An End-to-end Bi-objective Approach to Deep Graph Partitioning

Pengcheng Wei[a,1], Yuan Fang[b,*], Zhihao Wen[b], Zheng Xiao[c] and Binbin Chen[a]

[a]*Information Systems Technology and Design Pillar, Singapore University of Technology and Design, 485998, Singapore*
[b]*School of Computing and Information Systems, Singapore Management University, 178902, Singapore*
[c]*College of Computer Science and Electronic Engineering, Hunan University, Hunan, 410082, China*

ARTICLE INFO

Keywords:
Graph Partition
End-to-end
Bi-objective
Graph Neural Networks

ABSTRACT

Graphs are ubiquitous in real-world applications, such as computation graphs and social networks. Partitioning large graphs into smaller, balanced partitions is often essential, with the bi-objective graph partitioning problem aiming to minimize both the"cut" across partitions and the imbalance in partition sizes. However, existing heuristic methods face scalability challenges or overlook partition balance, leading to suboptimal results. Recent deep learning approaches, while promising, typically focus only on node-level features and lack a truly end-to-end framework, resulting in limited performance. In this paper, we introduce a novel method based on graph neural networks (GNNs) that leverages multilevel graph features and addresses the problem end-to-end through a bi-objective formulation. Our approach explores node-, local-, and global-level features, and introduces a well-bounded bi-objective function that minimizes the cut while ensuring *partition-wise* balance across all partitions. Additionally, we propose a GNN-based deep model incorporating a Hardmax operator, allowing the model to optimize partitions in a fully end-to-end manner. Experimental results on 12 datasets across various applications and scales demonstrate that our method significantly improves both partitioning quality and scalability compared to existing bi-objective and deep graph partitioning baselines.

1. Introduction

Graphs are ubiquitous in the real world and can be used to represent objects and their relationships, such as social networks Zhou, Liu, Ding, Jin and Li (2023); Ma, Zhang, Ma and Ma (2020), modules in biology Ma, Sun and Qin (2017); Ma, Zhao and Wu (2022); Ma, Tang, Wang, Guo and Gao (2016), and computation graphs Tanaka, Taura, Hanawa and Torisawa (2021). In computational intensive applications, such as distributed computing Tanaka et al. (2021), an input graph is usually partitioned into K partitions to be processed by K devices. The bi-objective K-*way* graph partitioning problem aims to divide the nodes in a graph into K partitions, such that the partitions are balanced with similar sizes and the "cut" between different partitions are minimized. The size of a partition is usually defined as the sum of weights of nodes in that partition, whereas the cut is measured as the sum of weights of the edges between different partitions.

Solving the graph partitioning problem will boost developments of various applications, such as distributed computing Shao, Li, Gu, Yin, Li, Miao, Zhang, Cui and Chen (2024); Ni, Li, Yu, Zhou and Wu (2020) and VLSI design Bustany, Gasparyan, Kahng, Koutis, Pramanik and Wang (2023). When performing distributed graph analysis tasks, minimizing the cut saves as much connectivity information as possible in the graph, and keeping the partitions in similar sizes helps balance the workloads among processors to approximate the linear speedup. Moreover, graph partitioning is a crucial step in VLSI design. Minimizing the cut between subcircuits can decrease circuit delay, and achieving balanced partitions helps in avoiding bottlenecks and ensuring uniform resource utilization. Hence, how to derive the partitions with both minimized cut and balanced sizes has become an urgent research topic.

Existing literature on the bi-objective optimization problem are mainly based on genetic algorithms Baños, Gil, Montoya and Ortega (2004); Farshbaf and Feizi-Derakhshi (2009). However, they incur high time costs to find good partitions in large graphs. Other heuristic methods show better scalability and partitioning quality by modeling the partitioning problem as a constrained single-objective optimization problem. That is, they first fix the size of the biggest partition to satisfy a given constraint, then aim to find the minimum cut. However, such a formulation suffers from two

*Corresponding author

pengcheng_wei@mymail.sutd.edu.sg (P. Wei); yfang@smu.edu.sg (Y. Fang); zhwen.2019@smu.edu.sg (Z. Wen); zxiao@hnu.edu.cn (Z. Xiao); binbin_chen@sutd.edu.sg (B. Chen)
ORCID(s):

[1]Part of the work done at Singapore Management University and Hunan University.

major limitations. First, these works only consider nodes and edges in a local view, ignoring a global view of the graph, resulting in inferior partitions. Second, they only concern with the size of the biggest partition, ignoring partition-wise balance across all partitions.

On another line, the success of graph neural networks (GNNs) has inspired several GNN-based graph partitioning models which divide nodes by embedding their features into K dimensions Gatti, Hu, Smidt, Ng and Ghysels (2022); Gatti and Hu (2022); Nazi, Hang, Goldie, Ravi and Mirhoseini (2019) and predicting the assignment probability distribution on each dimension representing a partition. These methods usually feed node weights into GNNs as a node-level feature, lacking a broader local or even global view of the graph, which leads to inferior performance on partitioning, especially for unweighted graphs (nodes and edges without features). Furthermore, none of these works is truly end-to-end. Specifically, their training process optimizes a *soft* assignment matrix containing the probability distribution, which cannot reflect the true partitioning quality resulting inferior performance. Moreover, actual partitions are discrete requiring a *hard* assignment matrix. The discrepancy between soft and hard assignment matrices can degrade the performance which will be shown in our experiment.

To address the issues of existing studies, we propose a novel deep graph partitioning method, which exploits multilevel graph features and contains a GNN-based model to optimize a well bounded bi-objective function in an end-to-end manner. Specifically, we tackle the following challenges in our model design.

Challenge 1: Multilevel graph features Exploiting more graph features is necessary and beneficial for generating high quality partitions. The chosen graph features should aid in characterizing nodes, be aligned with graph partitioning objectives and be easily accessible. Besides the node weight feature, we further consider adjacency encoding, i.e., adjacent matrix, as the local feature, which includes the local neighborhood of the node and is crucial for minimizing the cut. This feature is also handy and memory efficient for large graphs, since adjacency encoding can be stored as a sparse matrix Fey and Lenssen (2019). Finally, as a global-level feature, position encoding is calculated based on the shortest path between nodes, which is important to generating partitions that are consistent with the global topology.

Challenge 2: Bounded bi-objective function While the cut objective has been standard, such as the normalized cut Shi (2003) adopted in this paper, the balance objective is not well studied. Previous works either neglect the consideration of partition-wise balance or usually present an unbounded balance objective resulting in a large optimizing space Baños et al. (2004); Farshbaf and Feizi-Derakhshi (2009); Gatti et al. (2022); Gatti and Hu (2022); Nazi et al. (2019). We propose a bounded partition-wise balance objective function that encourages each partition to be perfectly balanced. Notably, we establish that the balance objective function can be bounded just as the normalized cut, implying that the two objectives can be combined and optimized smoothly in a bi-objective formulation.

Challenge 3: End-to-end optimization Existing deep learning-based graph partitioning works cannot optimize the hard assignment matrix (i.e, explicit partitions) directly. Although the hard assignment matrix can be obtained by applying the $\arg\max$ operation on the soft assignment matrix, $\arg\max$ is not differentiable and is thus incompatible with gradient-based optimization. To address this issue, we propose a differentiable Hardmax operator, which builds a tight connection between hard and soft assignment matrices and enables our model to optimize and output the hard assignment matrix directly.

In summary, our contributions in this work can be summarized as follows:

- To the best of our knowledge, for the bi-objective graph partitioning problem, we propose the first end-to-end deep partitioning method, demonstrating the promising potential of deep end-to-end models over both existing traditional heuristic and learning-based methods.
- We explore efficient multilevel (node-, local-, and global-) features aligning with graph partitioning objectives. A bounded bi-objective function is further proposed which considers both the cut and partition-wise balance to guide the model to generate high quality partitions. Moreover, we design a GNN model with the Hardmax operator to optimize hard assignments in an end-to-end manner.
- We conduct extensive experiments on 12 real-world datasets in various domains and scales, and the results demonstrate the notable improvement in partitioning quality and scalability than existing bi-objective baselines and deep graph partitioning baselines.

2. Related Work

In this section, we first introduce heuristic methods, followed by deep learning based graph partitioning works.

2.1. Node Partitioning and Edge Partitioning

In the graph partitioning problem, there are two main categories: node partitioning and edge partitioning Çatalyürek, Devine, Faraj, Gottesbüren, Heuer, Meyerhenke, Sanders, Schlag, Schulz, Seemaier et al. (2023). The node partitioning method, which we focus on in this paper, attempts to evenly assign nodes to partitions by cutting the edges, thereby minimizing the number of cut edges. In contrast, the edge partitioning method aims to evenly distribute edges across partitions by cutting the nodes, thus minimizing the number of cut nodes. Node partitioning is commonly adopted for load balancing in parallel computing Shao et al. (2024), whereas edge partitioning is frequently utilized in distributed graph processing, particularly for power-law graphs Hanai, Suzumura, Tan, Liu, Theodoropoulos and Cai (2019).

2.2. Heuristic *K-way* Graph Partitioning Methods

Existing bi-objective graph partitioning methods are based on evolutionary algorithms. The algorithms can be adopted to directly minimize the cut and the size of the biggest partition Baños et al. (2004); Farshbaf and Feizi-Derakhshi (2009). Besides the two objectives, other researchers consider minimizing the spread of a partition in a direction in order to make the partition of compact shape Datta, Figueira, Fonseca and Tavares-Pereira (2008); Datta and Figueira (2011). However, due to a large number of iterations required by evolutionary algorithms to converge, these works are time-consuming for partitioning large graphs.

Most existing graph partitioning works formulate the graph partitioning problem as a constrained single-objective optimization Çatalyürek et al. (2023); Sanders and Schulz (2013), and solve it follow a multilevel paradigm **?**Gottesbren, Heuer, Sanders, Schulz and Seemaier (2021), such as Metis Karypis and Kumar (1998a) and Scotch Chevalier and Pellegrini (2008). Specifically, they aim to find the minimum cut while ensuring the size of the biggest partition satisfy a given constraint. To achieve this, they first reduce the sizes of a graph to speedup (coarsening level), partition the reduced graph (partitioning level), and subsequently refine partitions to recover the graph (refining level). However, these works only consider nodes and edges in a local view, ignoring a global view of the graph, resulting in inferior partitions. Furthermore, they only concern the imbalance degree defined by the size of the biggest partition, ignoring a *partition-wise* balance across all partitions. Moreover, if they set the constraint as perfect balance (i.e., the biggest size equals to total number of nodes divided by K), the cut of partitions will usually be very high. That is, these methods are single-objective driven, which are difficult to generate as minimized and balanced partitions as possible. Finally, these methods require the input graphs to be undirected, connected, and free of self-loops. Checking these requirements for the graphs will incur additional time costs before partitioning.

2.3. GNN-based Graph Partitioning Methods

Graph neural networks (GNNs) Dwivedi, Joshi, Luu, Laurent, Bengio and Bresson (2023) can learn node, edge and structure representations in a graph, which has inspired GNN-based single- and multi- level graph partitioning models.

Regarding the single level partitioning methods, researchers first propose their bi-objective partitioning functions, and then design neural networks optimizing soft assignments directly to minimize the functions Shao, Wang, Zhao, Ma and Liu (2023); Nazi et al. (2019); Liu, Wang, Cao, Liu and Shen (2022); Cai, Guo and Huang (2024), such as GAP Nazi et al. (2019) and GON Liu et al. (2022). For example, GAP Nazi et al. (2019) is a deep approximate partitioning model, which is trained to learn soft assignments of partitions and tested to partition unseen graphs showing stronger generalizability than multilevel graph partition methods. In GON, the authors propose a variant of the normalized cut Shi and Malik (2000) as the edge-cut objective and a regularization loss as the balance objective where minimizing the former boosts strongly connected nodes to be partitioned together and minimizing the later encourages the partitions to hold similar size. Some graph pooling works Bianchi, Grattarola and Alippi (2020); Duval and Malliaros (2022); Tsitsulin, Palowitch, Perozzi and Müller (2023) can be categorized into the single level partitioning, because they share the similar idea of GON to learn more abstract and coarser representations of the graphs. For example, the loss functions of HoscPool Duval and Malliaros (2022) and DMoN Tsitsulin et al. (2023), contain a cut loss and a orthogonality loss. However, the above balance losses are only designed for the graphs with unweighted nodes. Moreover, due to without sufficient input features, these works usually output empty partitions.

Following the multilevel graph partitioning paradigm, deep multilevel methods are proposed Gatti et al. (2022). The graph firstly is coarsened, then a graph embedding module based on GNNs is designed to learn spectral embedding of the coarsened graph. Finally a partition module is used to generate the soft assignment matrix. The embedding module and the partition module are trained by different objective functions. However, this method can only perform bi-partition, not the general *K-way* ($K \geq 2$) graph partitioning. Besides GNNs, the authors integrated Deep Reinforcement Learning algorithms into the multilevel paradigm Gatti and Hu (2022). For example, after coarsening graphs as in Gatti et al. (2022), GraphSAGE Hamilton, Ying and Leskovec (2017) is adopted to learn node embeddings and A2C Williams (1992) algorithm is used for initial partitioning and refinement. From their experiments, they show the method generalizes well to unseen graphs, but do not show better performance than heuristic multilevel graph partition baselines in terms of cut or balance.

Notably, all the above GNN-based methods only adopt the node weight as features, without exploring multilevel graph features. Moreover, their balance objectives are only for unweighted graphs or usually are unbounded resulting in a large optimization space. Finally, they are not able to optimize explicit partitions in an end-to-end manner leading to inferior partitioning performance.

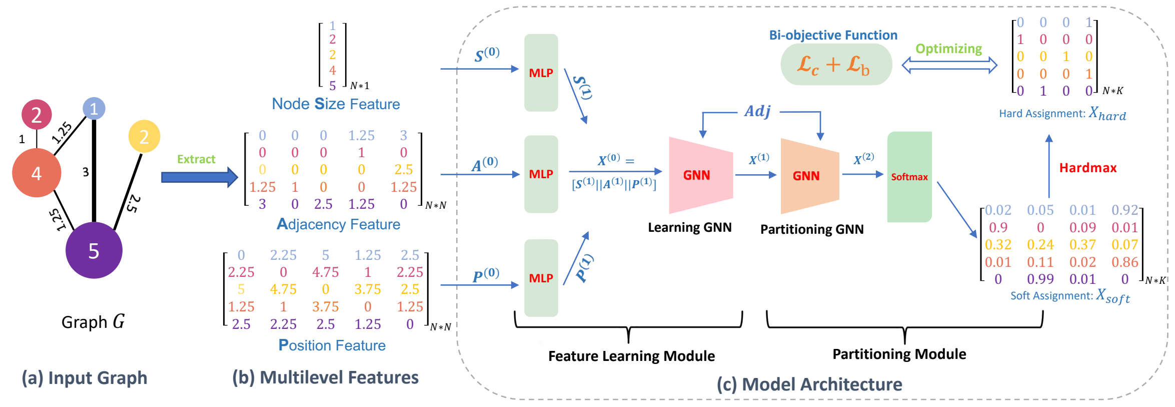


Figure 1: Overall framework of our proposed method for deep graph partitioning.

3. Proposed Method

In this section, we first present the bi-objective graph partitioning problem. Then, we explore how multilevel features can be utilized and why they can improve the partitioning quality. Finally, we illustrate our end-to-end model architecture.

3.1. Bi-objective Function

In this section, we introduce two optimization objectives for the *K-way* weighted graph partitioning. The first is the well-established normalized cut Shi (2003), and the second is our proposed partition-wise balance metric. These are detailed in the following sections.

3.1.1. The Normalized Cut Objective Function

In Shi (2003), the authors demonstrate that in the normalized form, minimizing the cuts between partitions is equivalent to maximizing the associations within partitions based on the fact that the total number of edges is fixed. Most existing deep partitioning works Nazi et al. (2019); Liu et al. (2022); Gatti et al. (2022); Tsitsulin et al. (2023) use variants of the normalized cut, such as GAP, which cannot reflect the true number of cut edges. In this paper, we adopt the standard normalized cut Shi (2003), which is measured by associations within partitions revealing the real partitioning quality, as our edge-cut objective in this paper. Moreover, for easy of implementation, we choose to minimize the negative normalized cut denoted as L_c, shown in Eq. 1:

$$L_c = -\frac{1}{K} \operatorname{tr}\left(\frac{\mathbf{X}^\top \mathbf{W} \mathbf{X}}{\mathbf{X}^\top \mathbf{D} \mathbf{X} + \mathbf{I}_K \times 10^{-3}}\right). \tag{1}$$

In Eq. 1, K is the number of desired partitions, and $\mathbf{X}$ is an $N \times K$ node assignment matrix with binary elements, where N represents the number of nodes. An element $\mathbf{X}_{ij} = 1$ indicates that node i is assigned to partition j, while

$\mathbf{X}_{ij} = 0$ indicates otherwise. $\mathbf{W}$ is the weighted adjacency matrix, $\mathbf{D}$ is the degree matrix of $\mathbf{W}$, $\mathbf{I}_K$ denotes a $K \times K$ identity matrix and $\mathrm{tr}(\cdot)$ represents the trace of a matrix.

In Eq. 1, both the numerator and denominator are diagonal matrices. The K diagonal values in the numerator show the sums of edges *staying* in each partition, and the diagonal values in the denominator represent the sums of both *staying* and *escaping* edges in each partition. Thus, it is easy to see Eq. 1 is bounded between 0 and 1, and maximizing Eq. 1 promotes the association within each partition. In addition, for more robustness optimization, we add a small constant (10^{-3}) into the denominator in Eq. 1. More details about this modification can be found in Appendix A.1.

3.1.2. The Balance Objective Function

Although the cut objective has been standard, such as the normalized cut L_c adopted in this paper, the balance objective is not well studied. Previous works either neglect the consideration of partition-wise balance or present an unbounded balance objective resulting in a large optimizing space Baños et al. (2004); Farshbaf and Feizi-Derakhshi (2009); Gatti et al. (2022); Gatti and Hu (2022); Nazi et al. (2019). To address these issues, we propose a well bounded partition-wise balance objective function L_b as shown in Eq. 2:

$$L_b = \frac{1}{\sqrt{K(K-1)}\overline{s}} \left\| \mathbf{X}^\top \mathbf{S}\mathbf{X} - \overline{s} \times \mathbf{I}_K \right\|_F \tag{2}$$

where $\mathbf{S} \in \mathbb{R}^{N\times 1}$ is the node weight feature, $\overline{s} = \frac{\sum_{i=1}^{N} s_i}{K}$ is the size of a perfectly balanced partition, and $\|\cdot\|_F$ indicates the Frobenius norm.

The L_b is a *partition-wise* balance objective, and minimizing L_b encourages the size of each partition to as close to the perfectly balanced size $\overline{s}$ as possible. Specifically, both $\mathbf{X}^\top\mathbf{S}\mathbf{X}$ and $\overline{s}\times I_K$ are diagonal matrices, and the ith diagonal value of $\mathbf{X}^\top\mathbf{S}\mathbf{X}$ is the size of the ith partition. Since $\overline{s} \times \mathbf{I}_K$ represents perfectly balanced K partitions, minimizing the elements in Frobenius norm is actually balancing the sizes of different partitions. This puts a balance requirement across all partitions, instead of just the biggest partition in traditional approaches Karypis and Kumar (1998a); Chevalier and Pellegrini (2008).

Then we present our Theorem 1 which is well bounded just as the normalized cut (the proof can be found in Appendix A.2), implying that the two objectives can be combined safely and optimized simultaneously. Moreover, Theorem 1 indicates that the lower L_b is, the more balanced partitions there are. In particular, when $L_b = 0$, the partitions are perfectly balanced.

Theorem 1. *For any assignment matrix* $\mathbf{X}$, $0 \le L_b \le 1$. □

Compared with balance objectives proposed by Nazi et al. (2019); Tsitsulin et al. (2023); Duval and Malliaros (2022), our L_b has three advantages. Firstly, our balance metric accurately represents the real sizes of each partition because it operates on hard assignments, whereas they uses soft assignments (the sum of all probabilities in i th column is the size of i th partition). Consider an extreme case when the model set equal probabilities on all partitions for each node, their balance metrics will consider this is perfect balanced, which is incorrect. Secondly, our balance metric is well-bounded and can be safely optimized alongside the normalized cut objective. Lastly, our metric accommodates both weighted and unweighted nodes, while they are designed solely for unweighted nodes.

3.1.3. Bi-objective Function and Optimization

Based on L_c and L_b, the bi-objective function given by

$$L = \alpha L_c + L_b, \tag{3}$$

where $\alpha > 0$ is a hyper-parameter that controls the trade-off between the two losses, and $-\alpha \le L \le 1$. When $\alpha > 1$, L_c will preferred over L_b, and vice versa.

In previous sections, we show that minimizing the normalized cut L_c encourages low edge-cut between partitions, and minimizing our L_b generates more balanced partitions which also has the exact same value range with L_c. Therefore, $L = L_c + L_b$ can be considered as the metric representing the overall partitioning quality and the lower L value means higher quality of partitions.

In the next sections, we will introduce how to optimize L efficiently from perspectives of graph features and model designing.

3.2. Node-level, Local and Global Features

Carefully designed graph features help to optimize the bi-objectives efficiently. Input features from different perspectives enhance the expressiveness of graph neural networks, to better distinguish nodes and structures in a graph for the K-way partitioning problem.

In Figure 1 (a), we show a weighted graph where nodes are represented by different colors, and the numbers on nodes represent nodes' sizes and the numbers on edges represent the edge weights. Ideally, the extracted graph features should be synergistic with the bi-objective and easy to obtain. To achieve this, we explore rich multilevel (node-, local- and global-) features as input to our deep partitioning model. Specifically, we extract three types of feature from a graph at different levels, i.e., *Node **S**ize Feature* at the node-level, ***A**djacency Feature* at the local level and ***P**osition Feature* at the global level, as shown in the Figure 1 (b).

Firstly, the *node size feature* is necessary for balancing partitions and is usually used as the default node features. Note that, the size of a node can be user-defined. For example, for a computation graph where a node represents an operator, e.g. a convolution operator, the node size can be defined as the operator's execution time. Secondly, the *adjacency feature*, i.e., the weighted adjacency matrix $\mathbf{W}$ of a graph, shows the neighbors of a node and their relationships (i.e. the edge weights), which provides a local view of each node in a graph and is essential to the calculation of cut. Note that, although the adjacency feature in Figure 1 (b) and the Adj in Figure 1 (c) represent the same matrix, they leverage adjacency information differently. Specifically, the adjacency feature is fed into our model as graph features, whereas the Adj functions as the node's neighbor indicator for GNNs This feature is still feasible when dealing with large graphs, since the adjacency matrix can be stored as a sparse matrix which is both computation and memory efficient. Finally, for the *position feature*, we represent the position encoding of a node as the shortest path of the node to other nodes, which is computed by Dijkstra's algorithm and presents a global position of a node in the whole graph. This feature provides more comprehensive information than the adjacency feature, and is important to minimize edge cut between partitions. Due to the algorithm's complexity, position encoding is implemented exclusively for small graphs.

3.3. Deep graph partitioning Model

Our deep graph partitioning model includes two main modules, as sketched in Figure 1 (c), the learning module and partitioning module. The feature learning module learns the multilevel features ($\mathbf{S}, \mathbf{A}, \mathbf{P}$) and outputs node embeddings. Then, in the partitioning module, we transform the node embeddings into a K-dimensional space where the ith dimension represents the ith partition. After this, a *softmax* layer is applied to generate a soft assignment matrix $\mathbf{X}_{\text{soft}}$ where $\mathbf{X}_{ij} \in \mathbf{X}_{\text{soft}}$ means the probability of the ith node belonging to the jth partition. Finally, to optimize hard assignments directly, we propose a Hardmax operator which drives the hard assignment matrix $\mathbf{X}$ from $\mathbf{X}_{\text{soft}}$ in a differentiable way. Next, we elaborate each module.

The feature learning module contains three multi-layer perceptron (MLP) layers and one GNN layer. The MLP layers learn input features ($\mathbf{S}^{(0)}, \mathbf{A}^{(0)}, \mathbf{P}^{(0)}$) independently and project them into appropriate dimensions for subsequent layers. The output of MLP layers will be concatenated and fed into a GNN layer to generate nodes embeddings $\mathbf{X}^{(1)}$ as the output of the module. Then, in the partitioning module, a partitioning GNN projects $\mathbf{X}^{(1)}$ into K dimensional space where each dimension corresponds to one partition. Next, a *softmax* layer is adopted to output a soft assignment matrix $\mathbf{X}_{\text{soft}}$ indicating probability distributions of each node over the K partitions.

However, as discussed in Section 1, actual partitioning requires discrete assignments, which can be suboptimal if a hard assignment matrix is not directly learned as shown in our experiment Section 4.5.1. Thus, we empower the model to optimize the hard assignment matrix $\mathbf{X}$ in an end-to-end manner. Note that the optimization process includes forward and backward propagation. In the forward propagation, the hard assignment matrix $\mathbf{X}$ is easily obtained using the arg max operator on $\mathbf{X}_{\text{soft}}$. However, when performing backpropagation, $\mathbf{X}$ is difficult to be optimized directly, because it is very sparse and the arg max operator cannot be differentiated .

To fix this, we propose a differentiable Hardmax operator to optimize and output $\mathbf{X}$ as shown in Eq. 4.

$$\begin{aligned}\mathbf{X} &= \text{Hardmax}(\mathbf{X}_{\text{soft}}) \\ &= \text{no\_grad}(\arg\max(\mathbf{X}_{\text{soft}}) - \mathbf{X}_{\text{soft}}) + \mathbf{X}_{\text{soft}}\end{aligned} \tag{4}$$

In the backpropagation, to optimize $\mathbf{X}$, we use the no−grad operator to stop the gradient calculation of "$\arg\max(\mathbf{X}_{\text{soft}}) - \mathbf{X}_{\text{soft}}$" and ensure Eq. 4 is differentiable. In the forward propagation, the no−grad operator will be transparent and $\mathbf{X} = \arg\max(\mathbf{X}_{\text{soft}})$, which is the output of our model and used for the calculation of our bi-objective.

To be more specific, since $\mathbf{X}$ are only influenced by the maximum probability in each row in $\mathbf{X}_{\text{soft}}$, it is equivalent to optimizing the positions of the maximum values in $\mathbf{X}_{\text{soft}}$. Thus, in the back propagation, we use $\mathbf{X}_{\text{soft}}$ to replace $\mathbf{X}$. Moreover, because of the guidance of the objective value from $\mathbf{X}$, optimizing $\mathbf{X}_{\text{soft}}$ finds a better partition assignment $\mathbf{X}$. Finally, $\mathbf{X}_{\text{soft}}$ is a dense matrix which helps solve the sparsity issue when optimizing.

The no−grad operator is provided in deep learning libraries such as Pytorch Paszke, Gross, Massa, Lerer, Bradbury, Chanan, Killeen, Lin, Gimelshein, Antiga et al. (2019), and enjoys widespread use, such as in Gumbel Softmax Jang, Gu and Poole (2022). Our contribution is introducing the Hardmax within the realm of graph partitioning, which is a key to improve partition quality.

3.4. The Graph Partitioning Algorithm

Our optimizing algorithm follows the typical training process in deep learning. Deep graph partitioning model learns the multilevel features and generates partitions to minimize the bi-objective function with the gradient descent optimizer. The partitioning with the minimum bi-objective function value during optimizing will be returned as the algorithm results.

We outline the optimizing process in Algorithm 1. Given a graph G, we first extract the multilevel features from it (line 1-4). Specifically, the node size feature can be user-defined, or all are set to 1 as default, the local feature is represented as the weighted adjacency matrix W, and the global feature is the node-pair shortest path from Dijkstra's algorithm. Next, we initialize our model M with initial random parameters $\theta^{(0)}$, and feed features into M (line 5). During the optimization (lines -13), M outputs a soft assignment matrix X'_{soft}, and our proposed Hardmax takes as input and outputs a hard assignment matrix X'_{hard}. Then, we calculate the bi-objective value l based on X'_{hard} (line 9). When the bi-objective value l becomes lower than the previous iteration, the assignment matrix $\mathbf{X}$ is updated to X'_{hard} (lines 10-11). In line 8, $\mathbf{X}$ is used for backpropagation and updating M's parameters. After that, we update the current iteration (line 13) and continue to the next iteration. When enough number of iterations are performed, our model converges and the training ends. The best partition found at the end of the training is then returned (line 14).

Algorithm 1: Our Deep Graph Partitioning Algorithm

```
Input: Graph $G$; Bi-objective function $L$;
       Partition model $M$ with initial parameters $\theta^{(0)}$;
       Number of partitions $K$ and iterations $Iter$.
Output: A hard node assignment Matrix $\mathbf{X}$.
1  Extract multilevel features:
2      The node-level user-defined feature: $\mathbf{S}^{(0)}$;
3      The local feature: $\mathbf{A}^{(0)} \leftarrow \mathbf{W}$;
4      The global feature $\mathbf{P}^{(0)} \leftarrow$ Dijkstra-algorithm($\mathbf{W}$);
5  Initialize and feed features into $M(\mathbf{S}^{(0)}, \mathbf{A}^{(0)}, \mathbf{P}^{(0)})$;
6  While $i < Iter$:
7      $\mathbf{X}'_{soft} \leftarrow M$;
8      $\mathbf{X}'_{hard} \leftarrow Hardmax(\mathbf{X}'_{soft})$;
9      $l \leftarrow L(\mathbf{X}'_{hard})$;
10     If $l^{(i)} < l^{(i-1)}$:
11         $X \leftarrow \mathbf{X}'_{hard}$;
12     Backpropagation: $\theta^{(i+1)} \leftarrow \theta^{i} - \alpha \frac{\partial M(\mathbf{X}, \theta^{i})}{\partial \theta^{i}}$;
13     $i \leftarrow i + 1$;
14 Return $\mathbf{X}$;
```

From Algorithm 1, the partitions are directly returned when training ends. That means no separate testing phase is required in our approach.

Table 1
Summary of datasets.

Datasets	Num. of Nodes	Num. of Edges	Description
Bert-3	235	250	Computation graphs[2]
Bert-12	783	843	Computation graphs[2]
PT	1,912	32,255	Social Networks[3]
add20	2,395	7,462	20-bit adder[4]
BlogCatalog	5,196	171,743	Social Networks[3]
Flickr	7,575	239,738	Social Networks[3]
musae-github	37,700	289,003	Social Networks[5]
twitch_gamers	168,114	6,797,557	Social Networks[5]
yelp	716,847	13,954,819	Social Networks[5]
com-youtube	1,134,890	2,987,624	Social Networks[5]
com-LiveJournal	3,997,96	34,681,18	Social Networks[5]
soc-LiveJournal1	4,847,571	68,993,773	Social Networks[5]

4. Experiment

We evaluate the benefits of our method against 12 state-of-the-art approaches on 12 datasets, with the goal of answering the following research questions:

- **Q1** Can our method yield higher quality partitions than state-of-the-art approaches?
- **Q2** What is the computational time required by our model to finish partitioning?
- **Q3** Can our method optimize partitions with preference in lower cut or lower imbalance?
- **Q4** Can the Hardmax operator improve partitioning quality, and how does our method perform with different combinations of inputs?
- **Q5** How does better partitions benefit downstream tasks?

We first introduce the datasets and baselines, then we will answer these questions sequentially.

4.1. Datasets, Baselines and Experiment Settings

4.1.1. Datasets

We validate our method using 12 real-world graph datasets, as shown in Table 1, with node counts ranging from hundreds to millions, covering various fields. Small and medium-scale graphs from real-world applications are considered to showcase our method's applicability across different scales and to cater to resource-limited devices like IoT devices. For example, the Bert-3 graph, containing only 235 nodes, is a computation graph constructed from a trained deep learning model with three transformer blocks Tarnawski, Phanishayee, Devanur, Mahajan and Nina Paravecino (2020). For computation graphs, a node represents an operator, e.g. a convolution operator, an edge represents a variable. In computation graphs, a node represents an operator (e.g., a convolution operator), and an edge represents a variable. The operator size is the memory usage of its associated weights, and the edge weight is the communication cost for transferring variables through a PIE bus. Bert-3 and Bert-12 have both node and edge weights, while the other 10 graphs are unweighted. For unweighted graphs, all node sizes and edge weights are set to 1.

4.1.2. Baselines

In the performance comparison on graph partition, we consider 12 state-of-the-art baselines.

Bi-objective Graph Partition. In Baños et al. (2004); Farshbaf and Feizi-Derakhshi (2009), the authors both proposed

[2] https://github.com/msr-fiddle/dnn-partitioning,
[3] https://pytorch-geometric.readthedocs.io/en/latest/modules/datasets.html,
[4] https://chriswalshaw.co.uk/partition, ,
[5] https://snap.stanford.edu/data,

genetic algorithms based on Pareto optimization to minimize the cut and the imbalance. We reproduced their methods using the NSGA-II genetic algorithm Deb, Pratap, Agarwal and Meyarivan (2002) as they did. The population size is set $1,000$ and the generation is set 100. We call the reproduced baseline *Genetic Algorithm*.

Single-objective Graph Partition. The Multi-class *Spectral Clustering* Shi (2003), which aims optimize the normalized cut L_c only, is considered as one baseline.

Constraint Single-objective Graph Partition. As discussed in Related Work, multilevel constraint single-objective methods are strong baselines for the graph partitioning problem. We choose two well-known multilevel baselines, i.e., *Metis* and *Scotch* Karypis and Kumar (1998b); Chevalier and Pellegrini (2008) with their default imbalance settings 0.05 for *Metis* and 0.03 for *Scotch*.

Deep Graph partition. *GAP* Nazi et al. (2019) and *GON* Liu et al. (2022) are adopted as baselines, as they have the normalized cut objective as ours and their own balance objectives. Note that, the official implementation of *GAP* is inefficient and time-consuming, and we only report *GAP*'s results on small scale graphs. For *GON*, since it is not open-source, we implement it following their methodology description and parameter settings, and will open source our baseline implementations. Additionally, it takes the identity matrix as the node feature matrix which has $N \times N$ dimension and may suffer scalability issue in memory usage for large scale graphs. To this end, we optimize it and use the sparse matrix to represent the identity matrix. Other deep graph partitioning methods Gatti and Hu (2022); Gatti et al. (2022) still follow the multilevel partitioning diagram, and only show similar performance with *Metis* and *Scotch*. If we can show better performance than *Metis* and *Scotch*, it is reasonable to infer we also outperform than Gatti and Hu (2022); Gatti et al. (2022).

Partitioning using Graph Pooling. *DMoN* Tsitsulin et al. (2023) and *HoscPool* Duval and Malliaros (2022) are two graph pooling layers and both contain the normalized cut loss and orthogonality loss. The losses are suitable for graph partition problem. We implement the two works using two GraphSAGE layers and one pooling layer.

Two-stage Partitioning Approaches. We consider the two-stage framework: learning embeddings then partitioning. Usually, the reconstruction loss between predicted Adj' and the adjacency matrix Adj is used in first stage, which is measured by the binary cross entropy. In the second phase, clustering methods will assign nodes into different partitions. We implement it using *GraphSAGE* Hamilton et al. (2017) and *DAEGC* Wang, Pan, Hu, Long, Jiang and Zhang (2019) to learn embeddings and using K-means Wu, Liu and Ma (2021) to cluster. For a fair comparison, these methods use two GNNs layers as ours. Due to running this framework is very time-consuming, we only report their results with small scale graphs.

Heuristic Community Detection Methods. Some heuristic community detection methods also consider the importance of number of edges within a intra-community or across inter-communities. For example, the *Girvan Newman* algorithm Girvan and Newman (2002) removes the edge with the highest betweenness centrality between inter-communities. The *Greedy Modularity* Maximization method Clauset, Newman and Moore (2004) finds the community with the largest modularity. Generally, the modularity shows how many edges remain within a community.

Note that, following existing deep graph partition works Gatti et al. (2022); Bianchi et al. (2020); Liu et al. (2022), although *Spectral Clustering* and *Community Detection Methods* are belong to graph clustering methods which do not have the balanced partitioning constraint, they are related with the normalized cut objective, i.e. L_c and show good performance for the bi-objective problem.

4.1.3. Experiment Settings

We conduct our experiments on an AMD EPYC 7543 CPU with 32 cores and a NVIDIA A40 GPU. We implement our model using Pytorch and integrate two GraphSAGE layers as our learning GNN layer and partitioning GNN layer, respectively. Our learning rate is 0.005. We set α in Eq. 3 as 10 and 1 for weighted and unweighted datasets, respectively. The number of training epochs is 500 with early stopping patience 100. These parameters are used for all experiments. For the baselines relying on node features, such as deep learning based baselines, we use the node size as the input feature for them as the default tradition. Besides node size feature, for datasets with less than ten thousands of nodes, we input both adjacency and position features into our method, while considering the scalability, we only adopt adjacency feature without position feature for large datasets. For all baselines and our method, we run 10 times using different seeds and pick the best result.

4.2. Evaluation on Partitioning Quality

To answer **Q1**, we conduct 3 sub-experiments in this section. The first experiment aims to show the overall performance, and the remaining experiments are for separate partition measurement metrics.

Table 2
The L value on 12 datasets.

Datasets	Bert-3			Bert-12			PT		
Methods \ K	2	6	10	2	6	10	2	6	10
Genetic Algorithm	$\underline{-0.9751}$	∅	∅	-0.6074	-0.1814	∅	-0.5203	∅	∅
Spectal Clustering	-0.9298	$\underline{-0.5144}$	$\underline{-0.4828}$	-0.5579	-0.712	-0.729	0.4138	-0.0942	-0.2215
Metis	-0.4629	-0.4573	-0.3814	-0.2859	-0.6507	-0.6317	$\underline{-0.728}$	$\underline{-0.4853}$	$\underline{-0.4003}$
Scotch	-0.9727	-0.3753	∅	$\underline{-0.997}$	$\underline{-0.89}$	$\underline{-0.9066}$	-0.7181	-0.4714	-0.3963
GON	0.9142	∅	∅	0.9288	∅	∅	0.9142	∅	∅
GAP	0.4649	0.7722	0.3822	0.8964	0.4426	0.523	-0.1106	0.0648	0.0836
Greedy Modularity	-0.7785	-0.5085	-0.4487	-0.15	-0.6133	-0.6111	-0.6991	0.1913	0.3788
Girvan Newman	-0.6118	-0.5118	-0.4086	-0.237	-0.6798	-0.7732	×	×	×
DAEGC	0.6748	0.3136	0.6037	0.5251	0.3377	0.3635	0.519	-0.0863	-0.0953
GraphSAGE	0.4649	0.7722	0.3822	0.8964	0.4426	0.523	-0.5407	-0.1438	-0.1108
HoscPool	-0.4252	∅	∅	-0.6389	∅	∅	-0.7109	∅	∅
DMoN	-0.8031	-0.0981	∅	-0.8353	0.0466	∅	-0.7093	0.053	∅
Our Method	**-0.9815**	**-0.5431**	**-0.5018**	**-0.9981**	**-0.9514**	**-0.9236**	**-0.7831**	**-0.5132**	**-0.4306**

Datasets	add20			BlogCatalog			Flickr		
Methods \ K	2	6	10	2	6	10	2	6	10
Genetic Algorithm	-0.5145	∅	∅	-0.5195	×	×	×	×	×
Spectal Clustering	0.454	0.1523	0.0827	0.3211	0.2096	0.2767	-0.5472	-0.0537	0.0224
Metis	$\underline{-0.8734}$	$\underline{-0.7833}$	$\underline{-0.7402}$	-0.6795	$\underline{-0.4466}$	$\underline{-0.3656}$	-0.664	$\underline{-0.3137}$	$\underline{-0.2444}$
Scotch	0.553	0.3712	0.2925	-0.6717	-0.4259	-0.3487	-0.4438	-0.2369	-0.1646
GON	-0.4350	∅	∅	0.5116	0.6220	∅	∅	∅	∅
GAP	-0.8293	-0.7605	-0.7368	-0.5407	-0.1438	-0.1108	-0.376	-0.0597	0.0999
Greedy Modularity	0.0829	-0.1304	-0.1964	-0.604	0.032	0.0286	-0.4839	0.2121	0.3328
DAEGC	0.6328	0.2204	0.1801	0.5562	0.4416	0.5379	0.7070	0.7400	0.8027
GraphSAGE	0.553	0.3712	0.2925	-0.1106	0.0648	0.0836	-0.3760	-0.0597	0.0999
HoscPool	-0.7781	∅	∅	-0.7631	∅	∅	-0.6631	∅	∅
DMoN	-0.5604	0.16	∅	$\underline{-0.765}$	∅	∅	$\underline{-0.6856}$	0.4983	∅
Our Method	**-0.8976**	**-0.7987**	**-0.7464**	**-0.7831**	**-0.5132**	**-0.5132**	**-0.7263**	**-0.3801**	**-0.2958**

Datasets	musae-github			twitch_gamers			yelp		
Methods \ K	2	6	10	2	6	10	2	6	10
Metis	$\underline{-0.7640}$	-0.5632	$\underline{-0.5268}$	$\underline{-0.7156}$	-0.5526	-0.4788	$\underline{-0.8844}$	-0.7474	-0.6859
Scotch	-0.7594	$\underline{-0.5842}$	-0.4946	-0.6686	$\underline{-0.5749}$	$\underline{-0.4920}$	-0.8614	$\underline{-0.7950}$	**-0.7346**
HoscPool	-0.5853	∅	∅	×	×	×	×	×	×
GON	-0.1880	∅	∅	0.4135	∅	∅	-0.4869	∅	∅
DMoN	-0.6884	∅	∅	×	×	×	×	×	×
Our Method	**-0.8253**	**-0.6044**	**-0.5286**	**-0.7993**	**-0.5931**	**-0.5213**	**-0.9202**	**-0.8021**	$\underline{-0.7299}$

Datasets	com-youtube			com-LiveJournal			soc-LiveJournal1		
Methods \ K	2	6	10	2	6	10	32	64	128
Metis	$\underline{-0.8892}$	$\underline{-0.8237}$	$\underline{-0.7766}$	$\underline{-0.8924}$	$\underline{-0.8264}$	$\underline{-0.8033}$	-	-	-
Scotch	-0.7950	-0.7694	-0.7457	-0.8569	-0.8217	-0.7833	-	-	-
GON	-0.3726	∅	∅	-0.2022	∅	∅	$\underline{-0.6485}$	$\underline{0.3962}$	∅
Our Method	**-0.9035**	**-0.8465**	**-0.8061**	**-0.9275**	**-0.8448**	**-0.8145**	**-0.7802**	**-0.745**	**-0.7026**

- The best L is bolded;
- The second best L is underlined;
- ∅: Exist empty partitions;
- ×: Timeout;
- −: Method not applicable.

4.2.1. Evaluation on Overall Quality of Partitions

In Section 3.1.3, we show L can represent the overall partitioning quality, and in Table 2, we report L values of 9 baselines on 12 datasets. $K \in \{2, 6, 10\}$ are chosen, because we have observed that the results show a decreasing trend with K gradually increasing, and it would be redundant and crowded if we put all results. Following the tradition of previous works Karypis and Kumar (1998a); Chevalier and Pellegrini (2008), we choose the even numbers for K. Note that, our method is flexible for both even and odd K values, and the evaluation on different K values is conducted in Section A.7. In Table 2, at each column, we bold the best L and underline the second best L. We use '∅' to represent baselines generate empty partitions which contain no nodes inside. We terminate baselines which run over 10 minutes and mark this case using '×'. Since some baselines have the scalability issue, such as GAP and Spectral Clustering, fewer baselines are shown in Table 2 for large scale graphs.

Firstly, the Table 2 shows that our method achieves the best L value on most datasets, which demonstrates our method can generate higher qualify partitions than baselines.

Secondly, compared with the bi-objective *Genetic Algorithm*, *GAP* and *GON*, our method achieves large margin improvements both in partitioning quality and scalability. For example, without partition-wise balance objective, there are empty partitions in *Genetic Algorithm*'s solutions, which means the sizes of partitions are very unbalanced. Lacking of multilevel graph features and end-to-end optimizing, Partitioning quality of *GAP* and *GON* is also unsatisfying, especially on weighted graphs. Furthermore, for the scalability, ten thousands scale graphs can be challenging for the two baselines. The scalability issue also lies in *Spectral Clustering* algorithm and heuristic community detection baselines. Because our model architecture is simple and multilevel features are feasible for large scale graphs, our method is able to partition million scale graphs. For the two-stage baselines *GraphSAGE* and *DAEGC*, the results, shown in Table 2, indicate that they perform poorly in terms of partition quality. This suggests that while they may be effective for graph clustering, they are not directly suitable for the graph partitioning task. As for graph pooling methods, i.e., *HoscPool* and *DMoN*, with the guidance of the similar loss function as ours, they can show good results in few cases. For example, in BlogCatalog dataset, *DMoN* shows the second-best L value. However, there are two limitations in these methods. The first is these methods often generate empty partitions when they meet large K, as lack of multilevel features, they can not distinguish and partition nodes in a high dimension space. The second is that they cannot optimize and output the node assignment matrix $\mathbf{X}$, which leads to inferior partitioning quality.

Finally, the Table. 2 shows that constraint single-objective methods, i.e. *Metis* and *Scotch*, are strong baselines. The multilevel partitioning paradigm enables them to process large graphs. However, they cannot optimize partitions in an end-to-end manner and lack of multilevel features to facilitate partitioning, thus their partitioning quality is inferior to ours. Moreover, these methods require the input graphs to be undirected, connected, and free of self-loops, making them unsuitable for partitioning the soc-LiveJournal1 graph, which is directed. We also use the biggest dataset soc-LiveJournal1 to show our model can generate over one hundred partitions without empty partitions, which demonstrates the effectiveness of our partition-wise balance metric L_b.

4.2.2. *Evaluation on Normalized Cut and Partition-wise Balance*

In this experiment, we evaluate how our method performs on the normalized cut L_c and partition-wise balance L_b. The results are shown in Figure 2. There are two sub-figures for each dataset showing the normalized cut (left) and balance (right) objective values. Among the compared baselines, since *GAP* and *GON* have the scalability issue, we only report their results on the small scale graphs.

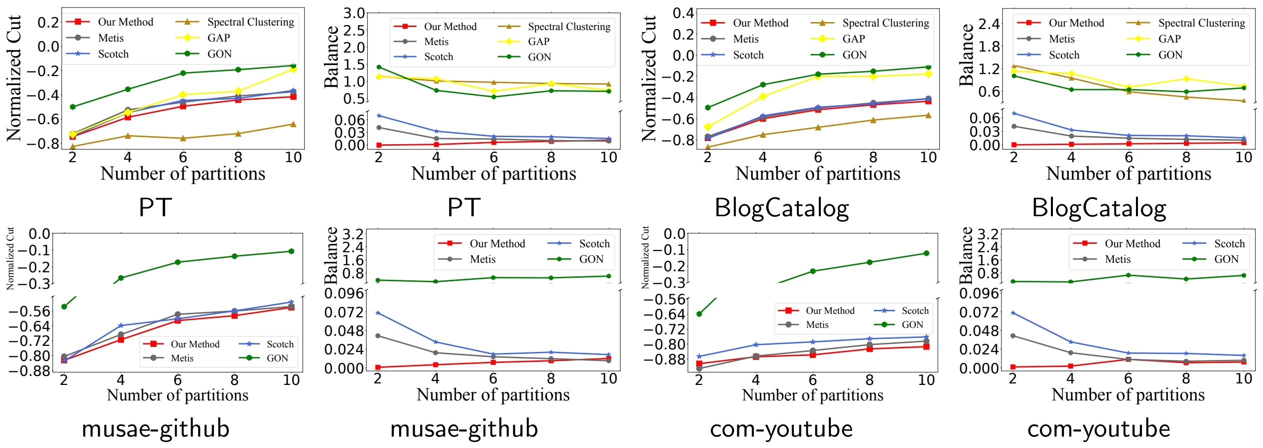


Figure 2: The objective values of Normalized Cut (L_c) and Partition-wise Balance (L_b). (The lower, the better)

From this figure, we observe that, for normalized cut, *GAP* and *GON* show the worst performance on PT and BlogCatalog graphs. This is because GAP and GON tend to generate empty partitions when K is large (e.g., $K \in 4, 6, 10$). The normalized cut metric considers the association within each partition, so the presence of empty partitions results in higher normalized cut. Moreover, empty partitions also lead to very unbalanced partitions, therefore the two methods also have higher L_b.

For traditional baselines, the Figure 2 shows that, compared with *Scotch* and *Metis*, our method achieves both lower cut and balance objective values in most situations. Although, *Spectral Clustering* always achieves the lowest *cut objective* value, these partitions are very imbalanced as shown in PT and BlogCatalog datasets. This experiment demonstrates that our superior overall partitioning quality in the last experiment is contributed by optimizing both the cut and balance objectives.

The right sub-figures for each dataset also show our method has lower balance objective values than baselines in most cases, which means these partitions are more balanced. For example, when $K = 2$ in PT and BlogCatalog datasets, our method drives the optimal balance value ($L_b = 0$) and finds perfect balanced partitions.

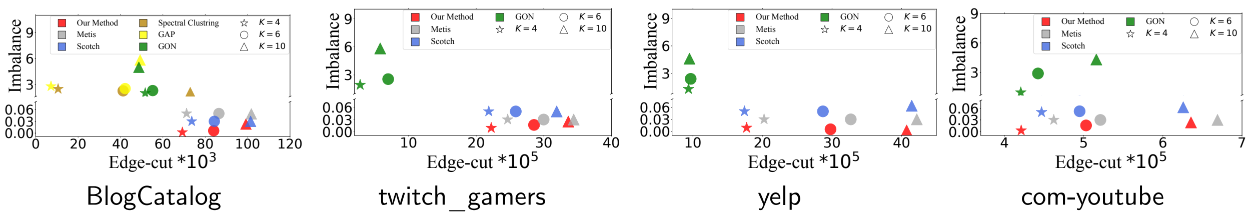


Figure 3: Comparison on Edge-Cut and Imbalance. (The lower, the better)

4.2.3. Evaluation on Edge-cut and Imbalance

The edge-cut and the degree of imbalance are also common metrics for measuring partition quality. The edge-cut is calculated by counting the number of cut edges across partitions. The imbalance degree is the ratio of the biggest partition's size to the perfect balanced size, which is defined as follows:

$$\text{Imbalance} = \frac{\max\{\sum_{v\in\mathbb{V}_i} s_v | i \in \{1, \cdots, K\}\}}{\bar{s}} - 1.$$

We use four datasets and report the results with $K \in \{4, 6, 10\}$. The results are shown in Figure 3. In this figure, different colors represent different methods and different markers represent different K values. The X-axis shows the imbalance and the Y-axis shows the scaled edge-cut. With the same K value, the markers located in the bottom-left area indicate better partitions.

In Figure 3, on the smallest graph BlogCatalog, *GAP* and *GON* show very low edge-cut. The reason is that empty partitions generated by GAP and GON lead to fewer non-empty partitions and consequently lower edge-cut. Strictly speaking, the presence of empty partitions indicates that *GAP* and *GON* cannot perform K-way partitioning. Additionally, empty partitions also result in high Imbalance value for the two baselines. The high Imbalance value represents unbalanced partitions be consistent with the results and analysis in Section 4.2.2.

From Figure 3, we can observe that with the same K, our method shows the lowest imbalance in most cases, which illustrates the effectiveness of our proposed partition-wise balance objective. Moreover, the Figure 3 shows the imbalance of *Scotch* and *Metis* is quite close or equal to the default imbalance setting (0.05 for *Scotch* and 0.03 for *Metis*), which means they do not explore more balanced partitions. Finally, *Spectral Clustering* achieves the lowest edge-cut in most cases, which demonstrates that maximizing the associations, i.e. the L_c, leads to fewer cut edges. However, its imbalance is usually the highest, which is consistent with the observation in Section 4.2.2.

The Figure 3 also shows our method only on BlogCatalog dataset has both lower edge-cut and imbalance than *Scotch* and *Metis*. In other datasets, our method achieves comparable cut with the two baselines. The reason is that our cut objective is not only the edge-cut, but the normalized version of it which considers both the cut between partitions and association within partitions.

4.2.4. Evaluation on Remaining Edges and Balanced Constraint Metric

Considering comprehensive comparison, we further evaluate the quality of partitions generated by minimizing L by adopting two new metrics, i.e., the remaining edge $|\mathcal{E}_{in}|$ and balance constraint metric BCI, from *GON*. Specifically, $|\mathcal{E}_{in}|$ indicates the number of remaining edges inside all partitions, and BCI measures the difference between the overall balance condition of partitions and the perfect balance, which are formulated as follows:

$$\begin{cases} |\mathcal{E}_{in}| & = \mathrm{tr}(\mathbf{X}^\top \mathbf{W}\mathbf{X}) \\ BCI & = \frac{1}{K}\sum_{i=1}^{K} \frac{\mathrm{abs}(|\mathbb{V}_i|-\overline{s})}{\overline{s}} \end{cases}$$

where K is the number of desired partitions, and $\mathbf{X}$ is a node assignment matrix with binary elements, where N represents the number of nodes. An element $\mathbf{X}_{ij} = 1$ indicates that node i is assigned to partition j, while $\mathbf{X}_{ij} = 0$ indicates otherwise. $\mathbf{W}$ is the weighted adjacency matrix, the i th diagonal element in $(\mathbf{X}^\top \mathbf{W}\mathbf{X})$ represents the number of remaining edges in the i th partition, and $\mathrm{tr}(\cdot)$ represents the trace of a matrix. For BCI, $|\mathbb{V}_i|$ indicates the sum of node in the i th partition and $\overline{s} = \lceil \frac{\sum_{i=1}^{K} |\mathbb{V}_i|}{K} \rceil$ is the size of a perfectly balanced partition. $\mathrm{abs}(\cdot)$ function returns an absolute value of a number.

In this experiment, we compare our method with 4 baselines (*GON*, *GAP*, *Spectral Clustering*, *Metis* and *Scotch*) on the same graphs with the same K values as in Section 4.2.3. However, due to the scalability issue of *Spectral Clustering* and *GAP*, they cannot partition the graphs with more than 100,000 nodes. Therefore, we can only report their results on small scale graphs. The experimental results are shown in Figure 4, where with the same K value, the markers located in the bottom-right area indicate better partitions..

Figure 4 shows compared with baselines, our method achieves the lowest BCI while comparable or even better $|\mathcal{E}_{in}|$. Specifically, on the 4 datasets, our method shows the lowest BCI with all K value cases, which implies minimizing our proposed L_b brings more balanced partitions. For the $|\mathcal{E}_{in}|$ metric, *GAP*, *GON* and *Spectral Clustering* achieve better/higher values showing more remaining edges within partitions than other methods. However, the good $|\mathcal{E}_{in}|$ values from the three baselines come from at the cost of very unbalanced partitions showing as the worse BCI values. For example, *GAP* and *GON* generate empty partitions. It is noteworthy that for the bi-objective partition problem, both two objective values should be optimized as good as possible. Therefore, we argue that our method shows superior performance than the baselines, because it achieves better trade-off between the two objectives. For example, in graph yelp with $K = 10$ and graph com-youtube with $K = 4$, our method shows both highest $|\mathcal{E}_{in}|$ values as well as lowest BCI values.

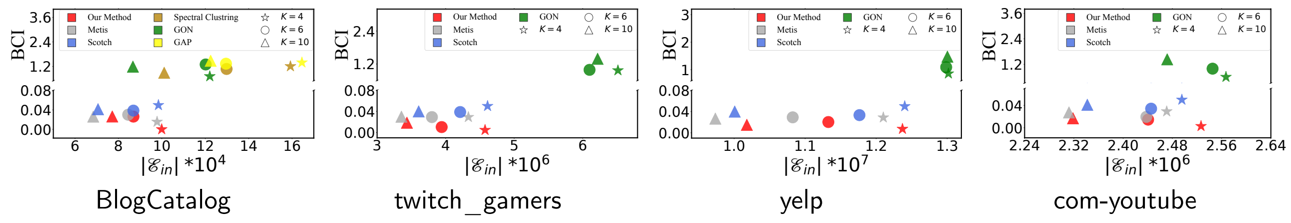


Figure 4: Comparison on $|\mathcal{E}_{in}|$ (The higher, the better) and BCI (The lower, the better).

4.3. Evaluation on Running Time

This experiment is conducted for **Q2** to show whether our method is computation-efficient. We used four large-scale datasets and $K = 2$. Baselines *Metis* and *Scotch* are compared to our method.

The Table. 3 shows *Metis* needs the least time to partition, and our method is faster than *Scotch*. Both *Metis* and *Scotch* follow the multilevel procedure as introduced in Section 2.2, which brings high partitioning speed for large graphs. Efficient C++ implementation of *Metis* further shortens the partitioning time cost. Our carefully designed graph features and model help to fill this latency gap. Specifically, for large datasets, we only use node size and adjacency features which are resource-efficient in terms of storage and computation. This is particularly advantageous for the adjacency feature, often represented as a sparse matrix.

Additionally, our model comprises only five deep layers, which has few weights and can converge fast within a few hundred of epochs. Figure 5, shows the objective values during the optimizing process on two datasets. We can observe our model converges at less than 200 epochs.

4.4. Evaluation on Different α Values

To answer **Q3** (optimizing partitions with cut or balance preference), we conduct this experiment to show how our method performs with different α values in the objective function L. In this experiment, we assign α with

Table 3
The running time on five datasets (unit: second).

Methods \ Datasets	twitch gamers	yelp	com-youtube	com-LiveJournal	soc-LiveJournal1
Metis	2.316	3.640	2.136	19.47	-
Scotch	66.61	84.58	43.99	427.49	-
Our Method	27.92	30.96	24.93	226.48	406.33

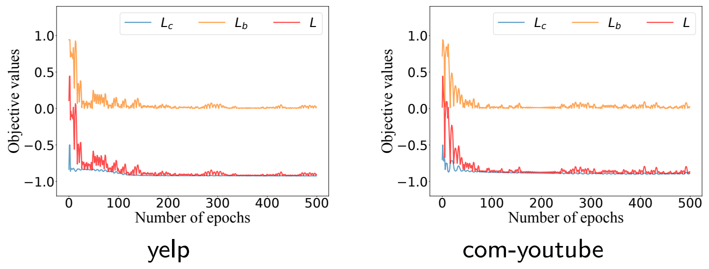


Figure 5: The objective values within 500 epochs.

{0.01, 0.1, 0.5, 1, 10, 20, 40, 60, 80, 100}. We run this experiment with $K = 10$ on four datasets. When $\alpha > 1$, we may prefer lower cut partitions. The results are shown in Figure 6. Note that, besides this, in Appendix A.4, we also design an experiment show whether our method can generate edge balanced partitions when nodes' degrees follow a power-law distribution in many real world graphs.

Figure 6 shows, overall, there is non-linear trend of objective values as α value increases. Specifically, as α increase from 0.01 to 10, all objective values decrease. The decreasing trend implies preferably optimizing L_c also brings lower L_b, since L_c considers the association within each partition. However, as α continues to increase to 100, L and L_b increase although L_c decreases. This indicates that the benefit of focusing on optimizing L_c diminishes, and the drawback of neglecting L_b becomes more pronounced. For instance, in PT dataset, L_b from about 0.01 ($\alpha = 1$) grows over 0.15 ($\alpha = 40$), and meanwhile L_c from about -0.48 becomes very close to -0.52.

Based on the L values in Figure 6, choosing α from [0.5, 10] can bring high partitioning quality, and we find $\alpha = 10$ for weighted datasets and $\alpha = 1$ for unweighted datasets have overall better performance, i.e., lower L value. Thus, we choose the two α values for the two kinds of datasets to run the previous partition evaluation experiments.

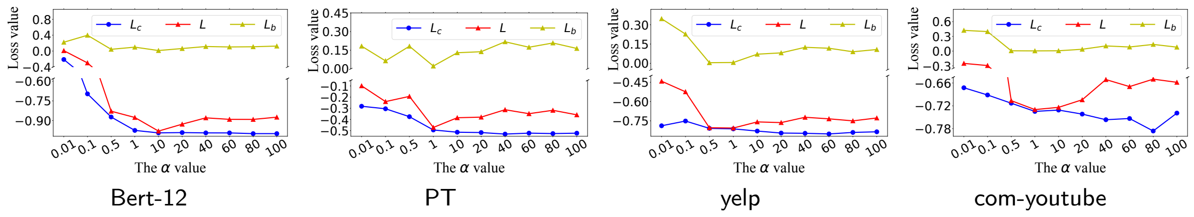


Figure 6: The L_c, L and L_b with different α values.

4.5. Ablation Study

To answer **Q4**, we design two experiments to explore the effect of introducing the Hardmax operator into our model and the performance of our model with different inputs. Moreover, we also show the results of our model with different model architectures in Appendix A.5.

4.5.1. *Effect of Hardmax operator*

In this experiment, we remove the Hardmax operator from our model as a variant named *Softmax only*, and others are remained as same. Then we run *Softmax only* on weighted graph Bert-12 and unweighted graph yelp with $K = 6$. The L_c and L_b values of *Softmax only* and *With Hardmax* are reported in Table 4.

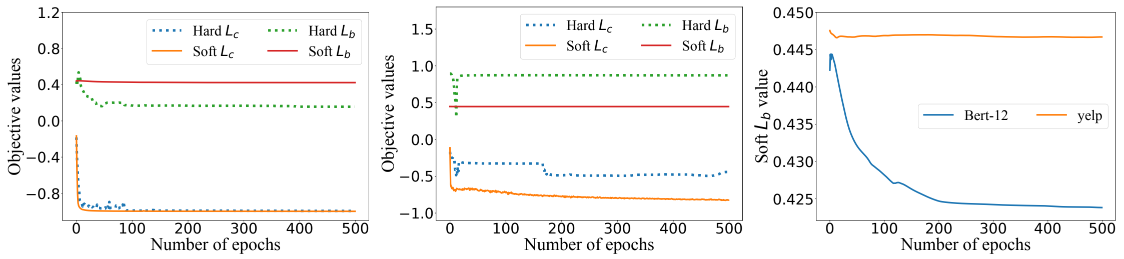
Table 4
The L_c and L_b values of Hardmax variants.

Datasets	Bert-12		yelp	
Models \ L_c and L_b	L_c	L_b	L_c	L_b
Softmax only	-0.9493	0.2617	-0.7021	0.6432 (∅)
With Hardmax	-0.9826	0.0312	-0.8038	0.0017

- ∅: Exist empty partitions;

Table 4 shows that on the two datasets, the L_c and L_b values of *With Hardmax* are much lower than those of *Softmax only*, especially for L_b. This indicates that the model guided by hard assignments can generate higher quality partitions.

Figure 7 presents, for each epoch, the L_c and L_b values calculated by soft assignments of *Softmax only* (solid lines), as well as the L_c and L_b values calculated by hard assignments derived from the soft assignments (dotted lines). We can observe significant gaps between the soft objective values and the true/hard objective values. Moreover, Figure 7 (a) and (b) show that optimizing soft objectives cannot lead to lower true objective values in all situations. For example, for the L_c on yelp dataset, the lowest hard L_c value appears at 10 epochs, but the Soft L_c continues to be optimized beyond this point until reaching the epoch limit. We can also observe that there are straight line trends of Soft L_b on the two graphs shown in Figure 7 (a) and (b). In Figure 7 (c), we show the zoomed in on Soft L_b values where the Soft L_b is optimized in a very small range, resulting the straight line trend. For example, the Soft L_b on yelp graph is optimized from 0.447 to 0.445. Compared with Soft L_c on yelp graph from 0.0 to -0.9 shown in Figure 7 (b), the difference of Soft L_b is subtle. Note that, Hardmax also helps optimizing L_c metric. Specifically, our method optimizes and outputs trustworthy hard assignments and L_c values, reflecting true partitioning quality. Moreover, the improvement on L_c can also be observed in Table 4. For example, on yelp graph, L_c is optimized from -0.7 on *Softmax only* to -0.8 on *With* Hardmax.



(a) Bert-12 (b) yelp (c) Soft L_b values (zoomed in)

Figure 7: The comparison of objective values between hard and soft assignments.

4.5.2. Different Combinations of Inputs

In this experiment, we analyze the influence of different combinations of inputs on Bert-12 and soc-LiveJournal1 datasets with $K \in \{2, 10\}$. As introduced in Section 3.2, we have three inputs, that is, the node size feature $\mathbf{S}$, adjacency feature $\mathbf{A}$ and position feature $\mathbf{P}$. Different combinations means that we remove one or two features from the three inputs. For example, '*Without* $\mathbf{A}$' means we remove the adjacency feature and only feed the node size feature and position feature into our model. Because the node size feature is usually fed as default node features, we do not have the '*Without* $\mathbf{S}$' option. Moreover, since for large graphs, we do not use the position feature, on soc-LiveJournal1 we only have '*With* $\mathbf{A}$' and '*Without* $\mathbf{A}$' options. The results of different combinations are shown in Figure 8. '∅' means there are empty partitions.

From Figure 8, we observe without the adjacency feature and position feature, our model shows the worst objective value on all datasets and usually generates empty partitions. This means the two features are necessary and helpful for generating K high quality partitions.

The quality improvements on weighted datasets Bert-12, are more obvious than unweighted datasets soc-LiveJournal1. On weighted datasets, without the position feature $\mathbf{P}$, the performance on objective value decreases

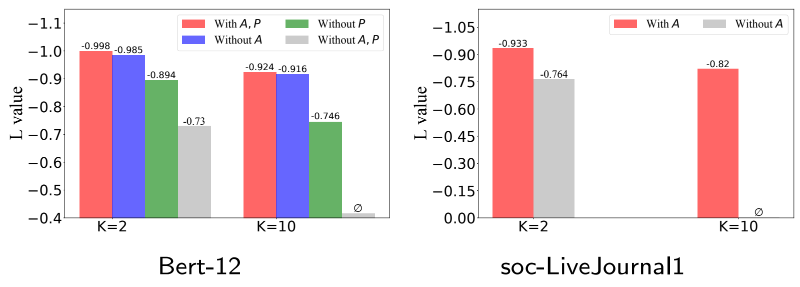


Figure 8: The bi-objective value L with different inputs.

Table 5
The L values with different GNN backbones.

Datasets	Bert-12			com-youtube		
GNNs \ K	2	6	10	2	6	10
GAT	$\underline{-0.9937}$	**-0.9697**	**-0.9355**	$\underline{-0.8941}$	-0.7981	-0.7361
GIN	-0.9760	-0.9141	$\underline{-0.9251}$	-0.8628	**-0.8471**	$\underline{-0.8022}$
GraphSAGE	**-0.9981**	$\underline{-0.9514}$	-0.9236	**-0.9035**	$\underline{-0.8465}$	**-0.8061**

- The best L is bolded;
- The second best L is underlined;

at a bigger space than without adjacency feature **A**. The reason may be **P**, which shows the shortest weighted path of nodes to others, provides more information than A, since **A** only contains the weight of nodes with their neighbours. However, since **P** is not available for large graphs, **A** plays an important role in such situations. As seen with the soc-LiveJournal1 dataset, the partitioning quality decreases without **A**, and the model can generate empty partitions

4.5.3. Different GNN Backbones

In this experiment, we explore the effect of different GNN backbones on the partitioning quality. We choose two well-known GNN backbones, GAT Veličković, Cucurull, Casanova, Romero, Liò and Bengio (2018) and GIN Xu, Hu, Leskovec and Jegelka (2018), and conducted this ablation experiment onboth weighted and unweighted graphs (Bert-12 and com-youtube). We used implementations of GAT and GIN from PyTorch Geometric, with 4 attention heads for GAT and other parameters (e.g., number of layers) consistent with those used for GraphSAGE.

The results, reported in Table 5, demonstrate that while GraphSAGE does not always achieve the lowest L values, it consistently shows more stable performance, achieving either the best or second-best results in most cases. The simplicity of its architecture and the minimal tuning required contribute to its reliable performance, making it our empirical choice for the GNN backbone.

4.5.4. Different Number of GNN Layers

This experiment investigates the impact of using different numbers of GNN layers on partitioning quality. Our method consists of two modules: a graph learning module and a graph partitioning module, each originally containing a single GNN layer. To explore the effect of additional layers, we created variants of our method by stacking 2, 4, and 8 GNN layers in each module and tested these variants on both weighted and unweighted datasets.

The results, presented in Table 6, reveal that increasing the number of GNN layers consistently leads to a decrease in partitioning quality, especially as the number of partitions K increases. For instance, on the Bert-12 graph with $K = 2$, the L value drops from -0.9981 with 1 GNN layer to -0.6425 with 8 GNN layers. The performance degradation is more pronounced for larger K values, as seen in the results for the com-youtube graph.

The primary reason for this decline in performance is that more GNN layers require more epochs to converge. However, in our experiments, the number of epochs was fixed at 500, leading to suboptimal convergence for deeper models. Additionally, more GNN layers demand greater computational resources, such as GPU memory. To balance partitioning quality and efficiency, we have opted to use a single GNN layer in our method.

Table 6
The L values with different numbers of GNN layers.

Datasets	Bert-12			com-youtube		
Methods \ K	2	6	10	2	6	10
8 GNN layers	-0.6435	-0.4524	-0.4036	<u>-0.9079</u>	-0.5957	-0.5478
4 GNN layers	-0.9568	-0.7782	-0.8346	-0.9003	-0.7996	<u>-0.7842</u>
2 GNN layers	<u>-0.9893</u>	<u>-0.9166</u>	<u>-0.9078</u>	**-0.9094**	<u>-0.8071</u>	-0.6486
1 GNN layer	**-0.9981**	**-0.9514**	**-0.9236**	-0.9035	**-0.8465**	**-0.8061**

- The best L is bolded;
- The second best L is underlined;

Table 7
Different metric values for the community detection task.

Methods \ Metrics	Running Time (second)	Modularity	Purity
Metis	167.6	0.7036	0.1562
Our Method	153.7	0.7211	0.1698

4.6. Evaluation on the Community Detection Task

Graph partitioning is typically used as a preprocessing step to obtain smaller subgraphs and accelerate downstream tasks, as directly applying such algorithms to large-scale graphs can be prohibitively time-consuming and sometimes infeasible. A key challenge in graph/network analysis is community detection, where nodes are grouped into tightly connected clusters that have more internal edges and fewer external ones Ma et al. (2020). To address **Q5** (better partitions benefit downstream tasks), similar to Stanton and Kliot (2012), we demonstrate that higher-quality partitions result in reduced time to complete community detection tasks and lead to better-detected communities. In this experiment, we use the well-known Louvain Algorithm Traag, Waltman and Van Eck (2019) to detect communities in the com-youtube dataset, with *Metis* serving as the baseline.

The com-youtube graph is partitioned into two sub-graphs, and we run Louvain Algorithm on each sub-graph. The total running time required for the task is determined by the longest running time among the two sub-graphs. Furthermore, to evaluate the goodness of detected communities, we choose two metrics: Modularity and Purity Chakraborty, Dalmia, Mukherjee and Ganguly (2017). The Modularity measures the goodness of communities from the topology view which encourages nodes within the same community are densely connected to each other. The Purity is to evaluate how well the communities aligns with a ground truth from a clustering task view. Note that, for Modularity and Purity, higher values indicate better performance.

We run five times for each method and report the average of metric values which are shown in Table. 7.

Table. 7 shows partitions generated by our method need less time to complete Louvain Algorithm. The reason is that the time complexity of this algorithm scales up with the number of nodes and our partitions are more balanced. Moreover, we can observe communities detected from our partitions are better than *Metis*'s. The reason might be our cut objective is the normalized cut rather than minimum cut adopted by *Metis*. Since minimizing normalized cut leads to strong association within partitions and sparse disassociation between partitions, our detected communities have higher Modularity, and nodes densely connecting to each other are tend to be the same community, i.e., higher Purity.

5. Conclusion

In this paper, we propose an end-to-end bi-objective GNN-based graph partitioning method, which employs multilevel graph features, guides the partitioning with not only a lower cut but also a partition-wise balance requirement, and optimizes and outputs the partitions in an end-to-end manner through our proposed Hardmax operator. Extensive experiments on graph partitioning tasks demonstrate the effectiveness of our method. This work sheds light on applying deep learning-based models in the graph partitioning problem, showcasing its superior learning ability in an end-to-end manner.

6. Acknowledgments

We first gratefully acknowledge anonymous reviewers who take their valuable time to read this draft and make any helpful suggestions. The work is supported by National Natural Science Foundation of China (Grant No. 62172145), and the Ministry of Education, Singapore, under its Academic Research Fund Tier 2 (Proposal ID: T2EP20122-0041). Any opinions, findings and conclusions or recommendations expressed in this material are those of the author(s) and do not reflect the views of the Ministry of Education, Singapore.

A. Appendices

A.1. The Modification on the Normalized Cut

During the optimizing process, all values in the jth column of $\mathbf{X}$ sometimes can be 0 which means there is a empty partition that contains no nodes inside. This will raise a *zero division error* and stop the optimizing process. To fix this, we add a small constant (10^{-3}) into the denominator in Eq. 1.

The small constant does not have a significant impact on L_c, because for zero diagonal values in $\mathbf{X}^\top\mathbf{DX}$, the association (i.e., the numerator of Eq. 1) is also 0, and for non-zero values in $\mathbf{X}^\top\mathbf{DX}$, they are usually much larger than the small constant.

A.2. Proof of the Boundness of Balance Objective Function

Proof. Let $\mathbf{P} = \mathbf{X}^\top\mathbf{SX}$ and $\mathbf{Q} = \overline{s}\times\mathbf{I}_K$. Both $\mathbf{P}$ and $\mathbf{Q}$ are a diagonal matrix, and $\mathbf{P}_{ii}$ represents the size of ith partition.

We first prove the left inequality. According to the definition of the Frobenius norm, we have

$$\|\mathbf{P}-\mathbf{Q}\|_F = \sqrt{\sum_{i=1}^{K}(\mathbf{P}_{ii}-\overline{s})^2} \tag{5}$$

It is easy to find $\|\cdot\|_F \geq 0$. When for $\forall i, \mathbf{P}_{ii} = \overline{s}$, i.e. achieving perfect balance, $L_b = 0$.

For the right inequality, we should note that $\mathbf{P}_{ii} \leq K * \overline{s}$, because $K * \overline{s}$ is the size of the whole graph. So, we have $\mathbf{P}_{ii} - \overline{s} \leq (K-1)\overline{s}$. If $\exists\, \mathbf{P}_{ii} \geq \overline{s}$, then $\mathbf{P}_{ii} - \overline{s} \geq 0$ and $(\mathbf{P}_{ii} - \overline{s})^2 \leq (K-1)\overline{s}(\mathbf{P}_{ii} - \overline{s})$. Again using $\mathbf{P}_{ii} - \overline{s} \leq (K-1)\overline{s}$, we have $0 \leq (\mathbf{P}_{ii} - \overline{s})^2 \leq [(K-1)\overline{s}]^2$. If $\exists\, 0 \leq \mathbf{P}_{ii} < \overline{s}$, we have $0 < \overline{s} - \mathbf{P}_{ii} \leq \overline{s}$, and $(\mathbf{P}_{ii} - \overline{s})^2 = (\overline{s} - \mathbf{P}_{ii})^2 \leq \overline{s}^2$. Because $K > 1$, $\overline{s}^2 \leq [(K-1)\overline{s}]^2$. Thus, when $\exists\, 0 \leq \mathbf{P}_{ii} < \overline{s}$, we still have $(\mathbf{P}_{ii} - \overline{s})^2 \leq [(K-1)\overline{s}]^2$. For K partitions, we can get $\sum_{i=1}^{K} |\mathbf{P}_{ii} - \overline{s}|^2 \leq K[(K-1)\overline{s}]^2$. Then, we will have

$$\frac{1}{\sqrt{K}(K-1)\overline{s}}\sqrt{\sum_{i=1}^{K}(\mathbf{P}_{ii}-\overline{s})^2} \leq 1. \tag{6}$$

Finally, the right inequality is proved. □

A.3. The Optimizing Algorithm

A.4. Evaluation on Edge Balanced Partitions

Many graphs used in this paper are power-law graphs, where a minority of hyper-nodes have a very large number of neighbors and others have relatively few neighbors, such as social networks He Li and .etc (2022). If we consider edges as the workload unit, the previous node-based balance objective L_b may bring highly unbalanced workload partition because of the hyper-nodes.

By slight modification on node-level features, our method can achieve edge balanced partitions. Specifically, since a node's degree represents the number of connected edges to it, we can define a node's degree as its size feature. Then, optimizing our L_b with the new node-level feature will generate edge balanced partitions.

With above feature setting, we run this experiment on two large social network datasets and set $K = 2$. The results compared with *Metis* are shown in the Figure 9.

From the Figure 9, we can observe that the numbers of edges in our partitions are closer to the green dash line, which is the perfect edge balanced value, than *Metis*. This experiment also shows our partition-wise balance function L_b is flexible and effective with customized node features and balance definitions.

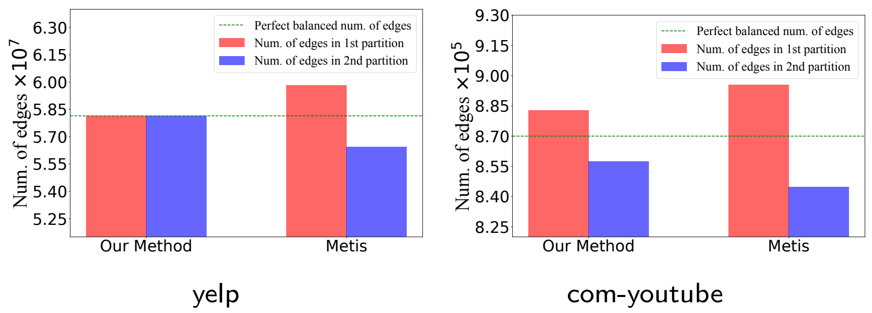


Figure 9: Number of edges in each partition.

A.5. Different Model Architectures

In this experiment, we design one variants of our method. The modifications are made on the *feature learning module* in our model. This variant is called '*Concatenate All*' which concatenates the three inputs in columns manner without the three MLP layers, and this is designed to see whether it is helpful learning features independently. The results are shown in Figure 10.

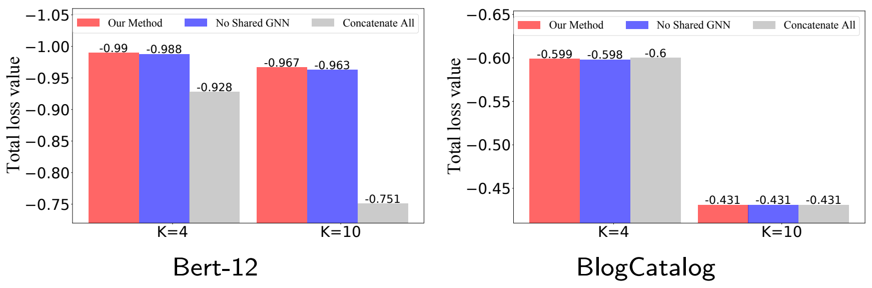


Figure 10: The bi-objective value L with different model architectures.

Figure 10 also shows, on the weighted dataset Bert-12, '*Concatenate All*' achieves the worse results, which indicates it is better the features are learned individually. For the unweighted dataset, BlogCatalog, the performances of our method and the variant are quite similar. In BlogCatalog, all values in S and all non-zero values in A are 1, and P also contains A's information, and the features are redundant. Learning these features may help to distinguish nodes, but may not be helpful for partitioning.

A.6. Evaluation on Different K values

To demonstrate the flexibility of our method with various K values, we performed partitioning with $K \in \{2, 3, 4, 5, 6, 7, 8, 9, 10, 32, 64, 128\}$ on the com-youtube graph. The results, compared with Metis and Scotch, are reported in Table 8. Our method effectively partitions graphs with different K values, whether even, odd, small, or large. Moreover, as K increases, the L value decreases, indicating that partitioning a graph into a larger number of balanced partitions with a low edge-cut becomes more challenging. Finally, across different K values, our method consistently achieves the lowest L values compared to the baselines, following the same trend on the com-youtube dataset observed in Section 4.2.1.

CRediT authorship contribution statement

Pengcheng Wei: Conceptualization of this study, Methodology, Writing - original draft. **Yuan Fang:** Funding acquisition, Investigation. **Zhihao Wen:** Investigation, Methodology. **Zheng Xiao:** Funding acquisition, Conceptualization of this study, Methodology. **Binbin Chen:** Methodology, Writing - review & editing.

Table 8
The L values on com-youtube graph.

Datasets	com-youtube											
Methods \ K	2	3	4	5	6	7	8	9	10	32	64	128
Metis	-0.8892	-0.8797	-0.8446	-0.8332	-0.8237	-0.8156	-0.7944	-0.7810	-0.7746	-0.6876	-0.6477	-0.6145
Scotch	-0.7950	-0.8297	-0.7699	-0.7951	-0.7694	-0.7683	0.7527	-0.7512	-0.7457	-0.6580	-0.6255	-0.6021
Our Method	**-0.9035**	**-0.8992**	**-0.8654**	**-0.8436**	**-0.8465**	**-0.8315**	**-0.8192**	**-0.8215**	**-0.8061**	**-0.7205**	**-0.6788**	**-0.6343**

- The best L is bolded;
- The second best L is underlined;

References

Baños, R., Gil, C., Montoya, M., Ortega, J., 2004. A new pareto-based algorithm for multi-objective graph partitioning, in: International Symposium on Computer and Information, Springer. pp. 779–788.

Bianchi, F.M., Grattarola, D., Alippi, C., 2020. Spectral clustering with graph neural networks for graph pooling, in: International Conference on Machine Learning, PMLR. pp. 874–883.

Bustany, I., Gasparyan, G., Kahng, A.B., Koutis, I., Pramanik, B., Wang, Z., 2023. An open-source constraints-driven general partitioning multi-tool for vlsi physical design, in: 2023 IEEE/ACM International Conference on Computer Aided Design, pp. 1–9.

Cai, Q., Guo, X., Huang, S., 2024. A self-supervised learning model for graph clustering optimization problems. Knowledge-Based Systems 290, 111549.

Çatalyürek, Ü., Devine, K., Faraj, M., Gottesbüren, L., Heuer, T., Meyerhenke, H., Sanders, P., Schlag, S., Schulz, C., Seemaier, D., et al., 2023. More recent advances in (hyper) graph partitioning. ACM Computing Surveys 55, 1–38.

Chakraborty, T., Dalmia, A., Mukherjee, A., Ganguly, N., 2017. Metrics for community analysis: A survey. ACM Computing Surveys (CSUR) 50, 1–37.

Chevalier, C., Pellegrini, F., 2008. Pt-scotch: A tool for efficient parallel graph ordering. Parallel Computing 34, 318–331.

Clauset, A., Newman, M.E., Moore, C., 2004. Finding community structure in very large networks. Physical Review 70, 06–11.

Datta, D., Figueira, J.R., 2011. Graph partitioning by multi-objective real-valued metaheuristics: A comparative study. Applied Soft Computing 11, 3976–3987.

Datta, D., Figueira, J.R., Fonseca, C.M., Tavares-Pereira, F., 2008. Graph partitioning through a multi-objective evolutionary algorithm: A preliminary study, in: Proceedings of the Conference on Genetic and Evolutionary Computation, ACM. pp. 625–632.

Deb, K., Pratap, A., Agarwal, S., Meyarivan, T., 2002. A fast and elitist multiobjective genetic algorithm: Nsga-ii. IEEE Transactions on Evolutionary Computation 6, 182–197.

Duval, A., Malliaros, F., 2022. Higher-order clustering and pooling for graph neural networks, in: Proceedings of the ACM International Conference on Information Knowledge Management, ACM. p. 426–435.

Dwivedi, V.P., Joshi, C.K., Luu, A.T., Laurent, T., Bengio, Y., Bresson, X., 2023. Benchmarking graph neural networks. Journal of Machine Learning Research 24, 1–48.

Farshbaf, M., Feizi-Derakhshi, M.R., 2009. Multi-objective optimization of graph partitioning using genetic algorithms, in: 2009 Third International Conference on Advanced Engineering Computing and Applications in Sciences, IEEE. pp. 1–6.

Fey, M., Lenssen, J.E., 2019. Fast graph representation learning with pytorch geometric. arXiv preprint arXiv:1903.02428 .

Gatti, A., Hu, Z., 2022. Graph partitioning and sparse matrix ordering using reinforcement learning and graph neural networks. Journal of Machine Learning Research 23, 1–28.

Gatti, A., Hu, Z., Smidt, T., Ng, E.G., Ghysels, P., 2022. Deep learning and spectral embedding for graph partitioning, in: Proceedings of the Conference on Parallel Processing for Scientific Computing, SIAM. pp. 25–36.

Girvan, M., Newman, M.E., 2002. Community structure in social and biological networks. Proceedings of the National Academy of Sciences 99, 7821–7826.

Gottesbren, L., Heuer, T., Sanders, P., Schulz, C., Seemaier, D., 2021. Deep multilevel graph partitioning. arXiv preprint arXiv:2105.02022 .

Hamilton, W., Ying, Z., Leskovec, J., 2017. Inductive representation learning on large graphs. Advances in Neural Information Processing Systems 30.

Hanai, M., Suzumura, T., Tan, W.J., Liu, E., Theodoropoulos, G., Cai, W., 2019. Distributed edge partitioning for trillion-edge graphs. Proceedings of the VLDB Endowment 12, 2379–2392.

He Li, Y.L., .etc, H.Y., 2022. Research on dynamic graph partitioning algorithms: A survey. Journal of Software 34, 539–564.

Jang, E., Gu, S., Poole, B., 2022. Categorical reparameterization with gumbel-softmax, in: International Conference on Learning Representations.

Karypis, G., Kumar, V., 1998a. A fast and high quality multilevel scheme for partitioning irregular graphs. Journal on Scientific Computing 20, 359–392.

Karypis, G., Kumar, V., 1998b. Multilevelk-way partitioning scheme for irregular graphs. Journal of Parallel and Distributed Computing 48, 96–129.

Liu, C., Wang, J., Cao, Y., Liu, M., Shen, W., 2022. Gon: End-to-end optimization framework for constraint graph optimization problems. Knowledge-Based Systems 254, 109697.

Ma, X., Sun, P., Qin, G., 2017. Identifying condition-specific modules by clustering multiple networks. IEEE/ACM Transactions on Computational Biology and Bioinformatics 15, 1636–1648.

Ma, X., Tang, W., Wang, P., Guo, X., Gao, L., 2016. Extracting stage-specific and dynamic modules through analyzing multiple networks associated with cancer progression. IEEE/ACM transactions on computational biology and bioinformatics 15, 647–658.

Ma, X., Zhang, B., Ma, C., Ma, Z., 2020. Co-regularized nonnegative matrix factorization for evolving community detection in dynamic networks. Information Sciences 528, 265–279.

Ma, X., Zhao, W., Wu, W., 2022. Layer-specific modules detection in cancer multi-layer networks. IEEE/ACM Transactions on Computational Biology and Bioinformatics 20, 1170–1179.

Nazi, A., Hang, W., Goldie, A., Ravi, S., Mirhoseini, A., 2019. Gap: Generalizable approximate graph partitioning framework. arXiv preprint arXiv:1903.00614 .

Ni, X., Li, J., Yu, M., Zhou, W., Wu, K.L., 2020. Generalizable resource allocation in stream processing via deep reinforcement learning, in: Proceedings of the AAAI Conference on Artificial Intelligence, pp. 857–864.

Paszke, A., Gross, S., Massa, F., Lerer, A., Bradbury, J., Chanan, G., Killeen, T., Lin, Z., Gimelshein, N., Antiga, L., et al., 2019. Pytorch: An imperative style, high-performance deep learning library. Advances in Neural Information Processing Systems 32.

Sanders, P., Schulz, C., 2013. Think locally, act globally: Highly balanced graph partitioning, in: The 12th International Symposium on Experimental Algorithms, Springer. pp. 164–175.

Shao, Y., Li, H., Gu, X., Yin, H., Li, Y., Miao, X., Zhang, W., Cui, B., Chen, L., 2024. Distributed graph neural network training: A survey. ACM Computing Surveys 56, 1–39.

Shao, Y., Wang, J., Zhao, Y., Ma, Y., Liu, M., 2023. A tiny graph neural network for inverse graph partitioning with imbalance constraints, in: 2023 IEEE 19th International Conference on Automation Science and Engineering, IEEE. pp. 1–6.

Shi, 2003. Multiclass spectral clustering, in: Proceedings of the IEEE International Conference on Computer Vision, IEEE. pp. 313–319.

Shi, J., Malik, J., 2000. Normalized cuts and image segmentation. IEEE Transactions on Pattern Analysis and Machine Intelligence 22, 888–905.

Stanton, I., Kliot, G., 2012. Streaming graph partitioning for large distributed graphs, in: Proceedings of the 18th ACM SIGKDD International Conference on Knowledge Discovery and Data mining, pp. 1222–1230.

Tanaka, M., Taura, K., Hanawa, T., Torisawa, K., 2021. Automatic graph partitioning for very large-scale deep learning, in: IEEE International Parallel and Distributed Processing Symposium, IEEE. pp. 1004–1013.

Tarnawski, J.M., Phanishayee, A., Devanur, N., Mahajan, D., Nina Paravecino, F., 2020. Efficient algorithms for device placement of dnn graph operators. Advances in Neural Information Processing Systems 33, 15451–15463.

Traag, V.A., Waltman, L., Van Eck, N.J., 2019. From louvain to leiden: guaranteeing well-connected communities. Scientific reports 9, 5233.

Tsitsulin, A., Palowitch, J., Perozzi, B., Müller, E., 2023. Graph clustering with graph neural networks. Journal of Machine Learning Research 24, 1–21.

Veličković, P., Cucurull, G., Casanova, A., Romero, A., Liò, P., Bengio, Y., 2018. Graph attention networks, in: International Conference on Learning Representations.

Wang, C., Pan, S., Hu, R., Long, G., Jiang, J., Zhang, C., 2019. Attributed graph clustering: a deep attentional embedding approach, in: Proceedings of the 28th International Joint Conference on Artificial Intelligence, pp. 3670–3676.

Williams, R.J., 1992. Simple statistical gradient-following algorithms for connectionist reinforcement learning. Reinforcement Learning , 5–32.

Wu, W., Liu, Z., Ma, X., 2021. jsrc: a flexible and accurate joint learning algorithm for clustering of single-cell rna-sequencing data. Briefings in bioinformatics 22, bbaa433.

Xu, K., Hu, W., Leskovec, J., Jegelka, S., 2018. How powerful are graph neural networks?, in: International Conference on Learning Representations.

Zhou, Z., Liu, Y., Ding, J., Jin, D., Li, Y., 2023. Hierarchical knowledge graph learning enabled socioeconomic indicator prediction in location-based social network, in: Proceedings of the ACM Web Conference 2023, pp. 122–132.