INSTITUTO POLITÉCNICO NACIONAL

CENTRO DE INVESTIGACIÓN EN COMPUTACIÓN

T E S I S

A GRAPH NEURAL NETWORK APPROACH FOR LARGE-SCALE GRAPH PARTITIONING

QUE PARA OBTENER EL TÍTULO DE:

MAESTRÍA EN CIENCIAS DE LA COMPUTACIÓN

PRESENTA:

JOSÉ CRISPÍN ALVARADO CALDERÓN

DIRECTORES DE TESIS:

DR. RICARDO MENCHACA MÉNDEZ
DR. ROLANDO MENCHACA MÉNDEZ

Ciudad de México
Mayo, 2022

RESUMEN

El diseño de algoritmos para resolver problemas de Optimización Combinatoria es una tarea desafiante debido a la naturaleza de intractabilidad que poseen los mismos. Uno de los problemas fundamentales, y también uno de los más estudiados en el área, es el problema del *Particionado de Grafos*. En años recientes, se han desarrollado distintos algoritmos basados en Deep Learning para lidiar con éste y otros problemas de Optimización Combinatoria, los cuales han mostrado resultados satisfactorios. Uno de las algoritmos más famosos, entre los antes mencionados, es *Generalizable Approximate Partitioning (GAP) framework*. El objetivo de este trabajo es presentar una extensión de este *framework* que funcione para grafos mas generales, i.e. grafos sin atributos, y generalice bien para grafos grandes.

Una de las primeras observaciones hechas en esta investigación es que debido al hecho que el framework GAP usa una arquitectura basada en *Graph Convolutional Networks*, éste tiende a desempeñarse lentamente en grafos grandes. Por lo tanto, se llevaron a cabo algunas modificaciones significantes que mantienen las ventajas que ofrecen las Graph Convolutional Networks y al mismo tiempo reducen el tiempo de cómputo sin perder la potencia del framework original.

Además, se observó que el framework GAP funciona únicamente con grafos con características en sus nodos como consecuencia de la arquitectura seleccionada por sus autores. Esta dependencia fue elmininada a través de un enfoque basado en caminatas aleatorias para generar las características de los nodos. Esta se puede considerar como una modificación importante porque el algoritmo ahora depende únicamente de la estructura del grafo lo cuál es suficiente para el problema del Particionado de Grafos.

La última parte de esta investigación muestra cómo se adaptó una técnica de *negative sampling* al algoritmo propuesto para acelerar y hacer más efectivo el proceso de encontrar particiones balanceadas sin sacrificar la calidad de las particiones antes mencionadas. El algoritmo propuesto mostró resultados comparables con algoritmos ampliamente usados para el particionado de grafos como lo es METIS.

Abstract

The design of algorithms that solve Combinatorial Optimization problems is a challenging task due to their intractable nature. One of the fundamental and most studied problems in the area is the Graph Partitioning problem. In recent years, several Deep Learning algorithms have been developed to deal with this and other Combinatorial Optimization problems, which have shown satisfactory results. One of the famous ones is the Generalizable Approximate Partitioning (GAP) framework. This work aims to present an extension of this framework that works for more general graphs and generalizes well to large ones.

One of the first observations made in this research is that the GAP framework uses an architecture based on Graph Convolutional Networks (GCN's) which tends to be slow for big graphs. Therefore, significant modifications were carried out that preserve the advantages that Graph Neural Networks offer and at the same time reduce the computation time without losing the power of the original framework.

In addition, it was noted that as a consequence of the model architecture chosen by its authors, GAP requires graphs with node features. That dependency was eliminated by using a random walk approach to generate node features. This was an important modification because the algorithm now relies purely on the graph's structure which is enough for the Graph Partitioning Problem.

The last part of this research shows how to use a negative sampling technique to accelerate and improve the process of finding balanced partitions without sacrificing the quality of said partitions. The proposed algorithm shows results comparable with widely used partitioning algorithms like METIS.

CONTENTS

List of Figures

List of Tables

CHAPTER 1

INTRODUCTION

1.1 Presentation

Combinatorial Optimization is a prominent field that results from the intersection of mathematics and theoretical computer science, specifically from areas such as combinatorics, algorithm theory, and operations research. Combinatorial Optimization problems are characterized by their solution space that consists of a finite collection of objects. This collection of objects typically grows exponentially in size, so going through all the objects and selecting the optimal one is not feasible [33].

The computational complexity behind solving Combinatorial Optimization (CO) problems is a very well-known concern, but the main motivation to solving them is the number of real-life problems that can be modeled using this framework. Thus, developing new techniques that provide Combinatorial Optimization algorithms which perform well has been in the spotlight for more than 50 years [25], and it is the subject matter of this research.

Considering the aforementioned, it is important to note what the essence of a CO algorithm is. As explained eloquently by Maltby and Ross [26], a CO algorithm uses mathematical methods either to make the search of possible solutions faster or to reduce the size of the set of feasible solutions. Some of the state-of-the-art techniques that have been proposed to produce those algorithms can be summarized as:

- Heuristic methods: a series of strategies where experience-based techniques are followed to solve the problem. Generally used when classical methods are too slow and

when an approximate solution is enough for the implicit purposes.

- Branch and bound method: consists of a systematic division of the solution space where the core elements are called *branches*. Then, branches are recursively explored and compared against estimated bounds of the optimal solution. [28]
- Mixed/Integer Programming: some of the decision variables in the problem are constrained to be integer values at the optimal solution, and then, the problem can be solved by one or several methods before mentioned.

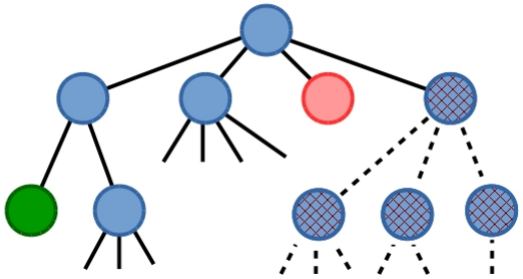


Figure 1.1: Illustration of the process of solving a CO algorithm with the branch and bound method [20].

For further information about the most popular strategies or an exhaustive explanation of those techniques, those considered to be good resources are [10] and [21].

It is well known that most CO problems can be modeled with mathematical graphs [25]. Hence, most of the efforts in developing CO algorithms are dedicated to solving graph problems. These efforts have incremented in the last decade on account of the rapid growth in digital technology. Along with this change, the computing needs and the usual assumptions that algorithms used to make have become more demanding in terms of resource allocation.

On one hand, there is a need to research new theoretical proposals and to further investigate the new constraints that modern problems imply. Then, one could use this knowledge to influence the way new strategies approach those problems. On the other hand, there is a pragmatic need to solve those problems in real life and come up with practical and efficient solutions in a considerably small amount of time. In short, the theoretical and practical aspects need to be considered as part of the equation in contemporary applications where strong optimization strategies, like heuristic methods and Integer Programming, are just not enough.

The ability to address the underlying issues that emerge in those kinds of problems has been improved drastically during the last years. Those huge steps are directed to obtain fast

approximations for CO problems and have given rise to new ways to find algorithms that provide empirically-efficient solutions.

Some recent approaches are being designed to take advantage of Machine Learning capabilities. In particular, Deep Learning strategies are providing alternative ways to deal with some of the underlying issues that arise with those kinds of problems. For specific problems, those approaches have shown to be the most successful in terms of running time efficiency, less human intervention, and the quality of the solutions.

Balanced graph partitioning is one of the fundamental Combinatorial Optimization problems and one which has gotten the best from Deep Learning. Stated in simple words, the Balanced Graph Partitioning Problem is the task of finding a partition of a given graph into mutually exclusive sub-graphs, i.e. they do not share nodes, and the partitions have the same size relative to a given measure. Finding a partition with the desired characteristics is a hard task, but the importance of solving it has incremented in recent years due to its numerous applications.

1.2 Objectives

1.2.1 General objective

The main target of this research is to develop a machine learning algorithm that solves the graph partitioning problem which is capable of performing on abstract graphs and large-scale graph instances and provide a more robust algorithm.

1.2.2 Specific objectives

The following objectives will help accomplish the general objective:

- To understand and analyze the Generalizable Approximate Partitioning (GAP) framework and to use it for solving the graph partitioning problem.
- To design an algorithm based on GAP that works for general (non-attributed) graphs and at the same time relies completely on their structure.
- To build a framework that is easy to modify in order to use different objective functions in the partitioning stage.

- To explore different types of sampling and study the impact they have in the efficiency of the algorithm.

1.3 Justification

Graphs are a mathematical representation of what is colloquially known as networks. They are used to describe the relationships between objects and allows one to model an extensive amount of real life problems. Due to their modeling capacity and their power of abstraction, they are widely studied in different areas of computer science and applied mathematics.

The Graph Partitioning (GP) problem is relevant in solving a big number of graph-related tasks. Frequently, GP is employed as a pre-processing subroutine to solve a distinct graph-related problem. If the edges that cross between the partition groups are relatively small compared to the original graph, then the partitioned graph may be more appropriate for analysis and problem-solving than the original [23].

Finding good-quality partitions of a graph is generally the first step of distributed graph computing tasks. The next paragraphs are dedicated to describe some of the most distinguished applications of GP. Applications of GP to solve graph-related and other abstract computer science problems are presented first followed by some applications to other areas and real-life situations.

One of the first and most useful applications of GP that one can find in literature is graph compression. A popular method for graph compression is the reordering method, which bisects a graph into two sets of equal cardinality aiming to minimize a compression-related objective function. *Graph bisection* is a special case of GP where the objective number of partitions is two. For instance, Bouritsas et. al. [2] showed a way to use a Machine Learning approach combined with a parametric GP algorithm to solve this task.

Another important application of GP is to implement *graph sparsification*. Graph sparsification is the task of representin big graphs in a way that takes less space in memory but preserves most of the relevant properties. Spielman and Teng [35] proposed a novel nearly-linear time partitioning algorithm that is used to produce spectral sparsifiers. The same partitioning algorithm is used by their authors to efficiently solve linear systems [37]. In an more recent work, Gatti et. al. [12] presented a way to solve sparse symmetric systems of linear equations using the *nested dissection ordering* algorithm. This algorithm is a divide and conquer heuristic based on GP. Previous work in this algorithms was made by Gupta [17].

Graph partitioning also plays an essential role in paralleling computations and the design

of new algorithms on large graphs. For example, in the *device placement* problem one aims to distribute work accross multiple devices. This is a relevant problem in Deep Learning in situations where it is important to train Neural Networks accross multiple devices [27].

In other applications, Sun et. al. [38] proposed a solution for the problem "intentional islanding in power systems considering load generation balance" where the main component is GP. For their part, Grady and Schwartz [16] made use of the isoperimetric constant to generate a GP algorithm that segmentate images.

Undoubtedly, the most recent direction where the GP problem has gained importance is in its application for clustering in complex networks. Those types of networks include but are not limited to social networks, transportation networks, web graphs, and biological networks. For an example, see Figure 1.2.

Figure 1.2: The global mobility network. Representation of 4069 airports worldwide where gray lines between them are direct connections [3].

An extensive report about the relationship between clustering in complex networks and GP can be found in [9]. That book is a collection of the classical approaches that involves this two related problems. The reader can also look over [36] for a more recently-related research.

The approaches followed by most of the mentioned research applications use state-of-the-art GP algorithms. Some of the same algorithms for those tasks are unexplored in modern large-scale graphs, and they do not consider the growth in size of the emerging graph data in the following years. The possibility that they become obsolete in the near future needs to be studied carefully alongside with the development of new strategies that overcome the impending challenges.

Something important to highlight is that the target is not only load-balance but some-

times the minimization of communication volume. This is a consideration not all of the existing approaches have and something worth researching.

In this context, the project "A Graph Neural Network Approach for Large-Scale Graph Partitioning" was created as a proposal to address some of those challenges. It took one of the most successful frameworks to create a flexible algorithm that can be used in a variety of graphs.

1.4 Project scope and limitations

During the development of this project the following considerations were taken:

- The algorithm has not been tested on real-world graphs. Most of the graphs where the experiments took place were taken from a famous dataset where some of the most successful partitioning algorithms have been tested. Even though this offers a good comparison point, the size of those graphs is pretty small compared to massive graphs like the Amazon or Pinterest networks. The largest graph in the dataset contains 500,000 nodes.

- In general, the graphs used for running the experiments are not dense. However, some of the graphs are dense, but they belong to the category of small-size graphs with less than 50,000 nodes. Although a number of real-world graphs are sparse, it is suspected that special considerations should be contemplated during the training of the algorithm in the future.

- Most of the parameter values are based on previous work where they were shown to have good performance or they were recommended by the original authors. Nonetheless, more experiments in different scenarios should be run to determine better values for the GP specific task.

- The machines where the experiments were executed have very limited computing power compared to the ambitions of this project. Though hardware specifications in those machines were enough to run experiments on the dataset, but it would be impossible to process complex networks with this technology.

- The proposed algorithm was not designed to run in a distributed environment. Experiments were run in single thread, and the algorithm was trained only in GPU. Modifying

the algorithm so it can run in parallel and distributing the training load between the GPU and the CPU are interesting research topics.

- The proposed algorithm has been tested only to bisect graphs. While it can work to partition a graph in more than two groups, the hyperparameters choice needs to be studied in more detail.

- The proposed framework only accepts certain input formats for the graphs, specifically JOSTLE and METIS formats.

1.5 Research Problem

Several new approaches and algorithms for the graph partitioning problem have arisen during the two decades. This effort has incremented during the last five years due to the creation of modern artificial techniques, in particular, Deep Learning Techniques.

The most successful techniques make use of Graph Neural Networks which have been shown The state-of-the-art algorithms for graph partitioning Graph partitioning algorithms based on DL

the algorithm cannot be easily adapted to more partitions due to the limited features they are using.

https://arxiv.org/pdf/2104.03546.pdf

1.6 Hypothesis

When analyzing GAP in depth some observations were made.

1.7 Project organization

The organization of the remaining parts of the thesis is as follows:

- In Chapter 2, basic mathematical terminology is presented which is going to be used in the next chapters. Next, this chapter provides a glance at the spectral methods used to solve the GP problem. Then, it finishes with a review of some of the most popular Deep Learning techniques for GP.

- In Chapter 3, the proposed graph partitioning algorithm for large-scale graphs is presented. It provides a description of the main components in the modules of the framework. Then, it shows how those modules can be used by the algorithm to effectively perform in abstract graphs. Finally, an incorporated sampling technique is used to improve the training process.
- In Chapter 4, the experiments carried out are described along with the dataset and the comparison metrics used in them. The proposed solution is compared to state-of-the-art methods given the proposed metrics. At the end, the obtained results are summarized.
- In Chapter 5, a summary of the contributions of this project is given. The advantages of the proposed algorithm over other existing approaches are mentioned as well as further research areas found during its development.

CHAPTER 2

THEORY AND CONCEPTUAL FRAMEWORK

A brief definition the mathematical concepts in the next chapters is found bellow. They are include some of the definitions and principal postulates used in spectral graph theory in which is based this work.

2.1 Preliminaries

Theorem 2.1.1. *For every $n \times n$ symmetric real matrix, the eigenvalues are real and the eigenvectors can be chosen real and orthonormal.*

Theorem 2.1.2 (Courant-Fisher Formula)**.** *Let A be an $n \times n$ real symmetric matrix with eigenvalues $\lambda_1 \leq \lambda_2 \leq \cdots \leq \lambda_n$ and corresponding eigenvectors $v_1, v_2, ..., v_n$. Then*

$$\lambda_1 = \min_{\|x\|=1} x^T Ax = \min_{x \neq 0} \frac{x^T Ax}{x^T x},$$

$$\lambda_2 = \min_{\substack{\|x\|=1 \\ x \perp v_1}} x^T Ax = \min_{x \neq 0} \frac{x^T Ax}{x^T x},$$

$$\lambda_n = \lambda_{max} = \max_{\substack{\|x\|=1 \\ x \perp v_1}} x^T Ax = \max_{\substack{x \neq 0 \\ x \perp v_1}} \frac{x^T Ax}{x^T x}.$$

In general, for $1 \leq k \leq n$, let S_k denote the span of $v_1, v_2, ..., v_k$ (with $S_0 = \{0\}$). Then

$$\lambda_k = \min_{\substack{\|x\|=1 \\ x \in S_{k-1}^{\perp}}} x^T Ax = \min_{\substack{x \neq 0 \\ x \in S_{k-1}^{\perp}}} \frac{x^T Ax}{x^T x}.$$

2.2 Graphs and Laplacian Matrices

For the rest of the chapter, let $G = (V, E)$ be a graph, where $V = \{v_1, v_2, ..., v_n\}$ is the non-empty set of nodes (or vertices) and E is the set of edges, composed by pairs of the form (v_i, v_j), where $v_i, v_j \in V$.

It is assumed that all graphs are undirected, meaning that if $(v_i, v_j) \in E$, then $(v_j, v_i) \in E$, for every $v_i, v_j \in V$. For that reason, the edge (v_i, v_j) will be represented as the unordered set $\{v_i, v_j\}$.

A convenient way to represent a graph is through an *adjacency matrix* $A \in \mathbb{R}^{|V|\times|V|}$. Giving a specific order to the graph nodes, one can represent the edges as binary entries in this matrix:

$$A[v_i, v_j] = \begin{cases} 1 & \text{if } \{v_i, v_j\} \in E \\ 0 & \text{otherwise} \end{cases}$$

Let $\mathcal{N}(v_i)$ denote the neighborhood of node v_i, i.e., the set of the adjacent nodes to it. The quantity that represents the number of nodes in $\mathcal{N}(v_i)$ is called the *degree of the vertex* v_i. This is one of the most obvious and informative feature for the structure of the graph and is denoted by

$$d_i = \sum_{j=1}^{n} A[v_j, v_i].$$

Finally, we can summarize that graph's information in the *degree matrix* D which is defined as the diagonal matrix with the degrees $d_1, d_2, ..., d_n$ on the diagonal.

2.2.1 The Unnormalized Laplacian

The unnormalized graph *Laplacian matrix* L is defined as

$$L = D - A$$

Proposition 2.2.1 (Some properties of L)**.** *The matrix L, as defined above, satisfies the following properties:*

1. *For every vector $x = (x_1, x_2, ..., x_n) \in \mathbb{R}$ we have*

$$x^T L x = \frac{1}{2} \sum_{i=1}^{n} \sum_{j=1}^{n} \left(x_i - x_j\right)^2$$

2. *L is symmetric and positive semi-definite*

3. *L has n non-negative, real-valued, eigenvalues $\lambda_1 \leq \lambda_2 \leq \cdots \leq \lambda_n$*

4. *The smallest eigenvalue of L is 0, the corresponding eigenvector is the constant one vector $\mathbb{1}$.*

2.2.2 Normalized Laplacians

The *symmetric normalized Laplacian* matrix L_{sym} is defined as

$$L_{sym} = D^{-\frac{1}{2}} L D^{-\frac{1}{2}}$$

while the *random walk Laplacian* is defined as

$$L_{rw} = D^{-1} L$$

2.2.3 The graph partitioning problem

In order to introduce the graph partitioning problem in its different settings, the mathematical definition of the concepts involved are presented below.

Definition 2.2.1. *Given a graph $G = (V, E)$ and an integer K, a partition of G is a collection of K subsets $P_1, P_2, ..., P_K \subset V$ such that:*

1. *$P_i \cap P_j = \emptyset$ for $i \neq j$, where $i, j \in \{1, 2, ..., K\}$*

2. *$\cup_{k=1}^{K} P_k = V$*

A partition of a graph can be seen as simply removing edges from the original graph in such way the obtained partitions are subgraphs. There are many ways a graph can be partitioned into subgraphs and the way it gets done depends completely on the application of interest. However, independently of the problem to solve, the objective relies on minimizing the connections between the partitions in the original graph. The following concepts provides a useful notation to turn the problem into an optimization one.

For a collection $S \subset V$ of vertices, we define the *edge boundary* $\partial(S)$ to consist of all edges in E with exactly one endpoint in S, that is,

$$\partial(S) := \{\{u, v\} \in E \mid u \notin S \text{ and } v \in S\}$$

Now the problem turns into finding a partition $P_1, P_2, ..., P_K$ such that minimizes the *cut value* of the partition, usually called just *cut*, which is defined as

$$\text{CUT}(P_1, P_2, ..., P_K) := \frac{1}{2} \sum_{k=1}^{K} |\partial(P_k)| \tag{2.1}$$

The notion of cut allows to measure the quality of any partition, and it can be solved in polynomial time. Nevertheless, solving the min cut problem in real-world problems often leads to very imbalanced partitions consisting of one-node groups.

For a collection of vertices $S \subset V$, consider the following quantity related to the edges of a subgraph

$$\text{VOL}(S) := \sum_{v_i \in S} d_i$$

This quantity allows to defining a different way to measure the quality of a partition that not only depends on the number of vertices but the total communication between them.

Inspired in that, and in the variety of applications that benefit from their formulation, the next quantities provides two different ways of measuring the size of the partitions. In colloquial terms, a partitioning algorihtm can aims to load-balance and/or to minimize the communication volume.

$$\begin{aligned}\text{RATIOCUT}(P_1, P_2, ..., P_K) &:= \frac{1}{2}\sum_{k=1}^{K}\frac{|\partial(P_k)|}{|P_k|} \\ &= \sum_{k=1}^{K}\frac{\text{CUT}(P_k, \overline{P_k})}{|P_k|}\end{aligned}$$

$$\begin{aligned}\text{NORMCUT}(P_1, P_2, ..., P_K) &:= \frac{1}{2}\sum_{k=1}^{K}\frac{|\partial(P_k)|}{\text{VOL}(P_k)} \\ &= \sum_{k=1}^{K}\frac{\text{CUT}(P_k, \overline{P_k})}{\text{VOL}(P_k)}\end{aligned}$$

2.2.4 The spectral method and Chegger's inequality

Cheeger's inequality

For a graph $G = (V, E)$ the *conductance* or *Cheeger ratio* of a set $S \subset V$ is the ratio of the fraction of edges in the cut $(S, \overline{S})$ o the volume of S,

$$\phi(S) = \frac{E(S, \overline{S})}{\text{VOL}(S)}$$

The *conductance* or *Cheeger constant* of a graph G is denoted by

$$\phi(G) = \min_S \phi(S)$$

Theorem 2.2.1. *In a graph G, the Cheeger constant $\phi(G)$ and the spectral gap λ_G are related as follows:*

$$2\phi(G) \geq \lambda_G \geq \frac{\alpha_G^2}{2} \geq \frac{\phi(G)^2}{2}$$

where α_G^2 is the minimum Cheeger ratio of subsets S_i consisting of vertices with the largest i values in the eigenvector associated with $lambda_G$, over all $i \in [n]$

The spectral method

1. Let v denote the second smallest eigenvector of $\mathscr{L}$. Sort the vertices i of G in increasing order of v_i. Let the resulting ordering be $v_1 \leq v_2 \leq \cdots v_n$
2. For each i, consider the cut induced by $\{1,2,...,i\}$ and its complement. Calculate its conductance.
3. Among these $n-1$ cuts, choose the one with minimum conductance.

Generalization to many partitions

1. Perform eigenvalue decomposition to find the eigenvectors of L_{sym}.
2. Select the k largest eigenvectors $e_1, e_2, ..., e_k$ of L_{sym} associated to the largest eigenvalues $\lambda_1, \lambda_2, ..., \lambda_k$
3. Form the matrix Y from the matrix $X = [e_1, e_2, ..., e_k]$ given by

$$Y_{ij} = \frac{X_{ij}}{\left(\sum_j X_{ij}^2\right)^{\frac{1}{2}}}$$

4. Treating each row of Y as a point in $\mathbb{R}^k$, cluster them into k clusters using $K-means$
5. Finally, assign the original vertex to cluster j if and only if row i of the matrix was assigned to cluster j

2.2.5 Generalizable Approximate Graph Partitioning (GAP) Framework

From all the Deep Learning approaches that solve the Graph Partitioning problem, one of the most notorious not only for its simplicity but for its exceptional results, is the *Generalizable Approximate Graph Partitioning* framework better known as GAP [15].

GAP is a Graph Neural Network approach that proposes a continuous relaxation of the problem using a differentiable loss function that is based on the normalized cut. According to Nazi et al. [29], it is an unsupervised learning algorithm that is capable of generalization, meaning that it can be trained in small graphs, which allows it to generalize into unseen much larger ones. This section describes the model described in the original paper which consists of two modules: the Graph Embedding Module and the Graph Partitioning Module. See Figure 2.1.

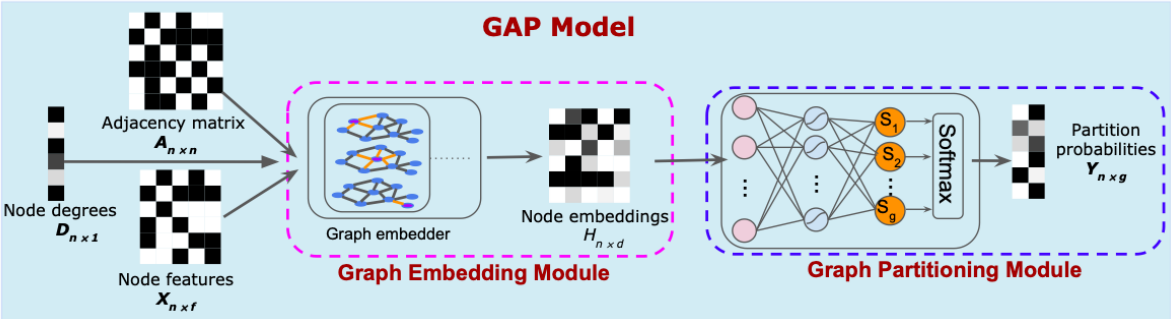


Figure 2.1: Architecture of the Generalizable Approximate Partitioning framework [29].

In the following subsections, it is assumed that the framework takes a graph $G = (V, E)$ as input, where $V = \{v_1, v_2, ..., v_n\}$, and outputs the probabilities tensor $Y \in \mathbb{R}^{n\times K}$, where Y_{ik} represents the probability that node v_i belongs to partition P_k. Before going into the model description, the deduction of the loss function is as follows.

2.2.5.1 Expected Normalized Cut Loss Function

Recall the normalized cut given by

$$\text{NormCut}(P_1, P_2, ..., P_K) = \sum_{k=1}^{K} \frac{\text{Cut}(P_k, \overline{P_k})}{\text{Vol}(P_k)} \tag{2.2}$$

In order to calculate the normalized cut expected value, one needs to compute the expected value of $\text{Cut}(P_k, \overline{P_k})$ and $\text{Vol}(P_k)$ from Equation 2.2. For the deduction of those quantities, an approach similar to the one presented in [13] will be followed .

Since Y_{ik} represents the probability that node $v_i \in P_k$, $1 - Y_{ik}$ is the probability that $v_i \notin P_k$, hence

$$\mathbb{E}[\text{Cut}(P_k, \overline{P_k})] = \sum_{i=1}^{|V|} \sum_{v_j \in \mathcal{N}(v_i)} Y_{ik}(1 - Y_{jk}) \tag{2.3}$$

as can it can be deducted from Figure 2.2.

Due to the fact that for a given node the adjacent nodes can be retrieved from the adjacency matrix A, Equation 2.3 can be rewritten as follows:

$$\mathbb{E}[\text{CUT}(P_k, \overline{P_k})] = \sum_{i=1}^{|V|} \sum_{j=1}^{|V|} Y_{ik}(1 - Y_{kj}^T) A_{ij} \tag{2.4}$$

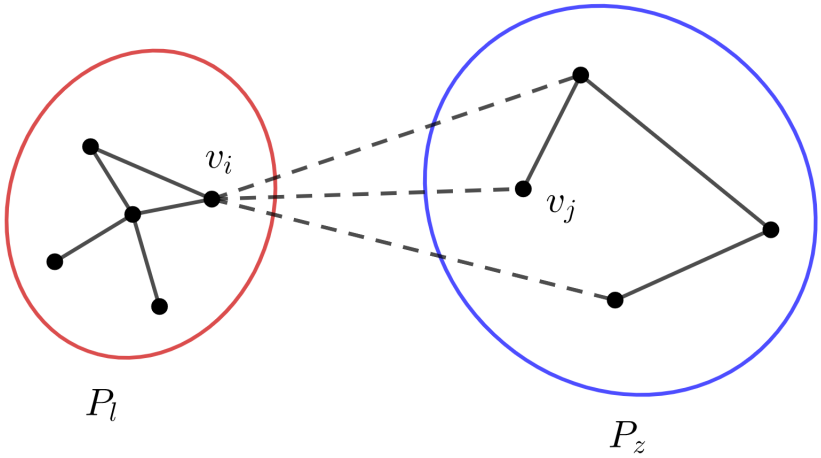


Figure 2.2: The expected value of the number of edges in the boundary of a determined partition P_l is given by the sum of the probabilities that each node $v_i \in V$ belongs to P_l, multiplied by the probability that each of its neighbors does not belong to P_l.

Keeping in mind that $\text{VOL}(P_k)$ is the sum of the degrees of the nodes in P_k, Δ is defined to be the column tensor where Δ_i is the degree of the node $v_i \in V$. Then, given Y, one can calculate the expected value of $\text{VOL}(P_k)$ as follows:

$$\begin{aligned} \Gamma &= Y^T \Delta \\ \mathbb{E}[\text{VOL}(P_k)] &= \Gamma_k \end{aligned} \tag{2.5}$$

From the results obtained in Equation 2.4 and Equation 2.5, a way to calculate the expected value of $\text{NORMCUT}(P_1, P_2, ..., P_K)$ is given by:

$$\mathbb{E}[\text{NORMCUT}(P_1, P_2, ..., P_K)] = \sum_{k=1}^{K} \sum_{i=1}^{|V|} \sum_{j=1}^{|V|} \frac{Y_{ik}(1 - Y_{kj}^T) A_{ij}}{\Gamma_k} \tag{2.6}$$

Nazi et al. [29] also showed that given the probability tensor Y, one can evaluate how balanced those partitions are. Note that the sum of the columns in Y is the expected number

of nodes in each partition, i.e., $\mathbb{E}[|P_k|] = \sum_{i=1}^{|V|} Y_{ik}$. On the other hand, in order to have balanced partitions, the number of nodes in each one should be $\frac{|V|}{K}$. As a consequence, the quantity $\left|\sum_{i=1}^{|V|} Y_{ik} - \frac{|V|}{K}\right|$ measures how balanced the partition P_k is.

Using the last result, and replacing the absolute value by the squared function, one can derive the loss function from Equation 2.6. This is the one originally used in GAP that intends to minimize the expected value of the normalized cut and at the same time balances the cardinalities of the partitions:

$$\mathscr{L} = \sum_{k=1}^{K} \sum_{i=1}^{|V|} \sum_{j=1}^{|V|} \frac{Y_{ik}(1 - Y_{kj}^{T}) A_{ij}}{\Gamma_k} + \sum_{k=1}^{K} \left(\sum_{i=1}^{|V|} Y_{ik} - \frac{|V|}{K} \right)^2 \tag{2.7}$$

2.2.5.2 The Embedding Module

In the graph embedding module, the algorithm learns node embeddings by encoding local structure information and the node features. The embeddings are calculated using Graph Neural Networks (GNN) which have become very popular during the recent years. To ensure generalization, the GAP authors opted for an inductive GNN approach by leveraging GraphSAGE with a Graph Convolutional Network (GCN) based approach.

In the paper where GAP was presented, the authors used a 3-layer GCN using Xavier initialization that can be found in [14]

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

where $\hat{A} = (D + I)^{-\frac{1}{2}}(A + I)(D + I)^{-\frac{1}{2}}$ is a normalized variant of the adjacency matrix with self loops, X is the feature matrix, and $\boldsymbol{W}^{(l)}$ is a learnable parameter matrix.

2.2.5.3 The Partitioning Module

The second module of GAP is composed of a fully connected layer that takes as input a node embedding vector $\boldsymbol{z}_u$ generated in the embedding module. This fully connected layer is then followed by a softmax layer trained to minimize the expected normalized loss function given by Equation 2.7.

This module is the responsible for partitioning the graph by returning $Y \in \mathbb{R}^{|V| \times K}$, the probabilities matrix that each node belongs to each of the partitions $P_1, P_2, ..., P_K$. At the same time, it ensures that for a given node, the sum of the probabilities of belonging each of the partitions is 1

$$\sum_{k=0}^{K} Y_{ik} = 1$$

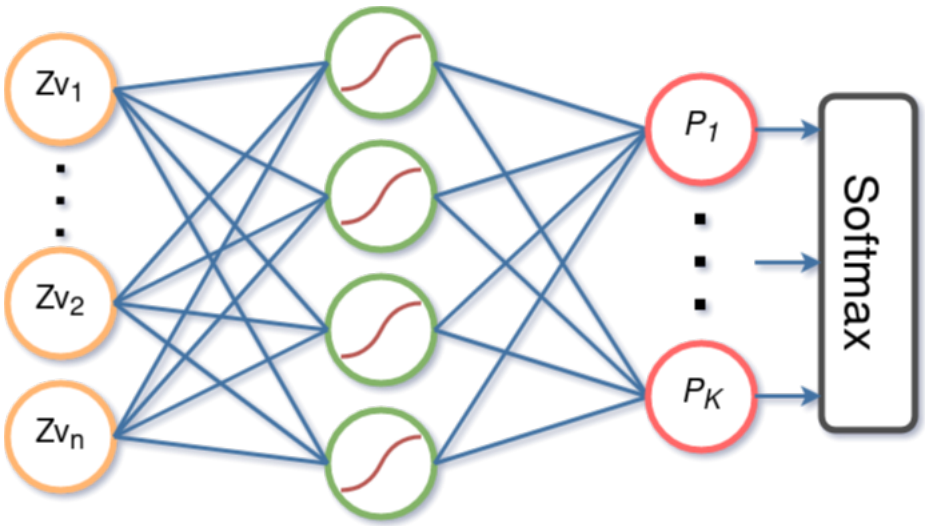


Figure 2.3: The partitioning module. A dense neural network that processes a node embedding and outputs the probability of the corresponding node to belong to each partition.

CHAPTER 3

GRAPH PARTITIONING FOR LARGE GRAPHS

This chapter is going to develop all the concepts related to the modifications made to the GAP framework. It contains the proposed solution and a description of why some decisions were made.

3.1 Node Embeddings

As mentioned by Zhang et. al. [45], Graph Convolutional Networks (GCN's) and their variants have become a very hot topic in Machine Learning, and they are being quite frequently used to solve plenty of problems. Indeed, the use of GCN's to generate *node embeddings*, which are just dense vector representations for the nodes in the involved graph, was a key factor that made the GAP framework very successful in producing good-quality partitions.

Graph Convolutional Networks can solve the following limitations found in traditional encoders [41]:

- Traditional encoders do not scale, i.e., they generate unique embeddings for each node.
- Traditional approaches can only generate embeddings for a single fixed graph.
- They only use the graph structure, and they do not consider node features.
- Those encoders focus on a specific architecture and cannot be adapted to train with different loss functions.

3.1.1 Graph Convolutional Networks

GCN's were originally proposed by Kifp et. al. [24]. When they first came up with their algorithm, they proposed a model where a single layer follows the following propagation rule:

$$H^{(l+1)} = f(H^{(l)}, A) = \sigma\left(\hat{D}^{-\frac{1}{2}}\hat{A}\hat{D}^{-\frac{1}{2}}H^{(l)}W^{(l)}\right) \tag{3.1}$$

$$H^{(0)} = X, \tag{3.2}$$

where $\boldsymbol{W}^{(l)}$ is a the learnable parameter weight matrix for the l-th network layer, $\hat{A} = A + I$ is the adjacency matrix with self loops, $\hat{D}$ is the diagonal node degree matrix of $\hat{A}$, X is the feature matrix, and σ is a non-linear activation function. Also, note that the term $\hat{D}^{-\frac{1}{2}}\hat{A}\hat{D}^{-\frac{1}{2}}$ is a symmetric normalized version of the graph's Laplacian.

Although GCN's are extremely powerful in generating node embeddings, in practice, they have the following downsides:

- GCN's are slow to train. As it can be seen in Equation 3.1, they multiply feature matrices by powers of the normalized graph's Laplacian, i.e., they are computationally very expensive.
- GCN's consume a lot of memory when training. They are not suitable for either dense graphs or large graphs.
- They present over-smoothing. After stacking multiple rounds of message passing, GCN's contain similar information in all node embedding representations [18].

To solve these issues in GCN's, several solutions have been proposed that deal with the downsides mentioned without losing the expressibility of the GCN. Some of the most relevant solutions are FastGCN, GraphSAGE and PinSAGE.

FastGCN [6] introduced sampling to the GCN framework and reformulated the loss function to provide a mechanism for inductive learning. They addressed the efficiency problems found in training GCN's by adding a batch-training scheme using Monte Carlo techniques. Finally, FastGCN shows better generalization than its preceding framework.

Similar to FastGCN, GraphSAGE [19] is an inductive version of GCN's. It introduces sampling techniques that remove the dependency of having all nodes during the training stage that improves the performance and running time. GraphSAGE naturally generalizes to unseen nodes and can learn about the local graph structure.

From all the mentioned solutions, PinSAGE [42] is perhaps the most complete and successful. This is the architecture used by Pinterest for their recommendation system. PinSAGE operates on an enormous graph with 3 billion nodes and 18 billion edges. It uses a combination of random walks and a MapReduce-based inference to compute the node embeddings which dramatically improves the scalability of GCN's.

For this research, GraphSAGE was chosen due to its flexibility and its superior performance on many graph-related tasks. As its authors suggested [19], new sampling techniques can be easily adapted as well as an application-driven loss function. Alternative approaches that can also be adapted are described in [46] or [18].

3.1.2 GraphSAGE

As claimed by the authors of the paper [19] where it was first proposed, GraphSAGE is an inductive framework that generates node embeddings for previously unseen data. It can be seen as an extension of the GCN framework to the inductive setting but it results advantageous in this process as it:

- Performs localized convolutions without needing to use the whole normalized Laplacian resulting in a reduction of the computing time. Due to the nature of the GCN's, they make use of the symmetrically-normalized graph's Laplacian to compute localized convolutions.
- Does not require that all nodes are present during the embeddings' training which results in a highly-scalable framework.
- Naturally generalizes to unseen nodes and even to entire sub-graphs.
- Is capable of recognizing local and global structural properties of nodes based only on its neighbors even though it still depends on node features.
- Uses an unsupervised loss function that tries to preserve graph structure without being specific on the task.

The basic idea of this framework to compute an embedding for node v can be summarized by the following three steps:

1. Uniformly and randomly sample a set of nodes from the neighborhood of v.
2. *Aggregate* feature information from neighbors

Having this idea in mind, the simplest form of a layer-wise propagation rule for GraphSAGE is given by

$$\begin{aligned} \boldsymbol{h}_v^{(l)} &= \sigma\left(\boldsymbol{W}^{(l)} \cdot \textsc{Concat}\left(\boldsymbol{h}_v^{(l-1)}, h_{N_l(v)}^{(l)}\right)\right) \\ \boldsymbol{h}_{N_l(v)}^{(l)} &= \textsc{Aggregate}_l\left(\left\{h_u^{(l-1)} \mid u \in N_l(v)\right\}\right) \\ \boldsymbol{h}_v^{(0)} &= \boldsymbol{X}_v \end{aligned} \tag{3.3}$$

where as usual $\boldsymbol{W}^{(l)}$ is a the learnable parameter weight matrix for the l-th network layer and σ is a non-linear activation function, $\{\boldsymbol{X}_v \mid v \in V\}$ are the input features, $\textsc{Aggregate}_l$ are differentiable aggregator functions, and $N_l : v \to 2^V$ are neighborhood sampling functions.

After computing the node embedding for the l-th layer, the algorithm also performs a normalization step by dividing by the vector norm. This step is a common technique to prevent gradient explosion.

To learn the GraphSAGE parameters and get useful predictive node representations, the model can be trained in a fully unsupervised manner by using the following graph-based loss function:

$$J_G(\boldsymbol{z}_u) = -\log\left(\sigma(\boldsymbol{z}_u^T \boldsymbol{z}_v)\right) - Q \cdot \mathbb{E}_{v_n \sim P_n(v)} \log\left(\sigma(-\boldsymbol{z}_u^T \boldsymbol{z}_{v_n})\right) \tag{3.4}$$

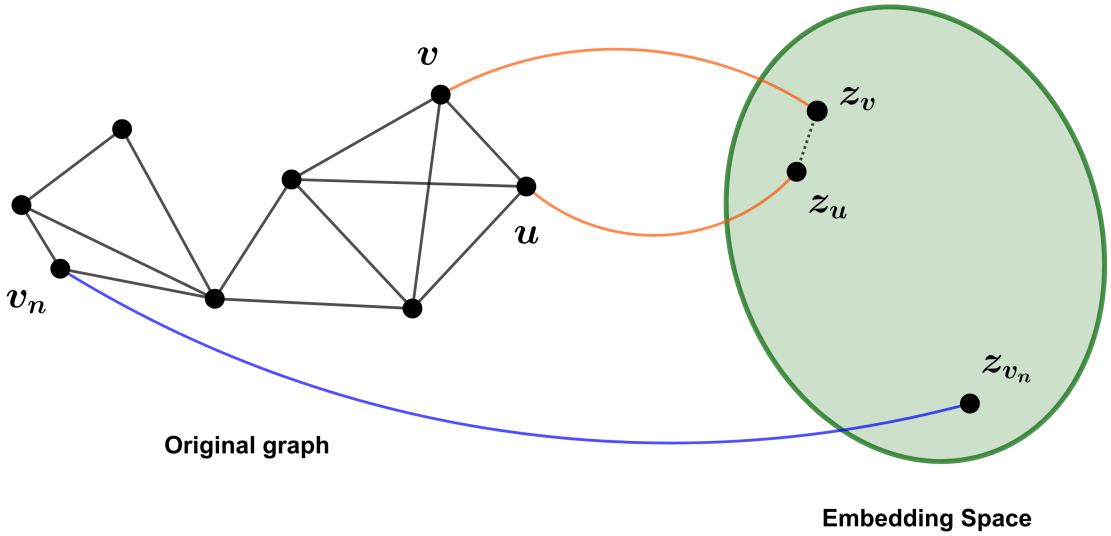


Figure 3.1: Graphical representation of the embeddings generated by GraphSAGE

As explained by Abraham [1], the supervised loss function given by Equation 3.4 is designed in a way that if nodes u and v are close in the original graph, their node embeddings should be similar. See Figure 3.1.

On one hand, if u and v are close, we expect the inner product of their representations z_u and z_v to be large and the first term of Equation 3.4 will be close to zero. On the other hand, if u and v are distant to each other, it is expected that their inner product is negative so the value of the second term will be close to zero. Since all distant nodes, also known as negative nodes, cannot be used to minimize the loss function, only Q of them are sampled from the distribution of negative nodes $P_n(v)$.

3.2 Node Features

For a graph $G = (V, E)$, where $V = \{v_1, v_2, ..., v_n\}$, the feature matrix X can be described as the $n \times F$ matrix where the rows X_{v_i} are node features that depends on the graph. For example, in the context of social networks, node features can be gender, city or any other user profile information.

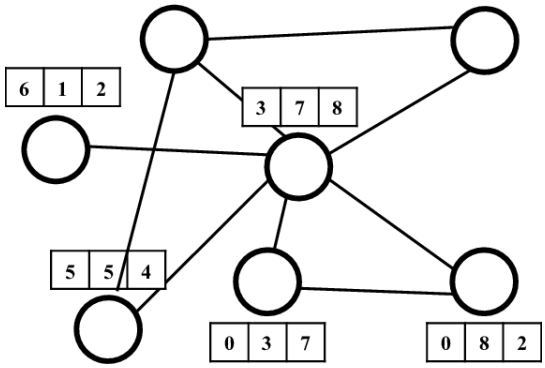


Figure 3.2: Graph with 7 nodes, five of them with 3 node features [11].

Many graphs from different applications come with very rich node feature information that can be very representative in the node embedding generation process. The key idea of Graph Neural Networks is to generate representations of nodes that depends not only on the structure of the graph but from any feature information of the graph.

As mentioned by Chen Wang et. al. [40], Graph Neural Networks (GNN's) aim at learning node representations by learning the similarities shared between connected nodes. However, the expressive ability of a GNN is highly dependent on the quality of node features.

In the version of the GP problem presented here, node features are irrelevant. The information that nodes or edges could contain does not determine if a node will belong to a specific partition or if an edge will be removed in the partitioned graph. Due to this fact and the inherent characteristic of GCN's before mentioned, one of the main limitations found in

the GAP framework is that it requires node features. Furthermore, in specific important GP applications, graphs do not have node features.

Learning inductive representations on graphs without node features it is still an open problem [46]. Nonetheless, for specific applications some efforts have been accomplished which generate new research opportunities. The next paragraphs are dedicated to study the impact of considering different sources of information as node features.

Two of the most common approaches where no node features are available are to use *identity features* and to use random feature initialization. However, those approaches have shown to be equivalent and to make the model incapable of generalizing to unseen nodes [18].

Another option is to use graph statistics like: node degree, node centrality, number of closed triangles or the clustering coefficient. Cai and Wang [4] proposed a degree-based approach. They used some statistics of the node's degree to generate the node features. Specifically, they used the feature vectors given by $(degree(v), \min(DN(v)), \max(DN(v)), mean((DN(v)), std(DN(v))$ where $DN(v) = \{degree(u) \mid (u,v) \in E\}$. This is an interesting research that showed prominent results.

Other methods propose using applying methods such as PCA to the adjacency matrix to extract the top k-eigenvalues [5, 22] but those approaches are computationaly very expensive, which is infeasible for large graphs.

In this work, to help capturing local information related to the graph's structure, a random walk approach was preferred. Given the the existent research, this type of approach looks very promising in terms of scalability to large graphs and structural representation capabilities. In experiments carried out independently by Duong et. al. [8] and Cui et. al. [7], who compared some of the already mentioned approaches, they concluded that matrix-decomposition-based methods are very powerful to extract positional node information. DeepWalk [30] produces a low-rank tranformation of the graph's normalized Laplacian matrix [31], i.e., DeepWalk is implicitly factorizing and reducing dimensionality. For that reason, DeepWalk is going to be used for this task.

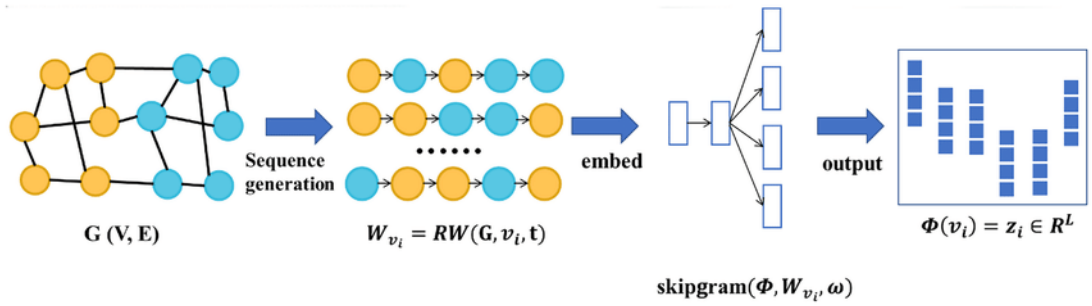


Figure 3.3: Summarized DeepWalk process for graph representation learning [44]

3.2.1 DeepWalk as feature extraction

Deep walk is graph representation method that can learn node embeddings by inferring local structures of the graph. This is an unsupervised learning algorithm that was inspired by the way traditional language models encode sequences of words. This algorithm is able to detect neighborhood similarities that can be used to distinguish densely-connected community structures [30, 43].

According to the original paper [30], the DeepWalk algorithm consists of two main components: a random walk generator and an update procedure. Those two components are described bellow.

In the first component, known as the *local structure stage*, a random node v_i is taken from the set of nodes uniformly and is set to be the root of a random walk W_{v_i}. Next, nodes are sampled recursively from the set of neighbors of the last-visited vertex, and they are added to the random walk until the maximum length t is reached. This process is repeated a number γ of times each time a root vertex is chosen.

The second component is known as the *skip-gram model*, and it is the analogous of the language model that maximizes the co-occurrence probability of words in a window. In the DeepWalk algorithm, a window size w is set to determine the context of a node given the random walks of the last stage. Then, a vector representation of dimension d is generated for each node and subsequently updated to maximize the probability of the neighbors that appear in the same random walk inside the same window. This last step can be performed using the Softmax function or Hierarchical Softmax to accelerate the training process [30].

DeepWalk is one of those encoders mentioned at the beginning of the chapter, it can only generate unique emmbedings for the nodes in a fixed graph. For the purposes of generating node features, it is adequate due to its topological representation capabilities.

CHAPTER 4

EXPERIMENTAL RESULTS

4.1 Technical details and implementation

playing with the parameters

4.1.1 Feature generation

[43] carried out a research for community detection using DeepWalk. After some experiments, they found convienent that for its problem the following parameters were eno As specified by Perozzi et. al. [30], their experiments suggest that the number of walks started per vertex should be greater or equal than $\gamma = 30$, the latent dimension greater or equal than $d = 64$, and they fixed the sensible values of $w = 10$ for the window size, and $t = 40$ for the walk length . Based on those recommendations, in the experiments carried out in [43], and the computational needs of the problem to be solved, it was found convenient to set $\gamma = 60$, $d = 64$, $w = 15$, and $t = 80$.

Taking into account the computational needs of the problem to be solved, and some

Saving and manipulating only the adjacency lists instead of the whole adjacency matrix, is an important

For the algorithm that is proposed here, the implementation by the Karate Club API [32] was used. The number of partitions was set to four differently from the original paper where three partitions were used. It was used this number of partitions to have a comparison point

with the result in The Graph Partitioning archive. There, the number of partitions is always a power of 2.

Quality metrics

METIS was used from the networkx interface

4.2 Dataset

For the dataset used to train and test the algorithm "The Graph Partitioning Archive" [34] The graphs are in the standard format used by JOSTLE and METIS. A description of the format and the requirements can be found in [39].

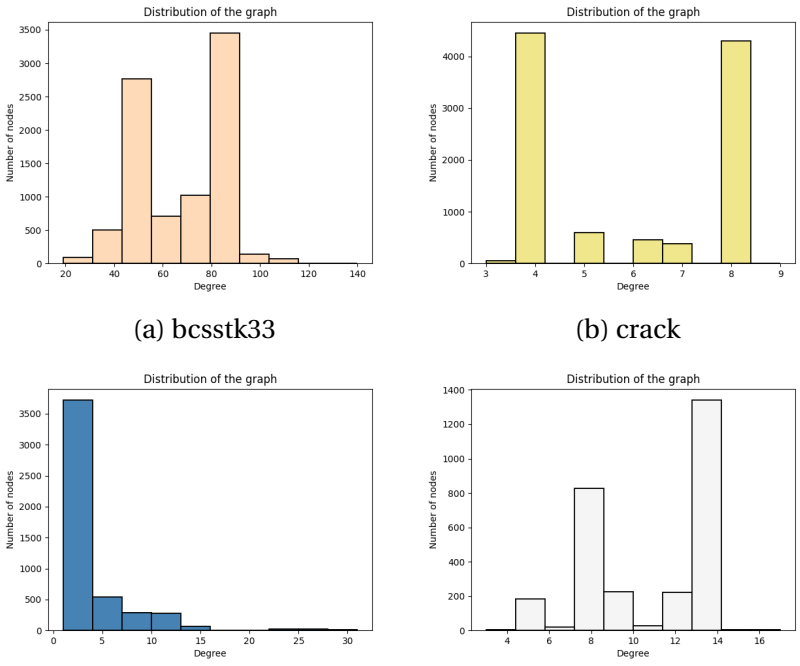
For the purposes of this research, the graphs contained in "The Graph Partitioning Archive" where categorized in one of the following categories according to their vertex cardinality:

- Tiny graphs: those with less than 10,000 nodes,
- Small graphs: those with node size between 45,000 and 75,000 and,
- Medium graphs: those with at least 99,000 nodes but no more than 500,000.

Ask Luis: is there a word for graphs with size more than big

Computation Graphs		
Name	**Nodes**	**Edges**
data	2851	15093
add32	4960	9462
bcsstk33	8738	291583
crack	10240	30380
fe_body	45087	163734
t60k	60005	89440
wing	62032	121544
finan512	74752	261120
fe_rotor	99617	662431
598a	110971	741934
m14b	214765	1679018
auto	448695	3314611

Table 4.1: Summary of the graphs characteristics. Taken from "The Graph Partitioning Archive" [34]



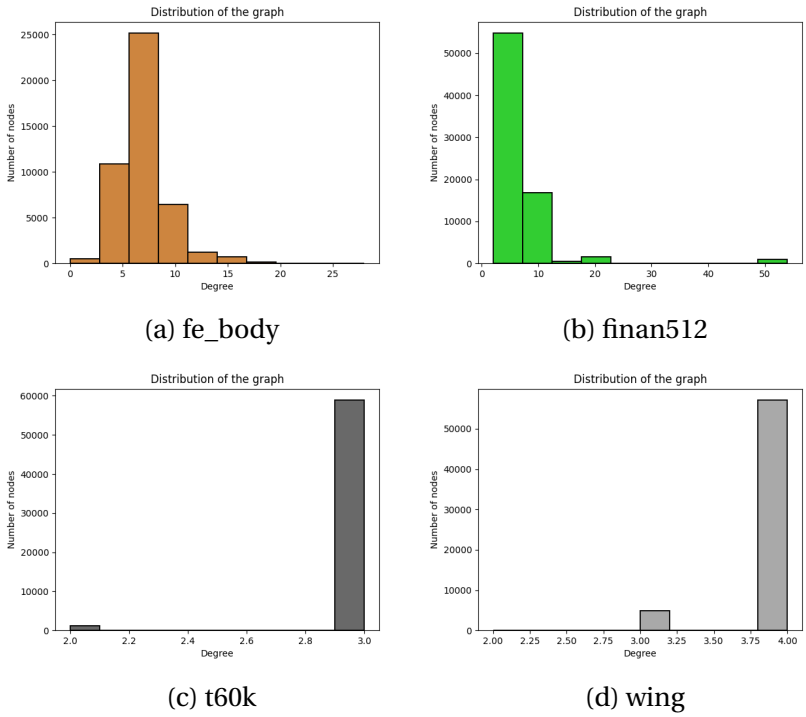
(a) bcsstk33 (b) crack

(c) add32 (d) data

Figure 4.1: Degree histogram of small graphs: bcsstk33, crack, add32, and data. Graphs are not so dense.

4.3 Results

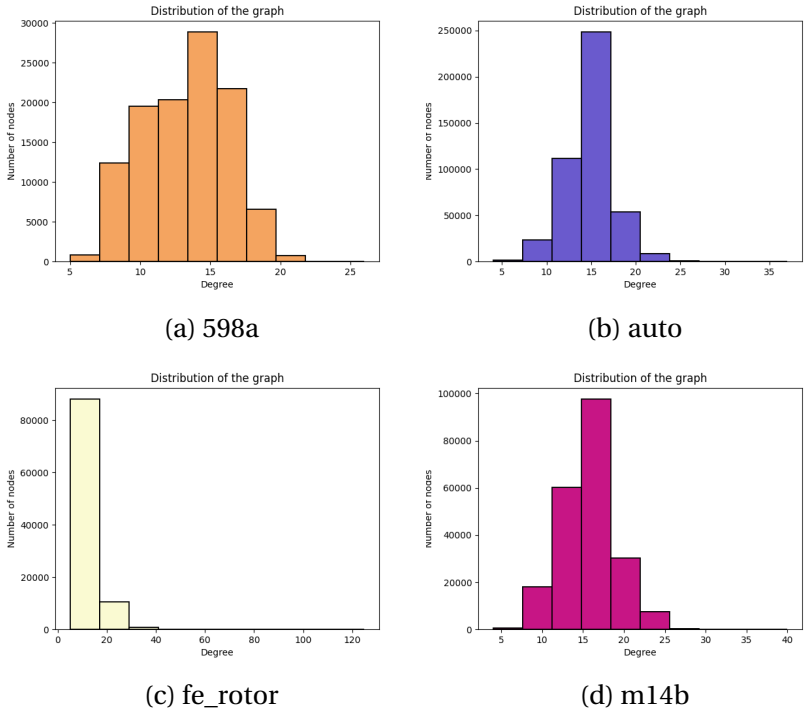
In table



(a) fe_body (b) finan512

(c) t60k (d) wing

Figure 4.2: Degree histogram of medium graphs. Graphs are not so dense.



(a) 598a (b) auto

(c) fe_rotor (d) m14b

Figure 4.3: Degree histogram of large graphs. Graphs are not so dense.

Graphs	**METIS**		**Modified GAP**	
Name	Edge Cut	Balancedness	Edge Cut	Balancedness
data	0	0	0	0
add32	0	0	0	0
bcsstk33	0	0	0	0
crack	0	0	0	0
fe_body	0	0	0	0
t60k	0	0	0	0
wing	0	0	0	0
finan512	0	0	0	0
fe_rotor	0	0	0	0
598a	0	0	0	0
m14b	0	0	0	0
auto	0	0	0	0

Table 4.2: Comparison of the results obtained by different algorithms in the computation graphs

CHAPTER 5

CONCLUSION

Future work extend to different problem statements, such as balanced graph partitioning with tolerance or weighted graph partitioning. Study the real impact on choosing the way to extract the features and try different algorithms look for new ways the algorithm relies only on topological graph's structure To generalize the algorithm and code to construct graphs from the features Explore new techniques for sampling Build a generic graph construction Training the algorithm with different graphs, look for useful training sets that allow to extend the recognition of different structural properties in graphs. For example, training the graph on only sparse or complete graphs Explore improvements in the training of the algorithm, parallel tasks or mix between CPU and GPU GraphSAGE still takes a lot of time to run on really large graphs Extend the accepted formats Study another way to extract features

5.1 Contributions

while it remains an open problem and it still very unexplored, the problem of feature generation...

5.2 Recommendations and future work

for future - Study the weighted problem Study better features related to the problem other ways to extract useful features for non-attributed graphs Study better ways to extract features

BIBLIOGRAPHY

[1] Nabila Abraham.
Graphsage and inductive representation learning.
`shorturl.at/oDEIY`, 2020.

[2] Giorgos Bouritsas, Andreas Loukas, Nikolaos Karalias, and Michael M. Bronstein.
Partition and code: learning how to compress graphs.
In *NeurIPS*, 2021.

[3] Dirk Brockmann and Dirk Helbing.
The hidden geometry of complex, network-driven contagion phenomena.
Science, 342(6164):1337–1342, 2013.

[4] Chen Cai and Yusu Wang.
A simple yet effective baseline for non-attribute graph classification.
arXiv preprint arXiv:1811.03508, 2018.

[5] Kamalika Chaudhuri, Fan Chung, and Alexander Tsiatas.
Spectral clustering of graphs with general degrees in the extended planted partition model.
Journal of Machine Learning Research, 23:35.1–35.23, 01 2012.

[6] Jie Chen, Tengfei Ma, and Cao Xiao.
Fastgcn: Fast learning with graph convolutional networks via importance sampling.
In *6th International Conference on Learning Representations, ICLR 2018, Vancouver, BC, Canada, April 30 - May 3, 2018, Conference Track Proceedings*. OpenReview.net, 2018.

[7] Hejie Cui, Zijie Lu, Pan Li, and Carl Yang.
On positional and structural node features for graph neural networks on non-attributed graphs.
ArXiv, abs/2107.01495, 2021.

[8] Chi Thang Duong, Thanh Dat Hoang, Haikun Dang, Quoc Viet Hung Nguyen, and Karl Aberer.
On node features for graph neural networks.
ArXiv, abs/1911.08795, 2019.

[9] K. Erciyes.
Graph Partitioning and ClusteringGraph partitioning, pages 199–218.
Springer International Publishing, Cham, 2021.

[10] Albert Einstein Muritiba Fernandes.
Algorithms and Models For Combinatorial Optimization Problems.
PhD thesis, 2010.

[11] Hongyang Gao, Zhengyang Wang, and Shuiwang Ji.
Large-scale learnable graph convolutional networks.
In *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '18, page 1416–1424, New York, NY, USA, 2018. Association for Computing Machinery.

[12] Alice Gatti, Zhixiong Hu, Pieter Ghysels, Esmond G. Ng, and Tess E. Smidt.
Graph partitioning and sparse matrix ordering using reinforcement learning.
ArXiv, abs/2104.03546, 2021.

[13] Alice Gatti, Zhixiong Hu, Tess Smidt, Esmond G. Ng, and Pieter Ghysels.
Deep Learning and Spectral Embedding for Graph Partitioning, pages 25–36.
2022.

[14] Xavier Glorot and Yoshua Bengio.
Understanding the difficulty of training deep feedforward neural networks.
In *AISTATS*, 2010.

[15] Anna Goldie, Sujith Ravi, and Azalia Mirhoseini.
A deep learning framework for graph partitioning.
2019.

[16] L. Grady and E.L. Schwartz.
Isoperimetric graph partitioning for image segmentation.
IEEE Transactions on Pattern Analysis and Machine Intelligence, 28(3):469–475, 2006.

[17] A. Gupta.
Fast and effective algorithms for graph partitioning and sparse-matrix ordering.
IBM Journal of Research and Development, 41(1.2):171–183, 1997.

[18] William L. Hamilton.
Graph representation learning.
Synthesis Lectures on Artificial Intelligence and Machine Learning, 14(3):1–159, 2020.

[19] William L. Hamilton, Rex Ying, and Jure Leskovec.
Inductive representation learning on large graphs.
In *Proceedings of the 31st International Conference on Neural Information Processing Systems*, NIPS'17, page 1025–1035, Red Hook, NY, USA, 2017. Curran Associates Inc.

[20] Máté Hegyháti.
Introduction into the modeling and optimization of linear systems.
`https://hegyhati.github.io/IMOLS/`, 2022.
Accessed: 2010-09-30.

[21] Karla L. Hoffman and Ted K. Ralphs.
Integer and Combinatorial Optimization, pages 771–783.
Springer US, Boston, MA, 2013.

[22] Qian Huang, Horace He, Abhay Singh, Ser-Nam Lim, and Austin Benson.
Combining label propagation and simple models out-performs graph neural networks.
In *International Conference on Learning Representations*, 2021.

[23] Vassilis G. Kaburlasos, Lefteris Moussiades, and Athena Vakali.
Fuzzy lattice reasoning (flr) type neural computation for weighted graph partitioning.
Neurocomputing, 72:2121–2133, 2009.

[24] Thomas N Kipf and Max Welling.
Semi-supervised classification with graph convolutional networks.
arXiv preprint arXiv:1609.02907, 2016.

[25] Bernhard Korte and Jens Vygen.
Combinatorial Optimization: Theory and Algorithms.
Springer Publishing Company, Incorporated, 5th edition, 2012.

[26] Henry Maltby and Eli Ross.
Combinatorial optimization.

https://brilliant.org/wiki/combinatorial-optimization/, 2022.

[27] Milko Mitropolitsky, Zainab Abbas, and Amir H. Payberah.
Graph representation matters in device placement.
Proceedings of the Workshop on Distributed Infrastructures for Deep Learning, 2020.

[28] David R. Morrison, Sheldon H. Jacobson, Jason J. Sauppe, and Edward C. Sewell.
Branch-and-bound algorithms.
Discret. Optim., 19(C):79–102, feb 2016.

[29] Azade Nazi, Will Hang, Anna Goldie, Sujith Ravi, and Azalia Mirhoseini.
Gap: Generalizable approximate graph partitioning framework.
ArXiv, abs/1903.00614, 2019.

[30] Bryan Perozzi, Rami Al-Rfou, and Steven Skiena.
Deepwalk: Online learning of social representations.
In *Proceedings of the 20th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '14, page 701–710, New York, NY, USA, 2014. Association for Computing Machinery.

[31] Jiezhong Qiu, Yuxiao Dong, Hao Ma, Jian Li, Kuansan Wang, and Jie Tang.
Network embedding as matrix factorization: Unifying deepwalk, line, pte, and node2vec.
In *Proceedings of the Eleventh ACM International Conference on Web Search and Data Mining*, WSDM '18, page 459–467, New York, NY, USA, 2018. Association for Computing Machinery.

[32] Benedek Rozemberczki, Oliver Kiss, and Rik Sarkar.
Karate Club: An API Oriented Open-source Python Framework for Unsupervised Learning on Graphs.
In *Proceedings of the 29th ACM International Conference on Information and Knowledge Management (CIKM '20)*, page 3125–3132. ACM, 2020.

[33] A. Schrijver.
Combinatorial Optimization - Polyhedra and Efficiency.
Springer, 2003.

[34] Alan Soper, Chris Walshaw, and Mark Cross.

A combined evolutionary search and multilevel optimisation approach to graph-partitioning.
Journal of Global Optimization, 29:225–241, 06 2004.

[35] Daniel A. Spielman and Shang-Hua Teng.
Spectral sparsification of graphs.
SIAM Journal on Computing, 40(4):981–1025, 2011.

[36] Daniel A. Spielman and Shang-Hua Teng.
A local clustering algorithm for massive graphs and its application to nearly linear time graph partitioning.
SIAM Journal on Computing, 42(1):1–26, 2013.

[37] Daniel A. Spielman and Shang-Hua Teng.
Nearly linear time algorithms for preconditioning and solving symmetric, diagonally dominant linear systems.
SIAM Journal on Matrix Analysis and Applications, 35(3):835–885, 2014.

[38] Zhonglin Sun, Yannis Spyridis, Thomas Lagkas, Achilleas Sesis, Georgios Efstathopoulos, and Panagiotis Sarigiannidis.
End-to-end deep graph convolutional neural network approach for intentional islanding in power systems considering load-generation balance.
Sensors, 21(5), 2021.

[39] Chris Walshaw.
Contents the jostle executable user guide: Version 3.1.
08 2005.

[40] Chen Wang, Yingtong Dou, Min Chen, Jia Chen, Zhiwei Liu, and Philip S. Yu.
Deep fraud detection on non-attributed graph.
CoRR, abs/2110.01171, 2021.

[41] Keyulu Xu, Weihua Hu, Jure Leskovec, and Stefanie Jegelka.
How powerful are graph neural networks?
In *International Conference on Learning Representations*, 2019.

[42] Rex Ying, Ruining He, Kaifeng Chen, Pong Eksombatchai, William L. Hamilton, and Jure Leskovec.
Graph convolutional neural networks for web-scale recommender systems.

In *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '18, page 974–983, New York, NY, USA, 2018. Association for Computing Machinery.

[43] Chen Yunfang, Li Wang, Dehao Qi, and Wei Zhang.
Community Detection Based on DeepWalk in Large Scale Networks, pages 568–583.
Springer Publishing Company, Incorporated, 08 2020.

[44] Boyao Zhang, Chao Yang, Haikuo Zhang, Zongguo Wang, Jingqi Sun, Lihua Wang, Yonghua Zhao, and Yangang Wang.
Graph representation learning for similarity stocks analysis.
Journal of Signal Processing Systems, 04 2022.

[45] Si Zhang, Hanghang Tong, Jiejun Xu, and Ross Maciejewski.
Graph convolutional networks: a comprehensive review.
Computational Social Networks, 6, 11 2019.

[46] Ziwei Zhang, Peng Cui, and Wenwu Zhu.
Deep learning on graphs: A survey.
IEEE Transactions on Knowledge and Data Engineering, PP:1–1, 03 2020.