A Sketch-based Approach towards Scalable and Efficient Attributed Network Embedding

A Thesis Report

submitted by

TAPADEEP CHAKRABORTY

Under the Supervision of

Dr. Debajyoti Bera

in partial fulfillment of the requirements

for the award of the degree of

MASTER OF TECHNOLOGY

COMPUTER SCIENCE AND ENGINEERING

INDRAPRASTHA INSTITUTE OF INFORMATION TECHNOLOGY DELHI

NEW DELHI- 110020

December 21, 2021

THESIS CERTIFICATE

This is to certify that the thesis titled "*A Sketch-based Approach towards Scalable and Efficient Attributed Network Embedding*" being submitted by **Tapadeep Chakraborty** to the Indraprastha Institute of Information Technology Delhi, for the award of the degree of Master of Technology, is an original research work carried out by him under my supervision. In my opinion, the thesis has reached the standards fulfilling the requirements of the regulations relating to the degree.

The results contained in this thesis have not been submitted in part or full to any other university or institute for the award of any degree/diploma.

Dr. Debajyoti Bera
Thesis Supervisor
Associate Professor
Dept. of Computer Science and Engineering
IIIT Delhi, 110020

Place: New Delhi
Date: December 21, 2021

ACKNOWLEDGEMENTS

At the end of a year-long project, it will hardly do justice to acknowledge everyone who played a part in this endeavor on a single page. Nevertheless, here is my attempt. First and foremost, I am incredibly humbled and thankful to my advisor Dr. Debajyoti Bera, for offering this project and guiding me throughout. From learning about the research area of Attributed Network Embedding to coming up with one of the fastest embedding schemes in the market, I can undoubtedly say that I have learned a lot. I remember reading the first few papers on the subject back in January of this year (2021) and being startled and wondering if I was even up for it, but Dr. Bera was always there to guide me throughout, responding promptly and enthusiastically to the numerous emails and doubts. Through the past year and a half, I have learned a great deal from him and his guidance, which I intend to carry forward in the future.

The second person I would like to thank for this project is actually not a single person but a division- IT Service Portal, IIITD. Since this project relied heavily on the computing infrastructure, the IT services of the college were almost instrumental to this study. I would especially like to extend my gratitude to Mr. Bhawani Shah from the IT Department for promptly addressing my many requests.

Furthermore, I would like to thank all the people who played even a tiny part in the making of this project, like providing the data for this project, providing access to the servers to run the code, and so on. I would also like to acknowledge all my teachers at IIITD for the brilliant courses I have so thoroughly enjoyed over here and which has, in turn, helped me make this study a success.

Finally, I would like to thank the CSE Department and my college Indraprastha Institute of Information Technology, Delhi, for the opportunity to study in such a brilliant atmosphere and my friends and parents who encouraged me to take up this thesis.

"*Essentially, all models are wrong, but some are ~~useful~~ fast.*"

- George E. P. Box and me

ABSTRACT

With the advent of big data, graphs have gained popularity as one of the most efficient data storage mechanisms. A graph can not only capture relationships between entities, but it can also store attributes associated with entities in the form of attributed nodes. This makes graphs quite a versatile data structure. Attributed network embedding refers to the task of representing each node of a graph as a low-dimensional vector so that it captures its neighborhood associations and attribute information. A downstream machine learning algorithm can use such an embedding to perform node classification, link prediction, and community detection tasks. Several learning-based methods were recently proposed that can produce high utility embeddings, but they scale poorly in terms of embedding space and embedding time with respect to network size, and stutter for massive billion-scale networks.

Our study addresses this problem by introducing BGENA (Binary-embedding GENerator for Attributed graphs), which uses a recently proposed fast and utility-preserving sketching method BinSketch along with a novel edge propagation mechanism to generate binary embeddings of each node. BGENA is designed to preserve any arbitrary order of proximity of nodes within its embedding. As a result of using only fast bitwise operations for the entire embedding process, BGENA achieves anywhere between $10\times$ to $100\times$ speedup compared to some existing methods. BGENA's binary embeddings allow for efficient bit-array/sparse-matrix representations to save space, making it four to eight times better in terms of the system's memory requirement. We also propose its parallelized version named PBGENA (Parallelized BGENA), which uses MPI to leverage the multi-core architecture of a system to further accelerate the embedding speed to nearly $16\times$ over BGENA.

PBGENA produced embedding results for all our graphs with 20,000 or fewer nodes in less than a second using an AMD 32-Core 3.2GHz server, and it did the job for TWeibo, a graph with over 2 million nodes and 50 million edges, in less than two minutes.

Further, BGENA is the only method known to us that was able to embed MAKG, a graph with nearly 60 million nodes and a billion edges, within the 270GB memory cap of the system in just 8 hours with comparable accuracy. We evaluate PBGENA embeddings on tasks like node classification, link prediction, and graph visualization with several real-world networks of varied sizes, and outperform the state-of-the-art baselines in performance, often by large margins and at a fraction of the time. Our experiments found that specific embedding methods prefer particular graphs where the results are in the top echelon but underperform significantly for other graphs. However, after hyperparameter tuning, no such effects were observed for PBGENA. All of these make PBGENA a robust, high-utility, cost-effective, and low space budget embedding method.

Keywords: Attributed Network Embedding, Network Representation Learning, Node Embedding, Sketching, Edge Propagation, Node Classification, Link Prediction, BinSketch, Parallelization, Message Passing Interface

ABBREVIATIONS

ACM	Association for Computing Machinery
AdONE	ADversarial ONE
AMD	Advanced Micro Devices
ANE	Attributed Network Embedding
ANRL	Attributed Network Representation Learning
API	Application Program Interface
ASNE	Attributed Social Network Embedding
AUC	Area Under Curve
BANE	Binarized Attributed Network Embedding
BFS	Breadth-First Search
BGENA	Binary embedding GENerator for Attributed graphs
BinSketch	BINary data SKETCHing
CABIN	CAtegorical data embedding using BINsketch
CAN	Co-embedding Attributed Networks
CBE	Circulant Binary Embedding
ComE	COMmunity Embedding
CPU	Central Processing Unit
CSE	Computer Science and Engineering
CUDA	Compute Unified Device Architecture
DFS	Depth First Search
DONE	Deep Outlier aware attributed Network Embedding
DOPH	Densified One Permutation Hashing
EPYC	Extreme Performance Yield Computing
FN	FeatherNode
GB	GigaByte
GNN	Graph Neural Network
GPU	Graphics Processing Unit
GraphSAGE	GRAPH SAmple and aggreGatE

HPC	High-Performance Computing
IANRW	Improved Attributed Node Random Walks
IBM	International Business Machines
IIITD	Indraprastha Institute of Information Technology, Delhi
IISc	Indian Institute of SCience
I/O	Input/Output
IPC	Inter-Process Communication
KDD	Knowledge Discovery in Databases
LDA	Linear Discriminant Analysis
LINE	Large-scale Information Network Embedding
LINQS	Lise's INQuisitive Students
LQANR	Low-bit Quantization for Attributed Network Representation learning
MAKG	Microsoft Academic Knowledge Graph
METIS	Serial Graph Partitioning and Fill-reducing Matrix Ordering
MHz	MegaHertz
ML	Machine Learning
MPI	Message Passing Interface
mpi4py	Message Passing Interface For Python
MUSAE	MUlti-Scale Attributed node Embedding
NLP	Natural Language Processing
NRL	Network Representation Learning
PANE	Parallel Attributed Network Embedding
ParMETIS	Parallel Graph Partitioning and Fill-reducing Matrix Ordering
PBGENA	Parallelized Binary embedding GENerator for Attributed graphs
PCA	Principal Component Analysis
PPI	Protein-Protein Interactions
QUINT	QUick thIN and biTty mapping
RAM	Random-Access Memory
RandNE	iterative RANDom projection Network Embedding
RDF	Resource Description Framework
ROC	Receiver Operating Characteristic
SAGE2VEC	Simple Attributed Graph Embedding to VECtor
SINE	Scalable Incomplete Network Embedding

SNAP	Stanford Network Analysis Project
TADW	Text-Associated DeepWalk
TENE	Text Enhanced Network Embedding
TF/IDF	Term Frequency/Inverse Document Frequency
t-SNE	T-distributed Stochastic Neighbor Embedding
TWeibo	Tencent WEIBO

NOTATIONS

$\mathbf{\Pi}$	Random Mapping Function between two sets
$\boldsymbol{\pi}$	Random Mapping Function from a categorical set to a boolean
$\boldsymbol{\psi}$	Maximum Sparsity among the rows of a binary matrix
$\boldsymbol{G}$	An undirected graph
$\boldsymbol{V}$	The vertex set of a graph
$\boldsymbol{E}$	The adjacency matrix/list of a graph
$\boldsymbol{A}$	The attribute matrix of a graph
$\boldsymbol{a}$	Number of attributes
$\boldsymbol{N}$	Embedding dimension
$\boldsymbol{\alpha}$	Attribute fraction
$\boldsymbol{b_t}$	Topology bitset probability
$\boldsymbol{b_a}$	Attribute bitset probability
$\boldsymbol{l_t}$	Number of levels in topology propagation
$\boldsymbol{l_a}$	Number of levels in attribute propagation
$\boldsymbol{f_t}$	Reduction fraction for b_t
$\boldsymbol{f_a}$	Reduction fraction for b_a
$\boldsymbol{emb}$	Output embeddings
$\boldsymbol{N_t}$	Topology dimension
$\boldsymbol{N_a}$	Attribute dimension
$\boldsymbol{S_t}$	Topology sketches
$\mathbf{\Pi}_t$	Topology mapping function
$\boldsymbol{S_a}$	Attribute sketches
$\mathbf{\Pi}_a$	Attribute mapping function
$\boldsymbol{E_t}$	Topology embedding
$\boldsymbol{E_a}$	Attribute embedding
$\boldsymbol{p}$	Number of processors
$\boldsymbol{rank}$	Processor identification
$\mathbf{\Phi}$	Mapping between nodes and processors
$\boldsymbol{in-edge}$	An edge whose vertices belong to the same partition
$\boldsymbol{cross-edge}$	An edge whose vertices belong to different partitions
$\boldsymbol{Ts}$	Time complexity for serial algorithm BGENA
$\boldsymbol{T_p}$	Time complexity for parallel algorithm PBGENA
$\boldsymbol{\mathcal{S}}$	Speedup of a parallel algorithm over its serial version
$\boldsymbol{\mathcal{E}}$	Efficiency of a parallel algorithm over its serial version
$\boldsymbol{m}$	Maximum degree of all nodes in a graph

LIST OF TABLES

LIST OF FIGURES

TABLE OF CONTENTS

CHAPTER 1

INTRODUCTION

Graphs are perhaps the most naturally occurring data structure in the real world. Any system with entities and interactions can be easily represented using graphs because graphs are abstract data types with vertices and edges capable of modeling complex, non-linear relationships between objects [1]. The most obvious example of a network encompassing all of us is the world wide web but it is hardly the only one. A few common examples of graphs in every day life include online social networks [2], road networks [3], citation networks [4], or even networks originating from protein interactions in human tissue [5] to name a few. Graphs, being omnipresent in the real world, have motivated the problem of Network Representation Learning (NRL). NRL refers to the task of learning low-dimensional representations of the network's vertices while preserving the network's topological structure and attributes [6]. This chapter will attempt to introduce and motive the problem of network representation learning and then present our contribution to the problem.

1.1 Motivation

Attributed Network Embedding (ANE), also known as NRL, is an essential task at the heart of graph mining which plays a pivotal role in many emerging applications. One application of network analysis is to predict which users one user might follow in social networks based on their past profile activity and connections. Such a recommendation system can not only be used to discover like-minded individuals online but also help in targetted advertising and refining personalized user search [7]. ACM had hosted its annual KDD cup in 2012 based on graph data obtained from the Chinese microblogging website Tencent Weibo. Politecnico di Milano and Oracle Labs had also organized a Kaggle competition based on multi-label vertex classification three years back on graph data obtained from protein interactions. Another application of ANE solvers lies in communication networks, where detecting community structures can help understand how information is spread. In biological networks, predicting the role of different proteins can accelerate drug discovery, and in chemistry networks, predicting the role of

the chemical structures can help us synthesize new compounds [6, 7]. All these tasks fall under the category of community detection and node classification, which are two of the most fundamental tasks in graph analysis.

Another well-studied problem in the area of NRL is link prediction, first introduced by Liben-Nowell et al. [8] in 2004, for addressing the dynamic nature of most real-world graphs. In the real world, graphs are never static and constantly evolving, so the link prediction task tries to quantify a model's ability to predict future links given a present snapshot of the network. Typically, this problem is formulated by randomly removing a percentage of edges from the graph and further generating dubious edges, and then testing the model's ability to correctly classify the actual unseen edges from the dubious ones. Just like community detection and node classification, link prediction has tremendous utility for emerging systems.

Other less known graph tasks include using embeddings for graph visualization [9] and performing attribute inference [10]. Graph visualization aims to generate meaningful visualizations through the node embeddings of the graph for visualizing class separation and to have a better idea of the natural clusters present in the graph. Attribute Inference, on the other hand, is the problem of predicting which attributes are most pronounced in a node.

1.2 Problem Statement

Given a graph $G(V, E, A)$, where V, E, and A are the vertex set, adjacency matrix, and attribute matrix of G, the task of attributed network embedding is to produce an embedding $emb : v \in V \mapsto \mathbb{R}^{|V| \times N}$; where N is the number of dimension of the embeddings and typically $N << |V|$.

1.3 Overview

Various real-world graphs often exhibit characteristics like non-linearity, sparseness, dynamicity, and heterogeneity [11] which make the task of NRL especially challenging. Even though great strides have been made in GNNs recently [12], most simple machine learning models still find it difficult to learn directly from graphical data. The primary

issue lies in the scale of the problem. Therefore scientists [13] and engineers [14] have come up with the notion of node embedding. Node embedding is closely related to the task of dimension reduction [15]. An embedding of a node in a graph is required to preserve both the topological and attribute information of each node in a network. A few desirable properties of a good embedding are that the nodes which are close to one another in the graph should also be close in the vector space of their embeddings. Similarly, two nodes sharing a large number of mutual neighbors should be close in the vector space of their embeddings. We further desire similar properties with respect to the attributes of each node in the network.

Many old factorization and learning-based methods stutter at large-scale graphs and is not a viable solution in the future for learning from massive graphs, therefore there has been a recent push for fast node embedding schemes [16, 17, 18]. Our method, PBGENA, is one such effort to provide a fast, alternate embedding scheme for massive graphs. PBGENA is not the first sketching-based embedding method; NetHash [17] is another attempt at trying to use MinHash [19] sketches for attributed network embedding. We believe that in a world of rapidly increasing connectivity, billion-scale graphs would become commonplace. In such a future, any learning-based embedding scheme would be rendered unworkably slow. Therefore, fast sketching-based methods is the way to go. To that effect, we have proposed a novel graph embedding method named **BGENA** (**B**inary embedding **GEN**erator for **A**ttributed graphs) that uses a binary dimension reduction method named BinSketch and a novel edge propagation mechanism to generate graph embeddings that preserves both topological and attributed features of a node in a graph. BGENA is nearly 10 to 100 times faster than most commonly used embedding schemes but remains competitive with all the modern ANE solvers in terms of its performance. We further propose a parallel version of BGENA named **PBGENA** (**P**arallelized **B**inary embedding **GEN**erator for **A**ttributed graphs), which makes use of a system's multi-core architecture with the help of MPI protocol to accelerate PBGENA. PBGENA is also perfect for graphs stored in a distributed manner.

To reduce the time needed for edge propagation we have exclusively used bitwise operations for the entire process. PBGENA is not just efficient in terms of its speed but also in terms of its memory requirement because PBGENA embeddings are binarized, that can be stored efficiently in bitarray and sparse matrix-like data structures. We have

demonstrated how PBGENA is capable of preserving any arbitrary order of proximity between two nodes when generalized by repeated level-wise edge propagation. We have also analyzed the space and time complexities for BGENA, and have shown that it is linear in terms of the graph structure. Finally, we have performed a rich array of experiments to demonstrate the superiority of PBGENA both in terms of performance and speed. These experiments include node classification, link prediction, graph visualization, and parameter sensitivity tests.

1.4 Contributions

We have compiled all the relevant information regarding our proposed methods of BGENA and PBGENA in the subsequent chapters, and this is a list of our major contributions to the field of NRL:

- We have proposed a novel space-preserving and scalable binary embedding method for attributed graphs named BGENA, for producing high-quality embeddings which are at par with the performance of the state-of-the-art contemporary methods.

- We are among the first generation of ANE solvers to use fast sketching methods to compute node embedding of a graph capable of preserving any arbitrary order of proximity.

- We also propose a parallelization scheme for BGENA named PBGENA to further accelerate its performance using a system's multi-core capability through a message passing interface.

- Finally, the numerous experiments performed in this study with 13 real-world graphs and 7 recently published methods demonstrate the superiority of PBGENA both in terms of speed and performance.

1.5 Organization

We have organized this report primarily into five chapters and an appendix. In chapter 2, we discuss the various studies which are closely related to this one, which includes some background about the history of ANE solvers and other closely related areas like community detection, graph partition, GNNs. Chapter 2 also looks at some of the most

popular sketching methods. In chapter 3, we introduce our main algorithm BGENA along with the idea of BinSketch, edge propagation, and its complexity analysis. Chapter 4 primarily deals with the parallelization of BGENA and sheds light on the speedup obtained through the process. We showcase all our experiments in chapter 5 including dataset and baseline descriptions, results obtained, and the numerous sensitivity analysis we have performed. Finally, in chapter 6, we conclude the thesis by summarizing the study and pointing to some interesting future directions in which this study can be extended. In the appendix, we provide the hyperparameters used for producing the results and also showcase some minor analyses.

CHAPTER 2

RELATED WORK

Network representation learning is not a recent endeavor. Yan et al. [15] had proposed back in 2006 that graph embedding could potentially be used as a unifying framework to test all dimension reduction methods owing to the fact that node embeddings of a graph will necessarily be required to preserve the proximity between nodes. Initially, the usual dimension reduction methods like PCA [20], LDA [21] had been tried to address the problem of NRL. These were typically underperforming because they were primarily linear algorithms and could not model the non-linearity inherent in graphs. To address this problem, the next generation of NRL methods included algorithms like Isomap [22] and SpectralClustering [23] which were able to model the non-linearity of graphs, but were extremely slow and could not reliably be used for large scale networks which is why they fell out of favor. These days NRL methods are able to encode within its node embedding both the structure and the attribute information of graphs. These methods can be broadly classified into categories like factorization-based, NLP-based, autoencoder-based, sketching-based, and GNNs. Other than these there are a few miscellaneous methods that do not strictly fit into any category. This chapter provides an overview of these methods along with introducing some well-known papers which have been the most successful in the area of NRL.

2.1 Factorization-based Methods

Factorization-based methods usually work in two stages- first, they build a proximity matrix that encodes the topological and/or attribute information of nodes into a more compact form. This step usually determines the order of proximity which will be preserved by the embedding. The second step involves factorizing the matrix using various techniques. For instance, TADW [24] uses low-rank matrix factorization for generating embeddings whereas TENE [25] uses non-negative factorization. Another important contribution by the authors of TADW was the proof that NLP-based random walk methods like DeepWalk are essentially similar to matrix factorization-based methods. BANE [26], another factorization-based ANE solver, uses the idea of Weisfiler-Lehman

graph kernels [27] to propose a novel matrix factorization scheme that outputs binary embeddings. LQANR [28] is like an extension to BANE which learns embeddings of the form $\{-2^b, \ldots, 2^b\}^N$, where b is the bit-width. BANE and LQANR are both attempts at improving space complexity while sacrificing performance.

2.2 NLP-based Methods

These are the class of methods that view the problem of NRL through the lens of NLP. The connecting bridge between NRL and NLP comes from Zipf's Law [29], which states that the word frequency in any document is inversely proportional to its rank in the corpus. Surprisingly, it has been observed that the frequency with which nodes appear in short random walks of a graph also follows Zipf's power law. This idea was first popularized by the authors of DeepWalk [13] who treated random walks over vertices of the network as sentences in a corpus and then fed this information to the popular language modeling tool SkipGram [30] to output the embeddings. DeepWalk has inspired a whole array of papers to improve and contribute to this idea. One such effort was made by Tang et al. in their paper LINE [31], who tried to concretize the idea of DeepWalk by introducing the notion of proximity measures in a network. LINE preserves second-order proximity in its embedding scheme which ensures nodes that share many mutual neighbors are closer in the vector space of their embeddings. node2vec [32] is another effort in the same direction that presents a unique metric to model two basic graph traversal techniques- BFS and DFS within its random walk. Their objective function has the flexibility of controlling which traversal mode to emphasize and empirically node2vec outcompetes DeepWalk and LINE. A common drawback of all these methods is the fact that they do not take the attributes of nodes into account while performing the embedding. This drawback has been lifted by recent papers like SINE [33] and IANRW [34] which uses the idea of second-order proximity between nodes even they share common attributes.

2.3 Autoencoder-based Methods

Since the task of ANE is essentially a modified dimension reduction problem [15], a common approach is to use autoencoders to solve ANE tasks. Hence, it is not surpris-

ing that the literature is abundant with autoencoder-based methods. Autoencoders are neural networks that compress and subsequently decompress data repeatedly with the objective of minimizing the decompression loss. Zhang et al. have proposed ANRL [35] that uses a neighbor enhancement autoencoder with an attribute-aware SkipGram model [30] to embed network structure and attributes. Sheikh et al. in their 2020 paper demonstrate SAGE2VEC [36], which uses an autoencoder with an enhanced decoder specifically designed to learn both network topology and attributes. SAGE2VEC can capture non-linearities (second-order proximity) and even works well with sparse graphs with sparse features. Researchers at IISc and IBM have recently published their autoencoder-based ANE solvers DONE and AdONE [37]. Both the models use two autoencoders (one for topology and the other for attributes) where DONE pays particular attention to community outliers whereas AdONE uses adversarial learning.

2.4 Sketching-based Methods

Due to the scale of graphs in the real world, research in NRL has recently shifted its attention towards fast ANE solvers. The present study is the product of a similar line of thinking. PBGENA is among the first generation of sketching-based NRL methods but it is not the only one. NetHash [17] constructs a reversed parent pointer tree for every node and passes on exponentially decaying MinHash [19] digests from leaves to the root. NodeSketch [38] uses consistent weighted sampling [39] that can be used to estimate a weighted variant of Jaccard similarity to model proximities of a network in its embeddings. QUINT [40], the precursor to PBGENA, is another sketching-based embedding scheme that uses BinSketch [41] for sketching the topological features of a graph.

2.5 Miscellaneous Methods

With such a rich and abundant source of literature, there are bound to be NRL methods that do not fit cleanly into any box. One such method is ComE [9], a community-aware node embedding strategy, that jointly performs community detection and node embedding in an expectation maximization algorithm-style method. ComE views communities in a graph as a multi-variate Gaussian distribution in 2D space and leverages

community detection to solve node embedding and vice versa. Another method for node embedding is RandNE [16], which uses an extremely fast Gaussian random projection method to map the network to a low embedding space. A slightly different approach to graph learning comes from GNNs that train deep learning models directly on graphs. GNNs are not a part of NRL in the strict sense of the term but GNNs have gained such popularity in recent times that there exists a general-purpose PyTorch library for it [1]. This is hardly the end of NRL methods, but these were the ones we found interesting enough to point out in our literature survey.

[1] https://github.com/pyg-team/pytorch_geometric

CHAPTER 3

THE BGENA ALGORITHM

The sparsity of a graph is a major challenge in network representation learning [13]. This, however, is not a problem for BinSketch [41] which thrives on sparse binary data making it ideal for network representation learning. BGENA is built on top of BinSkecth and bitwise edge propagation making it one of the fastest proximity-preserving embedding schemes available. This chapter discusses the background for BGENA and its construction along with its time and space complexity analysis.

3.1 The BinSketch Scheme

BinSketch [41] is an efficient similarity-preserving sketching method for sparse binary data proposed by Pratap et al. in 2019. BinSketch forms the backbone of the BGENA algorithm. We use the BinSketch scheme to compress both the topological and the attribute information of the input graph. BinSketch preserves distance metrics such as inner product and Jaccard similarity between two vectors even at high degrees of compression compared to other hashing schemes like DOPH [42] and CBE [43]. BinSketch is also one of the fastest sketching methods, which uses only random mapping and bitwise-OR for data compression, making it ideal for our purpose.

Algorithm 1 BinSketch Hashing Scheme

Input: $X \in \{0,1\}^{n \times d}$, N
Output: $S \in \{0,1\}^{n \times N}$ where $N << d$, Π
Initialisation: $S \leftarrow O_{n \times N}$
1: **def** BINSKETCH (X, N) :
2: $\Pi : \{1, 2, \dots\ d\} \mapsto \{1, 2, \dots\ N\}$ {random mapping}
3: **for** all non-zero entries in $(i, j) \in X$ {sketching}
4: $S[i, \Pi(j)] \leftarrow 1$
5: $S[j, \Pi(i)] \leftarrow 1$
6: **return** S, Π

In Algorithm 1, BinSketch takes a vector of d dimensions as input and outputs a compressed sketch of the vector in N dimensions ($N << d$) with the help of a mapping Π. The time complexity to generate a random integer in the range $[0, N)$ is $\mathcal{O}(\log N)$, and

the mapping Π generates d such random numbers so the time complexity for computing Π turns out to be $\mathcal{O}(d \log N)$.

The next part of the BinSketch algorithm involves hashing each vector in the dataset using Π, that is, $S[i,j] \leftarrow \bigvee_{k:\Pi(k)=j} X[i,k];\ \forall i \in [1,n], \forall j \in [1,N]$. Now the adjacency matrix and the attribute matrix of the input graph are both available to us in terms of sparse matrices, which is why we can simply go through the non-zero entries of the matrices and perform the appropriate bitwise-OR operations, as described in lines 3-5 in the Algorithm 1. The complexity of this step depends on the sparsity of the input matrix X. Let the maximum non-zero count among all the rows in X be denoted by ψ, then the time taken to sketch each row is given by $\mathcal{O}(\psi)$. BinSketch needs to sketch every vector in $X \in \{0,1\}^{n \times d}$ to produce the sketched matrix $S \in \{0,1\}^{n \times N}$, and finally returns the sketches along with the mapping. Therefore the complexity of hashing turns out to be $\mathcal{O}(n\psi)$. The total complexity of the BinSketch scheme for compressing n vectors of d dimension each into N dimensions turns out to be $\mathcal{O}(d \log N + n\psi)$.

3.2 Sketching Topology and Attributes

The key idea in trying to use BinSketch for attributed network embedding is to realize that both the adjacency matrix and the attribute matrix of a graph are available as sparse matrices. Therefore, these matrices can be provided directly to the BinSketch subroutine as input for sketching. In BGENA, we embed topology and attributes independent of one another. The adjacency matrix $E \in \{0,1\}^{|V| \times |V|}$ is compressed into the topology sketches $S_t \in \{0,1\}^{|V| \times N_t}$ using BinSketch; where N_t is the topology dimension.

Algorithm 2 CABIN Sketch

Input: $X \in \{0, 1, \ldots, c\}^{n \times d}$, N
Output: $S \in \{0,1\}^{n \times N}$ where $N << d$
Initialisation: $Y \leftarrow O_{n \times d}$

1: **def** CABIN (X, N) :
2: **for** $i \leftarrow 1$ to d:
3: $\pi[i] : \{1, 2, \ldots\ c\} \mapsto \{0, 1\}$ {random mapping for all input dimensions}
4: **for** $j \leftarrow 1$ to n:
5: $Y[j][i] \leftarrow \pi[i](X[j][i])$ {binarize}
6: $S, \Pi \leftarrow$ BINSKETCH (Y, N)
7: **return** S, Π

Sketching attributes are not as straightforward as sketching topology. Attributes can appear in various forms: binary, categorical, discrete, or real-valued. One way to convert categorical values to binary is by using one-hot encoding [44]. This, however, is not very helpful because the dimension of the data will explode immediately. A recent paper has proposed using an algorithm named CABIN [45] to perform embedding of categorical data using BinSketch. The CABIN algorithm [2] uses two mappings π and Π, unlike one in BinSketch, for the categories and hashing respectively. In lines 4-5 we first binarize the categorical data using π followed by calling BinSketch which uses Π to produce the binary sketches.

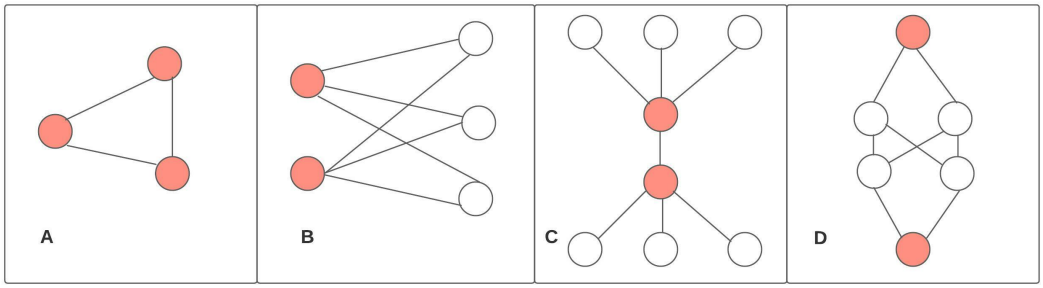
A graph, in general, can also have weighted edges and real-valued attributes thereby forcing the underlying embedding method to generalize. In such a case a logical extension for the sketching pipeline of PBGENA would be to perform binning [46] of the real values to categorical values followed by the use of CABIN. However, for the purposes of this study, we have only used binary adjacency matrices and binary attribute matrices to generate our results.

3.3 Edge Propagation

The BinSketch paper [41] proves that even though the sketches themselves do not preserve distance metrics like the inner product and Jaccard similarity precisely, we can obtain good approximations of them within decent error bounds. BinSketch alone has already been tried as an ANE solver by the paper QUINT [40]. Even though QUINT outperforms several older baselines and is extremely fast, it fails to match the performance of modern ANE solvers. Other than the performance fallback, QUINT also does not use attributes in its embedding scheme, meaning that it is not capable of using the full support provided by the network. These drawbacks prompted us to use an attribute pipeline and a novel edge propagation mechanism for enhancing the performance of BinSketch.

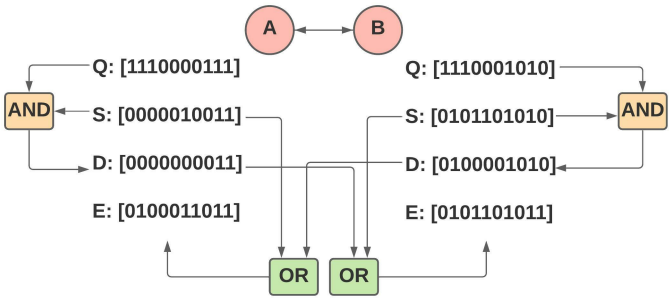
The idea behind edge propagation is that nodes that are topologically close in the graph should also be close in the vector space of their embeddings. The closeness can be described both in terms of topology and attributes or either. Edge propagation is related to the idea of proximity measure that was popularized by Tang et al. in their paper LINE

Figure 3.1: Idea of Proximity



[31]. The first-order proximity of a node v is the vertex set that shares a direct edge with v, and the attributes demonstrated by v (graph A in Figure 3.1). The second-order proximity between two nodes can be loosely defined as the degree of shared neighbors (graph B in Figure 3.1). In other words, two nodes can be close to one another if they have many mutual neighbors and/or attributes, even when they do not share a direct edge. BinSketch models the first and second order proximities of nodes. In some sense, BinSketch creates a reduced graph to preserve the proximities, but the first-order proximity is weakly expressed through BinSketch. For instance, consider graph C in Figure 3.1, if we were to take the inner product between the colored nodes after BinSketch, we would end up with a low value, but their inner product should be significant since they share a direct edge. To alleviate this problem, we introduce the notion of edge propagation which strengthens the first-order proximity of the embeddings and also has the ability to encode higher orders proximity within its embeddings.

Figure 3.2: A demonstration of the edge propagation mechanism



In edge propagation, we pass a digest of the sketch along with both the directions of the edge to update the embeddings of the connected nodes (Figure 3.2). We perform this once for the attributes (line 19 in Algorithm 3) and once for the topology (line 12). Now that we are performing an operation for every edge in the graph, the complexity for the edge propagation is in the order of $\mathcal{O}(|E|)$; meaning that we need to be economical in

every operation so as not to consume additional time. Therefore, we come up with edge propagation using only bitwise operations.

In order to compute digests for every node, we create a random binary vector Q of size equal to the embedding dimension. This vector is created such that every bit of Q is set with a probability of b and unset with a probability $(1 - b)$ (line 9 in Algorithm 3). We refer to b as the bitset probability of the vector Q. Q is generated independently for every node but we keep the bitset probability constant throughout. To create a digest $D[v]$ for every node v we perform: $D[v] \leftarrow S[v] \wedge Q$; where $S[v]$ is the BinSketched compressed vector of the node v (line 10 of Algorithm 3). This ensures that the digest will retain all the zeros in $S[v]$, but may flip some non-zeros. Thus, the digests contain a subset of the non-zero values of the sketches. This digest of node v is then propagated to all neighbors of v, using bitwise-OR as described in line 12 of Algorithm 3. The ultimate effect of this propagation is that some of the bits which were set in $S[v]$ are now set in $E[u]$, where u is a neighbor of v. Thus the inner product of the colored nodes in graph C in Figure 3.1, after single edge propagation, is no longer insignificant. In other words, we have strengthened the first-order proximity of the embeddings. A schematic demonstration of edge propagation in action is shown in Figure 3.3.

BGENA with single propagation also preserves third-order proximity if the bitset probabilities are high enough. To achieve higher orders of proximity, demonstrated via graph D in Figure 3.1, we propose repeated edge propagation through various levels. This can be achieved by performing several runs of edge propagation through the entire graph with exponentially decreasing bitset probability each time. Our experiments with real-world datasets have shown that repeated edge propagation does not really help in making predictions. We, therefore, concluded that for real-world datasets, being able to model first and second order proximities are enough for achieving good performance. With that knowledge, we perform all our experiments with a single pass of edge propagation. However, for a graph that requires support through higher-order proximities, BGENA can be easily generalized by setting $l > 1$ in line 7 of Algorithm 3.

3.4 Building BGENA

In this section, we discuss how to put everything together to construct the BGENA algorithm. We realize that certain graphs carry more support through their attributes while others carry more support through their topology. We further realize that the topology and attribute propagations can be performed independently of one another with different bitset probabilities. This makes the BGENA algorithm even more flexible. Later these independent pipelines will be exploited to parallelize BGENA.

Algorithm 3 The BGENA Algorithm

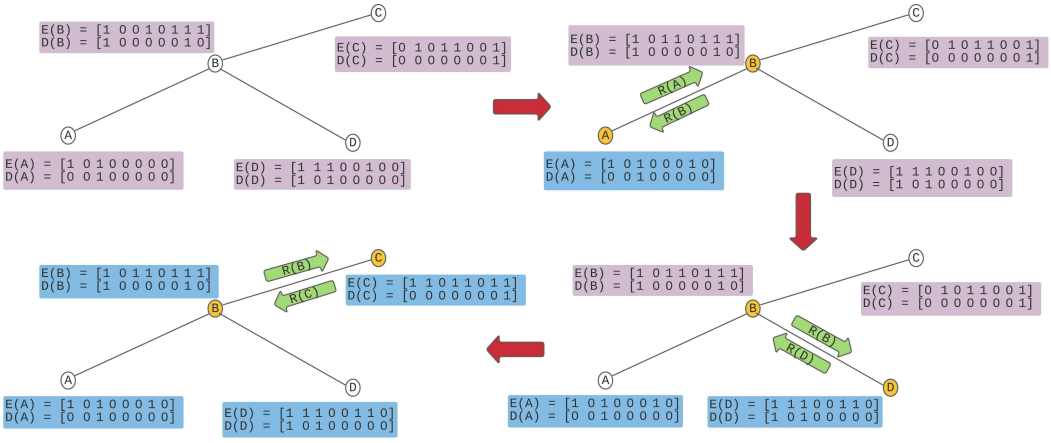
Input: Attributed Graph $G(V, E, A)$, α, N, b_t, b_a, l_t, l_a, f_t, f_a
Output: Node Embeddings $emb \in \{0,1\}^{|V| \times N}$

Initialisation: $emb \leftarrow O_{|V| \times N}$
def BGENA (G, α, N, b_t, b_a, l_t, l_a, f_t, f_a) :
  $N_a \leftarrow N * \alpha$, $N_t \leftarrow N * (1 - \alpha)$
  $S_t, \Pi_t \leftarrow$ BINSKETCH (E, N_t) {topology mapping and sketching}
  for $i \leftarrow 1$ to $|V|$:
    $S_t[i, \Pi_t(i)] \leftarrow 1$
  $S_a, \Pi_a \leftarrow$ BINSKETCH (A, N_a) {attribute mapping and sketching}
  for $i \leftarrow 1$ to l_t:
    for $j \leftarrow 1$ to $|V|$:
      $Q_t \leftarrow [\{0,1\}^{N_t} | P(Q_t[k] = 1) = b_t, \forall k \in \{1 \dots N_t\}]$
      $D_t[j] \leftarrow S_t[j] \wedge Q_t$, $E_t[j] \leftarrow S_t[j]$
    for (j, k) in E: {topology propagation}
      $E_t[j] \leftarrow E_t[j] \vee R_t[k]$, $E_t[k] \leftarrow E_t[k] \vee D_t[j]$
    $S_t \leftarrow E_t$, $b_t \leftarrow b_t / f_t$
  for $i \leftarrow 1$ to l_a:
    for $j \leftarrow 1$ to $|V|$:
      $Q_a \leftarrow [\{0,1\}^{N_a} | P(Q_a[k] = 1) = b_a, \forall k \in \{1 \dots N_a\}]$
      $D_a[j] \leftarrow S_a[j] \wedge Q_a$, $E_a[j] \leftarrow S_a[j]$
    for (j, k) in E: {attribute propagation}
      $E_a[j] \leftarrow E_a[j] \vee D_a[k]$, $E_a[k] \leftarrow E_a[k] \vee D_a[j]$
    $S_a \leftarrow E_a$, $b_a \leftarrow b_a / f_a$
  $emb \leftarrow$ CONCATENATE(E_t, E_a)
  return emb

The primary hyperparameters of BGENA are N which is the embedding dimension, and α which is the fraction of the embedding dimension to be used for attributes. This solves the problem of having to decide the number of dimensions to be used for attributes for a specific graph. The other hyperparameters include the bitset probabilities for topology and attribute b_t and b_a respectively.

The since all the vectors used in the BGENA algorithm are binary, we have imple-

Figure 3.3: Working of BGENA



mented BGENA entirely using *bitarray* [1], a fast and efficient array for booleans in Python. Bitarrays store booleans as bits instead of bytes, which enabled us to considerably outperform our competitors in terms of system memory required for embedding large networks.

The hyperparameters l_t and l_a indicate the level of edge propagation, i.e., the number of times BGENA should perform edge propagation through the topology embeddings and the attribute embeddings. For all our experiments, we have set $l_t = 1$, $l_a = 1$. If one decides to choose values greater than one for these two hyperparameters, it is not clear how to set the values for f_t, and f_a, which are the fractions by which the topology and attribute bitset probabilities are reduced in each pass. A value of 2 seems to be a good choice but a more rigorous analysis is required to settle the issue.

3.5 Time Complexity

Primarily BGENA can be decomposed into three operations: mapping, sketching, and propagation. Each of these three segments has two pipelines: one for topology and the other for attributes. Out of these three operations, mapping is the fastest and propagation is the most time-intensive. Let us now individually look at their respective complexities.

[1] `https://github.com/ilanschnell/bitarray`

3.5.1 Mapping

As discussed in section 3.1, mapping a set of d numbers to a set of N numbers randomly takes $\mathcal{O}(d \log N)$ time. Following the same logic for topology mapping, we compute a random mapping from a set with cardinality $|V|$ to a set with cardinality N_t, implying that the complexity would be $\mathcal{O}(|V| \log N_t)$. Similarly, for attribute mapping, we have $\mathcal{O}(a \log N_a)$, where a is the maximum number of attributes possible for a node in the graph. The cumulative mapping cost sums up to be $\mathcal{O}(|V| \log N_t + a \log N_a)$. If we want to represent explicitly in terms of the hyperparameters, then we will have the following expression: $\mathcal{O}(|V| \log(N(1-\alpha)) + a \log(N\alpha))$.

3.5.2 Sketching

The first sketching operation as described in lines 4-5 of Algorithm 3 takes $\mathcal{O}(|V|)$ time. The second sketching operation involves going through the non-zero entries of the adjacency matrix and setting the appropriate bits as described in Algorithm 1. This operation obviously takes $\mathcal{O}(|E|)$ time. Finally, we go through the non-zero entries of the attribute matrix and set the appropriate bits, which takes $\mathcal{O}(|A|)$ time, where $|A|$ is the number of non-zero entries of the attribute matrix. Hence the cumulative sketching cost becomes $\mathcal{O}(|V| + |E| + |A|)$.

3.5.3 Propagation

For topology propagation, we compute the random vector Q_t for every node, which consumes $\mathcal{O}(N_t|V|)$ time. After generating the digests D_t using Q_t [time upper bounded by $\mathcal{O}(N_t|V|)$], we need to perform the edge propagation over all edges $|E|$ in both directions using only bitwise operations, meaning that the total time for topology propagation comes out to be $\mathcal{O}(l_t N_t(|V| + |E|))$; the l_t multiplier is due to multiple passes of edge propagation. Using a similar analogy, we can conclude that the time for attribute propagation is $\mathcal{O}(l_a N_a(|V| + |E|))$.

Therefore complexity of the most generalized version of BGENA turns out to be:

$$\mathcal{O}(|V| \log N_t + a \log N_a + |V| + |E| + |A| + l_t N_t(|V| + |E|) + l_a N_a(|V| + |E|))$$

Simplifying this we get,

$$\boldsymbol{\mathcal{O}(a \log N_a + |A| + l_t N_t(|V| + |E|) + l_a N_a(|V| + |E|))} \tag{3.1}$$

For the purposes of experimentation in this paper we have set $l_t = l_a = 1$, this fact can help us further simplify Equation 3.1 as:

$$\boldsymbol{\mathcal{O}(a \log N_a + N|V| + N|E| + |A|)} \tag{3.2}$$

For most real-world datasets, $a \log N_a \ll |A|$ and N is constant (for this study $N = 2000$). Therefore we can say that the time complexity of BGENA is linear in terms of $\mathcal{O}(|V| + |E| + |A|)$.

3.6 Space Complexity

The Space Complexity of BGENA is given by the maximum amount of additional memory required by the algorithm at any point. This is known as the bottleneck of the algorithm. We can easily see that the maximum utilization of memory occurs in the edge propagation phase when we need to simultaneously maintain three matrices: the sketches S, the digests D, and the embeddings E. Each of these matrices is of type $\{0, 1\}^{|V| \times N}$. Therefore we can conclude that the final complexity of the BGENA algorithm is $\boldsymbol{\mathcal{O}(N|V|)}$ bits. This means that throughout the running time of the BGENA subroutine, the amount of space required never asymptotically exceeds the space of the embeddings themselves.

CHAPTER 4

PARALLELIZATION OF BGENA

Even with all the speed and might of BGENA, it can take nearly 8 hours for it to embed MAKG (as indicated in Table 5.6). Also, massive graphs can be stored in a distributed fashion (which is true for many ego networks and they require preprocessing), making it harder for BGENA or any other serial algorithm to process everything in a single CPU. Keeping all of these things in mind, we have proposed PBGENA, the parallel version of BGENA that retains all the embedding properties of BGENA but can provide excellent speedup by exploiting a system's multi-core capabilities using MPI. In this chapter, we explore the various schemes for parallelizing BGENA and discuss the underlying algorithm along with its complexity bounds.

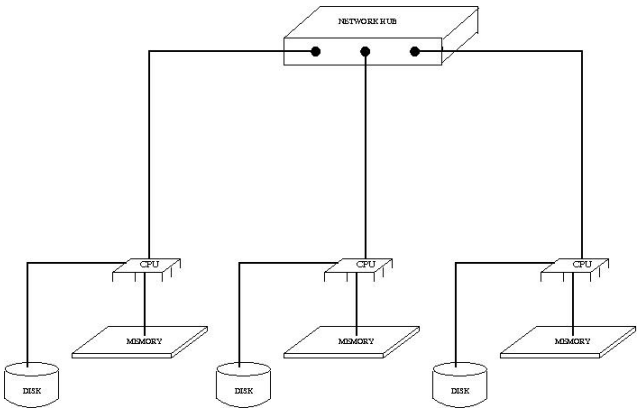
4.1 Parallelization Architecture

MPI [47] is the standard API for distributed computing featuring a message-passing paradigm for IPC, and it is the leading paradigm in HPC. MPI is an interface for distributed memory systems (Figure 4.1, Source: Wikipedia). Contrary to popular belief, MPI is not an implementation, it is a standardization and many open-source implementations are available, like OpenMPI[1] and MPICH[2]. MPI defines the syntax and semantics of parallelization for writing programs using languages like C and C++. These standards do not apply to high-level scripting languages like Python. This is where mpi4py [48] comes in to bridge the gap between MPI and Python.

mpi4py supports several features which are very useful for writing parallel programs at a high level. Unlike MPI for C/C++, which are stuck with message passing using arrays, mpi4py has the flexibility of sending pickled objects across various processors which makes it easier for PBGENA to pass the digests as bitarray objects across different processors. mpi4py comes bundled with an object-oriented API, making it very natural to code with. Finally, mpi4py also has the capability of sending out-of-band buffers

[1] `www.open-mpi.org`
[2] `www.mpich.org`

Figure 4.1: Distributed Memory System

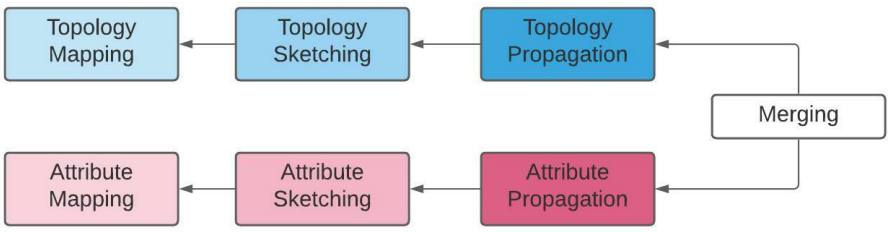


across processors using the Pickle-5 [49] protocol which may be useful for PBGENA for embedding massive graphs with a limited number of CPUs.

4.2 Parallelizable Tasks

As discussed earlier in section 3.5, the three major tasks in BGENA include mapping, sketching, and edge propagation where mapping is the most lightweight task and edge propagation is the most expensive. We present the dependency graph between the various tasks in Figure 4.2.

Figure 4.2: Task Dependency Graph



In the making of PBGENA, we investigated the prospect of parallelizing various combinations of operations and experimentally verified their speed. To split the graph across different processors, we perform a random partition of nodes (described in line 4 of Algorithm 4). We investigated the use of graph partitioning toolkits like METIS and ParMETIS [50] to reduce the communication overheads between processors, but METIS in itself turned out to be a bottleneck so we performed simple random partitioning. We observed that it is computationally cheaper to perform the mappings on a

single processor and then broadcast them instead of doing it parallelly. After the serial mapping is broadcasted from the first processor (line 7 of Algorithm 4), we perform the sketching of each node in parallel (lines 8-11 of Algorithm 4) with no need for IPC. Next, we perform edge propagation for the edges where both the vertices of the edge are present with the processor (lines 13-14 of Algorithm 4). For the edges that go across different processors, we first exchange the digests and then perform edge propagation (lines 16-17 of Algorithm 4). In our implementation of PBGENA, we always batch the data to be sent across any two processors to make it more efficient.

4.3 Parallelization Scheme

The traditional "master-slave" approach to parallel computing does not treat the I/O part of the parallel program differently from the serial code. In other words, the I/O should be performed by the "master" processor and be distributed to the other "slave" processors. The advantage of this protocol is that the RAM requirement for the inputting does not exceed that of the serial code. However, our experiments showed that BGENA is by itself so fast that distributing the data across processors itself takes significant time and a speedup of more than $2\times$ is not possible. To circumvent this problem, we perform a parallel reading of the graph into different processors. All our datasets, except MAKG, were parallelly read into our system by 32-cores using 270GB of physical memory.

As it may be evident from Algorithm 4, we do not consider the reading time of the graph as part of our algorithm. PBGENA assumes that the adjacency list of every node along with its attribute set is available in a dictionary-like [49] format. All the input parameters for PBGENA are identical with BGENA except p and $rank$ which identifies the number of processors and the processor ID respectively.

4.4 Complexity Overview

In this section, we discuss the time and space complexity of PBGENA. Experimentally it was observed that PBGENA is on average about $16\times$ faster than BGENA and more demanding in terms of memory. The following subsections provide a richer understanding of the underlying complexities:

Algorithm 4 The PBGENA Algorithm

Input: Attributed Graph $G(V, E, A)$, α, N, b_t, b_a, l_t, l_a, f_t, f_a, p, $rank$
Output: Node Embeddings $emb \in \{0,1\}^{|V| \times N}$

Initialisation: $emb \leftarrow O_{|V| \times N}$
1: **def** PBGENA (G, α, N, b_t, b_a, l_t, l_a, f_t, f_a, p, $rank$) :
2: $N_a \leftarrow N * \alpha$, $N_t \leftarrow N * (1 - \alpha)$
3: **if** $rank = 1$:
4: $\Phi : \{1, 2, \dots |V|\} \mapsto \{1, 2, \dots, p\}$ {random partition}
5: $\Pi_t : \{1, 2, \dots |V|\} \mapsto \{1, 2, \dots, N_t\}$ {topology mapping}
6: $\Pi_a : \{1, 2, \dots a\} \mapsto \{1, 2, \dots, N_a\}$ {attribute mapping}
7: **broadcast** Φ, Π_t, Π_a from Processor with $rank = 1$
8: **for** $i : \Phi(i) = rank$ **do in parallel**:
9: $S_t[i, \Pi_t(i)] \leftarrow 1$ {topology sketching}
10: $S_t[i, \Pi_t(j)] \leftarrow 1; \forall j \in E[i]$
11: $S_a[i, \Pi_t(j)] \leftarrow 1; \forall j \in A[i]$ {attribute sketching}
12: **for** all $(i, j) \in E$ **do in parallel**:
13: **if** $\Phi(i) \neq \Phi(j)$ **and** ($\Phi(i) = rank$ **or** $\Phi(j) = rank$): {for cross-edges}
14: **exchange** digests D_t, D_a with the other processor
15: **perform** Edge Propagation in the same way as Algorithm 3
16: **return** emb

4.4.1 Time Complexity

Similar to BGENA (section 3.5), the time complexity of PBGENA is dependent on the three stages of the algorithm: mapping, sketching, and propagation. Now since mapping is performed serially, the time complexity for mapping remains the same in BGENA and PBGENA: $\mathcal{O}(|V| \log N_t + a \log N_a)$ [subsection 3.5.1]. The next part of the process is sketching nodes. The sketching of nodes can be done in parallel without any need for inter-process communication [lines 8-11 of Algorithm 4]. Therefore the complexity for sketching in PBGENA becomes $\mathcal{O}(\frac{|V|+|E|+|A|}{p})$, where p is the number of processors [refer to subsection 3.5.2 for the derivation]. Finally, we need to find the complexity for edge propagation which relies on the number of edges that lie within a partition, and the number of edges that lie across two partitions. Let us call them *in-edges* and *cross-edges* respectively. Given that we have partitioned the vertex set into p processors using a random mapping Φ, the probability of an edge being an in-edge becomes $1/p$, which means that the expected number of cross-edges that reside across two different partitions becomes $|E| * (1 - \frac{1}{p})$. Assuming $\mathcal{O}(N)$ cost for propagation of each digest, the time complexity of edge propagation mechanism becomes $\mathcal{O}(\frac{l_t N_t(|V|+|E|)+l_a N_a(|V|+|E|)+(p-1)N|E|}{p})$. Summing everything up, the asymptotic time

complexity of PBGENA becomes (derivation in subsection 3.5.3):

$$\mathcal{O}(\frac{p|V|\log N_t + pa\log N_a + |A| + l_t N_t(|V|+|E|) + l_a N_a(|V|+|E|) + (p-1)N|E|}{p}) \tag{4.1}$$

Putting $l_t = l_a = 1$ we get,

$$\boldsymbol{\mathcal{O}(\frac{p|V|\log N_t + pa\log N_a + N|V| + (p+1)N|E| + |A|}{p})} \tag{4.2}$$

Realistically, the terms $|V|\log N_t$ and $a\log N_a$ are negligible, so we can safely assume that the serial time is $T_s = \mathcal{O}(N|V|+N|E|+|A|)$ (from Equation 3.2), and parallel time is $T_p = \mathcal{O}(\frac{N|V|+(p+1)N|E|+|A|}{p})$ (from Equation 4.2). Now the speedup $\mathcal{S}$ of a parallel algorithm is defined as $\mathcal{S} = T_s/T_p$. Hence the speedup obtained from PBGENA turns out to be:

$$\mathcal{S} = p * \frac{N|V| + N|E| + |A|}{N|V| + (p+1)N|E| + |A|} \tag{4.3}$$

A similar metric used for calculating the performance of a parallel algorithm is efficiency $\mathcal{E}$, which is defined as $\mathcal{E} = T_s/(pT_p)$

$$\mathcal{E} = \frac{N|V| + N|E| + |A|}{N|V| + (p+1)N|E| + |A|} \tag{4.4}$$

Experimentally PBGENA has demonstrated an average speedup of $16\times$ over BGENA by using 32 cores.

4.4.2 Space Complexity

The memory requirement of PBGENA is slightly more than BGENA due to storing the additional:

- Partition mapping Φ (line 4 of Algorithm 4) in every processor.
- The operation $i : \Phi(i) = rank$ (line 8 of Algorithm 4) is very expensive unless we store locally in each processor Φ^{-1}, indicating the nodes which are available to it.
- Some sketch digests D_t and D_a are duplicated multiple times in the processors because they are not available with the processor.

The space required to store Φ and Φ^{-1} are negligible as compared to the space for the embeddings $\mathcal{O}(nN)$ (derived in section 3.6). The main issue comes as a result of duplicated digests. Since one digest can be duplicated across p processors in the worst

case, the space complexity becomes $\mathcal{O}(npN)$. Practically, this bound is extremely loose because there are repeated cross-edges originating from the same node since the average degree of a node in real-world graphs is > 1. A tight bound on the space complexity of PBGENA is $\boldsymbol{\mathcal{O}(N|V|)}$ bits (same as BGENA, the proof is provided in Appendix B).

CHAPTER 5

EXPERIMENTS

The previous chapters have detailed the power of sketching and its use in building our parallel embedding scheme. Even though PBGENA looks good on paper we need to experimentally support our claims. Therefore, in this section, we showcase the various experiments that we have performed with BGENA and PBGENA along with how they compare with the state-of-the-art baselines and description of the datasets and baselines. Our experiments include testing PBGENA with its baselines for tasks like node classification, link prediction, graph visualization, and understanding the sensitivity of PBGENA hyperparameters through an array of detailed experiments and figures.

5.1 Dataset Description

We collect all our data from the PANE [18] authors, who provide the graphs in a preprocessed format [1]. The dataset description for the graphs is available in Table 5.1. The table provides the count of the number of nodes, edges, attributes, and labels in the graph. The fourth column of Table 5.1 indicates whether the graph is multi-labeled, meaning whether nodes may belong to more than one class. Multi-labeled graphs can further compound the problem of node classification. A single node in a graph like Twitter can potentially have more than 4000 labels. The datasets are organized in increasing order of their node count.

5.1.1 Wikipedia

Wikipedia is a graph dataset created by the authors of TADW [24]. This dataset was created by treating articles hosted at `www.wikipedia.org` as nodes and the hyperlinks between the articles and undirected edges. Each Wikipedia article is composed of long texts of 640 words on average, which were used to create the attributes for the graph. The labels mostly indicate the type of article they are. The Wikipedia dataset is available online[2], but we have used the preprocessed format from the PANE authors[1].

[1] `http://www4.comp.polyu.edu.hk/~jiemshi/datasets.html`
[2] `https://github.com/thunlp/TADW/tree/master/wiki`

Table 5.1: Dataset Description

Graph	#Vertices	#Edges	#Attributes	#Labels	Multi-Labeled?
Wikipedia	2,405	12,761	4,973	17	No
Cora	2,708	5,278	1,433	7	No
CiteSeer	3,312	4,660	3,703	6	No
Facebook	4,039	88,234	1,283	193	Yes
BlogCatalog	5,196	17,1743	8,189	6	No
Flickr	7,575	23,9738	12,047	9	No
PubMed	19,717	44,327	500	3	No
PPI	56,944	81,8716	50	121	Yes
Twitter	81,306	1,342,310	216,839	4,065	Yes
Google+	107,614	12,238,285	15,907	468	Yes
Reddit	232,965	57,307,946	602	41	No
TWeibo	2,320,895	50,133,382	1,657	9	No
MAKG	59,249,719	976,901,586	7,211	100	Yes

5.1.2 Cora

The Cora dataset was prepared by Lu et al. [51] using research papers from the website `www.cora.justresearch.com`. This is a citation network which means that the network is comprised of research papers in ML as nodes and citations as edges. The attribute set for the network is composed of keywords that appear in the papers. These keywords are filtered using stemming and stopword removal. The classes of the node indicate the subject matter which the papers deal with, like *Case-Based*, *Genetic Algorithms*, *Neural Networks*, *Probabilistic Methods*, *Reinforcement Learning*, *Rule Learning*, and *Theory*. Cora dataset is available online [3] for downloading.

5.1.3 CiteSeer

CiteSeer is yet another citation network prepared by Lu et al. [51]. The data for the network was obtained from the website CiteSeerX `https://citeseerx.ist.psu.edu/index` (previosly CiteSeer). The data preparation phase is identical to that of Cora's [subsection 5.1.2]. The labels of the papers in CiteSeer include *Agents*, *Artificial Intelligence*, *Database*, *Human-Computer Interaction*, *Machine Learning*, and *Information Retrieval*. The CiteSeer network is freely available online [3] at the website for the LINQS group at the University of California, Santa Cruz.

[3] `https://linqs.soe.ucsc.edu/data`

5.1.4 Facebook

The Facebook graph is an ego network that represents social relationships maintained by an individual (ego) with other people (alters). These relationships can be thought of as layers originating outwards from the ego and the associations are stronger with the ones close to the ego than others [52]. The ego network for Facebook is made up of friends lists from the website `www.facebook.com`. The data for this network was obtained from an anonymized survey. The Facebook network can be obtained online from the SNAP website[4].

5.1.5 BlogCatalog

BlogCatalog (`www.blogcatalog.com`) used to be a website for bloggers for posting blogs and related discussions. The nodes in the graph are the bloggers themselves and the attribute set is constructed from keywords generated from blog descriptions provided by the authors. The labels of the network represent the categories of the authors. The BlogCatalog network was originally prepared by Tang et al. [53] in 2009. The preprocessed data is available from CAN's [10] GitHub page [5].

5.1.6 Flickr

`www.flickr.com` is a media-hosting platform, where users interact by sharing photos. The connections between different users form the edges in the graph, and the interest tags are considered as attributes. The labels of this network represent the groups that exist on Flickr. This network was originally proposed by Huang et al. [54] in their 2017 paper. The Flickr network is available online [5].

5.1.7 PubMed

The PubMed website (`https://pubmed.ncbi.nlm.nih.gov`) is a repository for more than 33 million citations for biomedical literature. The nodes of the PubMed citation network [55] are diabetes articles obtained from the PubMed database. Similar to Cora and CiteSeer, PubMed interprets citations as links of a network. The attributes are made of TF/IDF-weighted word frequencies and the labels indicate the type of di-

[4] `https://snap.stanford.edu/data/ego-Facebook.html`
[5] `https://github.com/mengzaiqiao/CAN/tree/master/data`

abetes in the paper: *Type-1*, *Type-2*, and *Gestational Diabetes*. The data is publicly available online [3].

5.1.8 PPI

PPI stands for Protein-Protein Interaction, which is a network recently introduced by Hamilton et al. in 2018 in their paper GraphSAGE [5]. PPI, a multi-labeled graph, is essentially a collection of multiple graphs with each graph modeling the interactions between proteins in a different human tissue. The positional gene sets, motif gene sets, and immunological signatures are used as features in the graph and the gene ontology sets are used as labels. The raw data for the graph can be obtained from `https://thebiogrid.org` and the preprocessed graph is also available online [6].

5.1.9 Twitter

Twitter is another ego network in our dataset collection. This data was collected from public sources from the website `https://twitter.com`. The dataset includes profiles, circles, and ego networks [7]. It was first introduced by McAuley at al. [2] in 2012.

5.1.10 Google+

Similar to Facebook (subsection 5.1.4) and Twitter (subsection 5.1.9), Google+ is another ego network introduced by McAuley at al. [2] in 2012. `https://currents.google.com` used to be a social networking site operated by Google which is now shut down, but the graph data is still available online [8].

5.1.11 Reddit

`https://reddit.com` is a very old and popular website for content-sharing and discussions. Reddit is divided into sections or "subreddits" for segregating the type of content being discussed. GraphSAGE authors [5] treat posts made in the month of September 2014 on 50 large communities on Reddit as nodes, and two posts share an edge if the same user comments on both. The subreddit on which the post lies is treated as its label. The Reddit network is available online [6] for download.

[6] `http://snap.stanford.edu/graphsage`
[7] `https://snap.stanford.edu/data/ego-Twitter.html`
[8] `https://snap.stanford.edu/data/ego-Gplus.html`

5.1.12 TWeibo

TWeibo (http://t.qq.com/) was a Chinese microblogging website that was recently shut down. The KDD Cup 2012, Track 1 [9] presented the TWeibo dataset for participants to compete for making the best recommendation system for users on the website to follow. The PANE authors [18] subsequently extracted 1657 most popular tags and keywords from its user profile data as the node attributes, and the labels indicate the age of the users.

5.1.13 MAKG

MAKG, the largest graph among our networks, is a large RDF data set with over eight billion triples with information about scientific publications and related entities, such as authors, institutions, journals, and fields of study. The raw data is available at https://makg.org/. MAKG is essentially a citation network where each node represents a paper and each directed edge represents a citation. The PANE authors [18] extract 2000 most frequently used distinct words from the abstract of all papers as the attributes and the fields of study are treated as labels.

5.2 Baselines

We have used Karate Club [56], an open-source python framework for unsupervised graph learning, for all our baselines, except for PANE [18] whose code was obtained directly from the GitHub page of its authors [10]. A bigger and more descriptive explanation of embedding methods, in general, is available in chapter 2. This section only attempts to provide a brief description of the methods used to compare PBGENA. The baselines are provided in chronological order of their date of publication:

5.2.1 TADW

Yang et al. [24] proved that DeepWalk [13], a well-known NLP-based method, built using the SkipGram model is actually equivalent to the method of matrix factorization. They further go on to propose their own method TADW for embedding nodes using both

[9] https://www.kaggle.com/c/kddcup2012-track1
[10] https://github.com/AnryYang/PANE

the topology and the attribute information ("rich text"). TADW constructs a second-order proximity matrix M using the adjacency matrix of the graph and then proceeds to minimize the difference between the product of the embeddings learned and the attribute matrix with M. They also incorporate a regularization term involving the Frobenius norm of the learned embeddings. The complexity of TADW is quadratic with respect to the number of edges in the graph.

5.2.2 TENE

Text Enhanced Network Embedding [25] is yet another factorization-based method on our list. Similar to TADW, TENE constructs a second-order proximity matrix as $X = X^{(1)} + 5 * X^{(2)}$ where $X^{(1)}$ and $X^{(2)}$ denote the first and second order proximity matrix respectively. It then performs a non-negative factorization of both X and the attribute matrix T. Now, in order to make the learned embeddings be aware of the attributes, it performs a third matrix factorization which tries to minimize the difference between the topologically learned embeddings and the embeddings obtained from factorizing the attribute matrix. In other words, TENE tries a joint factorization of the adjacency matrix and the attribute matrix.

5.2.3 SINE

Scalable Incomplete Network Embedding [33] is an extension to DeepWalk [13]. SINE proposes using random walks to find pairs of vertices that belong together otherwise known as "context," and also tries to incorporate node-attribute relationships into the embedding. SINE formulates this probabilistic learning framework using a three-layer neural network whose output is the probabilities $P(v_i|v_j)$ and $P(a|v)$ where $v \in V$ and $a \in A$ and the input are the one-hot representation of each node. The weight matrices going from the input layer to the hidden layer are the learned embeddings.

5.2.4 ASNE

ASNE [57] is a deep-learning-based method that is specifically targeted towards the learning of social networks. ASNE uses a deep learning architecture to jointly learn structural and attribute associations between nodes. Unlike most methods that learn the attributes and topology of a graph separately, ASNE chooses to incorporate the node

attributes in the input layer itself for better-integrated learning of associations between nodes and attributes.

5.2.5 PANE

PANE [18] was published in the year 2020 and as of the writing of this thesis can be considered as the state-of-the-art of ANE solvers. PANE combines techniques like random walk and matrix factorization into its embedding scheme. Instead of using the original graph, PANE constructs an extended graph where bidirectional links are established between individual attributes and nodes. PANE is one of the few methods that respect the directedness of the underlying graph during its random walk. Using random walks to capture attribute and structural information, and cyclic coordinate descent [58] to jointly factorize the forward and backward affinities, PANE achieves significant performance gains. To enhance speed, PANE employs an effective greedy weight initialization scheme and parallelization through multiple threads.

5.2.6 FeatherNode

Feather [59] uses short random walks to represent the affinity between two nodes. Its authors define a characteristic function between a node and its multi-level neighbors, similar to Fourier transforms but for probability distributions, and also gives a linear time algorithm to evaluate this function for all nodes. Feather uses the value of these functions at several discrete points as feature vectors, effectively yielding an algorithm that performs very well for representing nodes in terms of their neighborhoods. FeatherNode is a graph neural network model that uses an r-scale random walk weighted characteristic function.

5.2.7 MUSAE

MUSAE [60] is an ANE solver that uses a random-walk-based method (similar to the ones described in section 2.2) to capture higher-order proximities of nodes from its local distribution of attributes and neighbouring nodes. MUSAE demonstrates through their experiments that using a "multi-scale" approach of storing distinct proximities obtained through various features may actually be advantageous for downstream machine learning tasks.

Apart from these, we have also experimented with BANE [26], another binary embedding method. According to our convention, embeddings with binarized values have been allowed to go as high as 2000 dimensions. However, BANE is considerably slower than BGENA even at $N = 128$, which is why we have decided to leave out BANE as a baseline.

5.3 Experimental Setup

We perform all our experiments on an **AMD EPYC 7452 32-Core 3200MHz Processor running Ubuntu 20.04.2 LTS with 270GB of RAM**. The dataset description for all our datasets is provided in section 5.1. We have selected eight state-of-the-art baselines for comparison in our study and a detailed description of them is provided in section 5.2.

Table 5.2: Embedding Dimensions (N) for various methods

Graph	Algorithm								
	TADW	**TENE**	**SINE**	**ASNE**	**PANE**	**FN**	**MUSAE**[a]	**BGENA**	**PBGENA**
Wikipedia	250	250	250	250	250	250	248	2000[b]	2000
Cora	250	250	250	250	250	250	248	2000	2000
CiteSeer	250	250	250	250	250	250	248	2000	2000
Facebook	250	250	250	250	250	250	248	2000	2000
BlogCatalog	250	250	250	250	250	250	248	2000	2000
Flickr	250	250	250	250	250	250	248	2000	2000
PubMed	250	250	250	250	250	250	248	2000	2000
PPI	250	250	250	250	100[c]	250	248	2000	2000
Twitter	250	250	250	250	-	250	-	2000	2000
Google+	-	250	-	250	250	250	-	2000	2000
Reddit	-	250	-	250	-	250	-	2000	2000
TWeibo	-	250	-	250	250	250	-	2000	2000
MAKG	-	-	-	-	-	-	-	2000	-

[a] The MUSAE source code available with us did not allow exactly 250 dimensions

[b] PBGENA stores 2000 bits, whereas the other baselines store 250 floats

[c] The PANE source code available with us requires $N/2 \leqslant a$

Since all our baselines output real-valued embeddings, we set the number of dimensions to approximately 250 (exact values are available in Table 5.2) and set PBGENA's dimensions to 2000 because PBGENA stores bits. Note that real-valued elements consume 4 bytes to 8 bytes for *float32* and *float64* representations respectively. This means even at $N = 2000$, PBGENA is at least $4\times$ better in terms of memory requirements. In node classification results (section 5.4), we have also demonstrated the power of PBGENA at $N = 8000$ which is at an equal footing compared to its baselines. For Node

Classification, we use 70% of the embeddings as a training set and the remaining is held out for testing. While performing Link Prediction (section 5.5), we randomly remove 30% of the edges from the graph and create a reduced graph with only 70% of the original edges, and generate embeddings using our various methods with the residual graph. We further generate dubious edges which do not exist in the graph and also divide that edge set into test and train. Finally, compute the cosine distance between two embeddings for the feature value and label the positive edges as 1 and negative edges as 0. For the predictions in both tasks, we have used scikit-learn's [61] Logistic Regression. We present our results broken into three sets of tables for small-sized, medium-sized, and large-sized graphs. We consider a graph to be small if it contains fewer than $5,000$ nodes and a graph to be large if it contains an excess of $100,000$ nodes, the remaining are classified as medium-sized graphs. All the results presented are averaged over five runs.

For graph visualization, we have used t-SNE [62] on the PBGENA embeddings to reduce the feature space into two dimensions and subsequently plotted them while coloring the nodes with their respective labels. This helps us to actually visualize class separation resulting from the underlying embedding scheme. In the remaining experiments, we test PBGENA's speed and accuracy with respect to various hyperparameters and also perform experiments to quantify how robust PBGENA actually is by reducing the training ratio in node classification and increasing the fraction of removed edges for link prediction.

5.3.1 System Requirements

The following are the system specifications and libraries required to run PBGENA:

- Windows 11 Home Single Language $21H2$ / Ubuntu $20.04.3$ LTS
- Python $3.10.0$
- Microsoft-MPI $10.1.1$ / Open MPI $4.1.2$
- mpi4py $3.1.3$
- pandas $1.3.5$

- numpy 1.21.4
- SciPy 1.7.3
- bitarray 2.3.4
- scikit-learn 1.0.1

The source code for PBGENA is made available on GitHub [11].

5.4 Node Classification

Node Classification refers to the task of predicting the labels of each node of the graph using the embeddings. Since we have multi-labeled graphs in our dataset (refer to Table 5.1), there may be nodes with more than a single label. The results of node classification are presented in the three tables- 5.3, 5.4, 5.5 below. We present the performance of the various methods in terms of the averaged Macro-F1 and Macro-F1 scores, along with their confidence margins in terms of the standard deviation values over five runs. We have used logistic regression to train on the generated embeddings with a training ratio of 0.3. Micro-F1 measures the accuracy of label predictions and Macro-F1 computes the mean accuracy over all the classes. We have compared the performance of PBGENA at both 2000 and 8000 dimensions. When at $N = 2000$, PBGENA is about four to eight times more economical in space, and at $N = 8000$ PBGENA is about twice as memory efficient. We have marked the best performances in bold and colored our novel algorithm in green for better readability. Since BGENA and PBGENA are essentially the same method, we do not perform the experiments separately for BGENA.

In Table 5.3, we have compiled the node classification results for small graphs. We can see that PBGENA at $N = 2000$ is superior to all other methods on the Facebook network and PBGENA at $N = 8000$ is the best performing on the Cora dataset. Except for Wikipedia, we are competitive with the other baselines on all other graphs. For Wikipedia, we have come a close second with PANE and even though we are behind MUSAE, it should be noted that MUSAE on the Wikipedia network is $1000\times$ slower than PBGENA (Table 5.6).

[11] `https://github.com/tapadeep/PBGENA`

Table 5.3: Node Classification results with small graphs

Algorithm	Graph							
	Wikipedia		Cora		CiteSeer		Facebook	
	Macro-F1	Micro-F1	Macro-F1	Micro-F1	Macro-F1	Micro-F1	Macro-F1	Micro-F1
TADW	62.37 ± 1.39	71.27 ± 1.33	62.69 ± 0.94	65.61 ± 1.04	60.45 ± 1.12	64.33 ± 1.23	05.98 ± 0.44	29.35 ± 1.64
TENE	63.22 ± 2.60	72.30 ± 1.64	55.90 ± 1.74	61.50 ± 1.33	58.06 ± 1.39	65.31 ± 1.65	02.30 ± 1.69	07.57 ± 1.48
SINE	66.22 ± 3.55	76.57 ± 1.08	79.23 ± 1.00	80.79 ± 1.03	62.70 ± 0.91	66.80 ± 1.05	21.81 ± 0.80	66.70 ± 1.68
ASNE	67.65 ± 4.26	78.34 ± 1.15	79.86 ± 1.54	81.70 ± 1.15	64.22 ± 1.37	67.81 ± 1.33	26.66 ± 1.96	73.57 ± 1.62
PANE	68.65 ± 2.75	82.05 ± 0.70	85.19 ± 1.12	86.59 ± 1.18	$\mathbf{71.30 \pm 0.92}$	$\mathbf{75.57 \pm 0.98}$	14.07 ± 0.34	63.91 ± 1.17
FN[a]	56.63 ± 3.40	71.16 ± 1.68	81.27 ± 1.09	82.66 ± 1.02	67.16 ± 1.42	73.18 ± 1.58	24.97 ± 0.65	69.48 ± 1.38
MUSAE	$\mathbf{76.90 \pm 3.11}$	$\mathbf{88.06 \pm 0.82}$	82.79 ± 1.14	83.79 ± 0.77	65.91 ± 1.42	69.62 ± 1.25	31.59 ± 1.82	75.64 ± 1.99
(P)BGENA	66.32 ± 2.68	79.89 ± 1.51	83.82 ± 0.94	85.05 ± 0.71	68.99 ± 1.34	72.61 ± 0.62	32.94 ± 1.61	$\mathbf{76.11 \pm 1.38}$
(P)BGENA (N=8000)	69.92 ± 2.02	80.03 ± 1.80	$\mathbf{85.95 \pm 0.95}$	$\mathbf{86.67 \pm 0.53}$	68.77 ± 1.02	72.75 ± 1.03	$\mathbf{33.82 \pm 1.78}$	75.31 ± 0.64

[a]FN: FeatherNode Embedding Strategy

*Best Performance in bold

Table 5.4: Node Classification results with medium-sized graphs

Algorithm	Graph									
	BlogCatalog		Flickr		PubMed		PPI		Twitter	
	Macro-F1	Micro-F1	Macro-F1	Micro-F1	Macro-F1	Micro-F1	Macro-F1	Micro-F1	Macro-F1	Micro-F1
TADW	84.39 ± 0.84	84.77 ± 0.93	73.73 ± 0.90	73.79 ± 0.95	83.71 ± 0.18	83.61 ± 0.12	18.52 ± 0.12	44.66 ± 0.15	14.38 ± 0.63	34.58 ± 1.02
TENE	81.41 ± 1.56	81.59 ± 1.50	21.32 ± 1.49	24.09 ± 1.61	30.21 ± 0.42	40.70 ± 0.63	13.58 ± 0.20	42.45 ± 0.14	09.19 ± 0.66	24.57 ± 0.84
SINE	93.19 ± 0.32	93.34 ± 0.33	85.18 ± 0.71	85.32 ± 0.67	$\mathbf{87.59 \pm 0.19}$	$\mathbf{87.78 \pm 0.16}$	19.65 ± 0.13	45.60 ± 0.14	19.01 ± 0.18	51.74 ± 0.54
ASNE	91.90 ± 0.78	91.99 ± 0.78	$\mathbf{85.40 \pm 0.45}$	$\mathbf{85.55 \pm 0.33}$	87.18 ± 0.62	87.25 ± 0.66	33.08 ± 0.51	52.50 ± 0.35	20.57 ± 0.50	52.67 ± 0.68
PANE	86.61 ± 0.61	86.76 ± 0.60	76.93 ± 0.74	77.20 ± 0.68	87.13 ± 0.49	87.29 ± 0.44	36.21 ± 0.20	$\mathbf{54.60 \pm 0.12}$	-	-
FN	70.97 ± 0.57	71.38 ± 0.66	56.22 ± 0.60	56.67 ± 0.42	83.18 ± 0.32	83.40 ± 0.36	24.70 ± 0.17	49.73 ± 0.07	14.57 ± 0.20	40.63 ± 0.28
MUSAE	72.17 ± 1.81	72.43 ± 1.66	56.18 ± 0.65	56.47 ± 0.58	84.57 ± 0.55	85.30 ± 0.49	35.23 ± 0.21	53.79 ± 0.16	-	-
(P)BGENA	91.36 ± 1.01	91.57 ± 0.96	76.64 ± 1.05	76.43 ± 1.06	86.51 ± 0.46	86.68 ± 0.45	40.21 ± 0.41	54.56 ± 0.28	25.85 ± 0.83	57.27 ± 0.47
(P)BGENA (N=8000)	$\mathbf{95.00 \pm 0.33}$	$\mathbf{95.08 \pm 0.35}$	85.21 ± 0.80	85.25 ± 0.75	87.41 ± 0.20	87.54 ± 0.24	$\mathbf{44.70 \pm 0.19}$	53.70 ± 0.14	$\mathbf{28.05 \pm 0.60}$	$\mathbf{60.15 \pm 0.77}$

-either ran out of memory or did not stop or crashed unexpectedly

Table 5.4 demonstrates the results of node classification on medium-sized graphs. We find that PBGENA is the best method for networks like BlogCatalog, PPI, and Twitter. We are very close for graphs like Flickr and PubMed, differing in the decimal places when compared to the winners. For PPI and Twitter, we are significantly ahead in Macro-F1 scores meaning that PBGENA embeddings are better equipped to distinguish between different classes.

Table 5.5: Node Classification results with large graphs

Algorithm	Graph							
	Google+		Reddit		TWeibo		MAKG[a]	
	Macro-F1	Micro-F1	Macro-F1	Micro-F1	Macro-F1	Micro-F1	Macro-F1	Micro-F1
TADW	-	-	-	-	-	-	-	-
TENE	-	-	-	-	-	-	-	-
SINE	-	-	-	-	-	-	-	-
ASNE	52.83 ± 0.31	78.85 ± 0.17	$\mathbf{92.47 \pm 0.12}$	$\mathbf{94.18 \pm 0.10}$	15.84 ± 0.06	56.54 ± 0.13	-	-
PANE	29.06 ± 0.20	59.56 ± 0.28	-	-	16.14 ± 0.02	57.43 ± 0.07	-	-
FN	30.77 ± 0.35	53.52 ± 0.68	84.69 ± 0.17	89.49 ± 0.10	12.93 ± 0.02	51.62 ± 0.02	-	-
MUSAE	-	-	-	-	-	-	-	-
(P)BGENA	58.53 ± 0.26	83.13 ± 0.35	86.65 ± 0.15	87.60 ± 0.06	16.35 ± 0.03	55.20 ± 0.09	$\mathbf{36.99 \pm 0.16}$	$\mathbf{49.35 \pm 0.21}$
(P)BGENA (N=8000)	$\mathbf{63.40 \pm 0.33}$	$\mathbf{89.15 \pm 0.35}$	89.85 ± 0.14	91.09 ± 0.14	$\mathbf{18.20 \pm 0.05}$	$\mathbf{57.87 \pm 0.04}$	-	-

[a]Experiments with MAKG were performed with a training ratio of 0.1

We present the results of node classification on large graphs in Table 5.5, and we are the best performers in Google+ and TWeibo. BGENA was the only method able to perform embedding on MAKG within the system's 270GB memory limit. For MAKG we have used a training ratio of 0.1 because of our system constraints.

From the tables above we can see that the older methods don't quite match up to PBGENA and PBGENA either beats or remains competitive when compared to the newer methods. These tables do not quite show the superiority of PBGENA in terms of its biggest asset, i.e., speed. Therefore we present the time required to generate the embedding in Table 5.6. This table demonstrates just how fast PBGENA is against its competitors. Note that some of these methods had the parallel capability and even then falls significantly behind in speed. The only methods which are comparable to PBGENA with respect to its embedding speed are FeatherNode and BGENA (serial version of PBGENA) and it should be noted that FeatherNode failed terribly in the task of node classification.

Table 5.6: Embedding Time (in seconds) for various methods

Graph	Algorithm								
	TADW	**TENE**	**SINE**	**ASNE**	**PANE**	**FN**	**MUSAE**	**BGENA**	**PBGENA**
Wikipedia	11.80	253.07*	69.65**	108.91	17.04	8.56	1211.09	4.25	**0.16**
Cora	2.43	21.20	58.75	15.37	1.68	0.31	156.84	0.72	**0.08**
CiteSeer	2.07	34.29	70.09	24.20	1.96	0.34	239.07	0.95	**0.12**
Facebook	52.31	44.30	90.03	30.32	4.17	0.50	163.95	1.59	**0.37**
BlogCatalog	294.14	119.23	175.58	68.11	107.10	1.48	629.77	3.18	**0.52**
Flickr	753.27	106.64	259.02	76.94	231.78	1.43	546.09	3.71	**0.67**
PubMed	29.87	221.07	416.14	156.50	10.08	1.31	2226.47	6.52	**0.56**
PPI	1106.12	437.02	1102.36	440.23	5.57	4.87	1066.34	25.56	**3.03**
Twitter	5074.39	14040.34	3648.08	9612.05	-	419.56	-	243.61	**6.26**
Google+	-	-	-	24709.18	11741.49	419.28	-	869.14	**17.87**
Reddit	-	-	-	18841.99	-	634.05	-	1006.16	**65.10**
TWeibo	-	-	-	20920.17	5672.88	526.61	-	1173.07	**115.93**
MAKG	-	-	-	-	-	-	-	**30089.47**	-

*Time 1000× of PBGENA marked in red

**Time 100× of PBGENA marked in orange

5.5 Link Prediction

Link Prediction tries to test how well can an embedding scheme cope with the dynamic nature of graphs. In the real world, graphs are not static but form links (or associations) over time. To model this problem, we have randomly removed 30% of the edges from the graph and then generated embeddings with the residual graph using PBGENA and the other baselines. We further generate a negative edge set containing pairs of nodes that do not have an edge between them and leave out 30% of the negative edges as a test set. Finally, we compute the cosine distance between all pairs of negative and positive edges and label the positive and negative edges as 1 and 0 respectively. We have used logistic regression to the training part (70%) of this data. We have used

the area under receiver operator characteristic graph and average precision as the two metrics to evaluate link prediction.

Table 5.7: Link Prediction results with small graphs

Algorithm	Graph							
	Wikipedia		Cora		CiteSeer		Facebook	
	AUC[a]	AP[b]	AUC	AP	AUC	AP	AUC	AP
TADW	77.45 ± 0.40	81.50 ± 0.40	59.33 ± 0.96	59.59 ± 0.50	64.00 ± 0.85	65.39 ± 0.10	57.40 ± 0.21	59.20 ± 0.27
TENE	87.50 ± 0.34	89.30 ± 0.36	48.76 ± 0.94	54.58 ± 0.85	59.74 ± 1.58	64.17 ± 1.13	59.97 ± 0.52	59.92 ± 0.78
SINE	76.82 ± 0.16	81.97 ± 0.15	87.05 ± 0.49	89.18 ± 0.49	92.75 ± 0.45	93.81 ± 0.50	87.97 ± 0.25	86.97 ± 0.26
ASNE	85.41 ± 0.52	88.76 ± 0.37	89.90 ± 0.57	91.39 ± 0.62	94.24 ± 0.29	95.17 ± 0.25	91.88 ± 0.16	90.96 ± 0.24
PANE	$\mathbf{94.94 \pm 0.27}$	$\mathbf{95.60 \pm 0.22}$	$\mathbf{91.01 \pm 0.44}$	$\mathbf{92.59 \pm 0.34}$	$\mathbf{95.78 \pm 0.38}$	$\mathbf{96.45 \pm 0.23}$	98.27 ± 0.07	98.02 ± 0.07
FN	87.94 ± 0.37	90.14 ± 0.30	82.32 ± 0.51	84.89 ± 0.69	84.30 ± 0.22	87.39 ± 0.35	96.72 ± 0.09	97.10 ± 0.08
MUSAE	91.55 ± 0.44	92.75 ± 0.30	90.03 ± 0.57	91.77 ± 0.27	94.58 ± 0.29	95.52 ± 0.32	$\mathbf{98.54 \pm 0.06}$	$\mathbf{98.24 \pm 0.13}$
(P)BGENA	86.63 ± 0.42	88.20 ± 0.42	89.44 ± 0.17	90.30 ± 0.44	93.45 ± 0.34	94.57 ± 0.21	97.72 ± 0.05	97.85 ± 0.05

[a]AUC: Area Under the Receiver Operating Characteristic Curve

[b]AP: Average Precision Score

Table 5.8: Link Prediction results with medium-sized graphs

Algorithm	Graph							
	BlogCatalog		Flickr		PubMed		PPI	
	AUC	AP	AUC	AP	AUC	AP	AUC	AP
TADW	51.25 ± 0.11	51.04 ± 0.06	51.13 ± 0.29	51.36 ± 0.18	60.97 ± 0.65	60.28 ± 0.61	50.64 ± 0.06	53.68 ± 0.10
TENE	51.65 ± 0.30	51.73 ± 0.29	71.11 ± 0.82	75.56 ± 0.79	53.66 ± 0.48	54.71 ± 0.60	50.88 ± 0.13	53.50 ± 0.11
SINE	65.23 ± 0.19	66.56 ± 0.12	48.83 ± 0.35	48.06 ± 0.25	81.84 ± 0.16	84.83 ± 0.15	50.34 ± 0.18	53.84 ± 0.16
ASNE	62.81 ± 0.13	63.88 ± 0.19	44.38 ± 0.13	45.48 ± 0.12	89.69 ± 0.13	90.53 ± 0.17	90.40 ± 0.04	90.31 ± 0.04
PANE	67.90 ± 0.12	68.84 ± 0.23	59.00 ± 0.12	62.99 ± 0.12	$\mathbf{93.58 \pm 0.11}$	$\mathbf{93.86 \pm 0.14}$	70.42 ± 0.09	67.88 ± 0.12
FN	80.98 ± 0.07	80.64 ± 0.08	85.03 ± 0.08	85.83 ± 0.10	86.49 ± 0.19	87.38 ± 0.15	81.38 ± 0.07	81.79 ± 0.04
MUSAE	$\mathbf{82.01 \pm 0.33}$	$\mathbf{81.91 \pm 0.31}$	71.30 ± 0.64	70.27 ± 0.60	94.86 ± 0.05	95.13 ± 0.05	$\mathbf{97.55 \pm 0.02}$	96.42 ± 0.05
(P)BGENA	79.07 ± 0.14	78.77 ± 0.13	$\mathbf{90.34 \pm 0.09}$	$\mathbf{90.41 \pm 0.10}$	91.51 ± 0.10	92.06 ± 0.16	95.51 ± 0.03	$\mathbf{96.42 \pm 0.02}$

Table 5.9: Link Prediction results with large graphs

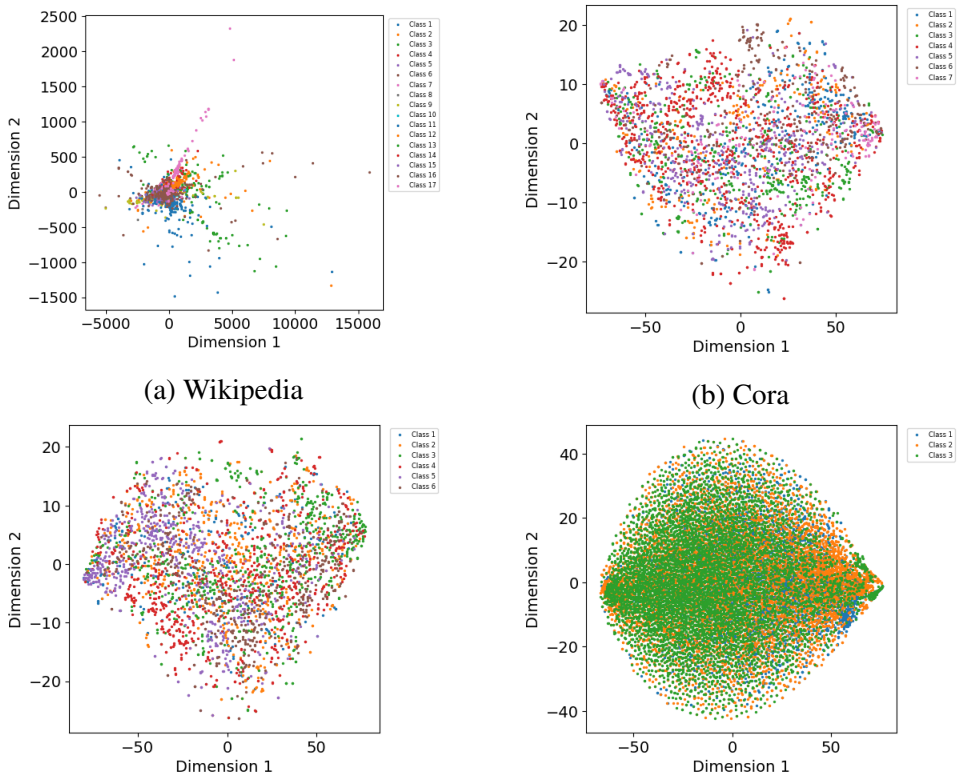
Algorithm	Graph							
	Twitter		Google+		Reddit		TWeibo	
	AUC	AP	AUC	AP	AUC	AP	AUC	AP
TADW	87.25 ± 0.07	89.80 ± 0.05	-	-	-	-	-	-
TENE	97.94 ± 0.17	97.61 ± 0.21	-	-	-	-	89.98 ± 0.80	92.99 ± 0.58
SINE	92.90 ± 0.06	94.68 ± 0.04	-	-	-	-	-	-
ASNE	91.17 ± 0.08	94.12 ± 0.04	74.56 ± 2.92	73.61 ± 2.51	78.88 ± 0.05	82.04 ± 0.17	85.28 ± 0.04	86.47 ± 0.03
PANE	-	-	$\mathbf{96.98 \pm 0.00}$	94.35 ± 0.01	-	-	80.82 ± 0.16	74.96 ± 0.11
FN	$\mathbf{98.59 \pm 0.01}$	$\mathbf{98.58 \pm 0.01}$	96.56 ± 0.00	$\mathbf{95.91 \pm 0.01}$	$\mathbf{93.95 \pm 0.01}$	$\mathbf{94.49 \pm 0.00}$	67.38 ± 0.02	55.58 ± 0.01
MUSAE	-	-	-	-	-	-	-	-
(P)BGENA	97.44 ± 0.01	97.51 ± 0.01	93.68 ± 0.04	93.34 ± 0.11	88.10 ± 0.01	86.16 ± 0.01	$\mathbf{96.21 \pm 0.04}$	$\mathbf{96.38 \pm 0.04}$

The results of link prediction have been compiled in tables 5.7, 5.8 and 5.9. We can see that PBGENA is the best performing in graphs like Flickr, PPI, and TWeibo and for the other graphs, we are always competitive. We can see some of the methods which work well on a specific graph perform badly on other graphs but PBGENA has consistently been among the top performers, with an average precision score of more than 90% in nine out of the twelve graphs. Also, PBGENA is faster than all the baselines listed above. This makes PBGENA embeddings highly scalable as compared to our baselines.

5.6 Graph Visualization

Graph visualization is the task of creating meaningful visualizations from the embeddings of the graphs. To perform the visualization, we first choose four citation networks: Cora, CiteSeer, PubMed, and Wikipedia, and compress the network using TADW, PBGENA, and some other embedding schemes. Then we perform t-SNE [62] to the node embeddings to bring the dimension all the way down to 2. Finally, we plot them on a 2D plane and color them according to the labels of the nodes. The results of the visualizations obtained from TADW and PBGENA are displayed with figures 5.1 and 5.2 respectively.

Figure 5.1: Visualizing graphs through TADW embeddings



(a) Wikipedia

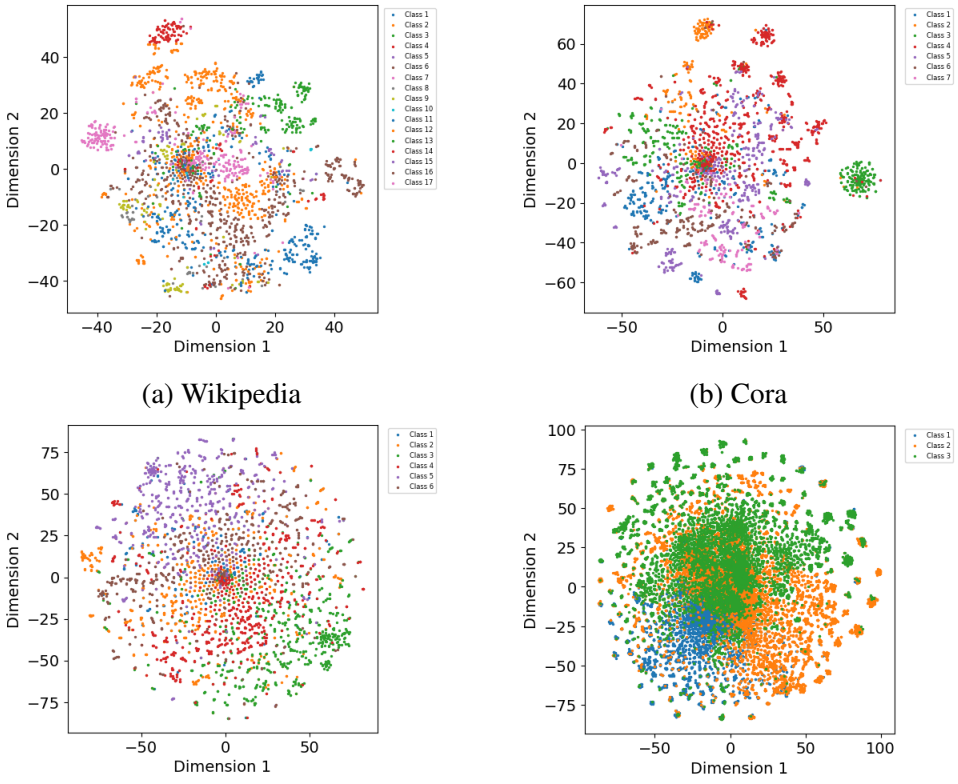
(b) Cora

(c) CiteSeer

(d) PubMed

The task of distinguishing clear boundaries between different classes is not trivial. Often the labels are very closely related like in the Cora dataset, where all the papers considered were from the field of machine learning (subsection 5.1.2) and it is a similar situation with the CiteSeer network. The PubMed network only considers studies in the area of diabetics. Even with all these limitations, we can clearly see from the figures that PBGENA was able to produce comparably superior visualizations than TADW. The various classes are much more clearly separated in PBGENA's embedding.

Figure 5.2: Visualizing graphs through PBGENA embeddings



(a) Wikipedia

(b) Cora

(c) CiteSeer

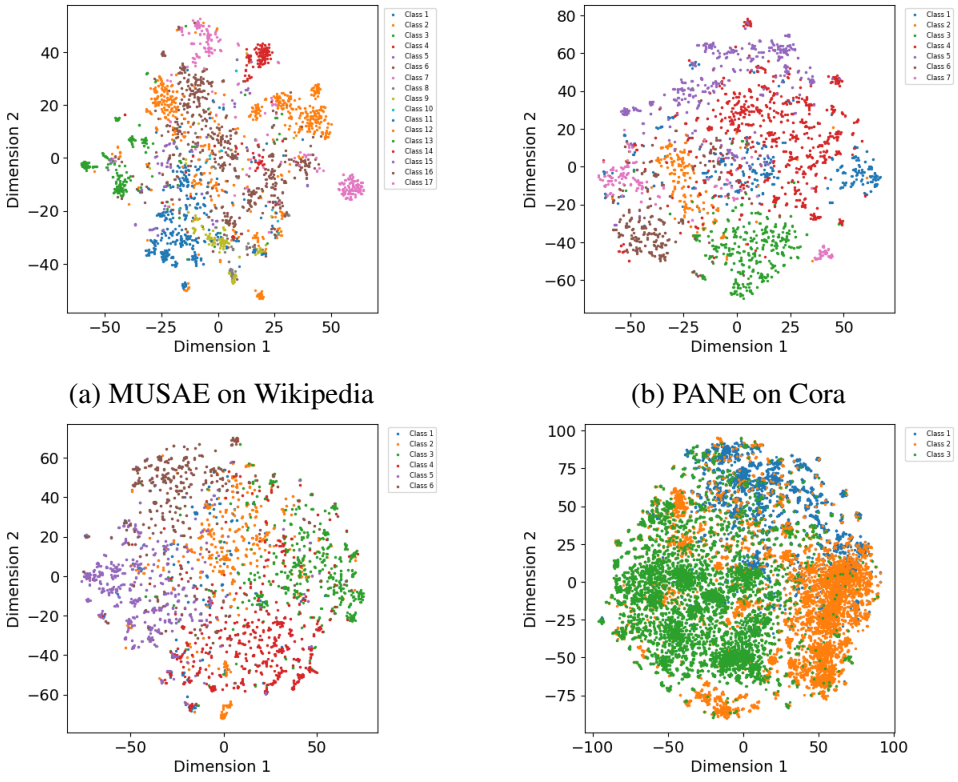
(d) PubMed

In Figure 5.3, we perform the same task of graph visualization using the embeddings of the best performers (second-best if PBGENA was the best) in node classification. We can see that PBGENA embeddings are just as capable of producing good visualization as the state-of-the-art methods. It should be noted that PBGENA produces binary embeddings while the others produce rich real-valued embeddings. Also, PBGENA embeddings were compressed from $N = 2000$ to $N = 2$ as compared to TADW and the other baselines whose embeddings were compressed from $N = 250$ to $N = 2$, meaning PBGENA incurred a greater loss due to compression.

5.7 Parameter Sensitivity

In this section, we lay out the various sensitivity experiments performed using the numerous hyperparameters of BGENA and PBGENA. We broadly categorize these experiments into three subsections involving testing the model's time, robustness, and stability in different situations.

Figure 5.3: Visualizing graphs through various baselines



(a) MUSAE on Wikipedia

(b) PANE on Cora

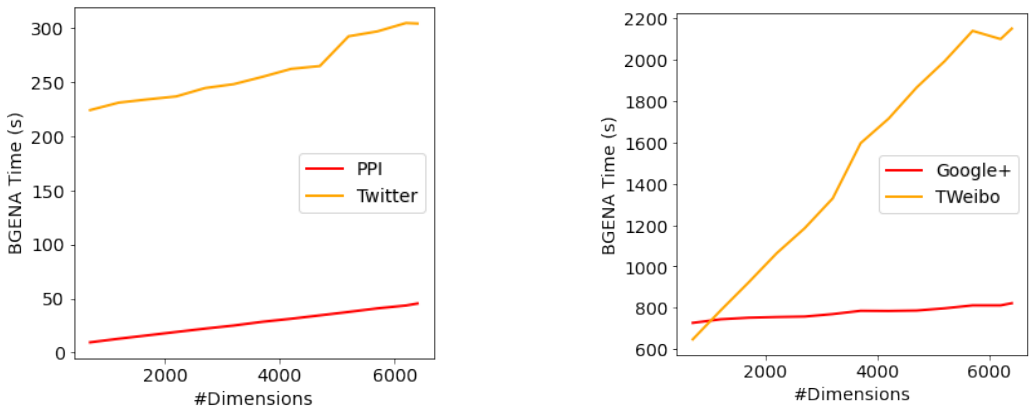
(c) PANE on CiteSeer

(d) SINE on PubMed

5.7.1 Speed

The first set of experiments we performed on time was to identify the relationship between the time for embedding and the number of dimensions (N) for both BGENA and PBGENA. According to Equation 3.2, BGENA is linear in terms of $\mathcal{O}(N|V|)$ so for a graph with a fixed number of nodes, we expect the curve to have linear growth and that is what we observe from Figure 5.4.

Figure 5.4: BGENA Embedding Time v Dimensions

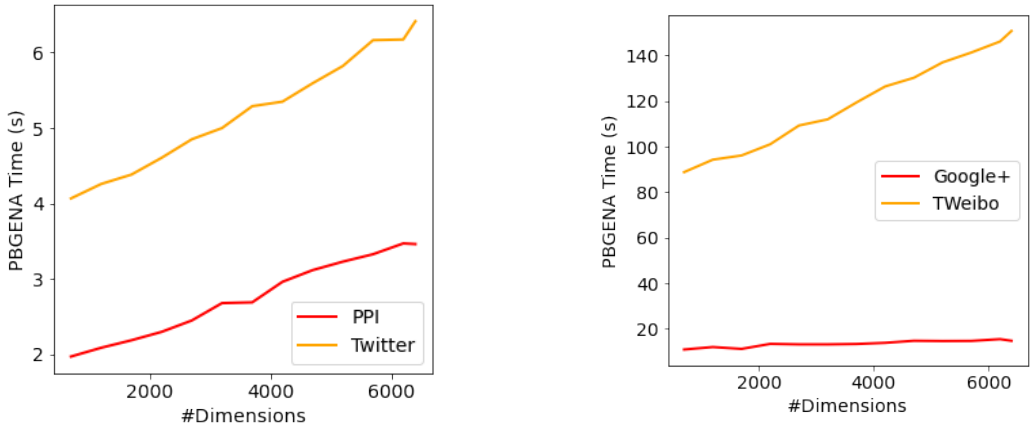


(a) Smaller Graphs

(b) Large Graphs

Figure 5.5 demonstrates the same experiment of embedding time versus time, but with PBGENA instead. Again, we have seen previously in subsection 4.4.1, that the time complexity of PBGENA is linear in $\mathcal{O}\left(\frac{N|V|}{p}\right)$. This means that we also expect a linear time growth for PBGENA but due to the presence of the factor p, the curve will be much flatter. This property can be observed in Figure 5.5 where even when the number of dimensions is increased by a factor of 6, the required time does not double. This can be explained by the fact that experiments showed in Figure 5.5 were performed with the number of cores set to 32.

Figure 5.5: PBGENA Embedding Time v Dimensions



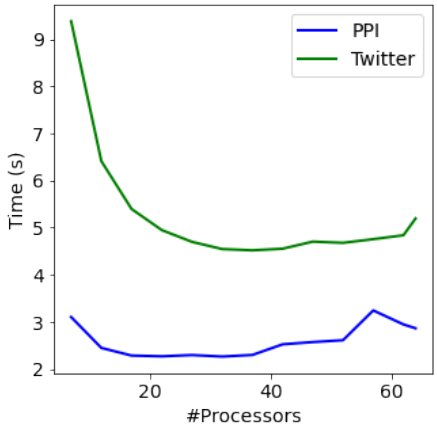
(a) Smaller Graphs

(b) Large Graphs

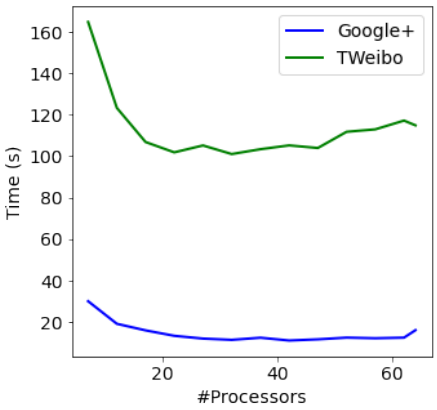
In the analysis section (Appendix B), we have shown that the speedup obtained using PBGENA is increasing with respect to the number of processors in use, at least in theory. In this section, we test the actual relationship between the number of cores and the time of embedding. Figure 5.6 displays the results and we can observe that the time required for embedding falls sharply with an increase in the number of processors but only to a certain extent and then saturates. Sometimes it may look as if the time is increasing when using too many cores, this is a common phenomenon in parallel computing which occurs when the communication cost between multiple cores starts to affect the throughput. Curves like these can potentially help us set a standard for PBGENA on the number of cores to practically employ, given the scale of the graph.

Finally, in timed experiments, we present the holy grail of this study in Figure 5.7, i.e., the speedup obtained by PBGENA over its competitors. Missing bars in the plot indicate that the algorithm in question was slower than $1000\times$ the speed of PBGENA for

Figure 5.6: Embedding Time v Number of Cores



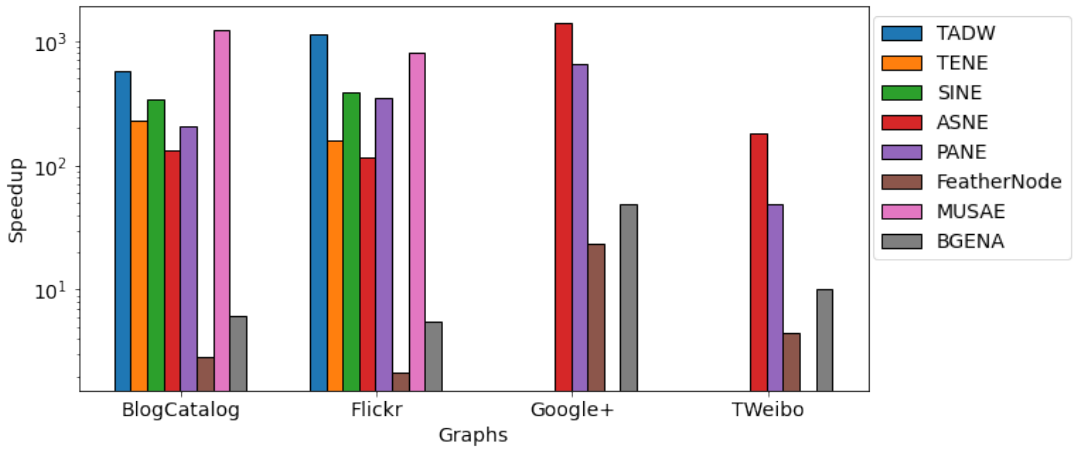
(a) Smaller Graphs



(b) Large Graphs

the same task. Note that some of our baselines are implicitly parallelized, and whenever possible we let our baselines run with at most 32 cores just like PBGENA. We can easily observe from Figure 5.7 that except for FeatherNode [59] and BGENA, no other method even comes close to matching PBGENA in terms of embedding speed. From the tables of section 5.4, it is apparent that FeatherNode severely lacks the capability to produce embeddings good enough for node classification, despite its speed.

Figure 5.7: PBGENA's Speedup

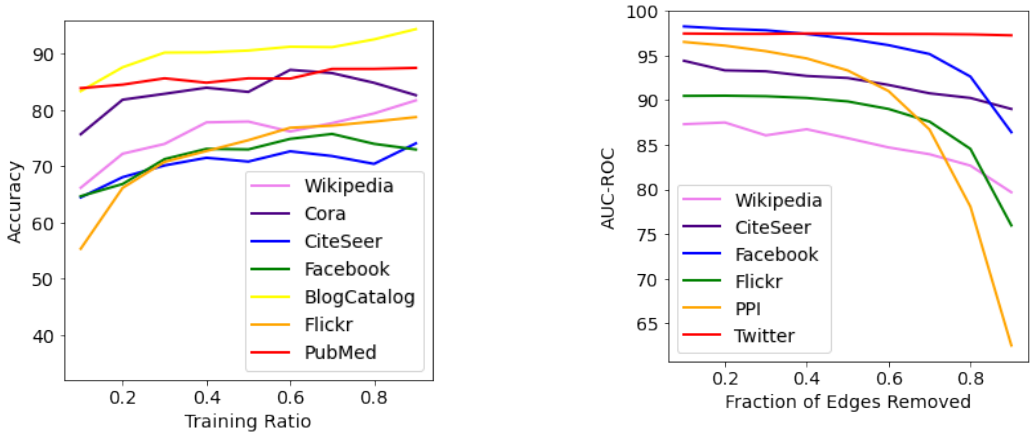


Some of our baselines are so far behind PBGENA in time that it hardly matters even if they outperform us by small margins in prediction because when it comes to large emerging networks, these methods will simply stutter. Therefore, from these results, we can conclude that PBGENA is one of the most lightweight, high-utility NRL schemes.

5.7.2 Robustness

In this section, we test the robustness of PBGENA on extremely challenging tasks and see how well it performs both in terms of node classification and link prediction. In node classification, we reduce the fraction of the embeddings supplied to logistic regression for training steadily from 0.9 to 0.1. On the other hand, in link prediction, we increase the fraction of edges removed from the original to the reduced graph from 0.1 to 0.9. The results can be found in Figure 5.8.

Figure 5.8: Testing for PBGENA's Robustness



(a) Accuracy v Training Ratio

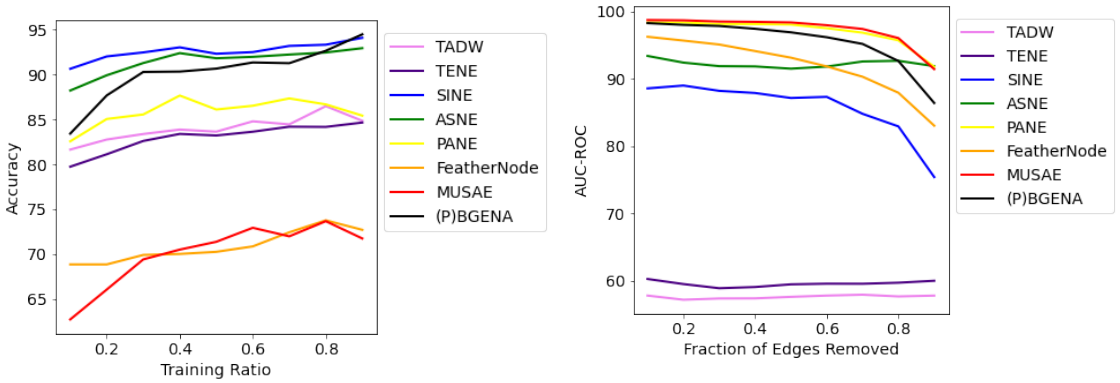
(b) AUC-ROC v Fraction of edges removed

From Figure 5.9a, we find the expected trend of node classification accuracy increasing with an increase in training ratio. However we can make some interesting observations here: the accuracy of none of the graphs falls below 50% even when training with only 10% of the data, and in some cases, the accuracy never falls below 80%. This ensures that our embeddings are robust to fluctuating training ratios.

Figure 5.9b demonstrates the results of link prediction on changing the fraction of removed edges. We can actually observe the obvious trend of the AUC-ROC score dropping when a larger chunk of the edge set is eliminated from the residual graph. However, we find that some curves dip more than others and this property is actually explainable through the hyperparameters of these graphs Table A.2. A graph like PPI has a low value for α (attribute fraction) used in its optimal hyperparameter setting. This means that PPI relies heavily on the topology of the graph for link prediction as compared to its attributes. Therefore removing a large fraction of the edges disrupts the structure of the graph and causes a drop in the AUC-ROC score. Similarly, graphs like Twitter

which has a high value of α are hardly affected by the fraction of edges removed. We can also find in Figure 5.9b that even with 90% edges removed, the performance of PBGENA never falls below 60% in the AUC-ROC score thus ensuring the robustness of PBGENA in link prediction.

Figure 5.9: Robustness for various methods



(a) Accuracy v Training Ratio of the Baselines

(b) AUC-ROC v Fraction of Edges Removed of the Baselines

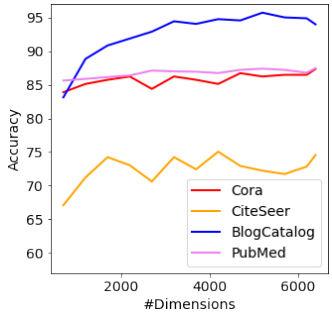
We have also performed robustness experiments for our baselines using a similar setup of experiments and again find that PBGENA is competitive with all the baselines at all levels. The results have been presented in Figure 5.9.

5.7.3 Stability

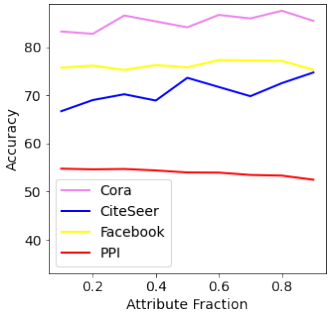
The primary purpose of stability experiments is to test how well a model performs with wildly fluctuating hyperparameters. These tests try to quantify how difficult it is to fine-tune a model for specific data (in our case, graphs). If the performance of the model fluctuates greatly with a slight change in the value of the hyperparameters, then the model is extremely sensitive and vice-versa. Ideally, we would like a steady performance of our method even with changing hyperparameters, and even when a model fails then it is better to fail gradually. We have compiled our parameter sensitivity results in Figure 5.10. These results were obtained by fixing the other hyperparameter values in accordance with Appendix A, and changing the specific hyperparameter under consideration from its high range to low range.

From Figure 5.10a, we note that the accuracy of embedding typically an increases with increase in dimension, as one would expect. However, more importantly, the accuracy

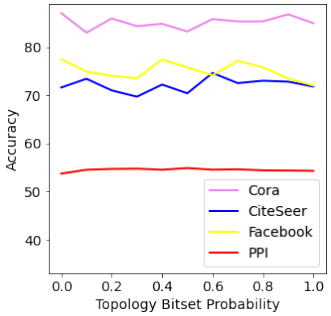
Figure 5.10: Testing for hyper-parameter sensitivity



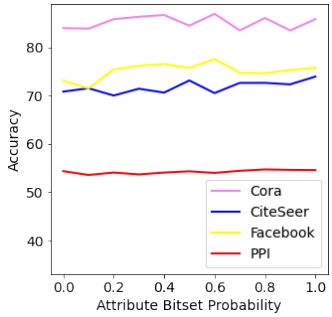
(a) Accuracy v Dimensions (N)



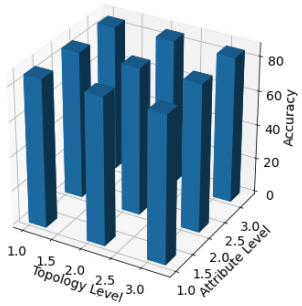
(b) Accuracy v Attribute Fraction (α)



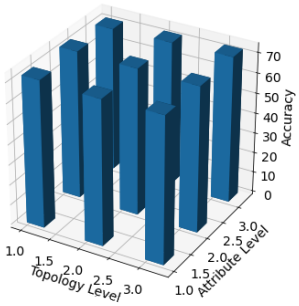
(c) Accuracy v Topology bitset probability (b_t)



(d) Accuracy v Attribute bitset probability (b_a)



(e) Cora Accuracy w.r.t Attribute Level (l_a) and Topology Level (l_t)



(f) Citeseer Accuracy w.r.t Attribute Level (l_a) and Topology Level (l_t)

does not plummet too much when the number of dimensions is decreased to $N = 700$, ensuring stability.

From figures 5.10b, 5.10c and 5.10d, we can make out that that the only parameter that fluctuates the most is α and once that is stabilized to its best setting, b_a and b_t remain relatively stable. Again, we can see that the variability of the hyperparameters is within acceptable range.

Finally, in figures 5.10e and 5.10f, we demonstrate the effect of multi-level edge propagation on node classification results using two small graphs. We find that for these two graphs we cannot gain a significant advantage from using the hyperparameters l_t and l_a. It should be noted that we had set $f_t = f_a = 2$ during our experiments (Algorithm 3).

CHAPTER 6

CONCLUSION

This is the final section of the thesis where we summarize the research work done over the past year with some salient points and also point to some interesting directions in which the study can be extended in the future.

6.1 Summary

In this study, we attempt to address the problem of network representation learning, which refers to the task of generating low dimensional embeddings for each node in a graph with attributes associated with each node. To that end, we propose a novel sketching-based ANE solver named BGENA which leverages efficient binary sketching method BinSketch and edge propagation to generate fast and high-utility embeddings. Edge propagation is the process of propagating a digest of the sketches to the neighboring nodes to strengthen the first-order proximity between directly connected nodes. The strength of the digests to be propagated can be controlled through a hyperparameter named bitset probability. This concept of using edge propagation for modeling proximity can be extended arbitrarily by using level-wise edge propagation to model any arbitrary order of proximity between nodes in a graph. BGENA produces sparse binary embeddings as its output thereby enabling fast bitarray/sparse-matrix representations to save memory. We then go on to provide a scheme for parallelizing BGENA named PBGENA using a system's multi-core capability. PBGENA is faster than any existing ANE solvers known to us with PBGENA achieving a speedup of $16\times$ over BGENA and sometimes over $1000\times$ over some of our baselines.

PBGENA embeddings achieve performance comparable to the state-of-the-art baselines, often outperforming them, at a fraction of the time because of the use of purely bitwise operations. We evaluate our results on thirteen real-world data sets and against an array of seven state-of-the-art baselines on tasks like node classification, link prediction, and graph visualization. Our dataset is composed of a variety of graphs from citation networks, to ego networks, to biological networks. Apart from performance

testing, we also present a rich collection of parameter sensitivity experiments to have an idea of how PBGENA works both in terms of performance and speed with changing circumstances. We also present some initial analysis in the appendix and point to some future directions for this study.

6.2 Takeaways

The major takeaways from this study are listed below:

- With modern data increasingly being stored as sparse billion-scale networks, it is difficult to scale learning-based methods and still maintain acceptable performance. Therefore, sketching-based methods are a great alternative for the future of ANE solvers. A method like PBGENA has the potential to become a benchmark for the fast sketch-based methods to come in the future.

- BGENA uses the efficient dimension reduction method BinSketch and a novel bitwise edge propagation mechanism to generate embeddings of nodes in a graph. BGENA outputs binary embeddings which allows storage into memory-efficient data structures like bitarrays and sparse matrices. BGENA is the only method that was able to embed MAKG within our system's 270GB memory cap in just 8 hours.

- PBGENA, parallelized BGENA, is the parallel version of BGENA which uses MPI to leverage a system's multi-core architecture to speedup BGENA significantly. PBGENA is $1000\times$ faster than some of our baselines and $100\times$ faster than the state-of-the-art baselines. PBGENA was able to embed TWeibo, a graph with 2 million nodes and 50 million edges, in less than two minutes.

- As shown in Figure 4.2, the task of embedding topology and attributes are independent in PBGENA. This means that PBGENA can easily work for graphs without any attribute support making it a very versatile embedding method.

6.3 Future Work

In this section, we discuss some of the future directions for this study. These include possible extensions to the already existing method, some ideas to generalize the algorithm, some alternate theory and implementations, and some improvements.

6.3.1 Hyperparameter Tuning

One of the only hurdles to using PBGENA is undoubtedly the fine-tuning of its hyperparameters. Since we are more than $100\times$ faster than our competitors, running PBGENA a few times usually does not hurt. Also, this is not a problem unique to PBGENA but it is still not insignificant. Since we often have a lot of meta-data associated with our graphs, it may be possible to use them to recommend the hyperparameters fit for particular graphs or at least reduce the space of hyperparameter values. We believe a future study in the direction of analyzing PBGENA can try to use graph properties like structure, communities, the average degree to recommend hyperparameters.

6.3.2 PBGENA with alternate hashing

The task of embedding nodes in a graph is akin to the task of dimension reduction [15]. The literature on dimension reduction is abundant with various types of hashing schemes capable of preserving complex similarity measures which can be ideal for modeling proximities in a network setting. One very well-known scheme is Feature Hashing [63] which typically uses MurmurHash3 function [1] to encode features and then produces the sketching in any desired dimension using a linear probing like approach. So we can potentially think of using alternate sketching methods to come up with the sketches and then use edge propagation to pass digests of those sketches to neighboring nodes.

6.3.3 Weighted Graphs with Real-valued Attributes

Graphs in the real world can come in all shapes and forms: weighted, directed, nodes with real-valued attributes, and so on. PBGENA has been tested on both directed (Cora, MAKG) and undirected (Facebook) graphs (refer to section 5.1 for information related to datasets) and we have produced excellent results in both cases. PBGENA is actually comparable with PANE which takes into account the directedness of the graph even when treating the edges of a directed graph to be undirected. For graphs with real-valued attributes and weighted edges, an easy solution would be to use the CABIN algorithm [2] which can sketch categorical data. Binning combined with CABIN can sketch real-valued data, but extensive experiments are needed to confirm if at all we can

[1] `https://github.com/aappleby/smhasher`

produce better results using CABIN by utilizing the exact values of the attributes and weights.

6.3.4 PBGENA for Attribute Inference

One lesser-known graph task is called attribute inference [64], which is similar to link prediction but for attributes. Attribute Inference tries to predict the attribute set of the nodes given its embeddings and requires embedding of both nodes and attributes. Typically, attribute inference tries to use the normalized inner product score between the node and attribute embeddings as the probability for the presence of the attribute in that specific node. We have not tried to use PBGENA to perform attribute inference but it is something that can be in the scope of a future study.

6.3.5 PBGENA with alternate partitioning

In the algorithm for PBGENA, we have used random partitioning to divide the nodes between different processors. This means that on average $|E| * (1 - 1/p)$ number of cross edges appear between different partitions which constitutes the main overhead for PBGENA (refer to subsection 4.4.1). To reduce this, we have investigated using the METIS toolkit [50] that provides a graph partitioning with the number of cross-edges minimized. METIS was not very useful because it itself became the bottleneck for PBGENA. However, this does not mean we cannot do better than random partitioning. Several fast graph partitioning [65] and community detection [66] algorithms can be utilized for this purpose.

6.3.6 Faster PBGENA Implementation in C++

The present implementation for PBGENA is available on GitHub [11] and is written in Python 3 [49]. Python is a very versatile language with a rich source of publicly available libraries that simplifies writing code for scientific computations. An example of this fact is the heavy use of libraries like bitarray and mpi4py in PBGENA's construction. However, a downside of Python is its speed as compared to other languages like C++ [67]. Therefore, an alternate implementation of PBGENA in C++ is badly desirable.

6.3.7 PBGENA for dynamic graphs

As we have already discussed in section 5.5, graphs in the real world are not static but very dynamic with links and sometimes nodes being volatile. PBGENA is perfectly suited for graphs of this sort because an added edge would simply mean performing a couple of propagation steps. Further study can go into understanding the exact modifications required to transform PBGENA into an online algorithm.

APPENDIX A

HYPERPARAMETERS

PBGENA has a total of 9 hyperparameters. However, for a constant number of processors (p), a constant number of dimensions (N), and by setting the level parameters to 1 ($l_t = l_a = 1$, $f_t = None$, $f_a = None$), we end up with three essential hyperparameters- the attribute fraction (α), the topolgy bitset probability (b_t), and the attribute bitset probability (b_a). These are the three hyperparameters we are most concerned about and need to optimize for a given setup. Grid searching over the entire space of the hyperparameter values is simply not possible, so we employ a greedy strategy to reach a good but sub-optimal result. We first set $b_t = b_a = 0$ and find the optimal value for the attribute fraction. Then we fine-tune b_a with the fixed value for α and $b_t = 0$. Finally, we fine-tune b_t. The results reported in section 5.4 and section 5.5 are based on the hyperparameters mentioned in tables A.1 and A.2.

Table A.1: PBGENA Node Classification Hyperparameters

Graph	α	b_a	b_t
Wikipedia	0.85	0.00	0.20
Cora	0.60	0.80	0.80
CiteSeer	0.80	0.90	0.40
Facebook	0.50	0.70	0.60
BlogCatalog	0.60	0.00	0.00
Flickr	0.90	0.00	0.85
PubMed	0.65	0.00	0.80
PPI	0.10	0.95	0.50
Twitter	0.60	0.00	0.00
Google+	0.15	0.50	0.00
Reddit	0.10	0.00	0.00
TWeibo	0.60	0.00	0.00
MAKG	0.85	0.86	0.60

To optimize the hyperparameters for large graphs we only looked at $\alpha = \{0.2, 0.4, 0.6, 0.8\}$ and $b_t = b_a = \{0.1, 0.2, 0.4, 0.6, 0.8, 1.0\}$ which means we need to perform 16 runs of the algorithm and sometimes we would understand the trend and only need to perform 4 to 6 runs. Since PBGENA is almost $100\times$ faster than most baselines, running PBGENA for a few extra times usually does not hurt. However, as discussed in subsection 6.3.1, we do understand that this may be a roadblock for PBGENA so there needs to be some

Table A.2: PBGENA Link Prediction Hyperparameters

Graph	α	b_a	b_t
Wikipedia	0.95	0.20	0.20
Cora	0.60	0.30	0.40
CiteSeer	0.90	0.20	0.40
Facebook	0.80	1.00	0.00
BlogCatalog	0.90	0.20	0.00
Flickr	0.90	0.00	0.00
PubMed	0.95	0.20	0.20
PPI	0.10	0.00	0.10
Twitter	0.95	0.00	0.20
Google+	0.90	0.05	0.10
Reddit	0.95	0.20	0.00
TWeibo	0.95	0.00	0.00

form of the theoretical basis for choosing the hyperparameters or at least limit the search space for the hyperparameters. This might lead us to use some structural features of the graph for generating a recommendation on what values may be most suited. Having said this, this topic is beyond the purview of this thesis and is a concern for a future endeavor.

APPENDIX B

ANALYSIS

Theorem B.0.1. *The speedup ($\mathcal{S}$) is increasing in terms of p*

$$\mathcal{S} = p * \frac{N|V| + N|E| + |A|}{N|V| + (p+1)N|E| + |A|}$$

Proof: Let $X = N|V| + N|E| + |A|$. Now we have,

$$\mathcal{S} = \frac{pX}{X + p|E|}$$
$$\frac{\partial S}{\partial p} = \frac{X^2}{(X + p|E|)^2} \quad \text{(quotient rule)}$$

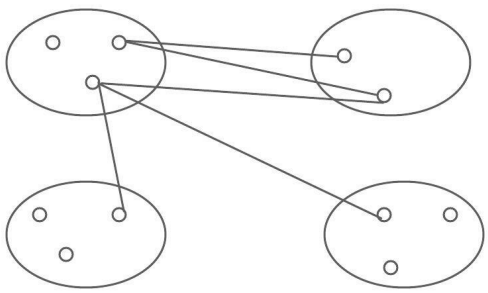
Now $\quad \dfrac{\partial S}{\partial p} > 0; \quad \forall p \in \mathbb{N}$

Therefore $\mathcal{S}$ is an increasing function with respect to p.

Theorem B.0.2. *The space complexity for PBGENA is $\mathcal{O}(N|V|)$ bits*

Lemma B.0.3. *The number of duplicated digests is at most twice the number of vertices having at least one cross-edge*

Figure B.1: Duplicated digests



Proof: This fact is almost self-evident because if an edge (u, v) is shared between two processors, then the digest for u ($D[u]$) must be shared with the processor hosting v and

vice-versa. This fact is demonstrated in Figure B.1 where the number of vertices with a cross edge is 6 and the number of duplicated digests is $8 \leqslant 2 * 6$.

Let m be the maximum degree of all nodes in the graph and let u be a node with degree $m' \leq m$. Now the probability that all neighbors of node u lie within its own partition is $(\frac{1}{p})^{m'}$ (where p is the number of processors/partitions, refer to subsection 4.4.1 for the justification). This means that the probability that at least one neighbor of u lie outside its partition is $1 - (1/p)^{m'}$.

Therefore, the expected number of nodes that has some neighbor in a different partition = $|V|(1 - (1/p)^{m'})$. Therefore the space complexity of PBGENA becomes, keeping in mind Lemma B.0.3:

$$\mathcal{O}\Big(N|V|\Big(1 - \big(1/p\big)^{m'}\Big) + N|V|\Big) \tag{B.1}$$

Now $(1 - (1/p)^{m'}) \leq 1$, therefore the space complexity of PBGENA reduces to $\mathcal{O}(N|V|)$.

REFERENCES

[1] W. L. Hamilton, R. Ying, and J. Leskovec, "Representation learning on graphs: Methods and applications," *IEEE Data Eng. Bull.*, vol. 40, no. 3, pp. 52–74, 2017. [Online]. Available: http://sites.computer.org/debull/A17sept/p52.pdf

[2] J. McAuley and J. Leskovec, "Learning to discover social circles in ego networks," in *Proceedings of the 25th International Conference on Neural Information Processing Systems - Volume 1*, ser. NIPS'12. Red Hook, NY, USA: Curran Associates Inc., 2012, p. 539–547.

[3] J. Leskovec, K. J. Lang, A. Dasgupta, and M. W. Mahoney, "Community Structure in Large Networks: Natural Cluster Sizes and the Absence of Large Well-Defined Clusters," *Internet Mathematics*, vol. 6, no. 1, pp. 29 – 123, 2009. [Online]. Available: https://doi.org/

[4] P. Sen, G. Namata, M. Bilgic, L. Getoor, B. Galligher, and T. Eliassi-Rad, "Collective classification in network data," *AI Magazine*, vol. 29, no. 3, p. 93, Sep. 2008. [Online]. Available: https://ojs.aaai.org/index.php/aimagazine/article/view/2157

[5] W. Hamilton, Z. Ying, and J. Leskovec, "Inductive representation learning on large graphs," in *Advances in Neural Information Processing Systems*, I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, Eds., vol. 30. Curran Associates, Inc., 2017. [Online]. Available: https://proceedings.neurips.cc/paper/2017/file/5dd9db5e033da9c6fb5ba83c7a7ebea9-Paper.pdf

[6] D. Zhang, J. Yin, X. Zhu, and C. Zhang, "Network representation learning: A survey," *IEEE Transactions on Big Data*, vol. 6, no. 1, pp. 3–28, 2020.

[7] X. Liu and J. Tang, "Network representation learning: A macro and micro view," *AI Open*, vol. 2, pp. 43–64, 2021. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S2666651021000024

[8] D. Liben-Nowell and J. Kleinberg, "The link prediction problem for social networks," in *Proceedings of the Twelfth International Conference on Information and Knowledge Management*, ser. CIKM '03. New York, NY, USA: Association for Computing Machinery, 2003, p. 556–559. [Online]. Available: https://doi.org/10.1145/956863.956972

[9] S. Cavallari, V. W. Zheng, H. Cai, K. C.-C. Chang, and E. Cambria, "Learning community embedding with community detection and node embedding on graphs," in *Proceedings of the 2017 ACM on Conference on Information and Knowledge Management*, ser. CIKM '17. New York, NY, USA: Association for Computing Machinery, 2017, p. 377–386. [Online]. Available: https://doi.org/10.1145/3132847.3132925

[10] Z. Meng, S. Liang, H. Bao, and X. Zhang, "Co-embedding attributed networks," in *Proceedings of the Twelfth ACM International Conference on Web Search and Data Mining*, ser. WSDM '19. New York, NY, USA: Association for Computing Machinery, 2019, p. 393–401. [Online]. Available: https://doi.org/10.1145/3289600.3291015

[11] B. Li and D. Pi, "Network representation learning: a systematic literature review," *Neural Computing and Applications*, vol. 32, no. 21, p. 16647–16679, 2020.

[12] J. Zhou, G. Cui, S. Hu, Z. Zhang, C. Yang, Z. Liu, L. Wang, C. Li, and M. Sun, "Graph neural networks: A review of methods and applications," *AI Open*, vol. 1, pp. 57–81, 2020. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S2666651021000012

[13] B. Perozzi, R. Al-Rfou, and S. Skiena, "Deepwalk: Online learning of social representations," in *Proceedings of the 20th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, ser. KDD '14. New York, NY, USA: Association for Computing Machinery, 2014, p. 701–710. [Online]. Available: https://doi.org/10.1145/2623330.2623732

[14] R. Zhu, K. Zhao, H. Yang, W. Lin, C. Zhou, B. Ai, Y. Li, and J. Zhou, "Aligraph: A comprehensive graph neural network platform," *Proc. VLDB Endow.*, vol. 12, no. 12, p. 2094–2105, aug 2019. [Online]. Available: https://doi.org/10.14778/3352063.3352127

[15] S. Yan, D. Xu, B. Zhang, H.-j. Zhang, Q. Yang, and S. Lin, "Graph embedding and extensions: A general framework for dimensionality reduction," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 29, no. 1, pp. 40–51, 2007.

[16] Z. Zhang, P. Cui, H. Li, X. Wang, and W. Zhu, "Billion-scale network embedding with iterative random projection," in *2018 IEEE International Conference on Data Mining (ICDM)*, 2018, pp. 787–796.

[17] W. Wu, B. Li, L. Chen, and C. Zhang, "Efficient attributed network embedding via recursive randomized hashing," in *Proceedings of the Twenty-Seventh International Joint Conference on Artificial Intelligence, IJCAI-18*. International Joint Conferences on Artificial Intelligence Organization, 7 2018, pp. 2861–2867. [Online]. Available: https://doi.org/10.24963/ijcai.2018/397

[18] R. Yang, J. Shi, X. Xiao, Y. Yang, J. Liu, and S. S. Bhowmick, "Scaling attributed network embedding to massive graphs," *Proc. VLDB Endow.*, vol. 14, no. 1, p. 37–49, sep 2020. [Online]. Available: https://doi.org/10.14778/3421424.3421430

[19] A. Z. Broder, M. Charikar, A. M. Frieze, and M. Mitzenmacher, "Min-wise independent permutations," *Journal of Computer and System Sciences*, vol. 60, no. 3, pp. 630–659, 2000. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0022000099916902

[20] I. T. Jolliffe, *Principal component analysis*. Springer, 2011.

[21] A. J. Izenman, *Modern multivariate statistical techniques: regression, classification, and manifold learning*. Springer, 2013.

[22] J. B. Tenenbaum, V. de Silva, and J. C. Langford, "A global geometric framework for nonlinear dimensionality reduction," *Science*, vol. 290, no. 5500, pp. 2319–2323, 2000. [Online]. Available: https://www.science.org/doi/abs/10.1126/science.290.5500.2319

[23] M. Belkin and P. Niyogi, "Laplacian eigenmaps and spectral techniques for embedding and clustering," in *Proceedings of the 14th International Conference on Neural Information Processing Systems: Natural and Synthetic*, ser. NIPS'01. Cambridge, MA, USA: MIT Press, 2001, p. 585–591.

[24] C. Yang, Z. Liu, D. Zhao, M. Sun, and E. Y. Chang, "Network representation learning with rich text information," in *Proceedings of the 24th International Conference on Artificial Intelligence*, ser. IJCAI'15. AAAI Press, 2015, p. 2111–2117.

[25] S. Yang and B. Yang, "Enhanced network embedding with text information," in *2018 24th International Conference on Pattern Recognition (ICPR)*, 2018, pp. 326–331.

[26] H. Yang, S. Pan, P. Zhang, L. Chen, D. Lian, and C. Zhang, "Binarized attributed network embedding," in *2018 IEEE International Conference on Data Mining (ICDM)*, 2018, pp. 1476–1481.

[27] N. Shervashidze, P. Schweitzer, E. J. van Leeuwen, K. Mehlhorn, and K. M. Borgwardt, "Weisfeiler-lehman graph kernels," *Journal of Machine Learning Research*, vol. 12, no. 77, pp. 2539–2561, 2011. [Online]. Available: http://jmlr.org/papers/v12/shervashidze11a.html

[28] H. Yang, S. Pan, L. Chen, C. Zhou, and P. Zhang, "Low-bit quantization for attributed network representation learning," in *Proceedings of the Twenty-Eighth International Joint Conference on Artificial Intelligence, IJCAI-19*. International Joint Conferences on Artificial Intelligence Organization, 7 2019, pp. 4047–4053. [Online]. Available: https://doi.org/10.24963/ijcai.2019/562

[29] M. Newman, "Power laws, pareto distributions and zipf's law," *Contemporary Physics*, vol. 46, no. 5, pp. 323–351, 2005. [Online]. Available: https://doi.org/10.1080/00107510500052444

[30] T. Mikolov, K. Chen, G. Corrado, and J. Dean, "Efficient estimation of word representations in vector space," in *1st International Conference on Learning Representations, ICLR 2013, Scottsdale, Arizona, USA, May 2-4, 2013, Workshop Track Proceedings*, Y. Bengio and Y. LeCun, Eds., 2013. [Online]. Available: http://arxiv.org/abs/1301.3781

[31] J. Tang, M. Qu, M. Wang, M. Zhang, J. Yan, and Q. Mei, "Line: Large-scale information network embedding," in *Proceedings of the 24th International Conference on World Wide Web*, ser. WWW '15. Republic and Canton of Geneva, CHE: International World Wide Web Conferences Steering Committee, 2015, p.

1067–1077. [Online]. Available: https://doi.org/10.1145/2736277.2741093

[32] A. Grover and J. Leskovec, "Node2vec: Scalable feature learning for networks," in *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, ser. KDD '16. New York, NY, USA: Association for Computing Machinery, 2016, p. 855–864. [Online]. Available: https://doi.org/10.1145/2939672.2939754

[33] D. Zhang, J. Yin, X. Zhu, and C. Zhang, "Sine: Scalable incomplete network embedding," in *2018 IEEE International Conference on Data Mining (ICDM)*, 2018, pp. 737–746.

[34] H. Wei, Z. Pan, G. Hu, G. Hu, H. Yang, X. Li, and X. Zhou, "Attributed network representation learning via deepwalk," *Intelligent Data Analysis*, vol. 23, no. 4, p. 877–893, 2019.

[35] Z. Zhang, H. Yang, J. Bu, S. Zhou, P. Yu, J. Zhang, M. Ester, and C. Wang, "Anrl: Attributed network representation learning via deep neural networks," in *Proceedings of the 27th International Joint Conference on Artificial Intelligence*, ser. IJCAI'18. AAAI Press, 2018, p. 3155–3161.

[36] N. Sheikh, Z. T. Kefato, and A. Montresor, "A simple approach to attributed graph embedding via enhanced autoencoder," in *Complex Networks and Their Applications VIII*, H. Cherifi, S. Gaito, J. F. Mendes, E. Moro, and L. M. Rocha, Eds. Cham: Springer International Publishing, 2020, pp. 797–809.

[37] S. Bandyopadhyay, L. N, S. V. Vivek, and M. N. Murty, "Outlier resistant unsupervised deep architectures for attributed network embedding," in *Proceedings of the 13th International Conference on Web Search and Data Mining*, ser. WSDM '20. New York, NY, USA: Association for Computing Machinery, 2020, p. 25–33. [Online]. Available: https://doi.org/10.1145/3336191.3371788

[38] D. Yang, P. Rosso, B. Li, and P. Cudre-Mauroux, "Nodesketch: Highly-efficient graph embeddings via recursive sketching," in *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery amp; Data Mining*, ser. KDD '19. New York, NY, USA: Association for Computing Machinery, 2019, p. 1162–1172. [Online]. Available: https:

//doi.org/10.1145/3292500.3330951

[39] W. Wu, B. Li, L. Chen, and C. Zhang, "Consistent weighted sampling made more practical," in *Proceedings of the 26th International Conference on World Wide Web*, ser. WWW '17. Republic and Canton of Geneva, CHE: International World Wide Web Conferences Steering Committee, 2017, p. 1035–1043. [Online]. Available: https://doi.org/10.1145/3038912.3052598

[40] D. Bera, R. Pratap, B. D. Verma, B. Sen, and T. Chakraborty, "Quint: Node embedding using network hashing," *IEEE Transactions on Knowledge and Data Engineering*, pp. 1–1, 2021.

[41] R. Pratap, D. Bera, and K. Revanuru, "Efficient sketching algorithm for sparse binary data," in *2019 IEEE International Conference on Data Mining (ICDM)*, 2019, pp. 508–517.

[42] A. Shrivastava, "Optimal densification for fast and accurate minwise hashing," in *Proceedings of the 34th International Conference on Machine Learning - Volume 70*, ser. ICML'17. JMLR.org, 2017, p. 3154–3163.

[43] F. Yu, S. Kumar, Y. Gong, and S.-F. Chang, "Circulant binary embedding," in *Proceedings of the 31st International Conference on Machine Learning*, ser. Proceedings of Machine Learning Research, E. P. Xing and T. Jebara, Eds., vol. 32, no. 2. Bejing, China: PMLR, 22–24 Jun 2014, pp. 946–954. [Online]. Available: https://proceedings.mlr.press/v32/yub14.html

[44] S. L. Harris and D. M. Harris, "2 - combinational logic design," in *Digital Design and Computer Architecture*, S. L. Harris and D. M. Harris, Eds. Boston: Morgan Kaufmann, 2016, pp. 54–106. [Online]. Available: https://www.sciencedirect.com/science/article/pii/B9780128000564000029

[45] B. D. Verma, R. Pratap, and D. Bera, "Efficient binary embedding of categorical data using binsketch," 2021, accepted in ECML/PKDD 2021 (DMKD Journal Track). [Online]. Available: https://arxiv.org/abs/2111.07163

[46] K. H. Knuth, "Optimal data-based binning for histograms and histogram-based probability density models," *Digital Signal Processing*, vol. 95, p. 102581, 2019. [Online]. Available: https://www.sciencedirect.com/science/article/pii/

S1051200419301277

[47] Message Passing Interface Forum, *MPI: A Message-Passing Interface Standard Version 4.0*, Jun. 2021. [Online]. Available: https://www.mpi-forum.org/docs/mpi-4.0/mpi40-report.pdf

[48] L. Dalcin and Y.-L. L. Fang, "mpi4py: Status update after 12 years of development," *Computing in Science Engineering*, vol. 23, no. 4, pp. 47–54, 2021.

[49] G. van Rossum and F. L. Drake, *The Python Language Reference Manual*. Network Theory Ltd., 2011.

[50] G. Karypis, *METIS and ParMETIS*. Boston, MA: Springer US, 2011, pp. 1117–1124. [Online]. Available: https://doi.org/10.1007/978-0-387-09766-4_500

[51] Q. Lu and L. Getoor, "Link-based classification," in *Proceedings of the Twentieth International Conference on International Conference on Machine Learning*, ser. ICML'03. AAAI Press, 2003, p. 496–503.

[52] V. Arnaboldi, M. Conti, A. Passarella, and F. Pezzoni, "Analysis of ego network structure in online social networks," in *2012 International Conference on Privacy, Security, Risk and Trust and 2012 International Confernece on Social Computing*, 2012, pp. 31–40.

[53] L. Tang and H. Liu, "Relational learning via latent social dimensions," in *Proceedings of the 15th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, ser. KDD '09. New York, NY, USA: Association for Computing Machinery, 2009, p. 817–826. [Online]. Available: https://doi.org/10.1145/1557019.1557109

[54] X. Huang, J. Li, and X. Hu, "Label informed attributed network embedding," in *Proceedings of the Tenth ACM International Conference on Web Search and Data Mining*, ser. WSDM '17. New York, NY, USA: Association for Computing Machinery, 2017, p. 731–739. [Online]. Available: https://doi.org/10.1145/3018661.3018667

[55] G. Namata, B. London, L. Getoor, and B. Huang, "Query-driven active surveying for collective classification," in *ICML Workshop on MLG*, 2012.

[56] B. Rozemberczki, O. Kiss, and R. 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 amp; Knowledge Management*, ser. CIKM '20. New York, NY, USA: Association for Computing Machinery, 2020, p. 3125–3132. [Online]. Available: https://doi.org/10.1145/3340531.3412757

[57] L. Liao, X. He, H. Zhang, and T.-S. Chua, "Attributed social network embedding," *IEEE Trans. on Knowl. and Data Eng.*, vol. 30, no. 12, p. 2257–2270, dec 2018. [Online]. Available: https://doi.org/10.1109/TKDE.2018.2819980

[58] S. J. Wright, "Coordinate descent algorithms," *Mathematical Programming*, vol. 151, no. 1, p. 3–34, 2015.

[59] B. Rozemberczki and R. Sarkar, "Characteristic functions on graphs: Birds of a feather, from statistical descriptors to parametric models," in *Proceedings of the 29th ACM International Conference on Information amp; Knowledge Management*, ser. CIKM '20. New York, NY, USA: Association for Computing Machinery, 2020, p. 1325–1334. [Online]. Available: https://doi.org/10.1145/3340531.3411866

[60] B. Rozemberczki, C. Allen, and R. Sarkar, "Multi-Scale attributed node embedding," *Journal of Complex Networks*, vol. 9, no. 2, 05 2021, cnab014. [Online]. Available: https://doi.org/10.1093/comnet/cnab014

[61] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay, "Scikit-learn: Machine learning in Python," *Journal of Machine Learning Research*, vol. 12, pp. 2825–2830, 2011.

[62] L. van der Maaten and G. Hinton, "Visualizing data using t-sne," *Journal of Machine Learning Research*, vol. 9, no. 86, pp. 2579–2605, 2008. [Online]. Available: http://jmlr.org/papers/v9/vandermaaten08a.html

[63] K. Weinberger, A. Dasgupta, J. Langford, A. Smola, and J. Attenberg, "Feature hashing for large scale multitask learning," in *Proceedings of the 26th Annual International Conference on Machine Learning*, ser. ICML '09. New York, NY, USA: Association for Computing Machinery, 2009, p. 1113–1120. [Online].

Available: https://doi.org/10.1145/1553374.1553516

[64] C. Yang, L. Zhong, L.-J. Li, and L. Jie, "Bi-directional joint inference for user links and attributes on large social graphs," in *Proceedings of the 26th International Conference on World Wide Web Companion*, ser. WWW '17 Companion. Republic and Canton of Geneva, CHE: International World Wide Web Conferences Steering Committee, 2017, p. 564–573. [Online]. Available: https://doi.org/10.1145/3041021.3054181

[65] G. Even, J. S. Naor, S. Rao, and B. Schieber, "Fast approximate graph partitioning algorithms," in *Proceedings of the Eighth Annual ACM-SIAM Symposium on Discrete Algorithms*, ser. SODA '97. USA: Society for Industrial and Applied Mathematics, 1997, p. 639–648.

[66] C. Peng, Z. Zhang, K.-C. Wong, X. Zhang, and D. E. Keyes, "A scalable community detection algorithm for large graphs using stochastic block models," in *Proceedings of the 24th International Conference on Artificial Intelligence*, ser. IJCAI'15. AAAI Press, 2015, p. 2090–2096.

[67] J. Akeret, L. Gamper, A. Amara, and A. Refregier, "Hope: A python just-in-time compiler for astrophysical computations," *Astronomy and Computing*, vol. 10, pp. 1–8, 2015. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S2213133714000687