A DEEP LEARNING FRAMEWORK FOR GRAPH PARTITIONING

Azade Nazi[†], **Will Hang**[††], **Anna Goldie**[†], **Sujith Ravi**[‡] **& Azalia Mirhoseini**[†]
[†]Google Brain; [‡]Google Research; [††]Stanford University
[†]{azade, agoldie, sravi, azalia}@google.com
[††]{willhang}@stanford.edu

ABSTRACT

Graph partitioning is the problem of dividing the nodes of a graph into balanced partitions while minimizing the edge cut across the partitions. Due to its combinatorial nature, many approximate solutions have been developed. We propose GAP, a *Generalizable Approximate Partitioning* framework that takes a deep learning approach to graph partitioning. We define a differentiable loss function that represents the partitioning objective. Unlike baselines that redo the optimization per graph, GAP is capable of generalization, allowing us to train models that produce performant partitions at inference time, even on unseen graphs. Furthermore, because we learn the representation of the graph while jointly optimizing for the partitioning loss function, GAP can be easily tuned for a variety of graph structures. We evaluate the performance of GAP on graphs of varying sizes and structures, including graphs of widely used machine learning models (e.g., ResNet, VGG, and Inception-V3), scale-free graphs, and random graphs.

1 INTRODUCTION

Graph partitioning is an important optimization problem with numerous applications in domains spanning computer vision, VLSI design, biology, social networks, transportation networks, device placement and more. The objective is to find balanced partitions while minimizing the number of edge cut. For example, in device placement, the goal is to evenly partition a computational graph and distribute the partitions across multiple devices (such as CPUs and GPUs), such that the total number of connections across devices is minimized. This problem is NP-complete, which is formulated as a discrete optimization problem, and solutions are generally derived using heuristics and approximation algorithms (Miettinen et al., 2006), (Andersen et al., 2006), (Chung, 2007), (Karypis & Kumar, 1998), (Karypis et al., 1999), (Karypis & Kumar, 2000), (Van Den Bout & Miller, 1990), (Kawamoto et al., 2018). Clustering problems with self-balanced k-means and balanced min-cut objectives have been exhaustively studied (Liu et al., 2017), (Chen et al., 2017), (Chang et al., 2014). One of the most effective techniques is spectral clustering, which finds node embeddings in the eigenspace of the graph Laplacian, and then applies k-means clustering to these vectors (Shi & Malik, 2000; Ng et al., 2002; Von Luxburg, 2007).

In this work, we introduce a learning based approach, GAP, for the continuous relaxation of the problem. We define a differentiable loss function which captures the objective of partitioning a graph into disjoint balanced partitions while minimizing the number of edge cut across those partitions. We train a deep model to optimize for this loss function. The optimization is done in an unsupervised manner without the need for labeled datasets. Our approach, GAP, does not assume anything about the graph structure (e.g., sparse vs. dense, or scale-free). Instead, GAP learns and adapts to the graph structure using graph embedding techniques while optimizing the partitioning loss function. This representation learning allows our approach to be self-adaptive without the need for us to design different strategies for different types of graphs.

Our learning based approach is also capable of generalization, meaning that we can train a model on a set of graphs and then use it at inference time on unseen graphs of varying sizes. In particular, we show that when GAP is trained on smaller graphs (e.g., 1k nodes), it transfers what it learned to much larger ones (e.g, 20k nodes). This generalization allows trained GAP models to quickly infer

partitions on large unseen graphs, whereas baseline methods have to redo the entire optimization for each new graph.

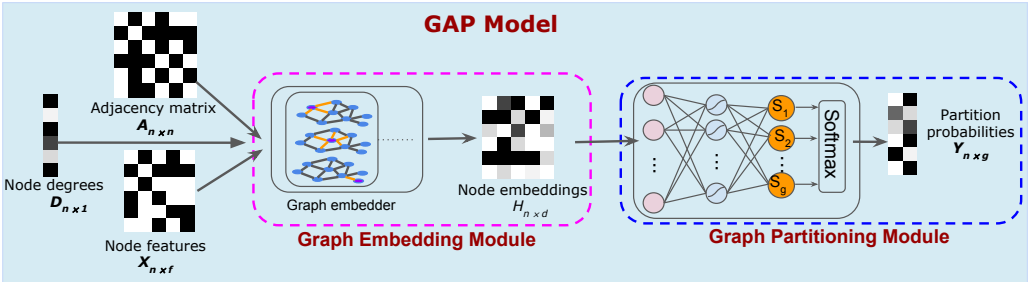

Figure 1: Generalizable Approximate graph Partitioning (GAP) (more details in Section 2).

2 Generalizable Approximate Partitioning

Let $G = (V, E)$ be a graph where $V = \{v_i\}$ and $E = \{e(v_i, v_j) | v_i \in V, v_j \in V\}$ are the set of nodes and edges in the graph. Let n be the number of nodes. Normalized cut (*Ncut*), which is based on the graph conductance, has been studied by (Shi & Malik, 2000; Zhang & Rohe, 2018), where the cost of a cut is computed as a fraction of the total edge connections to all nodes. $vol(S_k, V) = \sum_{v_i \in S_k, v_j \in V} e(v_i, v_j)$ is the total degree of nodes belong to S_k in graph G.

$$Ncut(S_1, S_2, \dots S_g) = \sum_{k=1}^{g} \frac{cut(S_k, \bar{S}_k)}{vol(S_k, V)} \tag{1}$$

One way to minimize the normalized cut is based on the eigenvectors of the graph Laplacian which has been studied in (Shi & Malik, 2000; Zhang & Rohe, 2018). In this paper, we introduce the *Generalizable Approximate Partitioning* framework (GAP). As shown in Figure 1, GAP has two main components: graph embedding module, and graph partitioning module. The GAP model ingests a graph definition, generates node embeddings that leverage local graph structure, and projects each embedding into logits that define a probability distribution to minimize the expected normalized cut (Equation 5). The purpose of the graph embedding module is to learn node embeddings using the graph structure and node features. Recently, there have been several advances on applying graph neural networks for node embedding and classification tasks using approaches such as Graph Convolution Network (Kipf & Welling, 2017) (GCN), GraphSAGE (Hamilton et al., 2017), Neural Graph Machines (Bui et al., 2017), Graph Attention Networks (Veličković et al., 2018) and other variants. In this work, we leverage GCN and GraphSAGE to learn graph representations across a variety of graphs, which helps with generalization. The second module in our GAP framework is responsible for partitioning the graph, taking in node embeddings and generating the probability that each node belongs to partitions $S_1, S_2, ..., S_g$ (Y in Figure 1). This module is a fully connected layer followed by softmax, trained to minimize GAP loss function introduced in Equation 5.

2.1 GAP Loss Function

As shown in Figure 1 GAP returns $Y \in \mathbb{R}^{n \times g}$ where Y_{ik} represents the probability that node v_i belongs to partition S_k. The probability that node v_j does not belong to partition S_k would be $1 - Y_{jk}$. We formulate $\mathbb{E}[cut(S_k, \bar{S}_k)]$ by Equation 2, where $\mathcal{N}(v_i)$ is the set of nodes adjacent to v_i.

$$\mathbb{E}[cut(S_k, \bar{S}_k)] = \sum_{\substack{v_i \in S_k \\ v_j \in \mathcal{N}(v_i)}} \sum_{z=1}^{g} Y_{iz}(1 - Y_{jz}) \tag{2}$$

Since the set of adjacent nodes for a given node can be retrieved from the adjacency matrix of graph A, we can rewrite Equation 2 as follows:

$$\mathbb{E}[cut(S_k, \bar{S}_k)] = \sum_{\text{reduce-sum}} Y_{:,k}(1 - Y_{:,k})^\intercal \odot A \tag{3}$$

From Equation 1, $vol(S_k, V)$ is the sum over the degree of all nodes that belong to S_k. Let D be a column vector of size n where D_i is the degree of the node $v_i \in V$. Given Y, we can calculate the $\mathbb{E}[vol(S_k, V)]$ as follows, where Γ is a vector in $\mathbb{R}^g$, and g is the number of partitions.

$$\Gamma = Y^\intercal D, \ where \ \mathbb{E}[vol(S_k, V)] = \Gamma_k \tag{4}$$

With $\mathbb{E}[cut(S_k, \bar{S}_k)]$ and $\mathbb{E}[vol(S_k, V)]$ from Equations 3 and 4, we can calculate the expected normalized cut in Equation 1 as follows ($\oslash$ is element-wise division):

$$\mathcal{L} = \mathbb{E}[Ncut(S_1, S_2, \dots S_g)] = \sum_{\text{reduce-sum}} (Y \oslash \Gamma)(1 - Y)^\intercal \odot A \tag{5}$$

3 Experimental Results

The main goals of our experiments are to (a) evaluate the performance of the GAP framework against hMETIS (Karypis & Kumar, 2000), a widely used partitioner that uses multilevel partitioning and (b) evaluate the generalizability of GAP over unseen graphs and provide insights on how the structural similarities between train and test graphs affect the generalization performance.

3.1 Setup

We conducted experiments on real and synthetic graphs. Specifically, we use five widely used TensorFlow graphs, i.e. *ResNet*, *Inception-v3*, *AlexNet*, *MNIST-conv*, and *MNIST-conv*.

- *ResNet* (He et al., 2016) is a deep convolutional network with residual connections. The TensorFlow implementation of *ResNet_v1_50* with 50 layers contains $20,586$ operations.
- *Inception-v3* (Szegedy et al., 2017) consists of multiple blocks, each composed of several convolutional and pooling layers. The TensorFlow graph of this model contains $27,114$ operations.
- *AlexNet* (Krizhevsky et al., 2012) consists of 5 convolutional layers, some of which are followed by max-pooling layers, and 3 fully-connected layers with a final softmax. The TensorFlow graph of this model has 798 operations.
- *MNIST-conv* has 3 convolutional layers for the *MNIST* classification task. The TensorFlow graph of this model contains 414 operations.
- *VGG* (Simonyan & Zisserman, 2014) contains 16 convolutional layers. The TensorFlow graph of *VGG* contains $1,325$ operations.

We also generate *Random* and *Scale-free* graphs as synthetic datasets to show the effectiveness of GAP on graphs with different structures. We generate random graphs using the *Erdös–Rényi* model (Erdos & Rényi, 1960), where the probability of having an edge between any two nodes is 0.1. A scale-free network is a network whose degree distribution follows a power law (Bollobás et al., 2003). We use NetworkX (Hagberg et al., 2008) to generate such graphs.

We compare GAP against hMETIS (Karypis & Kumar, 2000) which is a general framework and shown to provide high quality partitions in different domains (e.g., VLSI, road network (Miettinen et al., 2006; Xu & Tan, 2012). We set the hMETIS parameters to return balanced partitions with minimum edge cut. Moreover, since hMETIS is a randomized algorithm, we pick its best result over 100 iterations. We evaluate the performance of the resulting partitions by examining 1) *Edge cut (Cut)*: the ratio of the cut to the total number of edges, and 2) *Balancedness (Bal)*: is one minus the MSE of number of nodes in every partition and balances partition ($\frac{n}{g}$).

3.2 Performance

Since hMETIS does not generalize to unseen graphs and optimizes one graph at a time, we also constrain GAP to optimize one graph at a time for a fair comparison. We discuss the generalization ability of GAP in Section 3.3. Table 1 shows the performance of GAP against hMETIS on a 3-partition problem over real TensorFlow graphs. Both techniques generate very balanced partitions, with GAP outperforming hMETIS on edge cut for the VGG graph.

Computation graphs	**hMETIS**		**GAP**	
Name	Cut	Bal	Cut	Bal
VGG	0.052	0.999	0.046	0.998
MNIST-conv	0.054	0.998	0.057	0.999
ResNet	0.044	0.999	0.043	0.998
AlexNet	0.052	0.998	0.053	0.998
Inception-v3	0.047	0.999	0.046	0.998

Table 1: Performance of GAP against hMETIS over a given graph. For edge cut lower is better, for balancedness (Bal) higher is better.

	AlexNet		***Inception-v3***		***ResNet***	
Embedding	Cut	Bal	Cut	Bal	Cut	Bal
No embedding	0.166	0.717	0.242	0.740	0.450	0.908
GCN offline	0.078	**0.996**	0.123	0.984	0.110	0.941
GraphSAGE offline	0.070	**0.998**	0.088	**0.992**	0.093	0.959
GraphSAGE online	**0.069**	**0.998**	**0.064**	**0.987**	**0.084**	**0.983**

Table 2: Generalization results: GAP is trained on *VGG* and validated on *MNIST-conv*. During inference, the model is applied to unseen TensorFlow graphs: *ResNet*. *Inception-v3*, and *AlexNet*. According to Table 1, the ground truth for *VGG*, *MNIST-conv*, and *AlexNet* is 99% balanced partitions with 5% edge cut and for *ResNet* and *Inception-v3*, it is 99% balanced partitions with 4% edge cut. *GraphSAGE online* generalizes better than the other models.

3.3 GENERALIZATION

In this section, we show that GAP generalizes effectively on real and synthetic datasets. To the best of our knowledge, we are the first to propose a learning approach for graph partitioning that can generalize to unseen graphs.

3.3.1 GENERALIZATION ON REAL GRAPHS

In this set of experiments, we train GAP with a single TensorFlow graph, *VGG*, and validate on *MNIST-conv*. At inference time, we test the trained model on unseen TensorFlow graphs: *AlexNet*, *ResNet*, and *Inception-v3*. Table 2 shows the result of our experiments, and illustrates the importance of the graph embeddings in generalization. The operation type (such as Add, Conv2d, and L2loss in TensorFlow) is used as the node feature. We encode all features as one-hots. Following Section 2, we leverage GCN (Kipf & Welling, 2017) and GraphSAGE (Hamilton et al., 2017) to capture similarities across graphs. In *GCN offline* and *GraphSAGE offline*, we do not train the graph embedding module (Figure 1) without gradient flow from the partitioning module, while in *GraphSAGE online* both modules are trained jointly. Table 2 shows that the *GraphSAGE online* (last row) achieves the best performance and generalizes better than the other models. Note that this model is trained on a single graph, *VGG* with only 1325 nodes, and it is tested on *AlexNet*, *ResNet*, and *Inception-v3* with 798, 20586, and 27114 nodes, respectively.

Model Architecture and Hyper-parameters: Here, we describe the details of the model with the best performance (corresponding to the last row of Table 2). The number of features (TensorFlow operation types) is 1518. GraphSAGE has 5 layers of 512 units with shared pooling, and the graph partitioning module is a 3 layer dense network of 64 units with a final softmax layer. We use ReLU as activation function and all weights are initialized using Xavier initialization Glorot & Bengio (2010). We use the Adam optimizer with a learning rate of 7.5e-5.

3.3.2 GENERALIZATION ON SYNTHETIC GRAPHS

We further evaluate the generalization of GAP on *Random* and *Scale-free* graphs. Note that we train and test GAP on the same type of graph, but number of nodes may vary. For example, we train GAP on random graphs of 1k nodes and test on random graphs of 1k and 10k nodes (50k and 5M edges respectively). Similarly, we train GAP on scale-free graphs of 1k nodes and test on scale-free graphs of 1k and 10k nodes (2.5k and 21k edges respectively).

Table 3 shows the edge cut and balancedness of GAP against hMETIS on random and scale-free graphs. The results are the average over the 5 graphs of 1k and 10k nodes. For a given type of graph (random or scale-free), we train GAP with ten graphs and we then test the trained models on 5 unseen graphs of 1k and 10k nodes and we report the average results. Note that we could train with more graphs; however, even the results by training on 10 graphs is comparable with hMETIS. On the other hand, since hMETIS is a randomized algorithm, we run it 100 times for each graph and we pick the best result. As shown in Table 3, GAP provides high quality partitions over the unseen graphs.

Model Architecture and Hyper-parameters: Unlike computation graphs where node features are operation types, nodes in synthetic graphs have no features. Furthermore, we must train a model that generalizes to graphs of different sizes. For example, we train a model on a random graph with 1k nodes and test it on a random graph with 10k nodes. To do so, we apply PCA to the adjacency matrix of a featureless graph and retrieve embeddings of size 1000 as our node features. We use ReLU as our activation function and all weights are initialized using Xavier initialization. We also use the Adam optimizer. Here are the rest of the hyperparameters for each model.

GAP-Scalefree: Trained with 10 scale-free graphs. GraphSAGE has 4 layers of 128 units, and graph partitioning module is 1 layer dense network of 64 units with softmax. Learning rate is 7.5e-6.

GAP-Random: Trained with 10 random graphs. GraphSAGE has 2 layers of 256 units with shared pooling, and graph partitioning module is 3 layer dense network of 128 units with softmax. Learning rate is 7.5e-6.

	Random-1k		***Random-10k***		***Scalefree-1k***		***Scalefree-10k***	
	Cut	Bal	Cut	Bal	Cut	Bal	Cut	Bal
GAP	**0.654 ±9.4E-3**	0.983 ±1.3E-2	**0.654 ±8.9E-3**	**0.985 ±8.1E-3**	**0.439 ±6.5E-2**	0.938 ±5.3E-2	**0.411 ±9.2E-2**	0.979 ±5.7E-2
hMETIS	0.67 ±2.1E-2	0.99 ±4.6E-5	0.666 ±1.1E-2	0.984 ±2.5E-5	0.496 ±2.0E-2	0.968 ±5.3E-3	0.505 ±3.2E-2	0.99 ±1.3E-5

Table 3: Generalization results on synthetic graphs: GAP is trained on random graphs of 1k nodes and test on new random graphs of 1k and 10k nodes. Similarly, we train GAP on scale-free graphs of 1k nodes and test on new scale-free graphs of 1k and 10k nodes. The results are the average over the 5 graphs of 1k and 10k nodes. The results for hMETIS is the best among 100 runs on each graph.

4 CONCLUSION

We propose a deep learning framework, GAP, for the graph partitioning problem. Our GAP framework enables generalization: we can train models that produce performant partitions at inference time, even on unseen graphs. This generalization is an advantage over existing baselines which redo the optimization for each new graph. Our results over widely used machine learning models (ResNet, VGG, and Inception-v3), scale-free graphs, and random graphs confirm that GAP achieves high quality partitions even on unseen graphs.

REFERENCES

Reid Andersen, Fan Chung, and Kevin Lang. Local graph partitioning using pagerank vectors. In *FOCS*, pp. 475–486. IEEE, 2006.

Béla Bollobás, Christian Borgs, Jennifer Chayes, and Oliver Riordan. Directed scale-free graphs. In *Proceedings of the Fourteenth Annual ACM-SIAM Symposium on Discrete Algorithms*, SODA '03, pp. 132–139, 2003. ISBN 0-89871-538-5.

Thang D. Bui, Sujith Ravi, and Vivek Ramavajjala. Neural graph machines: Learning neural networks using graphs. *CoRR*, abs/1703.04818, 2017.

Xiaojun Chang, Feiping Nie, Zhigang Ma, and Yi Yang. Balanced k-means and min-cut clustering. *arXiv preprint arXiv:1411.6235*, 2014.

Xiaojun Chen, Joshua Zhexue Huang, Feiping Nie, Renjie Chen, and Qingyao Wu. A self-balanced min-cut algorithm for image clustering. In *ICCV*, pp. 2080–2088, 2017.

Fan Chung. Four proofs for the cheeger inequality and graph partition algorithms. In *Proceedings of ICCM*, volume 2, pp. 378, 2007.

Paul Erdos and Alfréd Rényi. On the evolution of random graphs. *Publ. Math. Inst. Hung. Acad. Sci*, 5(1):17–60, 1960.

Xavier Glorot and Yoshua Bengio. Understanding the difficulty of training deep feedforward neural networks. In Yee Whye Teh and Mike Titterington (eds.), *AISTATS*, volume 9, pp. 249–256, Chia Laguna Resort, Sardinia, Italy, 13–15 May 2010. PMLR.

Aric Hagberg, Pieter Swart, and Daniel S Chult. Exploring network structure, dynamics, and function using networkx. Technical report, Los Alamos National Lab.(LANL), Los Alamos, NM (United States), 2008.

William L. Hamilton, Zhitao Ying, and Jure Leskovec. Inductive representation learning on large graphs. In *Advances in Neural Information Processing Systems 30: Annual Conference on Neural Information Processing Systems 2017, 4-9 December 2017, Long Beach, CA, USA*, pp. 1025–1035, 2017.

Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 770–778, 2016.

George Karypis and Vipin Kumar. Multilevelk-way partitioning scheme for irregular graphs. *Journal of Parallel and Distributed computing*, 48(1):96–129, 1998.

George Karypis and Vipin Kumar. Multilevel k-way hypergraph partitioning. *VLSI design*, 11(3): 285–300, 2000.

George Karypis, Rajat Aggarwal, Vipin Kumar, and Shashi Shekhar. Multilevel hypergraph partitioning: applications in vlsi domain. *IEEE T VLSI SYST*, 7(1):69–79, 1999.

Tatsuro Kawamoto, Masashi Tsubaki, and Tomoyuki Obuchi. Mean-field theory of graph neural networks in graph partitioning. In *Advances in Neural Information Processing Systems*, pp. 4366–4376, 2018.

Thomas N Kipf and Max Welling. Semi-supervised classification with graph convolutional networks. In *ICLR*, 2017.

Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. In *Advances in neural information processing systems*, pp. 1097–1105, 2012.

Hanyang Liu, Junwei Han, Feiping Nie, and Xuelong Li. Balanced clustering with least square regression. In *AAAI*, pp. 2231–2237, 2017.

Pekka Miettinen, Mikko Honkala, and Janne Roos. *Using METIS and hMETIS algorithms in circuit partitioning*. Helsinki University of Technology, 2006.

Andrew Y Ng, Michael I Jordan, and Yair Weiss. On spectral clustering: Analysis and an algorithm. In *Advances in neural information processing systems*, pp. 849–856, 2002.

Jianbo Shi and Jitendra Malik. Normalized cuts and image segmentation. *IEEE Trans. Pattern Anal. Mach. Intell.*, 22(8):888–905, August 2000.

Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556*, 2014.

Christian Szegedy, Sergey Ioffe, Vincent Vanhoucke, and Alexander A Alemi. Inception-v4, inception-resnet and the impact of residual connections on learning. In *AAAI*, volume 4, pp. 12, 2017.

David E Van Den Bout and Thomas K Miller. Graph partitioning using annealed neural networks. *IEEE Transactions on neural networks*, 1(2):192–203, 1990.

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

Ulrike Von Luxburg. A tutorial on spectral clustering. *Statistics and computing*, 17(4):395–416, 2007.

Yan Xu and Gary Tan. hmetis-based offline road network partitioning. In *AsiaSim 2012*, pp. 221–229. Springer, 2012.

Yilin Zhang and Karl Rohe. Understanding regularized spectral clustering via graph conductance. In *NeurIPS*, pp. 10654–10663, 2018.