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 son METIS o SCOTCH.

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 problem 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 and at the same time reduce the computation time without losing the power of the original framewor.

In addition, it was noted that GAP requires graphs with node features as a consequence of the model architecture chosen by its authors. 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 or SCOTCH.

Contents

List of Figures

LIST OF TABLES

CHAPTER 1

Introduction

This chapter provides to the reader the necessary information to get familiar with the present work. It intends to contextualize the reader with the topics in the subsequent chapters and show them the path that should be followed.

1.1 Presentation

Brilliant [11]

Techniques proposed citefernandes

Combinatorial Optimization is a prominent field that results from the intersection of mathematics and theoretical computer science. A problem of this The computational complexity behind solving Combinatorial Optimization problems is a very well known concern. Combinatorial Optimization [10]...

On one hand, it is necessary to keep researching for new theoretical results limitations and to more assumptions and use this knowledge to influence the way we approach that kind of problems. On the other hand, there is a need to solve those problems in real life and come up with practical and efficient solutions in a considerably small amount of time. The theoretical and practical are tied To deal with the computational complexity in solving problems of the Combinatorial Optimization type, different techniques have been proposed which are based on optimization strategies like Heuristic Methods or Integer Programming, just to mention a few.

The ability to address the underlying issues that arise in this kind of problems has been improved drastically during the last years. Those huge steps forward towards need for fast approximations for

In terms of mathematical results and computing power, the technological progress that has taken place during the last decade has given rise to new ways to find algorithms that provide efficient solutions, particularly for graph-related Combinatorial Optimization problems.

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 determined 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 of those who has gotten the best from Deep Learning.

application to solve time consuming and computationally too expensive to get solutions

1.2 Objectives

1.2.1 General objective

- To develop an machine learning algorithm that solves the graph partitioning problem that works for large-scale graph instances.

1.2.2 Particular objective

- To understand and analyze the Generalizable Approximate Partitioning (GAP) framework and to use it for solving the graph partitioning problem.
- Based on GAP, design an algorithm 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 mathematical representations of what is colloquially known as networks. They are used to describe the relationships between objects which is adopted as a similitude measure and allows 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 mathematics and computer science.

Applications of graph partitioning to real life:

Graph partitioning plays an essential role in paralleling computations and the design of new algorithms on large graphs. For example device placement problem where one aims to distribute work accross multiple devices and have applications in Deep Learning to train Neural Networks accross multiple devices [12]

problem of intentional islanding in power systems considering load generation balance [17]

Image segmentation [6] Applications of graph partitioning to solve graph-related problems:

Graph partitioning it is used as a subrutine in tasks as graph compression [1]

1.4 Project scope and limitations

During the development of this project

- One of the first limitation found was the sizes of the dataset used to test the algorithm. Most of them were taken from "The Graph Partitioning Archive" and it do. While using this dataset offers a good comparison point, the algorithm have not been tested on biggest real-world graphs.
- In terms of edge size
- Values based on previous work
- What assumptions do I make?
- not distributed work which could improve
- It only accepts a certain format SCOTCH and Networkx

- Comparison with GAP was using graphs without features, an interesting thing would be to check performance with the same datasets they used but getting rid of the features, e.g., cora citation

1.5 Research Problem

The state-of-the-art algorithms for graph partitioning Graph partitioning algorithms based on DL

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

1.6 Hypothesis

When analyzing GAP in depth some observations were made.

1.7 Project organization

CHAPTER 2

THEORY AND CONCEPTUAL FRAMEWORK

2.1 Preliminaries

Description of concepts

Definition 2.1.1. *orthogonal complement*

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*

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

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

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

The notion of cut allows to measure the quality of any partition, nevertheless solving the min cut problem ...

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

$$\textsc{Vol}(S) := \sum_{v_i \in S} d_i$$

we want to agroupe by similarity so its natural to Cut value of that partition

solve the mincut problem

The next consider two different ways of measuring the size of the partitions

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

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

2.2.4 Spectral partitioning and Normalized Cut

Here will be presented the derivation of the normalized cut as relaxation of the problem to solve the spectral partitioning problem

2.3 Literature review

B-GRAP: BALANCED GRAPH PARTITIONING ALGORITHM FOR LARGE GRAPHS: https://www.rintonpre
2/116-135.pdf

Trading Quality for Efficiency of Graph Partitioning: An Inductive Method across Graphs https://openreview.net/forum?id=e6MWIbNeW1

Application Driven Graph Partitioning https://dl.acm.org/doi/abs/10.1145/3318464.3389745

2.3.1 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 [5].

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. [13], 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.

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.

2.3.1.1 Expected Normalized Cut Loss Function

Recall the normalized cut given by

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

In order to calculate the normalized cut expected value, one needs to compute the expected value of $\textsc{Cut}(P_k, \overline{P_k})$ and $\textsc{Vol}(P_k)$ from Equation 2.2. For the deduction of those quantities, an approach similar to the one presented in [3] 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

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

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:

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

Keeping in mind that $\textsc{Vol}(P_k)$ 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 $\textsc{Vol}(P_k)$ as follows:

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

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

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

Nazi et al. [13] 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:

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

2.3.1.2 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 [4]

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

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.

2.3.1.3 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

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

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 the principles to understand them.

3.1 Node Embeddings

As mentioned by Zhang et. al. [20], GCN's and its variants have become a very hot topic in Machine Learning and they are being extremely used to solve plenty of problems. Indeed, the use use of Graph Convolutional Networks (GCN) to generate node embeddings was a key factor that made the GAP framework very successful in producing good quality partitions.

3.1.1 Graph Convolutional Networks

Graph Convolutional Networks were originally proposed by Kifp et. al. [9]. When they first came up with its 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 Laplacian.

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

- In terms of efficiency, GCN's are . As it can be seen in Equation 3.1, they multiply feature matrices by powers of the graph's normalized Laplacian, i.e., they are computationally very expensive.
- Over-smoothing
-

To solve this issues in GCN's, several solutions have been proposed that deal with the downsides mentioned without loosing expression of GCN. FastGCN GraphSAGE PinSAGE

Different approaches that can be followed are found by the author in [21] or [7]

Describe Downsides

Mention all the approaches that use Graph Neural Networks and its importance in solving graph related tasks achieving superior performance on many graph-related tasks

Talk a little bit about traditional node embeddings approaches and its delimitations. Embeddings: dense vector representations Talk about the message passing framework Start with Grpah Convolutional Neural Networks (GCN) and its limitations. Emphasize in how GraphSAGE solve those limitations and how it extends the GCN capabilities

3.1.2 GraphSAGE

As claimed by the authors of the paper [8] where it was first proposed, GraphSAGE is an inductive framework that generate 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 Laplacian localized convolutions.
- Does not require that all nodes are present during training of the embeddings which turns out in a highly scalable framework.

- Naturally generalize to unseen nodes and even complete sub-graphs.
- Is capable of recognizing local and global structural properties of nodes based only on its neighborhood 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. Sample a set of nodes from the neighborhood of v uniformly at random
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:

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

3.2 Node Features

Many graphs come with very representative node feature information that can be very enlightening in the node embedding generation process. The key idea of Graph Neural

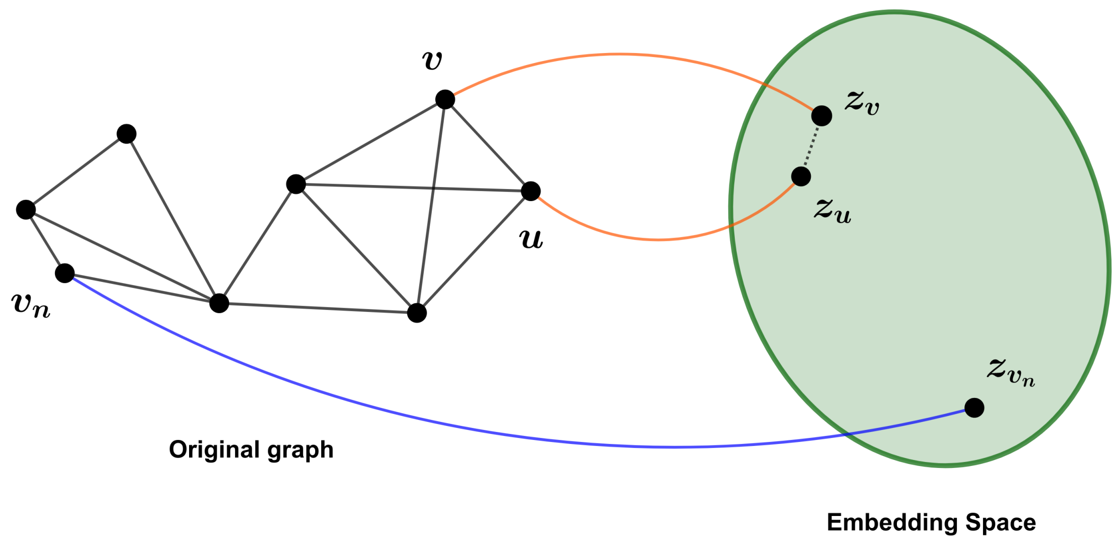


Figure 3.1: Graphical representation of the embeddings generated by GraphSAGE

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.

Citing textually to Chen Wang et. al. [18], 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.

An important thing to consider is that for solving the graph partitioning problem in the version presented here, node features are irrelevant.

Due to this inherent characteristic of GNN's, and particularly of GCN's, one of the main limitations of GAP is that it requires 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 that comes depending on the graph.

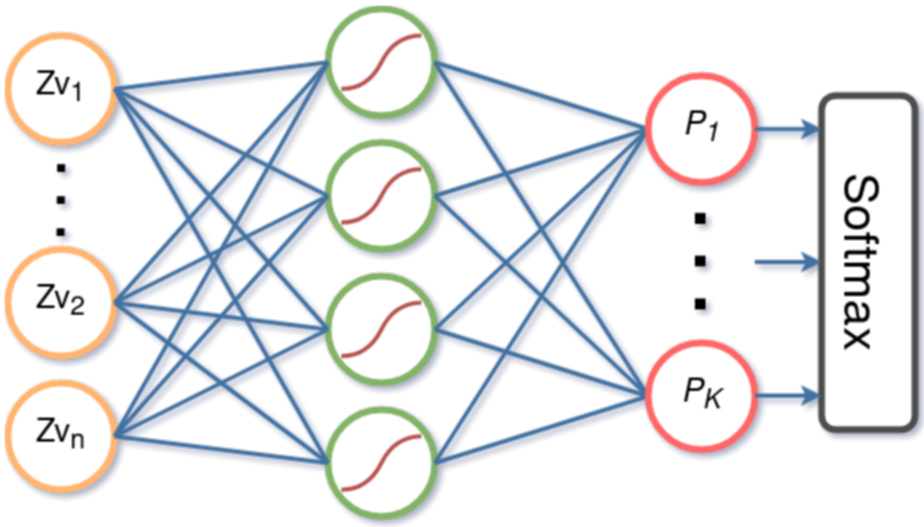
The way we generate node features independing task and also helps with the overall running time of the algorithm.S

GNNs aim Mention here: Deep Fraud Detection on Non-attributed Graph https://arxiv.org/pdf/2110.0 but cited here C. T. Duong, T. D. Hoang, H. T. H. Dang, Q. V. H. Nguyen, and K. Aberer, "On node features for graph neural networks," arXiv preprint arXiv:1911.08795, 2019. [11] H. Cui, Z. Lu, P. Li, and C. Yang, "On positional and structural node features for graph neural networks on non-attributed graphs," arXiv preprint arXiv:2107.01495, 2021

Eigendecomposition and top-k-eigenvalues are the k-dimensional feature vector Q.

Huang, H. He, A. Singh, S.-N. Lim, and A. R. Benson, “Combining label propagation and simple models out-performs graph neural net- works,” arXiv preprint arXiv:2010.13993, 2020.

Different feature initializations http://www.cs.emory.edu/ jyang71/files/gnnfeature.pdf



and after playing a little bit

although GCN have been shown to extract very useful information about graph topology we need to find a different way to get that information

Some work on using Graph Neural Networks without features has been made. Use one hot representation or Initialize random feature matrices (Those methods have been shown to be equivalents and they do not generalize as mention in [7]. Other methods propose using applying methods as PCA to the adjacency matrix to extract the top k-eigenvalues but those approaches are computationaly very expensive, which is infeasible for large graphs and only and does not To help capturing local information related to the graph's structure a random walk approach was chosen, have been shown efficient representation learning techniques for graphs https://arxiv.org/pdf/1901.01346.pdf, random walk kernels to produce high quality graph representations https://proceedings.neurips.cc/paper/2020/file/ba95d78a7c942571185308775a97a3a0-Paper.pdf

in particular Deep Walk to extract features about the topology of the network

According to the original paper [14], the DeepWalk algorithm consists of two main components: a random walk generator and an update procedure.

In the first component, a random node v_i is taken uniformly at random to be the root of a random walk $\mathcal{W}_{v_i}$ which in its turn samples recursively from the neighbors of the last visited vertex until the maximum length γ is reached.

As specified by Perozzi et. al. [14], 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 [19], and the computational needs of the problem to be solved, it was found convenient to set $\gamma = 60$, $d = 64$, $w = 15$, and $t = 80$.

For the algorithm that is proposed here, the implementation by the Karate Club API [15] was used.

Traditional approaches like METIS or SCOTCH implements different versions of the algorithm according to the balancedness measure, either the cardinality or the volume of the partitions. One of the great innovations presented in [13] is the introduction of a loss function derived from the expectation of the normalized cut. Even though they

Ratio cut it is still used and relevant to modern research cite here https://proceedings.neurips.cc/pape
Paper.pdf

or here https://www.springerprofessional.de/en/metaheuristic-approaches-for-ratio-cut-and-normalized-cut-graph-/20360832

the authors of [3] proposed some modifications to GAP so it can be used for graphs without features. However the presented

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

METIS was used from the networkx interface

a table for commonly used notation

3.2.1 PinSAGE and Markov Chain Negative Sampling (MCNS)

CHAPTER 4

EXPERIMENTAL RESULTS

The number of partitions was set to three

For the dataset used to train and test the algorithm "The Graph Partitioning Archive" [16]

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

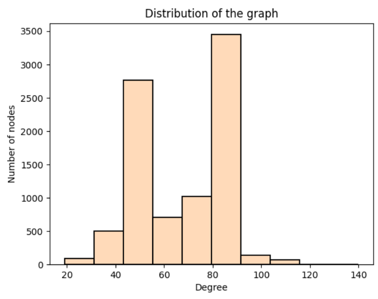
In table

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

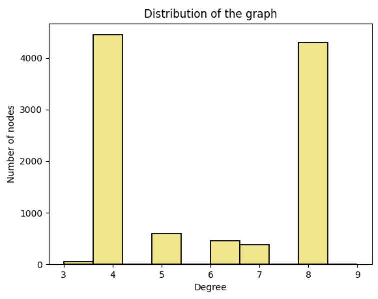
Table 4.1: Summary of the graphs characteristics. Taken from "The Graph Partitioning Archive" [16]

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

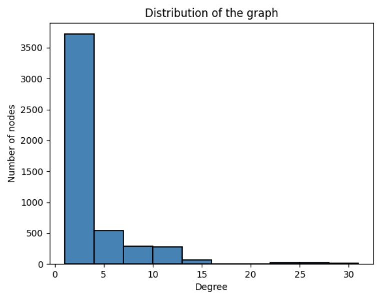
Table 4.2: Comparison of the results obtained by different algorithms in the computation graphs



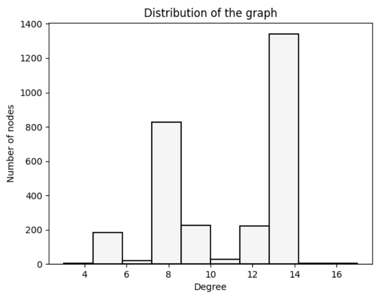
(a) bcsstk33



(b) crack

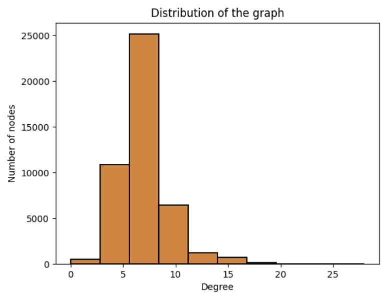


(c) add32

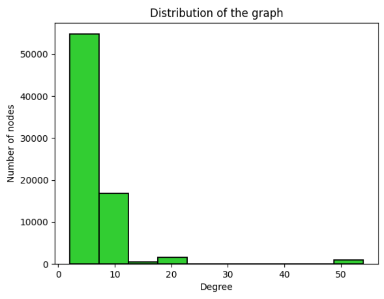


(d) data

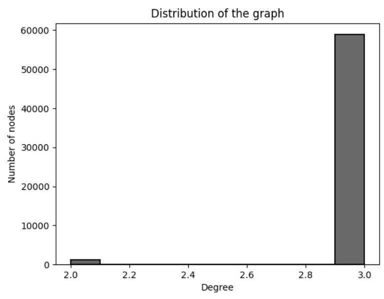
Figure 4.1: Degree histogram of small graphs: bcsstk33, crack, add32, and data. Graphs are not so dense.



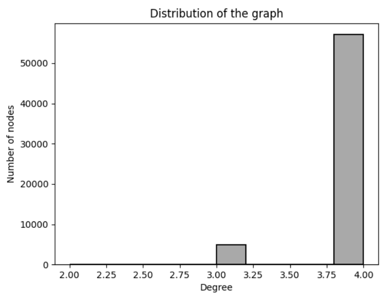
(a) fe_body



(b) finan512

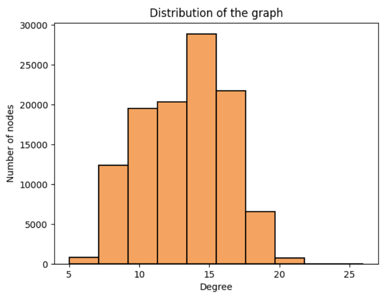


(c) t60k

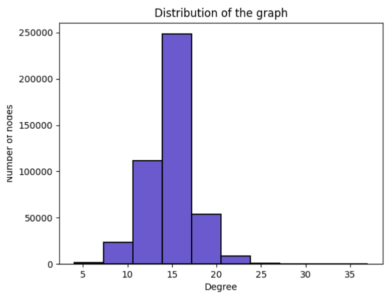


(d) wing

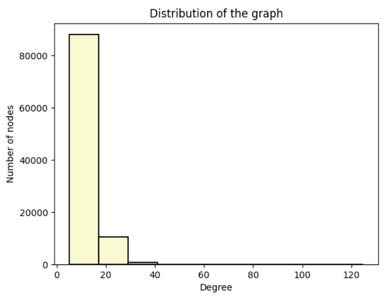
Figure 4.2: Degree histogram of medium graphs. Graphs are not so dense.



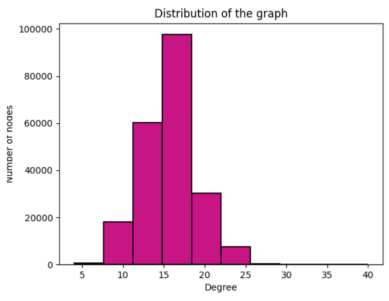
(a) 598a



(b) auto



(c) fe_rotor



(d) m14b

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

CHAPTER 5

Conclusion

Las conclusiones y el trabajo a futuro inicia aquí...

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

GraphSAGE still takes a lot of time to run on really large graphs

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

Bibliography

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

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

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

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

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

[6] 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.

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

[8] 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.

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

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

[11] Henry Maltby and Eli Ross.
Combinatorial optimization.
`https://brilliant.org/wiki/combinatorial-optimization/`, 2022.

[12] 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.

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

[14] 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.

[15] 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.

[16] 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.

[17] 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.

[18] 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.

[19] 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.

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

[21] 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.