GAP: Generalizable Approximate Graph Partitioning Framework

Azade Nazi[†*], Will Hang[††], Anna Goldie[†], Sujith Ravi[†], Azalia Mirhoseini[†]
[†]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, including variants of multi-level methods and spectral clustering. 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 and use backpropagation to optimize the network parameters. 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. We show that GAP achieves competitive partitions while being up to 100 times faster than the baseline and generalizes to unseen 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 and more. The objective is to find balanced partitions of a graph while minimizing the number of edge cut. This problem is NP-complete which is formulated as a discrete optimization problem and solutions are generally derived using heuristics and approximation algorithms. Some notable approaches include multi-level methods and spectral partitioning methods [Karypis and Kumar, 1998, Karypis et al., 1999, Karypis and Kumar, 2000, Miettinen et al., 2006, Andersen et al., 2006, Chung, 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

[*]Members of the Google Brain Residency Program

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

In summary, this paper makes the following contributions:

- We propose GAP, a *Generalizable Approximate Partitioning* framework, which is an unsupervised learning approach to the classic problem of balanced graph partitioning. We define a differentiable loss function for partitioning that uses a continuous relaxation of the normalized cut. We then train a deep model and apply backpropagation to optimize the loss.
- GAP models can produce efficient partitions on unseen graphs at inference time. Generalization is an advantage over existing approaches which must redo the entire optimization for each new graph.
- GAP leverages graph embedding techniques [Kipf and Welling, 2017, Hamilton et al., 2017] and learns to partition graphs based on their underlying structure, allowing it to generate efficient partitions across a wide variety of graphs.
- To encourage reproducible research, we provide source code in the supplementary materials and are in the process of open-sourcing the framework.
- We show that GAP achieves competitive partitions while being up to 100 times faster than top performing baselines on a variety of synthetic and real-world graphs with up to 27000 nodes.

2 Related Work

Graph Partitioning: Graph partitioning is an important combinatorial optimization problem that has been exhaustively studied. The most widely used graph partitioning algorithms generate partitions by performing operations on the input graph until convergence [Andersen et al., 2006, Chung, 2007]. On the other hand, multilevel partitioning approaches first reduce the size of the graph by collapsing nodes and edges, then partition on the smaller graph, and finally expand the graph to recover the partitioning for the original graph [Karypis and Kumar, 2000, Karypis et al., 1999, Karypis and Kumar, 1998, Miettinen et al., 2006]. These algorithms are shown to provide high-quality partitions [Miettinen et al., 2006].

Another approach is to use simulated annealing. [Van Den Bout and Miller, 1990] proposed mean field annealing, which combines simulated annealing with Hopfield neural networks. [Kawamoto et al., 2018] studied a different formulation of graph partitioning in which a graph is generated by a statistical model, and the task is to infer the preassigned group labels of the generative model. They developed a mean-field theory of a minimal graph neural network architecture for this version of the problem.

This line of inquiry formulates graph partitioning as a discrete optimization problem, while our GAP framework is one of the first deep learning approaches for the continuous relaxation of the problem. Moreover, GAP generalizes to unseen graphs, generating partitions on the fly, rather than having to redo the optimization per graph.

Clustering: Given a set of points, the goal of clustering is to identify groups of similar points. Clustering problems with different objectives such as self-balanced k-means and balanced min-cut have been exhaustively studied [Liu et al., 2017, Chen et al., 2017, Chang et al., 2014]. One of the most effective techniques for clustering is spectral clustering, which first generates node embeddings in the eigenspace of the graph Laplacian, and then applies k-means clustering to these vectors [Shi and Malik, 2000, Ng et al., 2002, Von Luxburg, 2007].

However, generalizing clustering to unseen nodes and graphs is nontrivial. To address generalization, SpectralNet [Shaham et al., 2018] is a deep learning approach to spectral clustering which generates spectral embeddings for unseen data points. Other deep learning approaches for clustering attempt to encode the input in a way that is amenable to clustering by k-means or Gaussian Mixture Models [Yang et al., 2017, Xie et al., 2016, Zheng et al., 2016, Dilokthanakul et al., 2016].

Although related, graph clustering and graph partitioning are different problems in that graph clustering attempts to maximize locality of clusters, whereas graph partitioning seeks to preserve locality while maintaining balance among partitions. Our approach also treats the partitioning problem as an end-to-end learning problem with a differentiable loss, whereas the aforementioned approaches generate embeddings that are then clustered using non-differentiable techniques like k-means.

Device Placement: The practical significance of graph partitioning is demonstrated by the task of device placement for TensorFlow computation graphs, where the objective is to minimize execution time by assigning operations to devices. [Mirhoseini et al., 2017] proposed a reinforcement learning method to optimize device placement for TensorFlow graphs. They used a seq2seq policy to assign operations to devices. The execution time of the generated placements is then used as a reward signal to optimize the policy. A hierarchical model for device placement has been proposed in [Mirhoseini et al., 2018], where the graph partition and placement are learned jointly. While this work also uses a neural network to learn the partitions, their objective is to optimize the runtime of the resulting partitions, forcing them to use policy gradient to optimize their non-differentiable loss function.

3 Problem Definition and Background

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. A graph G can be partitioned into g disjoint sets $S_1, S_2, \dots S_g$, where the union of the nodes in those sets are V ($\bigcup_{k=1}^{g} S_k = V$), and each node belongs to only one set ($\bigcap_{k=1}^{g} S_k = \emptyset$), by simply removing edges connecting those sets.

Minimum Cut: The total number of edges that are removed from G in order to form disjoint sets is called *cut*. Given sets S_k, and $\bar{S_k}$, the $cut(S_k, \bar{S_k})$ is formally defined as:

$$cut(S_k, \bar{S_k}) = \sum_{v_i \in S_k, v_j \in \bar{S_k}} e(v_i, v_j) \tag{1}$$

This formula can be generalized to multiple disjoint sets $S_1, S_2, \dots S_g$, where $\bar{S_k}$ is the union of all sets except S_k.

$$cut(S_1, S_2, \dots S_g) = \frac{1}{2} \sum_{i=k}^{g} cut(S_k, \bar{S_k}) \tag{2}$$

Normalized Cut: The optimal partitioning of a graph that minimizes the cut (Equation 2) is a well-studied problem and there exist efficient polynomial algorithms for solving it [Papadimitriou and Steiglitz, 1982]. However, the minimum cut criteria favors cutting nodes whose degree are small and leads to unbalanced sets/partitions. To avoid such bias, normalized cut (*Ncut*), which is based on the graph conductance, has been studied by [Shi and Malik, 2000, Zhang and Rohe, 2018], where the cost of a cut is computed as a fraction of the total edge connections to all nodes.

$$Ncut(S_1, S_2, \dots S_g) = \sum_{k=1}^{g} \frac{cut(S_k, \bar{S_k})}{vol(S_k, V)} \tag{3}$$

Where $vol(S_k, V) = \sum_{v_i \in S_k, v_j \in V} e(v_i, v_j)$, i.e., total degree of nodes belong to S_k in graph G.

One way to minimize the normalized cut is based on the eigenvectors of the graph Laplacian which has been studied in [Shi and Malik, 2000, Zhang and Rohe, 2018]. Previous research has shown that across a wide range of social and information networks, the clusters with the smallest graph conductance are often small [Leskovec, 2009, Zhang and Rohe, 2018]. Regularized spectral clustering has been proposed by [Zhang and Rohe, 2018] to address this problem.

In this paper, however, we propose GAP as an unsupervised learning approach with a differentiable loss function that can be trained to find balanced partitions with minimum normalized cuts. We show that GAP enables generalization to unseen graphs.

4 Generalizable Approximate Partitioning

We now introduce the *Generalizable Approximate Partitioning* framework (GAP). As shown in Figure 1,

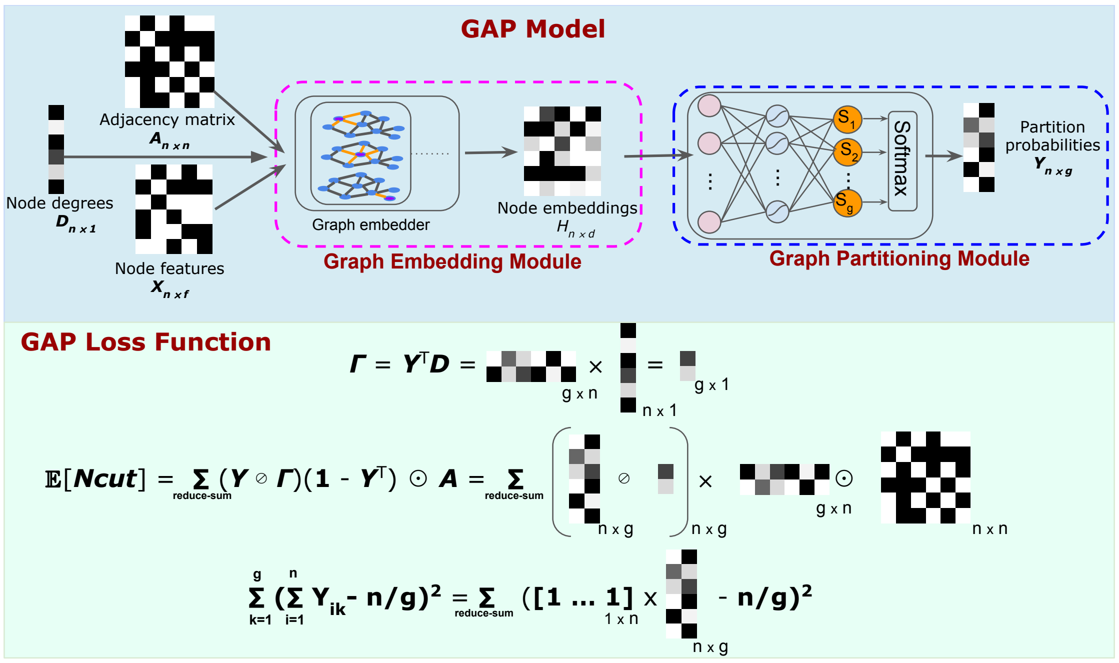

GAP Model
Adjacency matrix $A_{n\times n}$
Node degrees $D_{n\times 1}$
Node features $X_{n\times f}$
Graph embedder
Node embeddings $H_{n\times d}$
Graph Embedding Module
S_1 S_2 S_g
Softmax
Partition probabilities $Y_{n\times g}$
Graph Partitioning Module
GAP Loss Function
$\Gamma = Y^\top D$ = g x n × n x 1 = g x 1
$\mathbb{E}[Ncut] = \sum_{\text{reduce-sum}} (Y \oslash \Gamma)(1 - Y^\top) \odot A = \sum_{\text{reduce-sum}}$ n x g ⊘ n x g × g x n ⊙ n x n
$\sum_{k=1}^{g} (\sum_{i=1}^{n} Y_{ik} - n/g)^2 = \sum_{\text{reduce-sum}} ([1 \dots 1] \times - n/g)^2$ 1 x n n x g

Figure 1: Generalizable Approximate graph Partitioning (GAP) Framework (see Section 4 for more details).

GAP has two main components: graph representation learning for generating partition probabilities per node (the model), and a differentiable formulation of the normalized cut objective (the loss function). GAP enables us to train a neural network to optimize a previously undifferentiable objective by generating balanced partitions with minimum edge-cut. We first present the loss function before discussing the model.

4.1 GAP Loss Function

We assume that our model returns $Y \in \mathbb{R}^{n\times g}$ where Y_{ik} represents the probability that node $v_i \in V$ belongs to partition S_k. We propose a loss function based on Y to calculate the normalized cut in Equation 3 and evaluate the balancedness of the partitions. Later in subsection 4.2, we discuss the model that generates Y.

Normalized Cut: As we discussed in Section 3, $cut(S_k, \bar{S}_k)$ is the number of edges $e(v_i, v_j)$, where $v_i \in S_k$ and $v_j \notin S_k$. Let Y_{ik} be 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 $1-Y_{jk}$. Therefore, $\mathbb{E}[cut(S_k, \bar{S}_k)]$ can be formulated by Equation 4, where $\mathcal{N}(v_i)$ is the set of nodes adjacent to v_i (visual illustration in Figure 1).

$$\mathbb{E}[cut(S_k, \bar{S}_k)] = \sum_{\substack{v_i \in S_k \\ v_j \in \mathcal{N}(v_i)}} \sum_{z=1}^{g} Y_{iz}(1 - Y_{jz}) \tag{4}$$

Since the set of adjacent nodes for a given node can be retrieved from the adjacency matrix of graph A, we can rewrite Equation 4 as follows:

$$\mathbb{E}[cut(S_k, \bar{S}_k)] = \sum_{\text{reduce-sum}} Y_{:,k}(1 - Y_{:,k})^\intercal \odot A \tag{5}$$

The element-wise product with the adjacency matrix ($\odot A$) ensures that only the adjacent nodes are considered. Moreover, the result of $Y_{:,k}(1-Y_{:,k})^\intercal \odot A$ is an $n \times n$ matrix and $\mathbb{E}[cut(S_k, \bar{S}_k)]$ is the sum over all of its elements.

From Equation 3, $vol(S_k, V)$ 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:

$$\begin{gathered}\Gamma = Y^\intercal D \\ \mathbb{E}[vol(S_k, V)] = \Gamma_k\end{gathered} \tag{6}$$

Where Γ is a vector in $\mathbb{R}^g$, and g is the number of partitions.

With $\mathbb{E}[cut(S_k, \bar{S}_k)]$ and $\mathbb{E}[vol(S_k, V)]$ from Equations 5 and 6, we can calculate the expected normalized cut in Equation 3 as follows:

$$\mathbb{E}[Ncut(S_1, S_2, \dots S_g)] = \sum_{\text{reduce-sum}} (Y \oslash \Gamma)(1 - Y)^\intercal \odot A \tag{7}$$

$\oslash$ is element-wise division and the result of $(Y \oslash \Gamma)(1 - Y)^\intercal \odot A$ is an $n \times n$ matrix where $\mathbb{E}[cut(S_1, S_2, \dots S_g)]$ is the sum over all of its elements.

Balanced Cut: So far, we have shown how one can calculate the expected normalized cut of a graph given the matrix Y (probabilities of nodes belonging to partitions). Here, we show that given Y we can also evaluate how balanced those partitions are.

Given the number of nodes in the graph $|V| = n$ and the number of partitions g, to have balanced partitions the number of nodes per partition should be $\frac{n}{g}$. The sum of the columns in Y gives us the expected number of nodes in each partition due to the fact that Y_{ik} represents the probability that node $v_i \in V$ belongs to partition S_k. Thus, for the balanced partitions we minimize the following error:

$$\sum_{k=1}^{g}(\sum_{i=1}^{n} Y_{ik} - \frac{n}{g})^2 = \sum_{\text{reduce-sum}} (\mathbf{1}^\intercal Y - \frac{n}{g})^2 \tag{8}$$

Combining expected normalized cut (Equation 7) with the balanced partition error (Equation 8), we have the following loss function:

$$\mathcal{L} = \sum_{\text{reduce-sum}} (Y \oslash \Gamma)(1 - Y)^\intercal \odot A + \sum_{\text{reduce-sum}} (\mathbf{1}^\intercal Y - \frac{n}{g})^2 \tag{9}$$

Next, we discuss the GAP neural model that finds the graph partition Y to minimize the loss in Equation 9.

4.2 The GAP Model

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 9).

Graph Embedding Module: 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 and 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.

GCN: [Kipf and Welling, 2017] showed that untrained GCN with random weights can serve as a powerful feature extractor for graph nodes. In our implementation, we used a 3-layer GCN with weight matrices ($\mathbf{W}^{(l)}$) using Xavier initialization described in [Glorot and Bengio, 2010].

$$Z = \tanh(\hat{A} \tanh(\hat{A} \tanh(\hat{A}X\mathbf{W}^{(0)})\mathbf{W}^{(1)})\mathbf{W}^{(2)})$$

where $\hat{A} = \tilde{D}^{-\frac{1}{2}}\tilde{A}\tilde{D}^{-\frac{1}{2}}$, $\tilde{A} = A + I_n$ is the adjacency matrix of the undirected graph G with added self-connections. I_n is the identity matrix, and $\tilde{D}_{i,i} = \sum_j \tilde{A}_{ij}$. The input feature matrix X depends on the graph. In TensorFlow computation graphs, each operation type (such as MatMul, Conv2d, Sum, etc.) would be a feature.

GraphSAGE: [Hamilton et al., 2017] developed a node embedding technique that generates high dimensional graph node representations based on node input features. Central to this technique is *sample and aggregate*, where given a node v_i, we sample a set of v_i's neighbors from $\mathcal{N}(v_i)$, and aggregate their representations (with max pooling) to generate an embedding for the sampled neighbors of v_i. This neighbor representation, along with the representation of v_i itself, is combined to generate a new representation for v_i. Iterating this process multiple times results in message passing among nodes for an increasing number of hops.

Our implementation of GraphSAGE is based on Algorithm 1 in [Hamilton et al., 2017]. For each message passing step k, we perform the following operations per node $v_i \in V$:

$$\mathbf{h}^k_{\mathcal{N}(v_i)} = \text{maxpool}(\{\mathbf{W}^{\text{agg}}_k \mathbf{h}^{k-1}_{v_j} + \mathbf{b}^{\text{agg}}_k, \forall v_j \in \mathcal{N}(v_i)\})$$

$$\mathbf{h}^k_{v_i} = \text{relu}(\mathbf{W}^{\text{proj}}_k [\mathbf{h}^{k-1}_{v_i}, \mathbf{h}^k_{\mathcal{N}(v_i)}] + \mathbf{b}^{\text{agg}}_k)$$

$$\mathbf{h}^k_{v_i} = \mathbf{h}^k_{v_i} / ||\mathbf{h}^k_{v_i}||_2$$

where agg and proj denote the aggregation and projection matrices respectively.

Graph Partitioning Module: 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 Equation 9.

We also note that for particularly large graphs, it is possible to optimize on randomly sampled minibatches of nodes from the larger graph. Furthermore, it is possible to stop gradient flow from the partitioning module to the embedding module, resulting in unsupervised node embeddings.

5 Experiments

The main goals of our experiments are to (a) evaluate the performance of the GAP framework against hMETIS [Karypis and 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. Source code is provided for reproducibility and is in the process of being open-sourced.

5.1 Setup

We conducted experiments on real and synthetic graphs. Specifically, we use five widely used TensorFlow graphs. We also generate *Random* as well as *Scale-free* graphs as synthetic datasets to show the effectivenesss of GAP on graphs with different structures.

Real Datasets

- *ResNet* [He et al., 2016] is a deep convolutional network with residual connections to avoid vanishing gradients. 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 and Zisserman, 2014] contains 16 convolutional layers. The TensorFlow graph of *VGG* contains 1,325 operations.

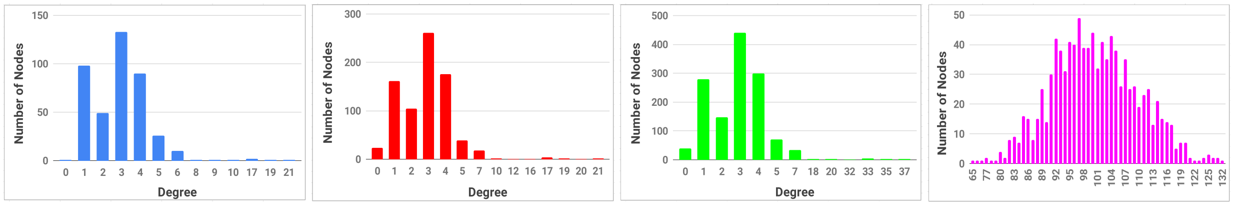
Synthetic Datasets

- *Random*: Randomly generated networks of size 10^3 and 10^4 nodes using the *Erdös–Rényi* model [Erdos and Rényi, 1960], where the probability of having an edge between any two nodes is 0.1.

- *Scale-free*: Randomly generated scale-free networks of size 10^3 and 10^4 nodes using NetworkX [Hagberg et al., 2008] (A scale-free network

Computation graphs	**hMETIS**		**GAP**	
Name	Edge cut	Balancedness	Edge cut	Balancedness
VGG	0.05	0.99	0.04	0.99
MNIST-conv	0.05	0.99	0.05	0.99
ResNet	0.04	0.99	0.04	0.99
AlexNet	0.05	0.99	0.05	0.99
Inception-v3	0.04	0.99	0.04	0.99

Table 1: Performance of GAP against hMETIS. Number of partitions is three and we run hMETIS and GAP over a given graph.For edge cut lower is better, for balancedness higher is better.



(a) MNIST-conv (b) AlexNet (c) VGG (d) Random graph

Figure 2: Degree histogram of *MNIST-conv*, *VGG*, *AlexNet* and synthetic *Random* graphs. *Random* graphs are denser than the others.

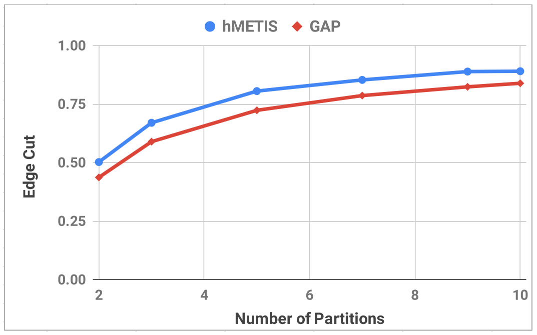


Figure 3: Edge cut of the partitions on random graphs by varying the number of partitions using GAP and hMETIS. Both GAP and hMETIS produce 99% balanced partitions.

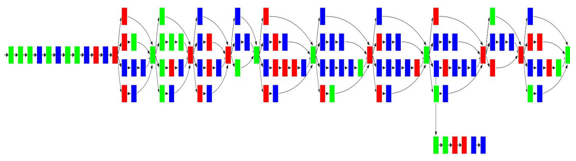
is a network whose degree distribution follows a power law [Bollobás et al., 2003]).

Baseline: Since graph partitioning is NP-complete, solutions are generally derived using heuristics and approximation algorithms. While there has been a substantial amount of work on graph partitioning for specific graph structure/applications [Gonzalez et al., 2012, Hada et al., 2018], *hMETIS* [Karypis and Kumar, 2000, Karypis et al., 1999] is a general framework that works across a wide variety of graphs and is shown to provide high quality partitions in different domains (e.g., VLSI, road network [Miettinen et al., 2006, Xu and Tan, 2012]. Similar to hMETIS, GAP is a general framework that makes no assumptions about graph structure. In our experiments, we compare GAP against hMETIS. We set the hMETIS parameters to return balanced partitions with minimum edge cut.

Performance Measures: As we discussed in Section 3, balanced partitions with minimum edge cut is the goal of graph partitioning. We evaluate the performance of the resulting partitions by examining 1) *Edge cut*: the ratio of the cut to the total number of edges, and 2) *Balancedness*: is one minus the MSE of number of nodes in every partition and balances partition ($\frac{n}{g}$).

Computation graphs		*AlexNet*		*Inception-v3*		*ResNet*	
Name	Embedding	Edge cut	Balancedness	Edge cut	Balancedness	Edge cut	Balancedness
GAP-op	-	0.16	0.71	0.24	0.74	0.45	0.90
GAP-id	*GCN offline*	0.28	0.97	0.19	0.98	0.17	0.93
GAP-op	*GCN offline*	0.07	**0.99**	0.12	0.98	0.11	0.94
GAP-op	*GraphSAGE offline*	0.07	**0.99**	0.08	**0.99**	0.09	0.95
GAP-op	*GraphSAGE trained*	**0.06**	**0.99**	**0.06**	**0.99**	**0.08**	**0.98**

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*. In *GAP-id*, we use node index features, while in *GAP-op*, we use TensorFlow operation types as features. 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. *GAP-op* with *GraphSAGE trained* (last row) generalizes better than the other models.


(a) Training a GAP model on *Inception-v3*, and testing on the same computation graph (*Inception-v3*) achieves 99% balanced partitions with 4% edge cut (Table 1)

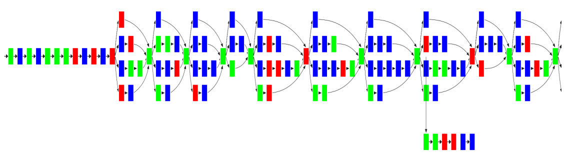

(b) Generalization: training a GAP model on *VGG*, and testing it on unseen graphs (*Inception-v3*) achieves 99% balanced partitions with 6% edge cut (last row of Table 2)

Figure 4: GAP partitioning of the *Inception-v3* (a) using the trained model on *Inception-v3* and (b) the trained model on *VGG*. Number of partitions is three and they are denoted by colors. We only show the nodes whose operation type is convolution.

5.2 Performance

In this set of experiments, we find that GAP outperforms hMETIS. 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 5.3.

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.

Figure 3 shows the performance of GAP against hMETIS on random graphs when the number of partitions is varied from 2 to 10. The plots represent the average value across 5 random graphs. Both GAP and hMETIS produce 99% balanced partitions. However, GAP is also able to find lower edge cut partitions than hMETIS. By examining the degree histograms of our datasets (Figures 2a to 2d), we found that while hMETIS heuristics work reasonably well on sparse TensorFlow graphs, GAP outperforms hMETIS on dense graphs.

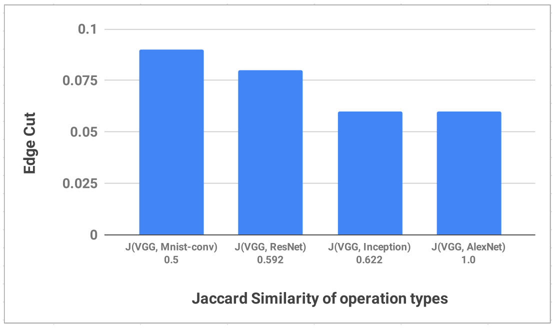


Figure 5: Here, GAP is trained on *VGG* and is tested on other computation graphs. We observe that the Jaccard similarity between the operation types in *VGG* and other graphs affects the generalization of GAP. Higher Jaccard similarities between the train and validation/test datasets enable GAP to find better partitions with smaller edge cut.

5.3 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.

5.3.1 Generalization on real graphs

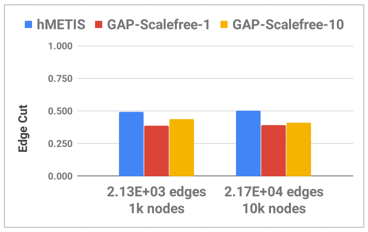
In this set of experiments, we train GAP with a single TensorFlow graph, *VGG*, and validate on *MNIST-conv*. At inference time, we test the trained model on unseen TensorFlow graphs: *AlexNet*, *ResNet*, and *Inception-v3*.

Table 2 shows the result of our experiments, and illustrates the importance of node features and graph embeddings in generalization. In *GAP-id*, we use the index of a node as its feature, while in *GAP-op*, 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 4.2, we leverage Graph Convolution Networks [Kipf and Welling, 2017] (GCN) 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 trained* both modules are trained jointly. Table 2 shows that *GAP-op* with *GraphSAGE trained* (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.

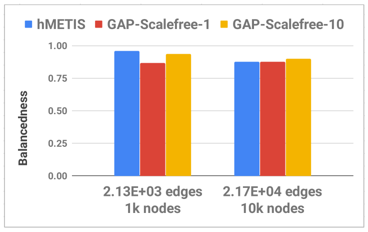
Figure 4 shows the GAP partitioning of *Inception-v3* using a model trained on the same graph (4a) and a model trained on *VGG* (4b). Note that partitions are denoted by colors and we only show nodes whose operation type is convolution. In the scenario (4a) where we train and test GAP on *Inception-v3*, we achieve 99% balanced partitions with 4% edge cut (Table 1). Where GAP is trained on *VGG* and tested over the unseen graph (*Inception-v3*), it achieves 99% balanced partitions with 6% edge cut (last row of Table 2). The partition assignments in Figures 4a and 4b are quite similar (75%), which demonstrates GAP generalization.

We also observed that the similarity of the node features (operation types) in *VGG* and other computation graphs used in inference and validation is correlated with the edge cut score of GAP partitioning (Figure 5). For example, let A and B be the set of the operation types in *VGG* and *ResNet*, respectively, with a Jaccard similarity of $\frac{|A \cap B|}{|A \cup B|} = 0.592$). Figure 5 shows that as Jaccard similarity of a graph with *VGG* increases, the edge cut decreases. In other words, the presence of similar node types across train and test graphs aids the generalization of our model.

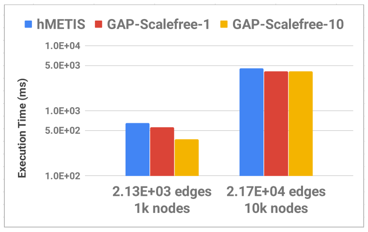
Model Architecture and Hyper-parameters: Here, we describe the details of the model with the best performance (corresponding to the last row of Table 2). The number of features (TensorFlow operation types) is 1518. GraphSAGE has 5 layers of 512 units with shared pooling, and the graph partitioning module 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 and Bengio, 2010]. We use the Adam optimizer with a learning rate of 7.5e-5.



(a) Edge cut of GAP vs hMETIS. GAP partitions unseen graphs of 1k and 10k nodes with smaller edge cut.



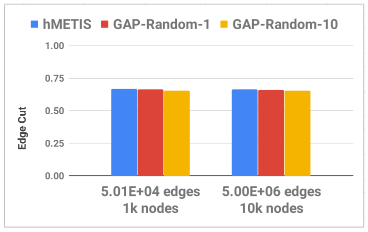
(b) Balancedness of GAP vs hMETIS. GAP-Scalefree-10 (trained on more graphs) improves the balancedness.



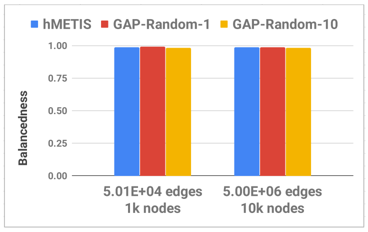
(c) Inference time of GAP vs running time of hMETIS. GAP is slightly faster than hMETIS.

Figure 6: Generalization of GAP on scale-free graphs. GAP-Scalefree-1 is trained on only one scale-free graph, while GAP-Scalefree-10 is trained on 10 scale-free graphs. The result is the average over the 5 scale-free graphs of 1k and 10k nodes. GAP-Scalefree-10 is slightly faster than hMETIS and it produces partitions which are as balanced as hMETIS partitions but with smaller edge cut.

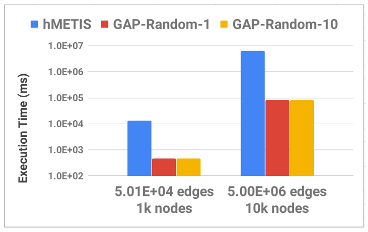
.



(a) The edge cut of GAP is slightly lower than that of hMETIS.



(b) The Balancedness of GAP and hMETIS are almost equal (99%).



(c) The inference time of GAP is 10 to 100 times faster than hMETIS.

Figure 7: Generalization of GAP on random graphs. GAP-Random-1 is trained on only one random graph, while GAP-Random-10 is trained on 10 random graphs. The result is the average over the 5 random graphs of 1k and 10k nodes. Performance of GAP-Random-1 and GAP-Random-10 is almost the same as hMetis but the inference time is 10 to 100 times faster than the runtime of hMETIS.

5.3.2 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. Similarly, we train GAP on scale-free graphs of 1k nodes and test on scale-free graphs of 1k and 10k nodes.

Figures 6a, 6b, and 6c show the edge cut, balancedness, and execution time of GAP against hMETIS over the scale-free graphs (every point is the average of 5 experiments). In GAP-Scalefree-1 we train GAP with only one scale-free graph, while GAP-Scalefree-10 is trained on 10 scale-free graphs. We then test the trained models GAP-Scalefree-1 and GAP-Scalefree-10 over 5 unseen scale-free graphs of 1k and 10k nodes and we report the average results. Figure 6a shows that both GAP-Scalefree-1 and GAP-Scalefree-10 partition the unseen graphs of 1k and 10k nodes with lower edge cut than hMETIS. Despite the balancedness of GAP-Scalefree-1 being lower than that of hMETIS, by increasing the number of graphs in the training set (GAP-Scalefree-10) balancedness is improved as shown in Figure 6b, while its edge cut is still smaller (6a). Furthermore, GAP-Scalefree-10 runs slightly faster than hMETIS (6c) and its partitions are just as balanced as those of hMETIS (6b) but with lower edge cut (6a).

Figures 7a, 7b, and 7c show the edge cut, balancedness, and execution time of GAP against hMETIS on random graphs. Every point is the average of 5 experiments. In GAP-Random-1, we train GAP on only one random graph, while in GAP-Random-10, we train on 10 random graphs. We then test the trained models GAP-random-1 and GAP-Random-10 on 5 unseen random graphs with 1k and 10k nodes and we report the average results. The performance of GAP when generalizing on unseen random graphs of 1k and 10k nodes is almost the same as the performance of hMETIS, while Figure 7c shows that during inference, GAP is 10 to 100 times faster than the runtime of hMETIS.

Model Architectures 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-1: model is trained with one scale-free graph. GraphSAGE has 5 layers of 512 units, and graph partitioning module is 3 layer dense network of 128 units with softmax. Learning rate is 2.5e-6.
GAP-Scalefree-10: 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-1: Trained with only random graph. GraphSAGE has 5 layers of 128 units with shared pooling, and graph partitioning module is 2 layer dense network of 64 units with softmax. Learning rate is 7.5e-4.
GAP-Random-10: 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.

6 Conclusion

We propose a deep learning framework, GAP, for the graph partitioning problem, where the objective is to assign the nodes of a graph into balanced partitions while minimizing the edge cut across the partitions. 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 competitive partitions while being up to 100 times faster than the baseline and generalizing to unseen graphs.

References

[Andersen et al., 2006] Andersen, R., Chung, F., and Lang, K. (2006). Local graph partitioning using pagerank vectors. In *FOCS*, pages 475–486. IEEE.

[Bollobás et al., 2003] Bollobás, B., Borgs, C., Chayes, J., and Riordan, O. (2003). Directed scale-free graphs. In *Proceedings of the Fourteenth Annual ACM-SIAM Symposium on Discrete Algorithms*, SODA '03, pages 132–139.

[Bui et al., 2017] Bui, T. D., Ravi, S., and Ramavajjala, V. (2017). Neural graph machines: Learning neural networks using graphs. *CoRR*, abs/1703.04818.

[Chang et al., 2014] Chang, X., Nie, F., Ma, Z., and Yang, Y. (2014). Balanced k-means and min-cut clustering. *arXiv preprint arXiv:1411.6235*.

[Chen et al., 2017] Chen, X., Huang, J. Z., Nie, F., Chen, R., and Wu, Q. (2017). A self-balanced min-cut algorithm for image clustering. In *ICCV*, pages 2080–2088.

[Chung, 2007] Chung, F. (2007). Four proofs for the cheeger inequality and graph partition algorithms. In *Proceedings of ICCM*, volume 2, page 378.

[Dilokthanakul et al., 2016] Dilokthanakul, N., Mediano, P. A., Garnelo, M., Lee, M. C., Salimbeni, H., Arulkumaran, K., and Shanahan, M. (2016). Deep unsupervised clustering with gaussian mixture variational autoencoders. *arXiv preprint arXiv:1611.02648*.

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

[Glorot and Bengio, 2010] Glorot, X. and Bengio, Y. (2010). Understanding the difficulty of training deep feedforward neural networks. In Teh, Y. W. and Titterington, M., editors, *AISTATS*, volume 9, pages 249–256, Chia Laguna Resort, Sardinia, Italy. PMLR.

[Gonzalez et al., 2012] Gonzalez, J. E., Low, Y., Gu, H., Bickson, D., and Guestrin, C. (2012). Powergraph: distributed graph-parallel computation on natural graphs. In *OSDI*, volume 12, page 2.

[Hada et al., 2018] Hada, R. J., Wu, H., and Jin, M. (2018). Scalable minimum-cost balanced partitioning of large-scale social networks: Online and offline solutions. *IEEE Transactions on Parallel and Distributed Systems*, 29(7):1636–1649.

[Hagberg et al., 2008] Hagberg, A., Swart, P., and S Chult, D. (2008). Exploring network structure, dynamics, and function using networkx. Technical report, Los Alamos National Lab.(LANL), Los Alamos, NM (United States).

[Hamilton et al., 2017] Hamilton, W. L., Ying, Z., and Leskovec, J. (2017). 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*, pages 1025–1035.

[He et al., 2016] He, K., Zhang, X., Ren, S., and Sun, J. (2016). Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778.

[Karypis et al., 1999] Karypis, G., Aggarwal, R., Kumar, V., and Shekhar, S. (1999). Multilevel hypergraph partitioning: applications in vlsi domain. *IEEE T VLSI SYST*, 7(1):69–79.

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

[Karypis and Kumar, 2000] Karypis, G. and Kumar, V. (2000). Multilevel k-way hypergraph partitioning. *VLSI design*, 11(3):285–300.

[Kawamoto et al., 2018] Kawamoto, T., Tsubaki, M., and Obuchi, T. (2018). Mean-field theory of graph neural networks in graph partitioning. In *Advances in Neural Information Processing Systems*, pages 4366–4376.

[Kipf and Welling, 2017] Kipf, T. N. and Welling, M. (2017). Semi-supervised classification with graph convolutional networks. In *ICLR*.

[Krizhevsky et al., 2012] Krizhevsky, A., Sutskever, I., and Hinton, G. E. (2012). Imagenet classification with deep convolutional neural networks. In *Advances in neural information processing systems*, pages 1097–1105.

[Leskovec, 2009] Leskovec, J. (2009). Community structure in large networks: Natural cluster sizes and the absence of large well-defined clusters. *Internet Mathematics*, 6(1):29–123.

[Liu et al., 2017] Liu, H., Han, J., Nie, F., and Li, X. (2017). Balanced clustering with least square regression. In *AAAI*, pages 2231–2237.

[Miettinen et al., 2006] Miettinen, P., Honkala, M., and Roos, J. (2006). *Using METIS and hMETIS algorithms in circuit partitioning*. Helsinki University of Technology.

[Mirhoseini et al., 2018] Mirhoseini, A., Goldie, A., Pham, H., Steiner, B., Le, Q. V., and Dean, J. (2018). A hierarchical model for device placement. In *ICLR*.

[Mirhoseini et al., 2017] Mirhoseini, A., Pham, H., Le, Q. V., Steiner, B., Larsen, R., Zhou, Y., Kumar, N., Norouzi, M., Bengio, S., and Dean, J. (2017). Device placement optimization with reinforcement learning. In *ICML*.

[Ng et al., 2002] Ng, A. Y., Jordan, M. I., and Weiss, Y. (2002). On spectral clustering: Analysis and an algorithm. In *Advances in neural information processing systems*, pages 849–856.

[Papadimitriou and Steiglitz, 1982] Papadimitriou, C. H. and Steiglitz, K. (1982). *Combinatorial Optimization: Algorithms and Complexity*. Prentice-Hall, Inc., Upper Saddle River, NJ, USA.

[Shaham et al., 2018] Shaham, U., Stanton, K., Li, H., Nadler, B., Basri, R., and Kluger, Y. (2018). Spectralnet: Spectral clustering using deep neural networks. *arXiv preprint arXiv:1801.01587*.

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

[Simonyan and Zisserman, 2014] Simonyan, K. and Zisserman, A. (2014). Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556*.

[Szegedy et al., 2017] Szegedy, C., Ioffe, S., Vanhoucke, V., and Alemi, A. A. (2017). Inception-v4, inception-resnet and the impact of residual connections on learning. In *AAAI*, volume 4, page 12.

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

[Veličković et al., 2018] Veličković, P., Cucurull, G., Casanova, A., Romero, A., Liò, P., and Bengio, Y. (2018). Graph Attention Networks. *International Conference on Learning Representations*.

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

[Xie et al., 2016] Xie, J., Girshick, R., and Farhadi, A. (2016). Unsupervised deep embedding for clustering analysis. In *International conference on machine learning*, pages 478–487.

[Xu and Tan, 2012] Xu, Y. and Tan, G. (2012). hmetis-based offline road network partitioning. In *AsiaSim 2012*, pages 221–229. Springer.

[Yang et al., 2017] Yang, B., Fu, X., Sidiropoulos, N. D., and Hong, M. (2017). Towards k-means-friendly spaces: Simultaneous deep learning and clustering. In *Proceedings of the 34th International Conference on Machine Learning-Volume 70*, pages 3861–3870. JMLR. org.

[Zhang and Rohe, 2018] Zhang, Y. and Rohe, K. (2018). Understanding regularized spectral clustering via graph conductance. In *NeurIPS*, pages 10654–10663.

[Zheng et al., 2016] Zheng, Y., Tan, H., Tang, B., Zhou, H., et al. (2016). Variational deep embedding: A generative approach to clustering. arxiv preprint. *arXiv preprint arXiv:1611.05148*.