. Gan, Z. Zhang, G. Karypis, Distdgl: distributed graph neural network training for billion-scale graphs, in: 2020 IEEE/ACM 10th Workshop on Irregular Applications: Architectures and Algorithms (IA3), IEEE, 2020, pp. 36–44.

[9] C. Zheng, H. Chen, Y. Cheng, Z. Song, Y. Wu, C. Li, J. Cheng, H. Yang, S. Zhang, Bytegnn: efficient graph neural network training at large scale, Proceedings of the VLDB Endowment 15 (2022) 1228–1242.

[10] H. Chen, B. Perozzi, Y. Hu, S. Skiena, HARP: Hierarchical Representation Learning for Networks, Proceedings of the AAAI Conference on Artificial Intelligence 32 (2018). Number: 1.

[11] M. Dedič, L. Bajer, J. Repický, P. Procházka, M. Holeňa, Adaptive graph coarsening in the context of local graph quality, Accepted to CEUR Workshop Proceedings (2022).

[12] P. F. Brown, V. J. Della Pietra, P. V. Desouza, J. C. Lai, R. L. Mercer, Class-based n-gram models of natural language, Computational linguistics 18 (1992) 467–480.

[13] T. Bonald, B. Charpentier, A. Galland, A. Hollocou, Hierarchical graph clustering using node pair sampling, arXiv preprint arXiv:1806.01664 (2018).

[14] P. Sen, G. Namata, M. Bilgic, L. Getoor, B. Galligher, T. Eliassi-Rad, Collective classification in network data, AI Magazine 29 (2008) 93.

[15] J. Tang, J. Zhang, L. Yao, J. Li, L. Zhang, Z. Su, Arnetminer: Extraction and mining of academic social networks, in: Proceedings of the 14th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, KDD '08, Association for Computing Machinery, New York, NY, USA, 2008, p. 990–998.