

**DO WE NEED ALL THE NEURONS WE UTILIZE?
AN EXPLORATION INTO COSTS AND SCALABILITY FOR LARGE-SCALE
MULTI-RELATIONAL GRAPH LEARNING**

A Dissertation
Presented to
The Academic Faculty

By

Lakshmi Sathidevi

In Partial Fulfillment
of the Requirements for the Degree
Master of Science in the
School of Electrical and Computer Engineering

Georgia Institute of Technology

May 2024

© Lakshmi Sathidevi 2024

**DO WE NEED ALL THE NEURONS WE UTILIZE?
AN EXPLORATION INTO COSTS AND SCALABILITY FOR LARGE-SCALE
MULTI-RELATIONAL GRAPH LEARNING**

Thesis committee:

Dr. Callie (Cong) Hao
Electrical and Computer Engineering
Georgia Institute of Technology

Dr. Tushar Krishna
Electrical and Computer Engineering
Georgia Institute of Technology

Dr. Hyesoon Kim
Computer Science
Georgia Institute of Technology

Date approved: April 29, 2024

What is that, knowing which, everything becomes known?

Mundaka Upanishad 1.1.3

To my mother and guru, Ambutiamma

ACKNOWLEDGMENTS

I have no words to express my utmost gratitude to my advisor, Dr. Callie Hao for her constant guidance and encouragement. I would cherish every single thing I learnt from her. Her warmth as a human being has been limitless.

I am thankful to Dr. Tushar Krishna and Dr. Hyesoon Kim for agreeing to be part of my thesis committee.

I would not have embarked on pursuing this scientific enquiry without the constant support and encouragement of my dearest Ambutiamma and my illustrious parents: Sathidevi and Satish. Their teachings and the values they imparted to me brought me to the place I am today. I can never recompense them for their unceasing support and abundant love.

TABLE OF CONTENTS

| | |
|---|-----|
| Acknowledgments | v |
| List of Tables | ix |
| List of Figures | x |
| List of Acronyms | xi |
| Summary | xii |
| Chapter 1: Introduction | 1 |
| 1.1 Background | 1 |
| 1.1.1 Biomedical Machine Learning Research | 1 |
| 1.1.2 Graph Representation Learning and Graph Neural Networks | 3 |
| 1.1.3 Heterogeneous Graph Neural Networks | 4 |
| 1.1.4 Knowledge Graphs | 5 |
| 1.1.5 HW-SW Co-design | 6 |
| 1.2 Problem Statement | 6 |
| 1.2.1 Need for Speed | 7 |
| 1.2.2 The Gap | 8 |
| 1.3 Delimitation | 8 |

| | | |
|---|--|-----------|
| 1.3.1 | Target | 8 |
| 1.3.2 | Demarcation | 9 |
| Chapter 2: Methodology | | 10 |
| 2.1 | Approach and Research Design | 10 |
| 2.2 | Components | 10 |
| 2.2.1 | Datasets Used | 10 |
| 2.2.2 | MinHashing | 11 |
| 2.2.3 | General formulation for a simple GNN | 12 |
| 2.2.4 | Replacing Neural Netowrks with MinHash | 12 |
| 2.2.5 | Enabling Multi-Relational Learning | 12 |
| 2.3 | Experiments and Evaluation | 14 |
| 2.3.1 | Scaling of Runtime | 14 |
| 2.3.2 | Scaling of Memory Usage | 19 |
| Chapter 3: Results | | 23 |
| 3.1 | Performance | 23 |
| 3.2 | Comparison | 24 |
| Chapter 4: Discussion and Conclusion | | 26 |
| 4.1 | Benefits | 26 |
| 4.1.1 | Algorithm | 26 |
| 4.1.2 | Implementation | 28 |
| 4.2 | Limitations | 29 |

| | | |
|-------------------|--|-----------|
| 4.3 | Contributions and Practical Implications | 29 |
| 4.4 | Future Scope | 30 |
| References | | 31 |

LIST OF TABLES

| | | |
|-----|--|----|
| 2.1 | Datasets utilized in this thesis, and their statistics | 11 |
| 3.1 | Heterogeneous link prediction performance of Het#GNN on the decagon dataset | 23 |
| 3.2 | Published works that cite their best performance on Decagon heterogeneous graph dataset | 24 |
| 3.3 | Het#GNN's Runtime (secs) and Peak Memory Usage (MB) against that of published works that cite their performance on Decagon dataset | 24 |

LIST OF FIGURES

| | | |
|-----|--|----|
| 2.1 | Scaling of runtime with hashed feature vector size for 1-hop receptive field . | 15 |
| 2.2 | Scaling of runtime with receptive field size hashdim=200 | 16 |
| 2.3 | Scaling of runtime with receptive field size at different hashdim values . . . | 17 |
| 2.4 | Scaling of runtime with hashed feature vector size and receptive field | 17 |
| 2.5 | Scaling of runtime with number of nodes | 18 |
| 2.6 | Scaling of runtime with hashed feature vector size for 1-hop receptive field . | 19 |
| 2.7 | Scaling of peak memory usage with hashed feature vector size and receptive field | 20 |
| 2.8 | Scaling of peak memory usage with number of nodes | 21 |
| 2.9 | Scaling of runtime with receptive field size in heterogeneous setting | 21 |

LIST OF ACRONYMS

| | |
|--------------|--|
| 3D | 3-Dimensional |
| ADE | Adverse Drug Event |
| GAT | Graph Attention Network |
| GCN | Graph Convolution Network |
| GIN | Graph Isomorphism Network |
| GNN | Graph Neural Network |
| GPU | Graphics Processing Unit |
| GRL | Graph Representation Learning |
| GSAGE | GraphSAGE |
| ICLR | International Conference on Learning Representations |
| LLM | Large Language Model |
| ML | Machine Learning |
| R-GCN | Relational Graph Convolutional Network |
| TPU | Tensor Processing Unit |
| US | United States |

SUMMARY

The first step in HW-SW Co-design is algorithm selection. The algorithm chosen places hard boundaries on the kind of efficiency and scalability that can be achieved through the design of a custom hardware accelerator. A smartly selected algorithm could enable several times more performance gains than a cutting-edge hardware accelerator designed for a poorly selected algorithm [1]. This study presents Het#GNN, an unsupervised network embedding algorithm that extends on top of the #GNN algorithm [2] to handle multi-relational graph learning. The goal is to provide a scalable, efficient, and low-latency solution for large-scale multi-relational graph learning.

Graph Neural Network (GNN) models have shown promising results in various applications. However, these models are often hindered by high memory and computational requirements, limiting their applicability to large-scale real-world scenarios. The biomedical community has shown growing interest in more efficient and less heavily parameterized methods for handling large-scale multi-relational graph learning.

Het#GNN is a simple yet powerful unsupervised algorithm that addresses the efficiency and scalability challenges faced by GNN algorithms. It enables a parameter-free approach to network embedding for large-scale multi-relational graphs. Het#GNN is able to compete with the best-performing models at less than a fraction of the runtime and power costs and Het#GNN achieves this on a consumer CPU, while all other methods utilize server GPUs to accelerate their algorithm.

The need for a scalable, efficient, and low-latency graph learning algorithm is increasingly important in the biomedical research community. Decagon [3] is a very popular work that applied the multi-relational GNN model R-GCN [4] to polypharmacy drug side-effect prediction, obtaining strong results and medically relevant predictions. However, just one epoch of training for this model takes 36 hours on a Tesla P40 GPU [5]. This kind of turnaround time is detrimental to the pace of biomedical research and discovery. Het#GNN

is presented as a solution. It is applied to the Decagon dataset to demonstrate how sometimes we may be utilizing too many time and energy expensive neurons where none may be required.

By focusing on simple yet smart choices at the algorithmic level, Het#GNN offers considerable benefits in terms of efficiency, scalability and latency compared to other best-performing methods without requiring high-end computational resources. With this work, it is hoped that this simple and accessible contribution enables biomedical researchers to accelerate the pace of their own biomedical research and discovery pipelines that involve large-scale multi-relational graph learning. The results of this work, however, are expected to be beneficial to other domains and areas of application as well, like real-time graph learning.

CHAPTER 1

INTRODUCTION

1.1 Background

1.1.1 Biomedical Machine Learning Research

Machine Learning (ML) has become an integral part of biomedical research by enabling the acceleration of the entire research and discovery process like never before. Many credible sources recognize this role of ML in biomedical research and have been focusing on ways to utilize it [3, 6, 7]. Previously, before the advent of ML methods, drug development for example, required an unimaginably large amount of effort and time in testing before being released into the market. For a measure of how long it takes, the rodent bioassay gold standard for identification of carcinogenicity of new compounds takes 2 years [8], and in drug discovery, the complete pipeline of identifying, developing, testing, and bringing the drug to market can take from 3 to 5 whole years! [7]

While the integration of machine learning methods marked a significant leap in accelerating biomedical applications, the sheer size and diversity of the data we need to learn from necessitate the acceleration of the ML methods themselves. There is a growing demand for rapid and scalable application of ML in the biomedical domain, a need further underscored by the emergence and increasing popularity of new, more efficient AI software platforms, like TorchDrug [9], ChemicalX [10] and DeepChem [11]. Biomedical graph datasets are also growing in number due to the effectiveness of learning in this format, like the MolHIV dataset from Stanford [12], the ENZYMES dataset [13], PPI [14], BioKG [15], and many more. Also, the success of services in the industry that applies ML for biomedical applications, like IBM Micromedex [16] and Amazon Neptune [17], indicate the need and demand for faster, more efficient methods in biomedical research, and also the effectiveness of ML

methods in serving that purpose.

Machine Learning methods have now become widely adopted in the biomedical domain and it has already been applied to an innumerable variety of tasks like MRI reconstruction [18], clinical image segmentation [19], molecular property prediction [12], drug discovery [7], and even to predict how proteins fold [6]. Among all these diverse and remarkable applications of ML to biomedical research, an application of growing importance is the prediction of drug polypharmacy side effects. This task has become more and more relevant over the years because of the increasing adoption of polypharmacy (the use of multiple drugs in combination at once). Polypharmacy has been shown to be very effective, for example, the drug combination of venetoclax and Idasanutlin has a very high antileukemic efficacy in treating acute myeloid leukemia [20]. At the same time, along with such powerful benefits of polypharmacy, it also introduces an increased chance of unexpected side effects. As larger and larger numbers of prescriptions start to include multiple drugs, the number of Adverse Drug Event (ADE) are also increasing [3]. More than 1 Million deaths are reported per year due to poly pharmacy side-effects in the US alone [21], and for further treatment up to 0.35 Million patients had to be hospitalized [22]. This severely affects drug development and time-to-market of new and important drugs. More importantly, the tight battle with time to test the drug and to release it in the market within the time frame when it is needed could considerably increase the risks associated with the drug and its combined use with other drugs. The demand for ways to solve the problem of polypharmacy side-effects is at its highest.

Being able to predict potential side-effects of combined use of drugs accelerates the entire process from research to development to market release of drugs, and enables a huge reduction in the effort and time required for extensive drug testing. This also enables companies to bring drugs to market in a faster and safer manner. A recent event where this was an immediate need, was the global COVID pandemic which started in 2019. Recent works have been fast to identify the potential of applying ML on graphs to this task [23].

Though ML plays the role of an accelerator in biomedical research, there is still huge benefits to reap by accelerating ML for biomedical research tasks. This can be seen in several recent applications, like AlphaFold [6] and Decagon [3]. AlphaFold from Google DeepMind, predicts the 3D structure of proteins and was built to accelerate research and discovery in all fields of Biology. It was a giant leap from the effort and years of time required to figure out 3D protein structures but still the time involved in training AlphaFold is very prohibitive for experimentation. The first version AlphaFold took a few weeks to train in a distributed manner across ~ 128 TPUs (equivalent to 100-200 GPUs) and the inference on a single protein took about a day [24]. This is so intense that there have been works trying to overcome this computational and runtime cost like FastFold [25]. Similarly, Decagon is another work in biomedical research that communicates the prohibitive nature of this issue [3]. It aims to predict the possible side effects that could occur upon taking a combination of drugs (polypharmacy side-effects). For this it uses a moderately large network of $\sim 19.5\text{K}$ nodes, ~ 1930 different edge types and $\sim 5\text{Million}+$ edges. Training their Relational Graph Convolutional Network (R-GCN) model on this network takes 36 hours per epoch on a Tesla P40 GPU [5]. This huge runtime involved in training can make it extremely difficult for iterative exploration, not to mention that it can be prohibitive for researchers who do not have the kind of hardware resources capable of handling such computationally intensive training. Here in this work, it is attempted to develop the initial iteration of a solution for the high cost of training and inference in multi-relational graph learning. It is hoped that this work brings forth a new opportunity to overcome a few of the restraints that block the pace of advancements in biomedical research.

1.1.2 Graph Representation Learning and Graph Neural Networks

A critical element of any learning system is the data that it learns from. Real-world data being of a highly interconnected and complex nature, is most effectively represented as a graph. Graph Representation Learning (GRL) is the method of forming an embedding

from the graph that represents the graph structure without the user having to engineer them manually. It can be thought of as the projection of non-Euclidean information from the graph onto a euclidean space. Once this is done, existing ML methods can easily take over. A foundational paper in GRL from ICLR 2017, introduced the Graph Convolution Network (GCN) which changed the way graphs were learned from [26].

GCN introduced the spatial GRL approach of aggregating a node's feature from its neighbours' features. Today there is a huge variety of GNNs and due to the increasing interest in graph learning, more and more varieties are being researched and applied to a variety of fields every year. GCN, Graph Attention Network (GAT), GraphSAGE (GSAGE) and Graph Isomorphism Network (GIN) are the more popular kinds of GNNs.

Due to the highly interconnected and diverse nature of biomedical data, GNNs have proven to be very effective for learning from highly interconnected biomedical data [3], [27], [28], [7], [29]. Most tasks in biomedical research, for example, drug discovery and polypharmacy side-effect prediction, involve large, diverse and highly interconnected data which can be leveraged and learned from by using GNNs. Works like Decagon [3] show the success of applying GNNs to biomedical data.

That said, real-world graph data are generally huge, and the parameterized message passing scheme of regular GNNs have proven to be computationally intensive despite the efficient nature of the underlying message passing methodology itself. This is a theme that is well understood by those interested in real-world production applications of graph learning at scale — like Google, Neo4j and Turbo ML. This theme is therefore of critical importance to efficient and scalable real-world application of graph learning, and will hence form a core focus of this thesis.

1.1.3 Heterogeneous Graph Neural Networks

Heterogeneous graphs or heterographs, are a class of graphs that include nodes or edges of different types. A simple demonstrative example is the ACM citation graph dataset

[4]. This dataset is an ‘Academic Social Network’ where there are ‘Venue’ nodes, ‘Paper’ nodes, and ‘Author’ nodes. It has edges that connect the ‘Author’ nodes to the ‘Paper’ which are ‘written-by’ edges, edges that connect ‘Paper’ nodes with ‘Venue’ which are ‘published-in’ edges, and also edges that connect a ‘Paper’ node to another, which are ‘cited-by’ edges. Often for large real-world graphs (millions of nodes) — which are commonplace in biomedical and healthcare applications — heterogeneous representation becomes almost necessary for effective representation. These characteristics of heterogeneity become new challenges and at the same time, unexplored opportunities for practical real-world application.

1.1.4 Knowledge Graphs

Knowledge Graphs are directed, heterogeneous, multigraphs that further generalize the heterogeneous graph view of real-world data. Several works suggest the use of Knowledge Graphs as a default data model for learning on interconnected real-world data. Knowledge Graphs can be considered as a more generalized form of large-scale heterogeneous graphs that can be used to flexibly learn from information and to intelligently access complex knowledge. There are already many large companies that use Knowledge Graphs in their production applications like Pinterest (Recommendation System), IKEA (Recommendation System), Alibaba (Recommendation System) [30], and Amazon Web Services (Fraud Detection) [31]. Knowledge graphs are very appealing to the industry because, unlike Large Language Model (LLM)s, Knowledge Graphs can be applied to real-time applications and scaled in a big way. Knowledge graphs also enable systematic reasoning that is not adulterated by hallucinations like we experience with LLMs [32]. This can be an extremely powerful capability when combined with other capabilities like knowledge graph completion, anomaly detection, neighbor node ranking, and others.

However, with the kind of scales at which knowledge graphs need to be applied, even the models that learn on such an efficient representation themselves face scalability and

efficiency challenges which call for more scalable approaches [30, 33]

1.1.5 HW-SW Co-design

Many of the existing works that attempt to accelerate graph learning often take directly to HW design to fit the given algorithm [34, 35, 36]. This could greatly limit the possible gains that could be extracted, had a HW-SW Co-design approach been taken [1]. On top of this, due to the challenging nature of hardware architecture design and test for complex ML models, the range of tasks that are in the focus of these works often tend to be quite limited. For example, most of them focus on just inference acceleration, and on a single GNN model alone.

In the long run, it is HW-SW Co-design that would enable us to squeeze the last bit of performance and efficiency for target applications, while also enable us to ensure applicability downstream. Especially for GNNs we have seen a significant increase in works that design hardware architectures for acceleration. The question here is whether we see these accelerator designs getting adopted and being applied in a real-world production setting. Hence, as part of HW-SW co-design this work focuses on the algorithm, as selection/discovery of the right algorithm could by itself yield several times the speed up obtained through the design and application of custom hardware architectures [1].

1.2 Problem Statement

Graphs are a form of representing data that is very natural and efficient because of their close association with the nature of the data itself. However, graphs have been one of the most challenging data formats to learn on because of the non-euclidean nature and because the computational complexity involved is often very intensive if the algorithms applied for learning are not intelligently designed. With the introduction of the GCN model [26], the popularity of GNNs truly blew up. However, one common theme that comes across repeatedly is the challenge of scaling and accelerating graph neural networks.

1.2.1 Need for Speed

Despite the expressivity and popularity, the need to improve efficiency, runtime and scalability of GNNs has been an important topic of research. We can see this from the fact that there have been several attempts at enabling scalable and fast computation of GCNs like FastGCN [37], SGCN [38], ClusterGCN [39], GraphSAGE [40], and GraphSAINT [41]. Moreover, if we look at application specific production environments where fast turnaround times are important, we can see that they hesitate to apply heavily parameterized neural network based models due to the fact that these are very computationally intensive. For example, companies like Google, Neo4j and TurboML realize that its not practical to deploy heavily parameterized, complex GNN algorithms in a scalable and useful manner. Highly expressive GNN based ML methods often tend to fall in this category. Real-world production environments resort to unsupervised techniques, binary representation vectors, and simplified similarity search mechanisms in order to meet the scalability, efficiency, and latency requirements while trying to limit the hit on performance. The matter gets worse when we move to the real-world applications where multi-relational graph learning is a necessity. For example, Google search infobox utilizes their in-house knowledge graph to fetch relevant information related to the search within a second. We could imagine the size of their knowledge graph to be greater than a trillion nodes, not to mention the number of edges. It would not be practical to apply GNNs at this kind of scale, but applications of most value are often large-scale.

In the biomedical domain, Decagon [3] is a very popular work that applied the multi-relational graph learning model R-GCN [4] to polypharmacy drug side-effect prediction, obtaining strong results and medically relevant predictions. However, just one epoch of training for this model takes 36 hours on a Tesla P40 GPU [5]. This severely disables effective research and discovery because there is no scope of iteration and exploration when one training run itself can take weeks. This kind of turnaround time is detrimental for the pace of biomedical research and discovery.

1.2.2 The Gap

The majority of the real-world graph learning challenges are of a heterogeneous nature [31, 3]. Moreover, scaling and accelerating a heterogeneous GNN model like R-GCN poses a much greater challenge than regular GNNs. One major challenge is the storage of the learnable weights for each edge type, another is the computational intensity. A third challenge is the gradient-based update of the node embeddings. For larger receptive fields, the number of neighbors exponentially more and, it can be very computationally intensive and not scalable to update all nodes involved in the forward computation, during the backward computation. There have been other works following the R-GCN model, like CompGCN [42]. However, even these face severe runtime/scalability issues [43]. All the while, the biomedical community recognises the need for faster ways to learn on graphs and specifically, multi-relational graphs. Their attention has moved towards more efficient and less heavily parameterized methods [44, 45, 46]. This indicates the need and desire for better turnaround times and efficiency for multi-relational graph learning in biomedical research.

This thesis seeks to discover and extend a scalable, efficient, and low-latency graph ML algorithm for multi-relational graph learning. It is hoped that this simple and accessible contribution enables any biomedical researcher to utilize the same to accelerate the pace of their own biomedical research and discovery pipelines.

The results of this thesis however, is expected to have considerable benefits for other domains and areas of application as well, like real-time graph learning.

1.3 Delimitation

1.3.1 Target

The current work that is presented as part of this thesis is an exploration towards finding a way for improving the costs and scalability of multi-relational graph learning to enable

fast-paced biomedical research and discovery. The research question that this thesis seeks to answer is, "What is the best way to structure graph learning at the algorithm level to enable best efficiency, scalability, and latency for large-scale multi-relational learning?". This work hopes that its findings can enable a faster pace of research and discovery in the field of biomedical research by enabling powerfully efficient machine learning on large-scale multi-relational graph data.

1.3.2 Demarcation

The current work explores ways to enable efficient, scalable and low-latency multi-relational graph learning. This work focuses on the first step in HW-SW co-design — the algorithm. Intelligent selection/discovery of the right algorithm could yield a large portion or even several times the speed up and efficiency obtained through the design and application of custom hardware architectures [1]. This current work, does not deal with development of efficient kernels, or design of custom hardware architectures. Its focus is solely on achieving good efficiency, scalability, and latency without compromising on predictive performance by means of algorithm selection/discovery, and algorithm extension.

CHAPTER 2

METHODOLOGY

2.1 Approach and Research Design

The approach followed in this thesis to seek out a way to enable efficient, scalable and low-latency learning on multi-relational graphs, is to first target the lowest-hanging fruit that delivers the most value. In the context of HW-SW Co-design, this becomes algorithm selection/discovery. Selecting the right algorithm can sometimes yield much better latency, power and memory usage than execution on a custom-built hardware architecture/accelerator [1].

Once we select a hardware-friendly, scalable and efficient algorithm, we then naturally seek to extend it to boost the benefits we can reap from it. In this direction, this thesis proposes an extension of the #GNN algorithm [2] — Het#GNN. #GNN utilizes a unique approach of replacing existing neural network based transformations (in a GNN) with Min-Hashing. Het#GNN extends the same approach to the heterogeneous setting where it applies a separate set of hash functions for the MinHashing of neighbor aggregates under each relation.

2.2 Components

2.2.1 Datasets Used

Table 2.1 lists the datasets utilized in this thesis, and also their statistics, namely, node count, edge count, feature vector size, and number of relations. Note that, here, for undirected graphs the edge count includes each direction of the undirected edges as separate edges.

Table 2.1: Datasets utilized in this thesis, and their statistics

| Name | Nodes | Edges | Feature Vector | Relations |
|-------------|--------------|--------------|-----------------------|------------------|
| Decagon | 19,762 | 10,671,808 | 19,726 | 1,932 |
| Facebook | 4,093 | 176,468 | 1,403 | 1 |
| Flickr | 7,564 | 478,730 | 12,047 | 1 |
| Cora | 2,708 | 10,858 | 1,430 | 1 |
| Citeseer | 3,312 | 4,715 | 3,703 | 1 |
| Pubmed | 19,717 | 44,338 | 512 | 1 |
| Google+ | 7,856 | 642,536 | 2,024 | 1 |
| BlogCatalog | 5,196 | 343,486 | 8,189 | 1 |

2.2.2 MinHashing

The MinHash transformation is a probabilistic transformation. It’s based on the principle of the min-wise independent permutations, which is a probabilistic method used to estimate the similarity between two sets. Het#GNN utilizes MinHash as an alternative transformation to replace the neural network based transformations in GNNs.

The MinHash algorithm works by creating a “signature” for each set in your data. This signature is a representative subset of the original set. The algorithm does this by applying a hash function to each element in the set and choosing the minimum hash value. This process is repeated with different hash functions to create a signature of multiple minimum hash values.

The key point here is that the MinHash signature represents the original set in a smaller, more manageable form, but it preserves the essential structure of the set, specifically the similarity between sets. This makes it possible to interpret the MinHash transformation: similar MinHash signatures correspond to similar sets.

So, unlike neural network based transformations that might seem like a “black box,” the MinHash transformation is a process that you can follow step by step, and the result has a clear and interpretable meaning in terms of the original data.

MinHash is also non-linear by nature because it does not satisfy the additivity property or homogeneity property. This means we can completely avoid activation functions like

ReLU when we replace the neural networks with MinHashing.

2.2.3 General formulation for a simple GNN

A general formulation for the message passing in GNNs can be considered as described below [47].

$$x_v^{(t)} = \sigma \left(U_1^{(t)} x_v^{(t-1)} \circ \left(U_2^{(t)} \sum_{u \in N(v)} \sigma \left(U_3^{(t)} x_u^{(t-1)} \right) \right) \right)$$

Here, this equation represents the update that happens to each node’s embedding upon moving from layer/time-step t-1 to t.

U_3 transforms each neighbor node embedding and σ applies a non-linear transform on each of them before they are aggregated together by means of some aggregation scheme, like mean, sum, min, or max. U_2 transforms the aggregated embedding before combining it with the result of the transformation of the ego node embedding by U_1 .

This formulation lays the foundation for a simple GNN.

2.2.4 Replacing Neural Netowrks with MinHash

The unique approach taken by #GNN is that it completely replaces the neural network based transforms U_1 , U_2 , and U_3 with MinHashing. This also completely de-parameterizes the embedding generation process. It is described more precisely in Algorithm 1. Here $\pi_1^{(t,k)}$, $\pi_2^{(t,k)}$, and $\pi_3^{(t,k)}$ are the three sets of hash functions that are used in MinHashing to represent U_1 , U_2 , and U_3 uniquely.

2.2.5 Enabling Multi-Relational Learning

Het#GNN extends #GNN to the heterogeneous setting. This is achieved by targeting U_2 . We know that U_2 is the transform unique to the neighborhood aggregate. So, we can enable a heterogeneous aggregation by having a unique U_2 for the aggregate under each relation. For #GNN, this can be achieved by having a separate set of π_2 hash functions for each relation r . This is described in detail in Algorithm 2

Algorithm 1 The #GNN Algorithm

Ensure: $G = (V, E, F), T, K, \{\pi_1^{(t,k)}, \pi_2^{(t,k)}, \pi_3^{(t,k)}\}_{t=1, k=1}^{T, K}$

Require: H

```
1: for  $t = 1, \dots, T$  do
2:   for  $k = 1, \dots, K$  do
3:     for  $v \in V$  do
4:        $x_{v,1}^{(t,k)} \leftarrow \arg \min \pi_3^{(t,k)}(x_v^{(t-1)})$ 
5:     end for
6:     for  $v \in V$  do
7:        $x_{v,\text{neighbors}}^{(t,k)} \leftarrow \bigcup_{u \in N(v)} x_{v,1}^{(t,k)}$ 
8:        $x_v^{(t)}[k] \leftarrow \arg \min(\pi_1^{(t,k)}(x_v^{(t-1)}) \cup \pi_2^{(t,k)}(x_{v,\text{neighbors}}^{(t,k)}))$ 
9:     end for
10:  end for
11:  for  $v \in V$  do
12:     $H[v, :] \leftarrow x_v^{(T)}$ 
13:  end for
14: end for
```

Algorithm 2 The Het#GNN Algorithm

Ensure: $G = (V, E, F), T, K, R, \{\pi_1^{(t,k)}, \pi_2^{(t,k,r)}, \pi_3^{(t,k)}\}_{t=1, k=1, r=1}^{T, K, R}$

Require: H

```
1: for  $t = 1, \dots, T$  do
2:   for  $k = 1, \dots, K$  do
3:     for  $v \in V$  do
4:        $x_{v,1}^{(t,k)} \leftarrow \arg \min \pi_3^{(t,k)}(x_v^{(t-1)})$ 
5:     end for
6:     for  $v \in V$  do
7:       for  $r \in R$  do
8:          $x_{v,\text{neighbors}}^{(t,k,r)} \leftarrow \bigcup_{u \in N_r(v)} x_{v,1}^{(t,k)}$ 
9:          $x_v^{(t)}[k] \leftarrow \arg \min(\pi_1^{(t,k)}(x_v^{(t-1)}) \cup \pi_2^{(t,k,r)}(x_{v,\text{neighbors}}^{(t,k,r)}))$ 
10:      end for
11:    end for
12:  end for
13:  for  $v \in V$  do
14:     $H[v, :] \leftarrow x_v^{(T)}$ 
15:  end for
16: end for
```

This extension enables the significant benefit of being able to apply this unsupervised node embedding algorithm to knowledge graphs, which are large-scale heterogeneous graphs that can greatly benefit from the efficiency, scalability, and latency benefits of Het#GNN.

2.3 Experiments and Evaluation

All experiments are conducted on a consumer-grade Intel i7 12450H CPU clocked at default 4.40GHz with the CPU governor set to 'performance'. They are performed on an Ubuntu-equivalent Linux operating system. Attention was taken to see that there was no interference due to CPU thermal throttling on the experiments. All runtime measurements are averaged across 5 trials. All memory measurements are peak memory usage observed in MB (1000 KB) across 5 trials.

2.3.1 Scaling of Runtime

With Hashed Feature Vector Dimension

In this experiment, the 'hashed' feature vector dimension (hashdim) is varied from 400 to 1200 at intervals of 200. The runtime for embedding generation is measured in seconds for each case. This is performed for three homogeneous graph datasets 'Citeseer', 'Pubmed', and 'Cora'. The receptive field size is set to 1-hop.

Figure 2.1 conveys the scaling of runtime observed. As expected, we observe a linear scaling of embedding generation latency. This would be as the implementation loops over each feature, and increasing the hashed feature vector size translates to increasing the number of iterations of the loop. We can also observe that the runtime varies between datasets under the same setting. Looking at it from a theoretical viewpoint, this would be because of the difference in the number of nodes and the sparsity of the binary input feature vector that create a difference in runtime between different datasets.

For this experiment, the i7 12450H CPU's clock was capped at 2.8GHz.

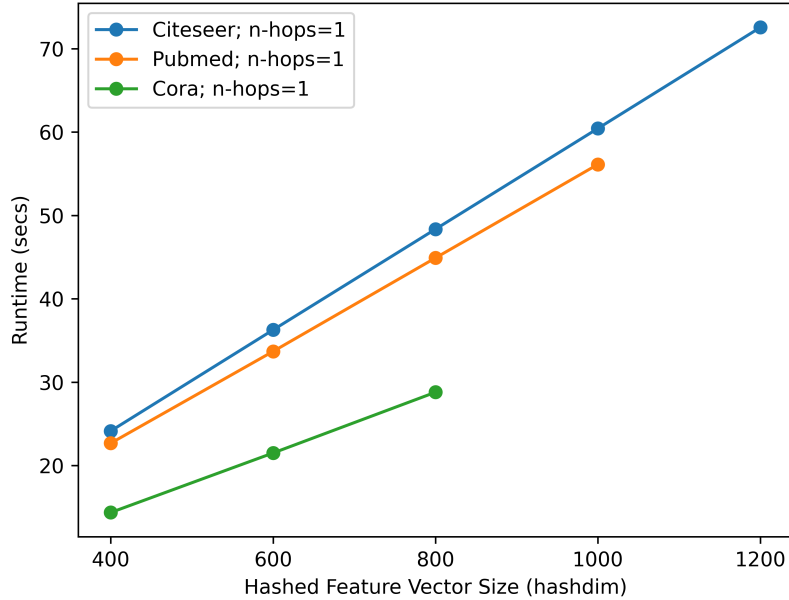


Figure 2.1: Scaling of runtime with hashed feature vector size for 1-hop receptive field

With Receptive Field Size

In this experiment, the receptive field size, or the node neighborhood size, or the number of message passing layers (n-hop) is varied from 1 to 4. The runtime for embedding generation is measured in seconds for each case. This is performed for a homogeneous graph datasets 'Flickr'. The hashed feature vector dimension is fixed at 200.

Figure 2.2 conveys the scaling observed. In this experiment too, a linear scaling of runtime is noted as expected. The number of iterations of message passing done for each node determines the receptive field size. In Het#GNN, incrementing the receptive field size is equivalent to incrementing the number of iterations in the respective loop, which supports this observed trend.

For this experiment, the i7 12450H CPU's clock was capped at 2.8GHz.

With combined change in Receptive Field Size and Feature Vector Size

Now we explore two experiments studying the combined influence of receptive field size and feature vector size on runtime. For experiment one, we wish to observe the influence

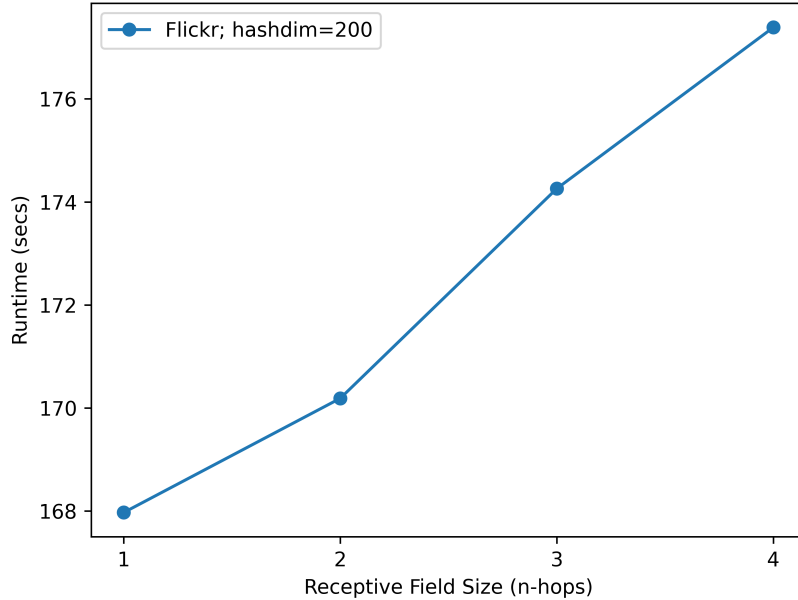


Figure 2.2: Scaling of runtime with receptive field size hashdim=200

of receptive field size at different values of hashed feature vector size, specifically, hashdim values 200, 400, and 600. The receptive field size is varied from 6 to 8, 4 to 6, and 2 to 4 respectively. This is performed for the homogeneous graph datasets 'Facebook'.

Figure 2.3 conveys the associated scaling of runtime observed with change in receptive field, at different values of hashdim. The key observation made here is the multiplicative influence of hashdim value on the rate at which runtime scales with the receptive field size.

For this experiment, the i7 12450H CPU's clock was capped at 2.8GHz.

For the second experiment, we vary the feature vector size from 200 to 1000 at intervals of 200 for different values of receptive field size, specifically, n-hop values 1, 2, and 3. This is performed for the homogeneous graph dataset 'BlogCatalog'

Figure 2.4 conveys the associated scaling across feature vector sizes, at different values of n-hop. Here, it is observed that at 1-hop receptive field size, the scaling of runtime is linear with hashdim. Then as the receptive field size is increased, it is observed that the scaling shifts slightly towards an exponential trend, but still not too far from linear. This interesting difference with respect to experiment one can be linked to the fact that increasing receptive field size introduces the phenomenon of neighborhood explosion (exponential

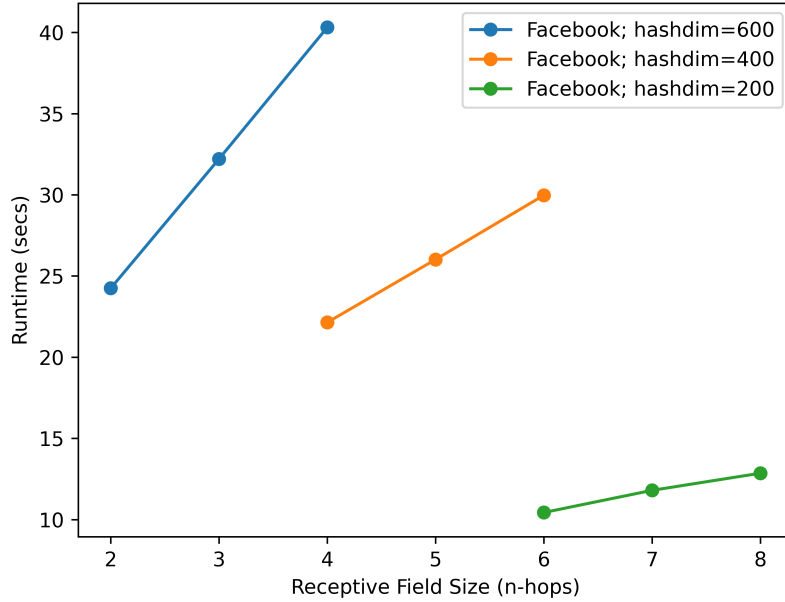


Figure 2.3: Scaling of runtime with receptive field size at different hashdim values

increase). Increasing the receptive field not only increases the number of iterations on each feature vector, but it also increases the computation needed per each iteration on each feature vector in an exponential manner.

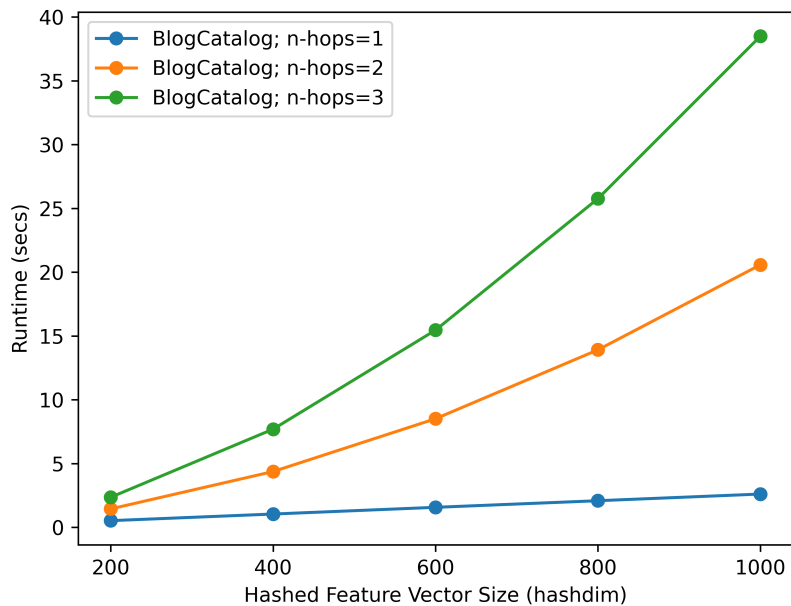


Figure 2.4: Scaling of runtime with hashed feature vector size and receptive field

With Node Count

In this experiment, Het#GNN is run on different subsets of the homogeneous graph dataset 'Google+' that 25%, 50%, 75%, and 100% of the nodes respective. This enables observing the scaling of runtime as the number of nodes increases. The hashdim and n-hop are fixed at decently high values of 400 and 5 respectively so as to exaggerate the trend so that it very clearly stands out.

Figure 2.5 communicates the associated trend in runtime. We can see that runtime increases exponentially, but still with a low net latency (< 18 secs). Here, the exponential increase is expected given that it not only increases the number of iterations feature iterations but it also enhances the node neighborhood explosion phenomenon. This exponential increase is observed as it is Google+ is not a really large graph. This would not be the expected trend for large graphs because, in that case, increase in nodes would not enhance the node neighborhood explosion phenomenon by a significant extent.

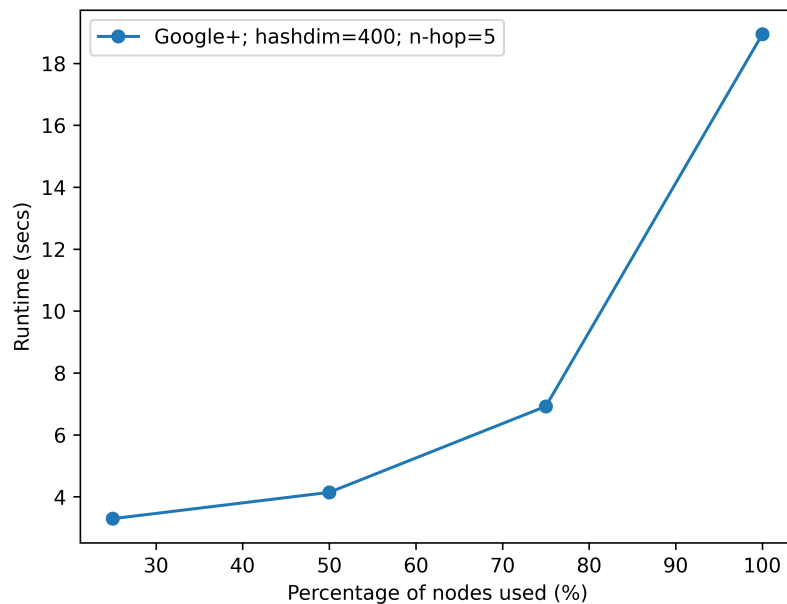


Figure 2.5: Scaling of runtime with number of nodes

With Receptive Field Size in the Heterogeneous Setting

In this experiment, Het#GNN is run on the heterogeneous graph dataset 'Decagon'. Receptive field sizes from 1 to 4 are attempted. Hashdim is fixed at 200.

We can clearly observe the linear trend indicating the heterogeneous extension does not cause a deviation from the default linear scaling observed with respect to receptive field size.

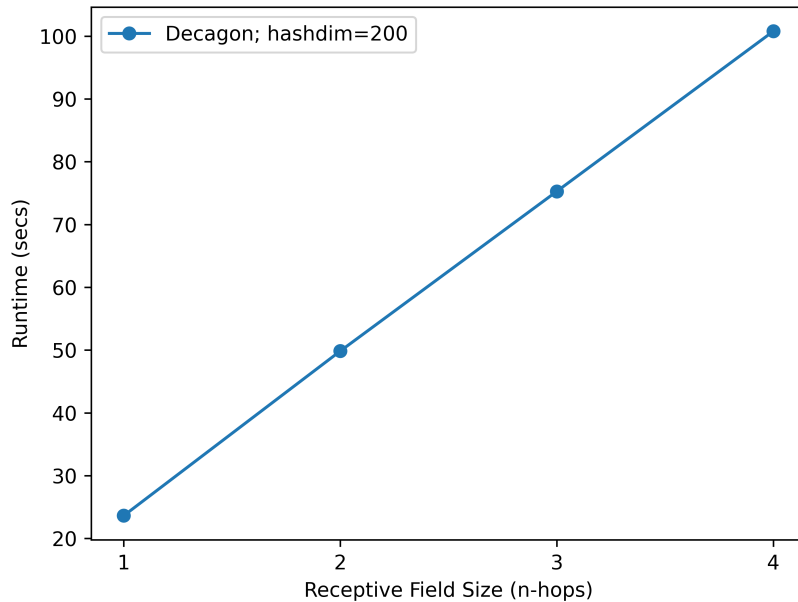


Figure 2.6: Scaling of runtime with hashed feature vector size for 1-hop receptive field

2.3.2 Scaling of Memory Usage

With Hashed Feature Vector Dimension and Receptive Field Size

In this experiment, Het#GNN is run on the homogeneous graph datasets 'BlogCatalog' and 'Google+'. The hashed feature vector size is varied from 200 to 1000 in intervals of 200. Instead of runtime like collected in the previous experiments, peak memory usage is collected in MB. The variation of hashed feature vector size is repeated across different values of n-hops, namely, 1-hop, 2-hop and 3-hop.

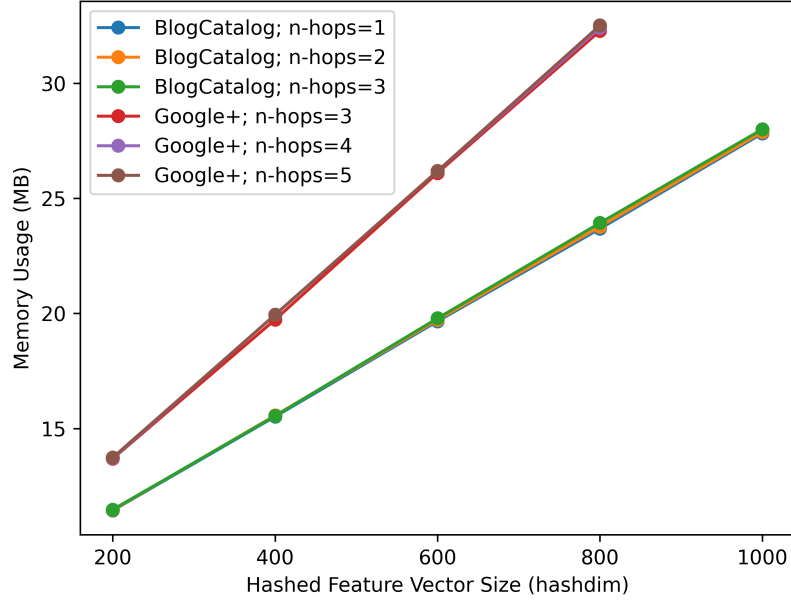


Figure 2.7: Scaling of peak memory usage with hashed feature vector size and receptive field

Figure 2.7 shows the associated trend. We can note how it appears as if increasing the receptive field size does not increase the peak memory usage, but in fact there is a linear scaling in memory usage but it is so low that it appears non-existent. With hashed feature vector size we can see a very linear trend and at the same time the peak memory used is very low in the range of 30MB which is often not the case with GNNs.

With Number of Nodes

In this experiment, Het#GNN is run on the homogeneous graph dataset 'Google+'. The percent of nodes used from the dataset is varied from 25% to 100% in intervals of 25%. hashdim and n-hop are fixed at 400 and 5 respectively.

Figure 2.8 conveys this the associated trend. We can see a linear scaling of memory usage with number of nodes. This linear increase would be because of the linear increase in the number of feature vectors on which the algorithm needs to be computed. This linear scaling of memory usage with node count is a necessity as we scale to larger and larger graphs.

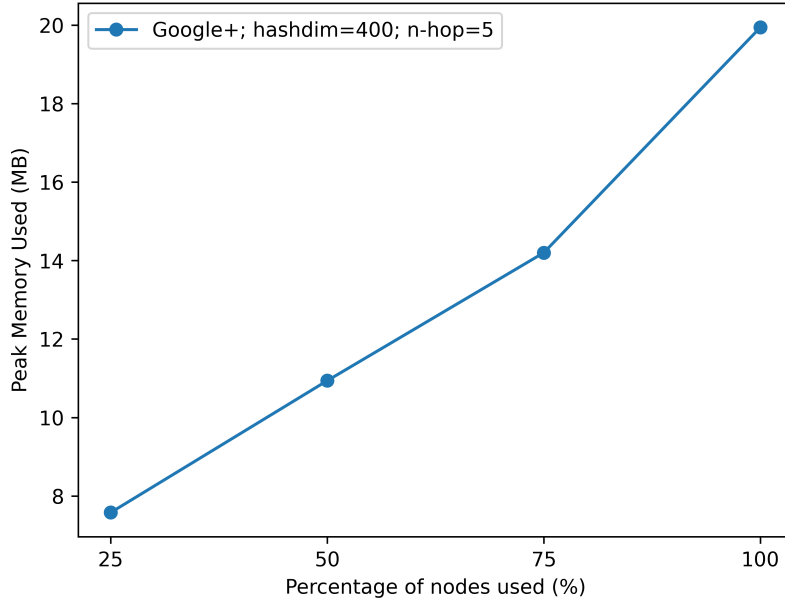


Figure 2.8: Scaling of peak memory usage with number of nodes

With Receptive Field Size in the Heterogeneous Setting

In this experiment, Het#GNN is run on the heterogeneous Decagon graph dataset. The receptive field size is varied from 1 to 4. hashdim and n-hop are fixed at 400 and 5 respectively.

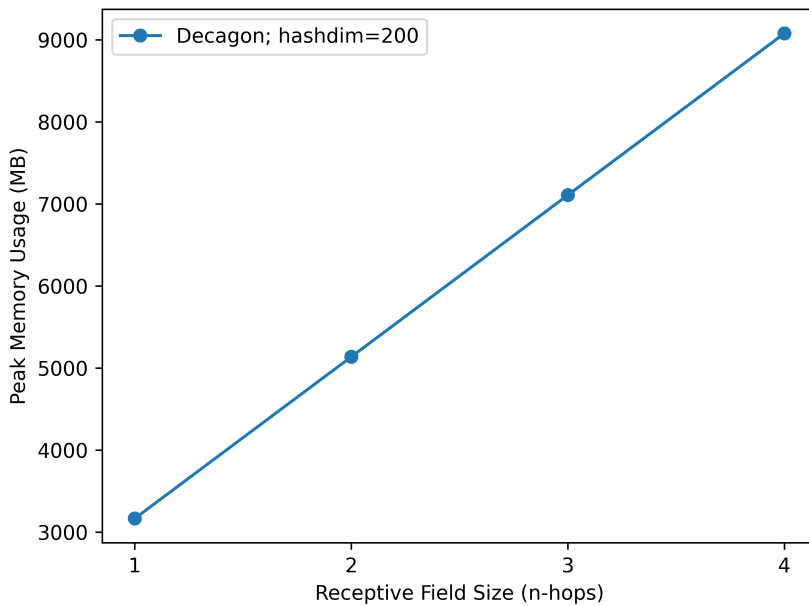


Figure 2.9: Scaling of runtime with receptive field size in heterogeneous setting

Figure 2.9 communicates the trend in memory usage with receptive field size for this heterogeneous setting. Consistent with what was discussed previously, linear scaling of memory usage is observed with increase in receptive field size.

CHAPTER 3

RESULTS

3.1 Performance

The predictive performance is measured in terms of AUROC and AUPRC scores. These scores are calculated per relation utilizing the 'roc_auc_score()' and 'average_precision_score()' functions of scikit-learn library. Decagon also utilizes the same scores, the same python library, and calculates them the same way. The results are tabulated in Table 3.1.

To calculate the final AUROC, and AUPRC, the per-relation AUROC and AUPRC are averaged across the relations. Here, the average scores are collected in two settings:

1. They are averaged across all 1932 relations which include forward relations, the corresponding inverse relations, the protein-protein relations, the drug-drug relations (which are not associated with side-effects), and the drug-protein relations.
2. They are averaged across 963 side-effect (forward) relations. This is the setting in which Decagon reports its scores.

Table 3.1: Heterogeneous link prediction performance of Het#GNN on the decagon dataset

| Relations Considered | AUROC | AUPRC |
|--|--------------|--------------|
| All Relations Including Inverse Relations (1932) | 0.944 | 0.908 |
| Only Forward Side-Effect Relations (963) | 0.945 | 0.909 |

We can note how Het#GNN exhibits a strong performance on such a complex dataset despite being non-parameteric. Another aspect to note here is that Decagon utilizes individual drug side-effects as drug node feature information. This is not done for the learning and evaluation executed in this work, which would mean Het#GNN is in a more disadvantaged position.

3.2 Comparison

Table 3.2 compares the performance obtained by Het#GNN with all published works found that cite their performance on the Decagon dataset [14]. Despite being non-parametric, we can see how Het#GNN has demonstrated a superior performance. Table 3.3 compares the efficiency in terms of memory and runtime, of Het#GNN with all other published works that cite their runtime on Decagon dataset. Het#GNN was able to outperform the state-of-the-art at less than a tiny fraction of the runtime, memory and power costs. This runtime achievement of Het#GNN is obtained on a consumer CPU, while all other methods utilize server GPUs to accelerate their algorithm.

Table 3.2: Published works that cite their best performance on Decagon heterogeneous graph dataset

| | Method | AUROC | AUPRC | Emb. Gen. |
|----------------------------|---------------------|--------------|--------------|--------------------------|
| This work | Het#GNN | 0.945 | 0.909 | Non-Parameterized |
| Xu et. al, 2020 [48] | TIP | 0.914 | 0.89 | Parameterized |
| Burkhardt et. al, 2019 [5] | ESP | 0.903 | 0.875 | Parameterized |
| Zitnik et. al, 2018 [3] | <i>Decagon RGCN</i> | <i>0.872</i> | <i>0.832</i> | <i>Parameterized</i> |
| Xu et. al, 2020 [48] | DistMult | 0.835 | 0.859 | Parameterized |

Table 3.3: Het#GNN’s Runtime (secs) and Peak Memory Usage (MB) against that of published works that cite their performance on Decagon dataset

| | Method | Runtime | Memory | Hardware |
|----------------------------|---------------------|-----------------------|-----------------|---------------------|
| This work | Het#GNN | 75 secs | 7.1GB | Consumer CPU |
| Xu et. al, 2020 [48] | TIP | 3.2 hrs | 9.47GB | Server GPU |
| Burkhardt et. al, 2019 [5] | ESP | 6.6 hrs | - | Server GPU |
| Marinka et. al, 2018 [3] | <i>Decagon RGCN</i> | <i>>>12days</i> | <i>>28GB</i> | <i>Server GPU</i> |
| Xu et. al, 2020 [48] | DistMult | 68 mins | 9.26GB | Server GPU |

This brings up a very important question that needs to be in our attention — **”Do we really need all the neurons that we utilize?”**. Neural networks are an expensive mechanism in ML. It comes with the power to learn an arbitrary function suited to achieve a strong fit. However, do we really need this learning capability, (and the associated heavy expenses) for all aspects of all ML pipelines that we are interested in? From the results

demonstrated here, it is observed that the proposed Het#GNN — a totally unsupervised network embedding algorithm — could handle the multi-relational graph learning task on a consumer-grade CPU, several times faster and more efficiently than what others achieved on server-grade GPUs, while still not compromising on the predictive performance.

CHAPTER 4

DISCUSSION AND CONCLUSION

4.1 Benefits

4.1.1 Algorithm

If an algorithm is decided for an application, that by itself sets hard boundaries on what a custom hardware accelerator can achieve for that application. Sometimes, simple and smart choices at the algorithm level can make a huge difference to the achievable efficiency, scalability, and latency. So it is important to discover/select suitable algorithmic techniques first before diving into hardware-level design. This reduces effort while greatly increasing the applicability of the custom hardware accelerators we design down the line.

Some of the key benefits of the proposed Het#GNN algorithm are:

- Het#GNN targets the core of graph machine learning — network embedding.
 - It is a very fundamental task that defines graph learning.
 - Het#GNN transforms the graph learning ML task to a regular ML task that can be handled by any existing non-graph ML pipeline. It achieves this without needing any kind of parameter learning.
- Multi-relational graph learning.
 - Unlike other algorithms, Het#GNN is able to learn on heterogeneous graph data while remaining fast, efficient, scalable, and parameter-free.
- The algorithm is bound by strong mathematical and statistical theory.
 - Not a black box like neural networks.

- MinHashing as a locality-sensitive hashing technique has a strong and interpretable mathematical backbone.
- The network embedding process is non-parametric, and doesn't involve a training phase.
 - Training incurs the greatest hit to memory and runtime for large-scale ML. Lack of a training phase completely removes a huge portion of the time, energy and effort costs from the pipeline. This also removes multiple uncertainties if considering a production deployed pipeline.
 - Propagating gradients all the way to the node embeddings through the message passing operation becomes hard to scale as the receptive field size increases. However, this is the nature of training for several of the existing methods like DistMult and R-GCN.
- Simple. Efficient. Hardware-friendly.
 - Can run on CPU and still achieve competitive prediction performance, better latency and better scaling than the parameterized R-GCN model.
 - Het#GNN achieves its demonstrated results despite being completely sequential and not exploiting any parallelism.
 - Each 'transformation' operation involved in message passing of Het#GNN is just an application of MinHashing, which is very efficient and involves very few multiplications compared to neural network-based transformation.
 - Uses binary feature vectors
 - * Possible to implement an entirely binary feature vector pipeline.
 - * Node embedding similarity in this setting is just jaccard similarity – one of the most hardware-friendly similarity calculations.

- * No need to generate dense FP32 input feature vectors to apply the Het#GNN algorithm.
- Can be incorporated into an entirely unsupervised end-to-end pipeline.
 - Although results demonstrated utilize logistic regression for the downstream task, it is possible to extend Het#GNN to form an end-to-end unsupervised graph learning pipeline.
 - Unsupervised pipelines enable the best scalability and runtimes for real-world applications due to the absence of the expensive optimization that we call training.
- Het#GNN runtime and peak memory usage scales well with receptive field size and feature vector size.
- Applicability in real-time setting.
 - Het#GNN does not learn node embeddings through backpropagation gradients like other multi-relational graph learning methods. Hence, Het#GNN does not require storage of 'learned' node embeddings. Embeddings could be generated online for just the nodes of interest.
 - Het#GNN can handle new unseen nodes without feature vectors.
 - Het#GNN utilizes binary input feature vectors which could even be generated real-time on the platform of deployment.

4.1.2 Implementation

Some of the key benefits of the implementation of Het#GNN are:

- Extremely simple and efficient sparse representation of binary vectors.
- Simple C++ code that does not use any ML frameworks.

- Avoids any framework-associated overheads and complexities.

4.2 Limitations

At the algorithm level:

- One of the significant advantages of Het#GNN is that it achieves competitive predictive performance while being completely non-parametric for embedding generation. This also forms a limitation of Het#GNN that it cannot handle cases where it is unavoidable to learn some complex function as part of message passing. This limitation does not apply if the goal is to just represent the graph structure in the form of node embeddings.
- Unlike neural networks, we cannot have a tunable knob of the number of parameters that can be increased to increase model performance.
- Het#GNN requires the input feature vectors to be binary vectors. However, in the experiments associated with this thesis, it has been observed that this can be worked around by applying a simple and efficient binarization technique of repeated classifications using random hyperplanes. Moreover, production-level research from big companies show a keen interest in forming binary feature vectors [30].

At the implementation level:

- Current Het#GNN implementation is fully sequential although there is potential for parallel execution in the algorithm.

4.3 Contributions and Practical Implications

Key contributions:

1. Algorithm Discovery: Finding a fast, efficient, scalable and hardware friendly algorithm that can generate expressive network embeddings — #GNN.

2. Proposed Het#GNN: Extended the #GNN algorithm to handle multi-relational network embedding.
3. Validated the performance of Het#GNN on Decagon dataset by comparing it to other methods in terms of runtime, memory usage, and prediction accuracy.
4. Explored in detail how the runtime and peak memory usage of Het#GNN scales with feature vector and receptive field size.

Practical Implication:

- Could be used in a real-time setting to generate embeddings for nodes in an online manner.
- Hints great potential for use with large-scale knowledge graphs.

4.4 Future Scope

- Currently, Het#GNN generates the node embeddings in a non-parametric way, and following that, logistic regression is trained on the generated embeddings for each relation so as to perform heterogeneous link prediction.
 - It is possible to extend Het#GNN to form a fully unsupervised pipeline for heterogeneous link prediction.
- Different kinds of application of hashing to different steps of the embedding generation and downstream pipeline can potentially avoid using expensive neurons, and save time and energy, while still performing really well. Het#GNN can serve as a baseline for such an exploration.

REFERENCES

- [1] B. Dally, [Research Talk @ GT, ECE] *Bill Dally — Directions in deep learning hardware*, <https://youtu.be/gofI47kfD28?&t=4272>, Accessed: 28, April, 2024.
- [2] W. Wu, B. Li, C. Luo, and W. Nejdl, “Hashing-accelerated graph neural networks for link prediction,” in *Proceedings of the Web Conference 2021*, ser. WWW ’21, Ljubljana, Slovenia: Association for Computing Machinery, 2021, pp. 2910–2920, ISBN: 9781450383127.
- [3] M. Zitnik, M. Agrawal, and J. Leskovec, “Modeling polypharmacy side effects with graph convolutional networks,” *Bioinformatics*, vol. 34, no. 13, pp. i457–i466, Jun. 2018.
- [4] M. Schlichtkrull, T. N. Kipf, P. Bloem, R. van den Berg, I. Titov, and M. Welling, “Modeling relational data with graph convolutional networks,” in *The Semantic Web*, A. Gangemi *et al.*, Eds., Cham: Springer International Publishing, 2018, pp. 593–607.
- [5] H. A. Burkhardt, D. Subramanian, J. Mower, and T. Cohen, “Predicting adverse drug-drug interactions with neural embedding of semantic predications,” *AMIA Annu. Symp. Proc.*, vol. 2019, pp. 992–1001, 2019.
- [6] J. Jumper *et al.*, “Highly accurate protein structure prediction with alphafold,” *Nature*, vol. 596, no. 7873, pp. 583–589, 2021.
- [7] R. Mercado *et al.*, “Graph Networks for Molecular Design,” *Machine Learning: Science and Technology*, 2020.
- [8] M. J. Jahid and J. Ruan, “An ensemble approach for drug side effect prediction,” *Proceedings (IEEE Int. Conf. Bioinformatics Biomed.)*, pp. 440–445, 2013.
- [9] Z. Zhu *et al.*, *Torchdrug: A powerful and flexible machine learning platform for drug discovery*, 2022. arXiv: 2202.08320 [cs.LG].
- [10] B. Rozemberczki *et al.*, “Chemicalx: A deep learning library for drug pair scoring,” in *Proceedings of the 28th ACM SIGKDD Conference on Knowledge Discovery and Data Mining*, ser. KDD ’22, Washington DC, USA: Association for Computing Machinery, 2022, pp. 3819–3828, ISBN: 9781450393850.
- [11] B. Ramsundar, P. Eastman, P. Walters, V. Pande, K. Leswing, and Z. Wu, *Deep Learning for the Life Sciences*. O’Reilly Media, 2019, <https://www.amazon.com/Deep-Learning-Life-Sciences-Microscopy/dp/1492039837>.

- [12] Z. Wu *et al.*, “MoleculeNet: A benchmark for molecular machine learning,” *Chem. Sci.*, vol. 9, no. 2, pp. 513–530, Jan. 2018.
- [13] K. M. Borgwardt, C. S. Ong, S. Schönauer, S. V. N. Vishwanathan, A. J. Smola, and H.-P. Kriegel, “Protein function prediction via graph kernels,” *Bioinformatics*, vol. 21, pp. i47–i56, Jun. 2005.
- [14] M. Zitnik and J. Leskovec, “Predicting multicellular function through multi-layer tissue networks,” *Bioinformatics*, vol. 33, no. 14, pp. i190–i198, Jul. 2017.
- [15] *BioKG Link Property Prediction* — *ogb.stanford.edu*, <https://ogb.stanford.edu/docs/linkprop/>, [Accessed 27-04-2024].
- [16] IBM, *IBM Micromedex: Improving efficiency and care with AI* — *ibm.com*, <https://www.ibm.com/case-studies/tidalhealth-peninsula-regional/>, [Accessed 27-04-2024].
- [17] *AWS Marketplace: Metaphactory with Amazon Neptune for Pharma & Life Sciences* — *aws.amazon.com*, <https://aws.amazon.com/marketplace/pp/prodview-pio5i6lyqocoo>, [Accessed 27-04-2024].
- [18] F. Knoll *et al.*, “FastMRI: A publicly available raw k-space and DICOM dataset of knee images for accelerated MR image reconstruction using machine learning,” *Radiol. Artif. Intell.*, vol. 2, no. 1, e190007, Jan. 2020.
- [19] J. Quesada *et al.*, “MTNeuro: A Benchmark for Evaluating Representations of Brain Structure Across Multiple Levels of Abstraction,” *Adv Neural Inf Process Syst*, vol. 35, pp. 5299–5314, 2022.
- [20] R. Pan *et al.*, “Synthetic lethality of combined bcl-2 inhibition and p53 activation in AML: Mechanisms and superior antileukemic efficacy,” *Cancer Cell*, vol. 32, no. 6, 748–760.e6, Dec. 2017.
- [21] H. Liang, L. Chen, X. Zhao, and X. Zhang, “Prediction of drug side effects with a refined negative sample selection strategy,” *Comput. Math. Methods Med.*, vol. 2020, p. 1 573 543, May 2020.
- [22] *Adverse Drug Events in Adults — Medication Safety Program* — *CDC* — *cdc.gov*, https://www.cdc.gov/medicationsafety/adult_adversedrugsafety.html, [Accessed 27-04-2024].
- [23] E. Muñoz, V. Nováček, and P.-Y. Vandebussche, “Facilitating prediction of adverse drug reactions by using knowledge graphs and multi-label learning models,” *Brief. Bioinform.*, vol. 20, no. 1, pp. 190–202, Jan. 2019.

- [24] *AlphaFold - Wikipedia* — *en.wikipedia.org*, <https://en.wikipedia.org/wiki/AlphaFold>, [Accessed 27-04-2024].
- [25] S. Cheng *et al.*, “Fastfold: Optimizing alphafold training and inference on gpu clusters,” in *Proceedings of the 29th ACM SIGPLAN Annual Symposium on Principles and Practice of Parallel Programming*, New York, NY, USA: Association for Computing Machinery, 2024, pp. 417–430.
- [26] T. N. Kipf and M. Welling, “Semi-supervised classification with graph convolutional networks,” in *International Conference on Learning Representations (ICLR)*, 2017.
- [27] D. Jiang *et al.*, “Could graph neural networks learn better molecular representation for drug discovery? a comparison study of descriptor-based and graph-based models,” *J. Cheminform.*, vol. 13, no. 1, p. 12, Feb. 2021.
- [28] E. Alsentzer, S. G. Finlayson, M. M. Li, and M. Zitnik, “Subgraph neural networks,” in *Proceedings of Neural Information Processing Systems, NeurIPS*, 2020.
- [29] R. Mercado *et al.*, “Practical Notes on Building Molecular Graph Generative Models,” *Applied AI Letters*, 2020.
- [30] Q. Tan, N. Liu, X. Zhao, H. Yang, J. Zhou, and X. Hu, “Learning to hash with graph neural networks for recommender systems,” in *Proceedings of The Web Conference 2020*, ser. WWW ’20, Taipei, Taiwan: Association for Computing Machinery, 2020, pp. 1988–1998, ISBN: 9781450370233.
- [31] S. Adeshina *et al.*, “Propinit: Scalable inductive initialization for heterogeneous graph neural networks,” in *2022 IEEE International Conference on Knowledge Graph (ICKG)*, 2022, pp. 6–13.
- [32] M. Yasunaga, H. Ren, A. Bosselut, P. Liang, and J. Leskovec, “QA-GNN: Reasoning with language models and knowledge graphs for question answering,” in *Proceedings of the 2021 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, K. Toutanova *et al.*, Eds., Online: Association for Computational Linguistics, Jun. 2021, pp. 535–546.
- [33] T. Zhang, H. Yin, R. Wei, P. Li, and A. Shrivastava, *Learning scalable structural representations for link prediction with bloom signatures*, 2023. arXiv: 2312.16784 [cs.LG].
- [34] M. Yan *et al.*, “Hygcn: A gcn accelerator with hybrid architecture,” in *2020 IEEE International Symposium on High Performance Computer Architecture (HPCA)*, 2020, pp. 15–29.

- [35] T. Geng *et al.*, *I-gcn: A graph convolutional network accelerator with runtime locality enhancement through islandization*, 2022. arXiv: 2203.03606 [cs.AR].
- [36] T. Geng *et al.*, “Awb-gcn: A graph convolutional network accelerator with runtime workload rebalancing,” in *2020 53rd Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*, Los Alamitos, CA, USA: IEEE Computer Society, Oct. 2020, pp. 922–936.
- [37] J. Chen, T. Ma, and C. Xiao, “FastGCN: Fast learning with graph convolutional networks via importance sampling,” in *International Conference on Learning Representations*, 2018.
- [38] F. Wu, T. Zhang, A. H. de Souza Jr. au2, C. Fifty, T. Yu, and K. Q. Weinberger, *Simplifying graph convolutional networks*, 2019. arXiv: 1902.07153 [cs.LG].
- [39] W.-L. Chiang, X. Liu, S. Si, Y. Li, S. Bengio, and C.-J. Hsieh, “Cluster-gcn: An efficient algorithm for training deep and large graph convolutional networks,” in *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, ser. KDD ’19, Anchorage, AK, USA: Association for Computing Machinery, 2019, pp. 257–266, ISBN: 9781450362016.
- [40] W. L. Hamilton, R. Ying, and J. Leskovec, “Inductive representation learning on large graphs,” in *NIPS*, 2017.
- [41] H. Zeng, H. Zhou, A. Srivastava, R. Kannan, and V. Prasanna, “Graphsaint: Graph sampling based inductive learning method,” in *International Conference on Learning Representations*, 2020.
- [42] S. Vashishth, S. Sanyal, V. Nitin, and P. Talukdar, *Composition-based multi-relational graph convolutional networks*, 2020. arXiv: 1911.03082 [cs.LG].
- [43] S. Vashishth, *Model efficiency · Issue 5 · malllabiisc/CompGCN — github.com*, <https://github.com/malllabiisc/CompGCN/issues/5>, [Accessed 28-04-2024].
- [44] R. Liu and A. Krishnan, “PecanPy: a fast, efficient and parallelized Python implementation of node2vec,” *Bioinformatics*, vol. 37, no. 19, pp. 3377–3379, Mar. 2021.
- [45] A. Noori, A. L. Tan, M. M. Li, and M. Zitnik, “Metapaths: Similarity search in heterogeneous knowledge graphs via meta paths,” *arXiv: 2209.0000*, 2022.
- [46] C. Ruiz, M. Zitnik, and J. Leskovec, “Identification of disease treatment mechanisms through the multiscale interactome,” *Nat. Commun.*, vol. 12, no. 1, p. 1796, Mar. 2021.

- [47] T. Lei, W. Jin, R. Barzilay, and T. Jaakkola, “Deriving neural architectures from sequence and graph kernels,” in *Proceedings of the 34th International Conference on Machine Learning*, D. Precup and Y. W. Teh, Eds., ser. Proceedings of Machine Learning Research, vol. 70, PMLR, Jun. 2017, pp. 2024–2033.
- [48] H. Xu, S. Sang, and H. Lu, *Tri-graph information propagation for polypharmacy side effect prediction*, 2020. arXiv: 2001.10516 [cs.LG].