Addressing Scalability and Performance for Community Detection and Clustering in Complex Graphs

BRENNAN, JOHN,DAVID,PATRICK

Addressing Scalability and Performance for Community Detection and Clustering in Complex Graphs

John Brennan

A Thesis presented for the degree of
Doctor of Philosophy

Innovative Computing Group
Department of Computer Science
University of Durham
England

October 2020

Dedicated to

Leanne - the one who always puts up with me.

Addressing Scalability and Performance for Community Detection and Clustering in Complex Graphs

John David Patrick Brennan

Submitted for the degree of Doctor of Philosophy
August, 2021

Abstract

Graphs are increasingly being used to provide structured representations of data as they are able to well encapsulate complex relationships within the data. This has led to the development of an abundance of graph centric analysis methods. These methods are used across many academic and industrial domains and aim to extract structural and spacial information from the topology of a graph.

Current approaches for identifying community memberships and clusters within complex networks often rely on a global view of a graph, normally requiring the entire graph to be held in memory. With the ever-growing size of graph datasets, processing this global view in memory is becoming increasingly difficult. In addition there are currently gaps in the existing literature related to community classification, identification and measurements of similarity without maintaining an expensive global view of the graph.

This work aims to address some of these issues through a number of approaches. These include an investigation into finding optimal ways of comparing the similarity between graphs as well as identification of optimal way of identifying subgraphs, with common features, within a larger graph or, more simply, Community Detection. Development of ideal strategies for reducing computational and memory requirements of graph processing is also a major contribution of this work.

Declaration

The work in this thesis is based on research carried out at the Durham University Department of Computer Science, England. No part of this thesis has been submitted elsewhere for any other degree or qualification and it is all my own work unless referenced to the contrary in the text.

Acknowledgements

I would like to thank my supervisory team including Dr Boguslaw Obara, Dr Stephen McGough and Professor Georgios Theodoropoulos for their guidance through each stage of the process.

This thesis would not have been possible without the unending support of my wife (Mrs Leanne Brennan), my parents (Mrs Jacqueline Brennan and Mr James Brennan) and the rest of my family. Thank you all for always being there to support me.

I would also like to acknowledge, the various forms, of invaluable help and support I received have from my colleagues at Durham University throughout my studies - Dr Ibad Kureshi, Dr Stephen Bonner, Dr Amir Atapour-Abarghouei, Dr Philip Jackson, and Dr Chas Nelson.

Finally, I would like to thank the Engineering and Physical Sciences Council, UK for providing funding for this research (EP/M020576/1).

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Clustering, the grouping of objects which share similar characteristics, and identification of communities within complex graph structures has been heavily studied for many years. However, in recent years, the level of digitisation (conversion of material from the real world to a digital format) has increased by many orders of magnitude, leading to a greater prevalence of both data and data formats. This increased interaction and data gathering across services, such as social media and e-commerce providers, has produced enormous datasets that can no longer be easily processed as they are far too large to fit within the memory of a single system (McCune et al., 2015). By definition, data gathered from these sources is a network, usually represented as graphs. These graphs consist of vertices depicting individuals and edges representing relationships between said individuals. Additional features are generally associated with the vertices and are sometimes applied to edges. Furthermore, as these networks have grown in size and complexity, it has become more difficult to interpret exactly which features, or a combination thereof, represent any particular type of relationship between vertices.

This chapter will primarily focus on the aims of this thesis and provide an introduction into the field of Network Science.

1.1 Motivation

One of the greatest challenges often seen in various areas of computing is scalability – the ability of a system to cope at different scales. Large and complex datasets usually require significant computational resources for processing and analysis and as the size and complexity of the data increases, special measures need to be considered to ensure scalability. This issue is particularly prevalent when dealing with graphs.

Within both academia and industry, graphs have been widely used to provide a structure for encapsulating complex relationships present within data (Milenković et al., 2008). This has led to the development of a large number of graph specific analysis methods (Berkhin, 2006). These methods are used across a wide range of applications and aim to extract structural and spacial information from the topology of a graph.

Existing approaches (Berlingerio et al., 2012), (Koutra et al., 2011), commonly used for determining clusters and community memberships often rely on a global view of the graph. With the ever-growing size of these datasets, holding this global view in memory is becoming increasingly difficult (Papadopoulos et al., 2012). There are still large knowledge gaps in existing work related to determining clusters or community membership, without maintaining an expensive global view of the data. Likewise, in how to efficiently capture relationships or similarity without maintaining this expensive global view of the data.

1.2 Overview

Current approaches to determining the memberships of clusters based on the nodes and edges within networks rely on simplistic models, such as undirected graphs, of the systems that they represent (Yano & Wadayama, 2011). Although there are many algorithms for network analysis, currently the underlying data these tech-

niques are applied to is incomplete. An example of this is observed in situations where undirected or weighted graphs are used to represent complex networks. When constructing network models from raw data (ingestion), much of the data is discarded or only considered in a very superficial way. Weighted networks, where some value of importance is given to each edge, are a prime example of this as all interconnections can only be represented by a weight representing some global network factor. This means that criticality studies are generally flawed as the connections between network components have no context and there is no way to follow a class of edges with common attributes through a network. Network analysis could be greatly improved with the inclusion of attributes for nodes and edges. This would allow the contextual analysis of network structures, considering multiple inter-node relationships, which would enable greater confidence in studies of criticality regarding not only nodes but the interconnecting edges.

1.3 Aims

The overall aim of this work is the development of advanced techniques for identifying similarities between graphs and common patterns within a graph, primarily utilising machine learning.

In order to achieve this, the following avenues were explored:

- Objective 1 - Development of a simple mechanism for ingesting graph data from its many forms. To address this SemNetConn was developed, as discussed in Chapter 3.
- Objective 2 - An investigation of the optimal ways for comparing the similarity between graphs, evaluated in Chapter 4.
- Objective 3 - Chapter 5 covers an exploration of the optimal ways for identi-

fying sub-graphs with common features within a larger graph or, more simply, Community Detection.

- Objective 4 - Given that graph analysis of any kind often comes with intensive computational and memory requirements, an evaluation of whether it is possible to reduce the amount of computing resources needed for processing large graphs is highly important. This is discussed in Chapter 6.

1.4 Thesis Structure

A background to Network Science is presented in Chapter 2 along with the core related work for this thesis. The issue of how to construct graphs from a disparate range of input sources is discussed in Chapter 3, including a discussion of the developed tool SemNetCon, which is designed to ingest graph descriptions provided in various formats and export them in standard formats required for machine learning. Chapter 4 presents work on how a representation of a graph (known as an embedding) can be easily constructed along with how this embedding can be used for clustering graphs. Additionally, techniques for using neural networks to identify communities within a graph are presented in Chapter 5. The computational and memory costs of using deep learning on graphs can be very high – especially as graph size increases – hence an evaluation of the use of mixed-precision computations to reduce this is presented in Chapter 6. Overall conclusions and a discussion of the possible directions for future work are presented in Chapter 7.

1.5 Research Contribution

The work and contributions outlined in this thesis will focus on four primary areas.

1. To develop an intuitive method for the construction of uniform graph datasets

from a range of sources.

2. To develop and evaluate methods for scalable comparison of graph structures.

3. To develop and evaluate methods for scalable machine learning methods for identification of communities within graph structures.

4. An evaluation of the benefits of mixed-precision neural networks.

1.6 Publications

The following works have been published or are under review and have been completed during the course of study for this work:

- **John Brennan**, Stephen Bonner, Amir Atapour-Abarghouei, Philip T Jackson, Boguslaw Obara and Andrew Stephen McGough. Not Half Bad: Exploring Half-Precision in Graph Convolutional Neural Networks. Under review in the Fourth IEEE International Workshop on Benchmarking, Performance Tuning and Optimization for Big Data Applications (BPOD), 2020.

- **John Brennan**, Stephen Bonner, Georgios Theodoropoulos, Andrew Stephen McGough, and Boguslaw Obara. Semantic Enabled Python Tool For The Construction of Complex Networks From Disperse Data Sources (SemNetCon). Under review in the Journal of Open Research Software, (2020).

- Stephen Bonner, **John Brennan**, Ibad Kureshi, Andrew Stephen McGough, and Georgios Theodoropoulos. Efficient Comparison of Massive Graphs through the Use of 'Graph Fingerprints'. In the KDD Workshop on Mining and Learning with Graphs (MLG), 2016.

- Stephen Bonner, **John Brennan**, Georgios Theodoropoulos, Ibad Kureshi, and Andrew Stephen McGough. GFP-X: A Parallel Approach to Massive

Graph Comparison using SPARK. In the IEEE International Conference on Big Data, pages 3298–3307, 2016.

- Stephen Bonner, Amir Atapour-Abarghouei, Philip T Jackson, **John Brennan**, Ibad Kureshi, Georgios Theodoropoulos, Andrew Stephen McGough, and Boguslaw Obara. Temporal neighbourhood aggregation: Predicting Future Links in Temporal Graphs via Recurrent Variational Graph Convolutions. In the IEEE International Conference on Big Data, 2019.

- Stephen Bonner, Ibad Kureshi, **John Brennan**, Georgios Theodoropoulos, Andrew Stephen McGough, and Boguslaw Obara. Exploring the Semantic Content of Unsupervised Graph Embeddings: An Empirical Study. In Data Science and Engineering, 4(3):269–289, 2018.

- Stephen Bonner, **John Brennan**, Ibad Kureshi, Georgios Theodoropoulos, Andrew Stephen McGough, and Boguslaw Obara. Temporal Graph Offset Reconstruction: Towards Temporally Robust Graph Representation Learning. In the IEEE International Conference on Big Data, pages3737–3746. IEEE, 2018.

- Daniel Justus, **John Brennan**, Stephen Bonner, and Andrew Stephen McGough. Predicting the Computational Cost of Deep Learning Models. In the IEEE International Conference on Big Data, pages 3873–3882. IEEE, 2018.

- Andrew Stephen McGough, Matthew Forshaw, **John Brennan**, Noura Al Moubayed, Stephen Bonner. Using Machine Learning to Reduce the Energy Wasted in Volunteer Computing Environments. In the Ninth International Green and Sustainable Computing Conference (IGSC), 2018.

- Stephen Bonner, Ibad Kureshi, **John Brennan**, and Georgios Theodoropoulos. Exploring the Evolution of Big Data Technologies. In Software Architec-

ture for Big Data and the Cloud, pages 253–283. Elsevier, 2017.

- Stephen Bonner, **John Brennan**, Ibad Kureshi, Georgios Theodoropoulos, Andrew Stephen McGough, and Boguslaw Obara. Evaluating the Quality of Graph Embeddings via Topological Feature Reconstruction. In the IEEE International Conference on Big Data, pages2691–2700. IEEE, 2017.

- Stephen Bonner, **John Brennan**, Georgios Theodoropoulos, Ibad Kureshi, and Andrew StephenMcGough. Deep Topology Classification: A New Approach for Massive Graph Classification. In the IEEE International Conference on Big Data, pages 3290–3297. IEEE, 2017.

- Stephen Bonner, Andrew Stephen McGough, Ibad Kureshi, **John Brennan**, Georgios Theodoropoulos, Laura Moss, David Corsar, and Grigoris Antoniou. Data Quality Assessment and Anomaly Detection via Map/Reduce and Linked Data: A Case Study in the Medical Domain. In the IEEE International Conference on Big Data, pages 737–746. IEEE, 2015.

Chapter 2

Background and Related Work

2.1 Problem Domain

This thesis aims to investigate the development of algorithms, methodologies, and modelling strategies to reliably and efficiently identify similarities between graphs and the nodes within them. This problem is of particular interest as it is becoming increasingly difficult to process internet scale datasets, such as the Facebook social graph, to identify communities or community membership. This will require robust modelling of complex networks and reliable algorithms for the determination of which nodes and edges within a network have relationships that are not immediately observable.

Graphs, which represent a number of entities (referred to as vertices or nodes – without loss of generality the term node will be used in this thesis) and the links between these entities (referred to as edges or links – without loss of generality these will henceforth be referred to as edges), have become an indispensable tool for analysis of data across many disciplines including social sciences, security and medicine. A graph $G = (V, E)$ is defined as a set of nodes V, with a corresponding set of edges E. E is composed of unordered tuples (u, v) where $u, v \in V$. Their

ability to represent the links between different entities makes them a more natural representation for tasks such as registering relationships between different entities (edge prediction) than other data representation formats.

While one might use unordered tuples to represent the edges of a given graph, a graph could potentially contain ordered tuples, where an edge can only be traversed in one direction. Thus it is possible to move on the edge from u to v but not along the edge from v to u – here it is said that $(u, v) \neq (v, u)$. In the case of ordered tuples, it is said that the edges are directed, conversely, for unordered tuples the edges are undirected and represented as $(u, v) \Leftrightarrow (v, u)$.

2.2 Network Science

The basis of network science is fundamentally grounded in graph theory. Graph theory began with Leonhard Euler's solution to the Seven Bridges of Kõnigsberg problem (Euler, 1741). The problem was to determine a path which would cross each of the seven bridges separating the four landmasses within the city once, and only once. Euler's solution was to take a simple topological approach to the problem which along with his solution gave rise to modern graph theory (Biggs et al., 1976).

The order of a graph, defined as the number of nodes, is represented by $|G|$. Within a graph edges can be considered as directed or undirected. That is to say where an edge is directed the 'flow', of whatever property is represented by the edge, can only travel in one direction. Conversely an undirected edge is considered to be a bi-directional connection (Bollob'as, 1998). Graphs are generally studied in terms of paths. The path P is given by the form $V(P) = \{x_o, x_1, ..., x_n\} \in V$ which represents a set of edges forming a path between x_0 and x_n (Bollob'as, 1998). Graph theory also allows for the existence of hypergraphs. A hypergraph occurs when there exists hyperedges, which are defined as a set of edges which can connect

any number of nodes together. Hyperedges could be used to represent product flows within a business supply network, for example, n businesses connected to each other by virtue of being intermediaries, in the supply chain, for a common product (Newman, 2003). Hyperedges are essentially defined paths within a network that have a fixed route based on node function.

Network science takes the graph theory paradigm as a basis and becomes a broad-ranging field which utilises a number of theories and methods from many disciplines. The foundation of this field is built upon graph theory, but is extended with the implementation of theories from mathematics, statistical mechanics, data mining and information visualisation, statistical analysis and sociology (Foltz et al., 2005). Brandes *et al.* (2013) state that network science is "exploding" and is a field which penetrates a broad range of traditional disciplines.

Networks exist everywhere in modern life. These can take many forms, such as: natural networks, infrastructure networks, socioeconomic networks, network decision or control systems. Networks fall into two categories natural and engineered. Natural networks are those that occur spontaneously and can potentially contain agents that act selfishly or myopically, such that these agents may behave in a way that is detrimental to the overall network. These types of networks are usually studied in terms of computational modelling, developmental and predictive analysis in order to explain certain phenomena or to gain other insights. Engineered networks are those which are artificially made and can contain agents, that also have the potential for myopic behaviour but this is unlikely in a well-designed system, programmed with objective-based behaviour.

Engineered networks are usually considered in terms of design. How should the nodes within the network be programmed/designed to behave in order to achieve a specific goal? The goal could be anything within the realm of the network function,

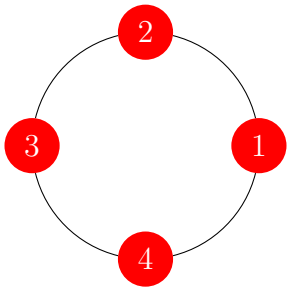


Figure 2.1: Simple Network

such as achieving suitable quality of service within a computer network, be it a dynamic (subject to a sequence of modifications) or a static environment. However in reality this distinction is not so clearly defined and the designed behaviour of engineered networks can be influenced by the actions of unpredictable agents, such as human users within a computer network. Also a network that originally evolved as natural could be influenced or controlled by an engineered component. Such as a social network provider suggesting 'friendships', based on user analysis, which may never have occurred naturally (Strogatz, 2001).

In order to analyse any kind of network quantitatively mathematical models need to be constructed. One way of doing this is by utilising an adjacency matrix. Consider a network G which consists of n nodes labelled 1 to n. Using Figure 2.1 as an example, an edge list can be generated to describe the network. This example has $n = 4$ nodes and an edge set of (1,2),(1,4),(2,3),(3,4). A better way to represent this edge list is as an adjacency matrix A with elements A_{ij}, where A_{ij} denotes the number of vertices between nodes i and j, described as:

$$A_{ij} = \begin{cases} 1 & \text{if at least one edge exists between nodes } i \text{ and } j, \\ 0 & \text{otherwise.} \end{cases} \tag{2.2.1}$$

This enables the construction of:

$$A = \begin{vmatrix} 0 & 1 & 0 & 1 \\ 1 & 0 & 1 & 0 \\ 0 & 1 & 0 & 1 \\ 1 & 0 & 1 & 0 \end{vmatrix}, \tag{2.2.2}$$

to fully describe the current example in a mathematical model which will form, in general, a sparse matrix. Note that this example is an undirected network and therefore the matrix is symmetric along the leading diagonal. This would not be the case for a directed network as adjacency would only exist in the direction of travel. Adjacency matrices can also be used to represent multi-edge and weighted networks by expanding the number of values available for A_{ij} beyond the simple binary system.

2.3 Network Metrics

Here within is a brief overview of the most commonly used measures and metrics for network analysis taken from the work by Newman (2010):

- **Degree Centrality** is the measure of node degree as an integer value for the number of edges connected to it. This concept is extended slightly in directed networks and is divided into in-degree and out-degree.

- **Eigenvector Centrality** adds a weighting to degree centrality based on the centrality of the node neighbours. With the aim to better identify influential nodes within the network.

- **Closeness Centrality** is a measurement of the mean distance between a node and all other nodes in the graph. If multiple paths exist between two nodes this method will only consider the shortest, or geodesic, path.

- **Betweenness Centrality** measures the degree to which a node lies on paths, or hyperedges, between other nodes. This can be used to find the important intermediary nodes within a network.

- **Groups of Nodes** describe distinct communities of nodes. This is highly important in the analysis of social networks.

- **Transitivity** is particularly useful to social networks, it helps to facilitate inference of extended friendships. It relies on the mathematical notion that a relation "$\oplus$" is said to be transitive. For example if $a \oplus b$ and $b \oplus c$, then it is implied that $a \oplus c$.

- **Similarity** is how similar one node is to others within the network, there are however a great many methods of specifying similarity including structural properties or values of other metrics such as centrality.

- **Components**. If a graph is fully connected i.e. it has, at least, one edge connecting every pair of nodes, then it only has a single component. Where some connections are absent each disjoint sub-graph is considered a component.

- **The Small World Effect** the measurement of the observed phenomena that within large natural networks the shortest path between two nodes is, on average, extremely small relative to the size of the network.

- **Degree Distributions** is one of the most commonly used network properties and describes the frequency distribution of node degrees. This can be used to demonstrate whether networks follow power-law or scale-free distributions, allowing a determination of whether the network is random or has some underlying structure (Barabási & Bonabeau, 2003).

2.3.1 Global Features

Here, those graph features that are considered to be global will be outlined. A global feature is a summary statistic for an entire graph, generally derived from a sum or mean of some localised feature.

- **Graph Order** - Defined as: $|V|$.
- **Number of Edges** - Defined as: $|E|$.
- **Number of Triangles** - The number of triangles, α, for a given graph is the number distinct sub-graphs which consist of three nodes and three edges where each node has a total degree of two, ignoring any edges that connect to a node outside the sub-graph.
- **Global Clustering Coefficient** - This is a measure of how connected the graph is overall. A triplet is a set of three connected nodes that can either be a triangle or, if not fully connected, a potential triangle, the total number of triplets is given by β. A measure of how interconnected a graph is can be computed as: $gc = 3(\alpha/\beta)$, this gives a measure of how interconnected a graph is.
- **Maximum Total Degree Value** - This represents the total number of edges the most connected node in the graph has to other nodes, irrespective of direction.
- **Connected Components** - This is the total number of components within the graph. A component is defined as a set of nodes, or sub-graph, where any selected pair of nodes can be connected through a set of edges within the component, but will not have a path to any node in another component.

2.3.2 Node Level Features

The aim of Chapter 4 is to identify the subtle differences between graph topology. In order to achieve this a number of features are evaluated for each node within the graph. Through experimentation, it has been determined that the following seven feature metrics give the best balance between structural sensitivity and computational complexity. For each of the six node features detailed in Table 2.1, a value is evaluated for each $v \in V$.

Table 2.1: Node Level Features

Feature Name	Comment	Label	Equation	Source
Eigenvector Centrality	λ is the largest eigenvalue, A is the graph adjacency matrix and x is the eigenvector centrality.	Ax	$Ax = \lambda x.$	(Bonacich, 2007)
PageRank Score	N is the number of nodes, $\Gamma^-(v)$ is the set of incoming neighbours of v, $d^+(u)$ is the out-degree of u and d is a damping factor (0.85 for this work).	$PR(v)$	$PR(v) = \frac{1-d}{N} + d \sum_{u \in \Gamma^-(v)} \frac{PR(u)}{d^+(u)}$	(Page et al., 1998)
Total Degree	Sum of the in & out degree of node v.	td_v	$td_v = \Gamma^-(v) + d^+(v)$	(Bonner et al., 2016)
Two-Hop Away Neighbours	$N(v)$ is each node incident on node v.	th_v	$th_v = \frac{1}{\lvert N(v) \rvert} \sum_{\forall j \in N(v)} d^+(j)$	(Berlingerio et al., 2012)
Local Clustering Score	Φ is the number of pairs of $v's$ neighbours which are themselves connected.	c_v	$c_v = \frac{2\Phi}{d^+(v)(d^+(v)-1)}$	(Watts & Strogatz, 1998)
Avg Neighbour Clustering	c_j is the local clustering score.	nc_v	$nc_v = \frac{1}{\lvert N(v) \rvert} \sum_{\forall j \in N(v)} c_j$	(Berlingerio et al., 2012)

2.4 Classic Community Detection Algorithms

General overview - Communities are simply groups of elements in a network where members are more likely to interact with each other than with elements outside the group. In some small networks, communities are easy to spot visually. For others, you need complex calculations to help tease out the community lines.

2.4.1 Bipartitions

The Kernighan–Lin algorithm (Kernighan & Lin, 1970) is a heuristic graph partitioning method. For input, the algorithm accepts an undirected graph $G = (V, E)$ where V is a finite set of vertices and E is a set of edges, with optional weights W for each edge in E. This algorithm aims to split V into two disconnected sets, A and B, of, ideally, equal size. In the case where G is weighted the algorithm attempts to minimise $T = \sum W(b)$, where b is the set of edges that cross the boundary of A and B. Where the graph is un-weighted $T =\mid E(b) \mid$ is minimised instead, with $\mid E(b) \mid$ being the count of edges that cross the boundary. The algorithm iteratively improves the partitioning by greedily pairing vertices with A with vertices from B such that the move between partitions will improve the balance of the partitions. After this step further pairs are chosen in order to best minimise T. The complexity of this algorithm can be described by $O(n^2 \, log \, n)$, for a graph with n vertices.

2.4.2 Cliques

Within graph theory, a clique C is described as $C \subset V$ taken from an un-directed graph $G = (V, E)$. Cliques can be further categorised into the following categories. A maximal clique is a clique in which further adjacent vertices can not be added without that clique becoming wholly a subset of a larger clique. The maximum clique of a graph G is the singular clique in a graph which has the largest number

of vertices. A k-Clique is a clique consisting of k vertices contained within a fully-connected sub-graph (Palla et al., 2005). Finding cliques within a graph is a non-trivial problem, indeed, the process of finding all cliques within a graph has been proven to be NP-Complete (Karp, 1972).

2.4.3 Modularity-based communities

The Clauset-Newman-Moore (Clauset et al., 2004) algorithm is a bottom-up hierarchical approach that attempts to identify communities within graphs using fast greedy modularity maximisation and is an extension of the work by (Clauset et al., 2004). Greedy modularity maximisation begins with each vertex v in the graph G having singular membership of its own community. Following this, pairs of communities that represent the greatest sum of modularity are joined in an iterative process until no further joins exist that meet the condition. Modularity Q is defined as the proportion of edges that fall within a subgraph minus the assumed proportion if all edges were distributed at random. More formally this can be represented as:

$$Q = \sum_{c=1}^{n} \left[\frac{L_c}{m} - \left(\frac{k_c}{2m} \right)^2 \right], \tag{2.4.3}$$

where c is a community, the number of edges is represented by m, the count of intra-community links for community c is L_c and the sum of all vertex degrees within c is k_c.

2.4.4 Label propagation

The asynchronous label propagation algorithm proposed by (Raghavan et al., 2007) is able to isolate communities in near-linear time. It is a probabilistic method and the communities that are found may vary on different executions of the algorithm. The algorithm proceeds as follows. After initialising each vertex v with an exclusive

community C, the algorithm repeatedly sets the community of a vertex to be the community that has the highest incidence among the neighbours of the vertex currently being considered. The algorithm finishes when every vertex has been labelled according to the community membership of its peers. As each vertex is updated without consideration of other vertex updates the algorithm can be considered as asynchronous and is well suited to parallelisation.

2.4.5 Fluid Communities

Another asynchronous method for community detection has been proposed by (Parés et al., 2018). The authors of this paper present the fluid communities algorithm that is based on the concept of fluid dynamics. Where vertices interacting within in an environment, or graph, expand and exert force on one another. The algorithm uses a random initialisation, assigning each vertex a community from an initial set of k communities. The algorithm then iterates in a similar fashion to label propagation in that community membership is assigned based on neighbourhood membership. However, there is a further condition in this approach in that total community density must always remain at 1. The density with respect to a graph, or sub-graph, is defined as $D = \frac{|E|}{|V|}$. When any vertex moves between communities all densities of all affected communities are updated immediately. The algorithm completed when during a single iteration no vertices change community membership.

2.5 Clustering Approaches

Graph Clustering (GC) is a sub-set of graph theory concerned with finding sets of nodes, within the same graph, that share some form of 'community' (Schaeffer, 2007). There are many methods for GC, some notable examples are as follows: The Modularity method, proposed by Newman (2004), is a system for assigning

weights to nodes/edges based on some implementation-specific criteria. Shiokawa et al. (2013) demonstrate a fast modularity based approach (SFA) to this problem where the density of edges is used to evaluate what cluster a node belongs to. SFA is considered to be the current state of the art approach for modularity based clustering having the ability process web-scale graphs in $O(|E| - cn|V|)$ time, where c is the clustering coefficient and n is the number of neighbours for each adjacent node. Saltz et al. (2015) use a distributed Weighted Community Clustering (WCC) approach which is derived from the number and distribution of triangles within a graph, where a triangle is a set of three nodes where each node has a degree of two ignoring any edges that leave the set. WCC based approached have been shown to provide clusters that are more densely connected than those using modularity based approaches Prat-Pérez et al. (2012). Using these methods to group distinctly separate graphs would not be possible as they do not consider cluster similarity.

2.5.1 Spacial Clustering

Spacial clustering (SC) algorithms are predominately considered to belong to one of two groups, Hierarchical clustering and Partitional clustering (Berkhin, 2006).

Hierarchical clustering: algorithms are either Divisive (bottom-up) or Agglomerative (top-down) Tan et al. (2005). Divisive algorithms initially treat each node as a single cluster and then iteratively merges pairs of clusters until all clusters have been merged into a single cluster that contains every node. Throughout this process information of each merge is maintained so an entire hierarchy can be constructed. Agglomerative clustering begins with a single cluster that contains every node and then splits clusters recursively until each individual node is considered a single cluster. Within Divisive and Agglomerative algorithms the decision of which clusters to merge or split is determined by calculating an appropriate distance metric for each pair of nodes within a cluster and selecting the most favourable outcome, which will

be a minimum or a maximum for a merge or split respectively (Ding & He, 2002). While any distance metric is valid, Euclidean distance is one of the most commonly used (Madhulatha, 2012).

Partitional clustering: algorithms typically determine all clusters at once but can also be Divisive, as in hierarchical clustering. These generally use the same metrics that are employed by hierarchical. However, in this case, the nodes are separated into a collection of non-overlapping subsets, where each node is a member of exactly one subset. Conversely to hierarchical methods, each of these subsets have no defined relationship to one another (Tan et al., 2005).

2.5.2 Node Classification

Li et al. (2011) propose a topological and label feature-based approach as a computationally efficient method for classification. The limitations of this method are that it has a very narrow domain focus and is dependent on graph databases. Additionally some graph features, such as eccentricity (the maximum distance between a node to all other nodes) and shortest path (the smallest path from given source to any other node) are computationally expensive. Therefore this approach does not scale well when considering very large graphs, such as those considered in this work.

A boosting based multi-graph classification framework has been proposed by Shun et al. (2016). Boosting aims to sort multiple sub-graphs into 'bags', each of which has been manually assigned a static label or class. The evaluation and labelling of sub-graphs on a very large dataset is a non-trivial task. The authors claim to mitigate this factor by making correlations between such things as keywords, similar to those found in scientific publication networks. However, this mitigation requires a level of domain-specific knowledge, something that is not present in synthetic graphs, and the overhead required to evaluate many sub-graphs limits the scaleability of the boosting approach.

2.6 Neural Network Approaches

In the field of Deep Learning, supervised learning is probably the most studied and understood (Goodfellow et al., 2016). In supervised learning, the datasets contain labels which help guide the model in the learning process. In the field of graph analysis, these labels are often present at the vertex level and contain, for example, the meta-data of a user in a graph representation of a social network.

2.6.1 Graph Convolutional Networks

Possibly the most heavily researched area of supervised graph embeddings is that of *Graph Convolutional Neural Networks (GCNs)* (Bruna et al., 2013), both spectral (Defferrard et al., 2016) and spatial (Niepert et al., 2016) approaches. These approaches use a sliding window filter over an entire graph. This approach is similar to the approach used in Convolutional Neural Networks (CNN)s (Goodfellow et al., 2016) from the computer vision field. The primary difference being that the neighbourhood of a node represents the sliding window as opposed to adjacent pixels. Current GCN approaches are supervised and require that nodes are labelled. This precondition has two notable drawbacks. First, any dataset used for training must have fully labelled nodes in order for it to be used. Second, any embeddings generated using this approach are unlikely to generalise to other datasets meaning re-training would be needed if inference were to be required on a node from a different network.

2.6.2 Inductive Representation Learning

Before the GraphSAGE (Hamilton et al., 2017a) approach was presented, most node embedding models were based on spectral decomposition and matrix factorisation methods. The GraphSAGE authors argue that this is a problem because these methods are transductive and such methods do not perform well when being tested on unseen data. Such that the entire graph needs to be visible for training and if any additional nodes are added the whole thing will need to be re-trained. GraphSAGE is capable of generating embeddings for a previously unseen node, without requiring retraining of the model. In order to achieve this GraphSAGE learns aggregation functions, using information from neighbouring nodes, that can be applied to a new node, given its features and neighbourhood are present. This is referred to as inductive representation learning. The baseline GraphSAGE neighbourhood method is to use nearest neighbours with arbitrary additions/deletions to achieve consistent input size for the aggregation input. The types of aggregation function presented in the paper for use with GraphSAGE are; Mean, Long Short Term Memory (LSTM) and Max Pooling.

2.6.3 Graph Attention Networks

Veličković et al. (2018) presented the concept of a *Graph Attention Network (GAT)*. GATs use the spatial information related to a vertex in order to learn an embedding which can be used for classification. This use of spatial information is what differentiates this method from GCNs which use a spectral approach, directly analogous to CNNs used in image processing. GATs use stacked attention layers which are able to attend over the features of their neighbours. This implicitly allows each node in a neighbourhood to have different weights without the need for any further matrix operations and without prior knowledge of the entire graph structure.

2.7 Reduced-Precision Neural Networks

With the growing popularity of deep neural networks and hence the increasing focus on training and inference efficiency, the use of reduced precision has received significant attention within the existing literature (Ginsburg et al., 2017; Micikevicius et al., 2018; Gupta et al., 2015; Hubara et al., 2017; Courbariaux et al., 2015; N. Wang et al., 2018). For instance, there have been attempts to binarise model weights and activations while gradient calculations are kept within the full-precision format (Hubara et al., 2017). In (Rastegari et al., 2016), even gradients are binarised along with all other tensors in order to improve training and inference efficiency in terms of both memory usage and run-time. However, despite their impressive computational efficiency, such approaches always lead to significant losses in accuracy with larger model architectures.

To resolve the issue of accuracy loss, the majority of the recent work has shifted towards using at least 16 bits for data and gradient computation. The approach proposed by Micikevicius et al. (2018) uses a 16-bit floating-point format accumulating results into 32-bit arrays and ensures gradients with a small magnitude are preserved via loss-scaling. Accuracy is also maintained in (Das et al., 2018; Köster et al., 2017) using a custom format with a 16-bit mantissa and a shared exponent to train large neural networks. Despite their promising performance, such approaches keep a 32-bit copy of the model weights to enable precise weight updates and partial products are accumulated in a 32-bit format.

For N. Wang et al. (2018), 8-bit floating-point numbers are used for both the numerical representation of the data and all the computations required for the operations involved in the forward and backward passes of the model. The approach outlined in by Mellempudi et al. (2019) enhances the use of 8-bit floating-point representation by compensating for the reduced subnormal range of 8-bit floating-point

representation for improved error propagation leading to better model accuracy.

It is important to note, however, that the use of any reduced-precision approach, such as those reviewed above, heavily depends on model size, input data modality and the nature of the task. Extensive exploration and benchmarking of various reduced-precision methodologies have been carried out for Convolutional Neural Networks and Transformer models for computer vision and natural language processing tasks (Micikevicius et al., 2018; Kuchaiev et al., 2018), whilst the use of neural networks for graph-based applications is not yet fully investigated. Consequently this work provides a detailed study of the use of mixed-precision operations using specialised hardware for graph convolutional neural networks.

2.8 Summary

This section has laid the foundation of graph theory and network science upon which this thesis will build. The concepts of global and node level features have been described with examples of such observations given. Also included here is a discussion of the approaches that have been most commonly used for community detection, including clustering and those methods adopted due to the emergence widespread neural network use. The following section will present the Semantic Network Constructor application for the creation of network datasets from disparate data sources.

Chapter 3

Network Construction

Network science is an interdisciplinary field which enables the studying of detailed real-world phenomena by viewing them as a series of connected components within a complex system. There are numerous examples of systems from across the spectra of scientific disciplines which are composed of individual elements linked together in some manner (Newman, 2010). Some examples of networks include the Internet - the emergent phenomena created by the global interconnection of computer systems, and human societies - the linking of humans via social interaction.

The field of network science can be defined as the study of the collection, management, analysis, interpretation, and presentation of relational data (Brandes et al., 2013). Networks are often constructed from real-world datasets and are a series of nodes, with pairs of nodes connected together via an edge. These edges can be undirected, to create what is known as a 'simple network', or directed edges, with implied direction between two nodes creating a 'directed network'. Additionally, two nodes can be connected via multiple edges, or an edge can connect a node to itself, creating a self-loop. Edges can have weights, often in the form of a numeric value. According to (Barrat et al., 2004), these networks are known as weighted networks and are used to embed a greater quantity of information within the structure of

a network. A weighted edge could, for example, represent the strength of a social connection between two people (nodes).

In addition to weights, other information can be included within the network in the form of attributes. These attributes can be attached to nodes, edges or even the overall graph itself and can embed a greater level of the context within the network. Network science uses concepts from graph theory to provide a mathematical basis from which to work, as such, the term graph and network are often used interchangeably (including in this thesis).

Networks are created from a disparate range of data sources from across the various academic disciplines. However, currently, there exists no tool which can convert the diverse set of possible source graph representation formats into a standard network file format. Several such standardised network formats exist and are designed to make networks portable between various software packages which can be used to study and visualise them. The lack of a standardised conversion tool for network construction means that users are often required to create their own custom file parsers before running any network analysis algorithms.

The creation of networks from disparate data sources is a complex problem, as in addition to extracting the vertex and edge information, users need to be able to attach numerous custom attributes to these elements. It is possible that each of these items of information will be stored in different files and potentially in different formats. To combat this problem, the Semantic Enabled Python Tool For The Construction Of Complex Networks From Disperse Data Sources (SemNetCon) application has been developed. SemNetCon streamlines the process of creating a complex network from a range of possible data sources and allows the inclusion of an unlimited number of custom attributes. SemNetCon is seamlessly able to construct a single network, including associated attributes, even if they are stored across a range of physical files and file formats. For example, a file describing the edges, stored as a CSV, could

be used to construct the base network, whilst a file containing node names, stored in JSON, could be used to attach a custom attribute to each node. As far as the author is aware, SemNetCon is the first software that implements this functionality.

3.1 Implementation

SemNetCon uses technologies from the semantic web stack to provide a structured description of features within the source dataset which enables easy conversion into a number of network file formats. The semantic web is a framework initially developed for metadata allowing contextual data to be embedded into a dataset, enabling a deeper understanding of the underlying data (Berners-Lee et al., 2001). The semantic web stack is implemented as a series of layers, the most important being the Resource Description Framework (RDF) (W3C, 2016a), SPARQL Protocol and RDF Query Language (SPARQL) (W3C, 2016b) and the Web Ontology Language (OWL) (Antoniou & Harmelen, 2008). As part of SemNetCon, a new OWL ontology has been created to describe the components required to create a network from a range of possible data sources. Figure 3.1 shows an overview of the ontology. This ontology can be used to create a custom RDF file for a given dataset, describing everything required for SemNetCon to convert the data into a standardised network file format.

In terms of end-user functionality, SemNetCon allows via the use of a graphical user interface (GUI), a user to answer a small number of questions about the content of their source dataset. A screen-capture of the SemNetCon's GUI can be seen in figure 3.2. Further details and examples of how to utilise the GUI are given in the readme located in the project's repository [1]. With this information, and the

[1] `https://github.com/grossular/SemNetCon`

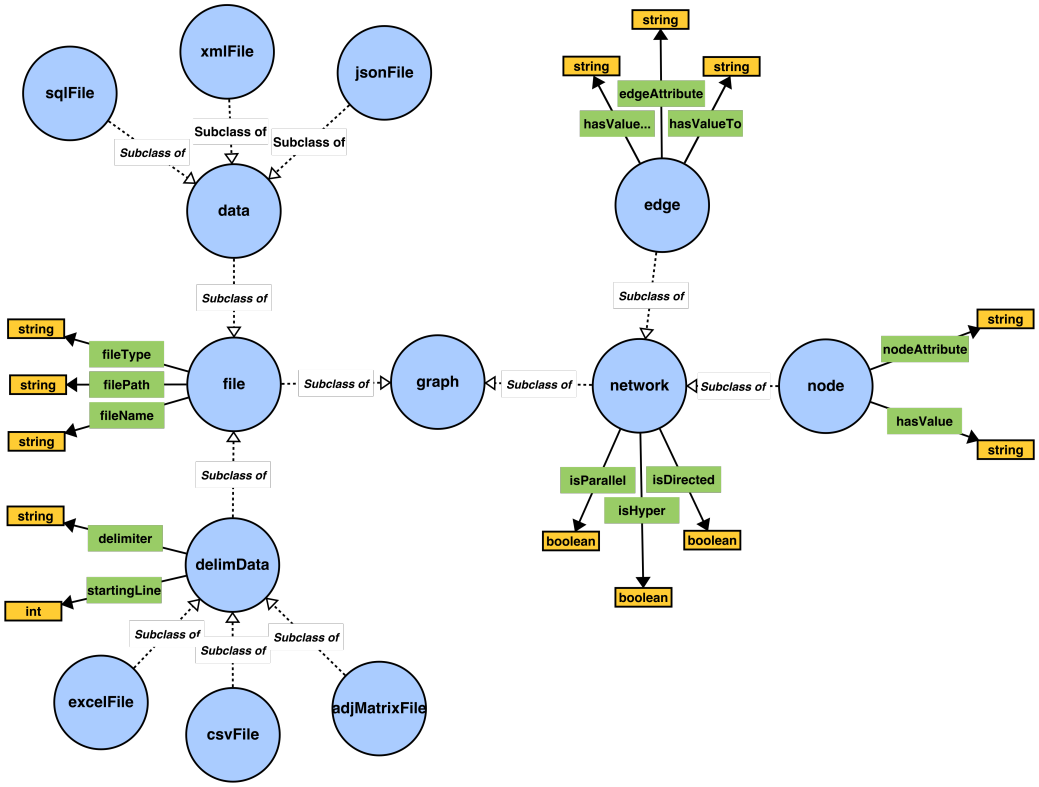


Figure 3.1: Diagram depicting a representation of SemNetCon's OWL Network Ontology

metadata structure provided by the OWL ontology, an RDF description of the data can be constructed. This description is structured and consistent across all datasets. The RDF for each piece of data can be considered as a logical funnel for SemNetCon parser functions as they now only need to extract very specific features. This enables the use of very generic file parsers where previously, potentially non-reusable, data specific parsers would have needed to be developed. Once the graph structure has been completed the resulting object is written back to disk in a standardised network format selected by the user. Currently supported output formats are; GML (Himsolt & Passau, 1996), GraphML (Eiglsperger et al., 2013), adjacency list (Blandford et al., 2004), multi-line adjacency list (Blandford et al., 2004) and Pajek (Batagelj & Mrvar, 2002).

SemNetCon currently has support for four input file formats; JSON, CSV, XML

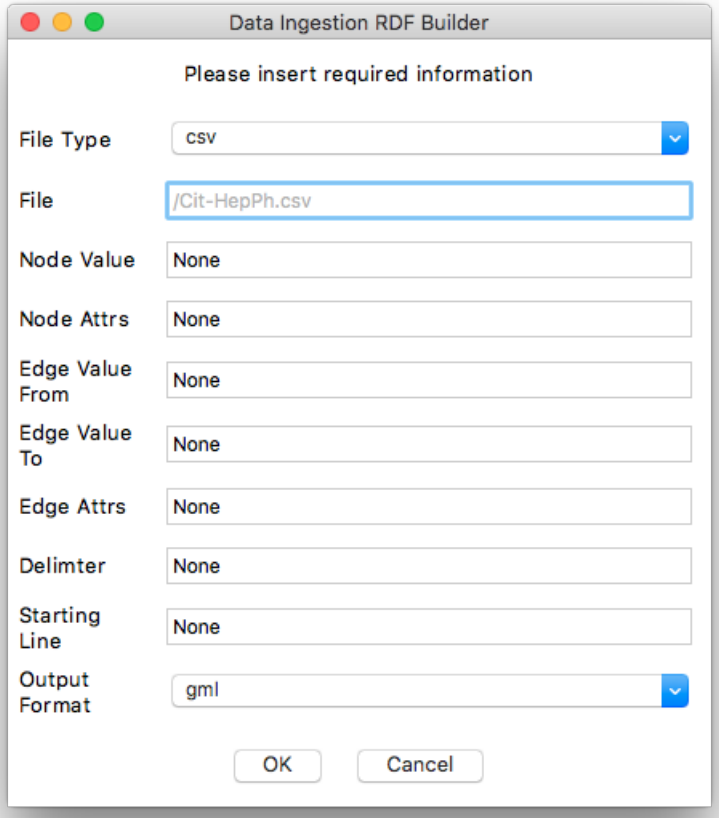


Figure 3.2: Diagram of SemNetCon's GUI

and Excel. The code has been written in such a way that users can easily implement an additional file parser for their required data format. End users are not required to create the required RDF by hand, instead, they are able to make use of the GUI. This allows the user to describe the features in their files which they would like to be represented as vertices, edges and attributes. The tool will then create the network and allow the users to save the generated RDF file to disk so that it can be used at a later date, bypassing the requirement for further invocation of the graphical tool. Overall SemNetCon is primarily designed to be used through the graphical interface, although users are able to use the command line interface if they already have a valid RDF file describing their data.

3.2 Architecture

SemNetCon is implemented in Python 2.7 and utilises many commonly available Python packages to provide efficient and portable file parsing and semantic web processing. For the semantic web components, such as RDF parsing, the *rdflib* package is used. The GUI is constructed using the Tkinter python package, this was chosen for portability. The software is implemented as three main Python classes, $FileParser$, $NetworkConstructor$ and $RdfBuildGUI$. The core class for the end-user is the $RdfBuildGUI$ which is used to launch an instance of the GUI. When interacting with the GUI, users are asked a series of questions about the dataset they are transforming into a network. The GUI then constructs a custom RDF file for the particular dataset, which is expressed against the OWL ontology. The GUI then calls the $NetworkConstructor$ class, passing to it the created RDF file. This class utilises SPARQL to query the passed RDF object and extract the encoded information. Depending upon the file type of the data source, the relevant parser is then called from the $FileParser$ class. The parser can then use the information passed to it to parse the data source and use the relevant features to construct the nodes, edges and their attributes required for the final network, as shown in Figure 3.3.

3.3 Objectives

SemNetCon has been designed to allow end-users to easily convert a given data source into a range of possible common network formats. In addition to this, it has been designed so that it can be extended by the research community to support new features and file formats. For example, if a new source data format is required, a user can simply create the required parsing method and place it in the $FileParser$ class. Further details of how to expand the SemNetCon codebase are given in readme of

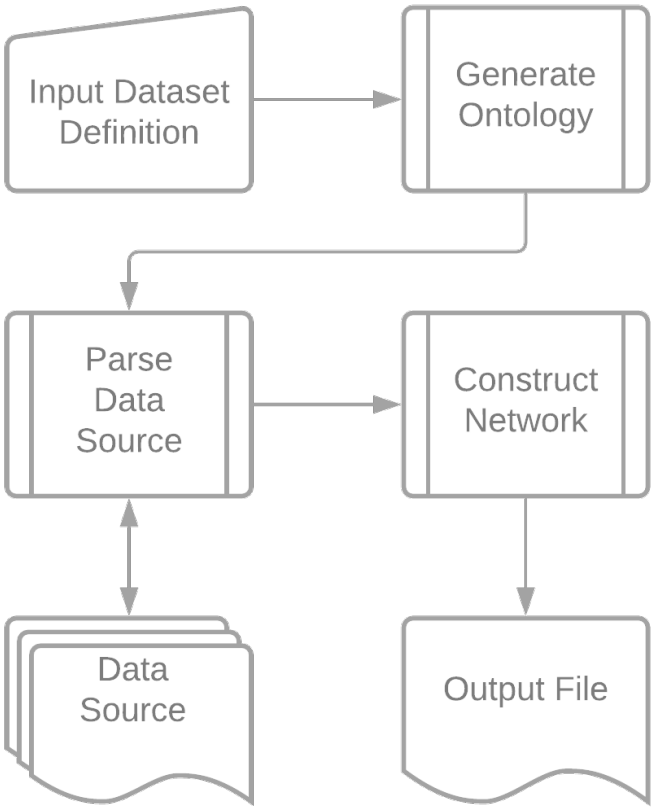


Figure 3.3: SemNetCon Overview

the code repository. The core objectives of the SemNetCon project are as follows:

- Create a formal ontology representing a generic graph structure using OWL.
- Generate a custom RDF file by describing the raw data to be converted into a network against the ontology.
- Leverage the RDF to enable generic parsers to construct topological representations of the data.
- Create a GUI to simplify the creation of the RDF for end users.
- Allow for easy expansion of the underlying OWL ontology and the list of file parsers so the tool can be applicable in a wide range of fields.

3.4 Quality control

Development and testing has been performed on OSX and various Linux distributions. Tutorials and sample datasets are provided with the source and are available on the main GitHub repository. The functionality of the software has been extensively tested using network datasets with known quantities from the Stanford Large Network Dataset Collection (Leskovec & Krevl, 2014), checking results for accuracy to ensure that a valid graph object has been created. SemNetCon is currently being utilised within the author's research group for the preparation of network datasets for analysis via machine learning. SemNetCon is proving to be very useful in creating a codebase for network analysis which can convert a large number of datasets from a selection of common file formats.

3.5 Availability

SemNetCon is available from `https://github.com/grossular/SemNetCon`.

3.5.1 Operating systems supported

SemNetCon has been designed and tested on various UNIX like platforms, specifically Mac OS X and Linux. As such it should work on any modern OS X version or Linux distribution. SemNetCon has been explicitly tested on the following platforms - Ubuntu 14.04, Fedora 22, CentOS 7 and OSX 10.11.

3.5.2 Programming language

SemNetCon is built using Python 2.7 and has not been tested on older versions of the Python 2 branch.

3.5.3 Dependencies

SemNetCon has the following dependencies, all of which are available via the python package manager pip -

Package Name	Min Required Version
NetworkX	1.10
jq	0.1.5
jsonpath_rw	1.4.0
lxml	3.4.4
rdflib	4.2.1
SPARQLWrapper	1.7.4
xlrd	0.9.4

Table 3.1: SemNetCon: Software Dependencies

3.6 Reuse potential

The analysis of complex networks is widely employed in many disciplines of science to map and measure relationships and flows between people, groups, organisations, computers, organic cells, and other connected entities. This software has been developed to allow researchers to construct useful network representations quickly and efficiently. The software had been developed to be modular, allowing it to be extended to read and process further file formats not considered by the author. This software can be useful across all domains where there is a requirement for constructing graph structures from non-graph native file formats.

3.7 Summary

This section has presented the first contribution of this thesis, SemNetCon. This open-source tool provides a simple interface allowing network information to be extracted from disparate data sources and stored in a choice of standard graph data formats. The rationale and implementation of this tool has been discussed here along with details of the architecture. Fundamentally, SemNetCon allows for easy construction, use and sharing of graph datasets. The following chapter will discuss how these datasets can be used to identify groups within the individual data points using feature extraction and spacial clustering.

Chapter 4

Embeddings and Spacial Clustering

The problem of categorising massive graphs accurately and efficiently by clustering them into their different types – be they social, biological, or technological – is an area of great interest within the field of complex network science.

Accurately clustering multiple massive graphs is computationally very expensive, and scales badly as the graph size, or the number of graphs, increases. This is because many problems involving graph traversal are NP-Complete. However, if it is possible traverse the graph in a linear manner to construct a feature set which can be evaluated through a single walk of a graph, it will be possible to create a simple proxy for the graph which is easier to analyse and scales linearly with the size of the graph.

In this work, a method is proposed for the clustering of complex networks, through the analysis of their Graph FingerPrint (GFP) feature vectors using spacial clustering algorithms. This method allows for the efficient clustering of massive, unlabelled, graphs. Three clustering algorithms are considered: Ward's hierarchical method, K-means++ and t-SNE. Graphs generated using the SNAP software pack-

age are used to evaluate the method. The t-SNE, spacial clustering, algorithm, is shown to be most effective at accurately identifying clusters even when considering graphs generated using algorithms designed to produce random graphs. Overall the GFP + t-SNE approach demonstrates 99% accuracy with very high stability.

4.1 Introduction

The Analysis of complex networks is widely employed in many disciplines to measure relationships and flows between people, organisations, computers or organic cells for purposes such as community (M. Wang et al., 2015) or anomaly (Akoglu et al., 2015) detection. These networks are difficult to understand and analyse either at system or unit level while maintaining the relationships that exist within them, often due to inherent interdependencies between components within the network (McCune et al., 2015).

The primary question addressed here is one of similarity, can topological features alone be used to determine if two graphs are of the same 'type'. It is possible to consider two distinct graphs have enough in common that they both belong to some shared class of graph, such as a shared generation method or similar social communities. There are multiple definitions of similarity when considering graphs (Berlingerio et al., 2012) (Koutra et al., 2011). These approaches can be separated into two groups, those which require labelled data, where each data point has some sort of meaningful 'label' associated with it, and those that do not. The work presented in this work falls into the latter category, using topological features alone as a basis for comparison in order to group graphs with comparable structures using Spacial Clustering (SC). SC considers objects within an n dimensional space, grouping objects which the algorithm considers similar into classes.

Another clustering approach often used in graph analysis is Graph Clustering

(GC). GC methods are concerned with identifying sub-graphs based upon relationships between individual nodes or discovery of common node features.

GC approaches rely on being able to touch the entire graph in order to determine [dis]similarity between sets of nodes. Something which is becoming increasingly difficult as web-scale graphs continue to grow. Many GC algorithms leverage random walks or sampling operations to minimise computational complexity (Shun et al., 2016). A random walk of length N is performed on a graph by first selecting a starting point V_i. From this point, a neighbour is randomly selected and 'walked' to and some form of metrics are gathered, a neighbour of this new node is then randomly selected to 'walk' to and this process is continued for N iterations.

Sampling is used where a graph is considered too large to be fully touched. Therefore a representative set of samples for the graph are derived. Using these methods to group distinctly separate graphs would not be possible as they give no consideration of cluster similarity.

In this work the author presents an approach to identify large complex network types by applying clustering methods to Graph FingerPrint (GFP) (Bonner et al., 2016) vectors. GFP components have been selected to produce meaningful metrics while minimising computation and memory footprint by gathering all information in a single traversal of each graph. This vector can be considered as an illustrative representation of a graph that, while still highly dimensional, can be analysed using unsupervised learning methods. The clustering methods evaluated are: Hierarchical; using the Ward method (Ward, 1963), K-means++ (Arthur & Vassilvitskii, 2007) and t-SNE (Van Der Maaten & Hinton, 2008). Using these methods it is possible group graphs into classes in order to identify which of these share common traits thus allowing the identification of a previously unseen graph. As far as the author has been able to determine there has been no prior work in the area of using unsupervised machine learning to identify the class of a graph from a given set of graph features.

SC, discussed in Section 2.5.1 is ideal for this work as each GFP provides a multi-dimensional spacial data point for each graph considered. The rest of the chapter is broken down as follows: Section 4.2 details the generation of Graph FingerPrints, Section 4.3 presents an overview of the spacial clustering algorithms used, Section 4.4 details the experimental data, Section 4.5 presents empirical results and in Section 4.6 conclusions are presented.

4.2 Generating Graph Fingerprints

The proposed approach for the clustering of graphs uses a Graph FingerPrint (GFP) embedding. This takes a high dimensional graph object and reduces the complexity down to a fixed-length vector. The GFP achieves this by extracting node-level and global features from a given graph, allowing it to capture both the macro and localised topological features.

A greater exploration of the features used in this clustering work can be found in (Bonner et al., 2016). Additional, or replacement, features can be used with this approach if required. Further investigations of features used in creating GFPs can be found in (Schaeffer, 2007) (Xu, 2005).

4.2.1 Global Features

The GFP method extracts multiple global features from a graph. These features were chosen to give a good overall representation of the graph structure, whilst also being computationally efficient. A total of six global features are evaluated for the GFP vector. These features were; graph order, number of edges, number of triangles, global clustering coefficient, maximum, total degree value and connected components. Each of these metrics are explained in Section 2.3.1.

4.2.2 Node Level Features

In order to make the GFP approach sensitive to subtle differences in graph topology, a number of features are evaluated for each node within the graph. Through experimentation, it has been determined that the following seven feature metrics give the best balance between structural sensitivity and computational complexity. For each of the seven node features detailed in Table 2.1, a value is evaluated for each $v \in V$. These features are further explored in Section 2.3.2.

4.2.3 Feature Creation

The matrix, $VF_{m,n}$, where $m = |V|$, contains all the node feature scores as defined in Section 2.3.2 respectively, and $n = |F|$ (F is the vector of features for each node):

$$VF_{m,n} = \begin{pmatrix} f_{1,1} & \cdots & f_{1,n} \\ f_{2,1} & \cdots & f_{2,n} \\ \vdots & \ddots & \vdots \\ f_{m,1} & \cdots & f_{m,n} \end{pmatrix} \tag{4.2.1}$$

To create the graph fingerprint, it is required to reduce the dimensionality of the matrix down to a more compact vector. This is achieved by taking a series of metrics for each of the columns in the matrix. The metrics chosen are median $\bar{x}_1$, mean Mo_1, standard deviation σ_1, variance σ_1^2, skewness $Skew[x]_1$, kurtosis $Kurt[x]_1$, minimum value $x(1)_n$ and maximum value $x(n)_n$. These are frequently used and well understood methods to capture the numerical variation within a range of values. These are concatenated into a single vector along with the global features described in Section 2.3.1. After this has been completed, the feature vector $\overrightarrow{a_{g1}}$ for graph G can be created as:

$$\begin{aligned}\overrightarrow{a_{g1}} = (&\bar{x}_1, Mo_1, \sigma_1, \sigma_1^2, Skew[x]_1, Kurt[x]_1, x(1)_1, x(n)_1 \\ &, ..., \bar{x}_n, Mo_n, \sigma_n, \sigma_n^2, Skew[x]_n, Kurt[x]_n, x(1)_n, x(n)_n).\end{aligned} \tag{4.2.2}$$

4.3 Clustering of Graphs

Spacial Clustering considers objects within an n dimensional space, grouping objects which the algorithm considers similar into classes. This allows us to leverage the GFP as a single multidimensional data point that can be grouped with similar points, allowing a determination of a class for a graph. For the purposes of this work, the ground truth class of a graph is the method, including parameters, used to generate it. The following clustering algorithms were evaluated during this work:

4.3.1 K-means++

The K-means algorithm clusters data by trying to separate n data points, that are members of the set X into k clusters $C_1...C_k$ of equal variance while minimising the inertia J, which is the distances between data points and their cluster centre for each cluster. This algorithm requires that the number of clusters (k) be specified. It scales well to large numbers of data points and has been used across a broad range of applications in many different disciplines such as document classification and image segmentation. The most popular K-means variant, known as Lloyd's algorithm (Lloyd, 1982), has a squared error function given by:

$$J = \sum_{j=1}^{k} \sum_{i=1}^{n} (||x_i^j - c_j||^2). \tag{4.3.3}$$

Where n is the number of data points, c_j is the centroid for cluster j and $(||x_i^{(j)} - c_j||^2)$ is the Euclidean distance between the ith sample in cluster j, x_i^j and c_j. Each cluster centroid is initially randomly selected and each data point is assigned to an appropriate c based on closest centre according to the Euclidean distance function.

The position of c_j is then recalculated to c'_j based on the mean value of all points, i, assigned to it:

$$c'_j = \frac{1}{|c_j|} \sum_{i=1}^{c_i} x_i^{(j)}, \tag{4.3.4}$$

Once each c'_j has been calculated, the distance between each data point and the new obtained cluster centres is calculated and the data points are assigned to their closest c'_j. If no data points were reassigned then the algorithm finishes.

Lloyd's algorithm has an average complexity of $O(kni)$, where i is the number of iterations. This can be improved when more consideration is given to the initial placement of centroids. K-means++ (Arthur & Vassilvitskii, 2007) randomly initialises the first centroid and then initialises all other c_j to be collocated with a chosen x_i. Each point is selected by using weighted probability distribution where a point x_i is chosen with probability proportional to the squared distance between x_i and the nearest decided centroid. This leads to vastly improved results, and complexity of $O(\log k)$ - Competitive with the optimal K-means (Arthur & Vassilvitskii, 2007).

K-means does not work well for high dimensional data (Jolliffe et al., 2011), due to the curse of dimensionality (Wilcox, 1961) that states, as the number of dimensions grow data becomes more sparse. Therefore the amount of data required to give a statistically significant result often grows exponentially with the dimensionality. This problem can be mitigated by first performing some function on the data to compress the number of dimensions that are present. One of the most popular forms of which is Principal Component Analysis (PCA). PCA is a linear dimensionality reduction method that provides a low dimensional vector representation of a given high-dimensional observation (Tipping & Bishop, 1999) (Jolliffe, n.d.). This is achieved through using statistical analysis to identify the amount of variance within each dimension. The algorithm identifies the dimension with the highest variance and rotates the feature space such that this is the first dimension. It then identifies

the dimension with the next largest variance and rotates the feature space such that this is the second dimension. This continues until all dimensions have been evaluated. Once the data is ordered in this fashion the algorithm can 'safely' discard some of the higher-order dimensions with low variance.

4.3.2 Hierarchical Clustering

The Hierarchical Clustering (Müllner, 2011) method aims to minimise the total within-cluster variance, which is the sum of all pairwise distances within a cluster used as a measure of compactness. While any valid distance metric can be used Euclidean distance is the most common. This is achieved by initially defining a set of clusters, called a forest, that are not yet included in the hierarchy. At each iteration the pair of clusters, s and t, that have the lowest sum of inter-cluster variance are found. These clusters are then merged into a new cluster u. At this point u is added to the forest, while removing s and t. The algorithm finishes when a single cluster remains in the forest, which becomes the root of the hierarchy. Using Ward's minimum variance method (Ward, 1963) pairwise distances are calculated at each iteration by:

$$d(u,v) = \sqrt{\frac{|v|+|s|}{T}d(v,s)^2 + \frac{|v|+|t|}{T}d(v,t)^2 - \frac{|v|}{T}d(s,t)^2}, \tag{4.3.5}$$

where $d(u,v)$ is the inter-cluster distance, v is an unused cluster in the forest and $T = |v|+|s|+|t|$ and $|*|$ is the number of points in the set. The general hierarchical clustering method used in this work has a complexity of $O(n_2 \log n)$ (Zhao & Karypis, 2002).

4.3.3 t-SNE

t-Distributed Stochastic Neighbour Embedding (t-SNE) (Van Der Maaten & Hinton, 2008) is an optimisation of Stochastic Neighbour Embedding (SNE) (Hinton &

Roweis, 2002). The aim of the SNE based approaches is to capture the structure of high dimensional data and create a low, usually two, dimensional 'map' which maintains as much of the significant structure from the original data as possible. This is achieved by first converting Euclidean pairwise distances of the original data into conditional probabilities, $p_{j|i}$ indicating the probability that datapoint i would pick datapoint j as its neighbour, in order to represent similarities. Given n data points $\{x_1, ..., x_n\}$ the probability that x_i would be a neighbour of x_j is based on a probability density function under a Gaussian, centred at x_i. The variance of the Gaussian for each datapoint is evaluated based on the ideal number of neighbours. The low-dimensional map is initially populated with n data points, $\{y_1, ..., y_n\}$, with random coordinates. From the low-dimensional map another set of conditional probabilities, $q_{j|i}$, for every pair of data points is calculated, however, these have a fixed variance value of $\frac{1}{\sqrt{2}}$. A gradient descent algorithm is then used to redistribute points $y_1, ..., y_n$ by minimising the sum of the Kullback-Leibler (KL) divergence (Kullback & Leibler, 1951) over all data points in $p_{j|i}$ and $q_{j|i}$ such that the difference between $p_{j|i}$ and $q_{j|i}$ is minimised.

SNE has two significant shortcomings, the KL divergence is difficult to optimise and the 'crowding problem' where moderately distant data points are 'crowded' into a relatively small area of the map. t-SNE proposes solutions to these issues. In order to better optimise the KL minimisation the conditional probabilities $p_{j|i}$ and $q_{j|i}$ are replaced with joint probabilities p_{ji} and q_{ji} which produces a simpler form of the gradient in the decent function meaning it is less computationally intensive. The crowding problem is addressed by using a Student t-distribution in place of the Gaussian distribution, with a single degree of freedom in q_{ji}. This type of heavy-tailed distribution approaches an inverse square law for large distances in the map, meaning the map can scale without impacting the joint probabilities allowing moderately distant points to be better separated within a larger space.

4.4 Experimental Data

In order to evaluate the effectiveness of the proposed approach, experimental data is required that has known similarities. This is not something that occurs naturally in large quantities, therefore it was decided that the most appropriate approach was to use synthetically generated graphs. The experimental data was created with a number of graph generation algorithms, using multiple parameters, using the SNAP 3.0 C++ system for analysis and manipulation of large networks (Leskovec & Sosič, 2016).

4.4.1 Barabási-Albert Model (BA)

The Barabási-Albert model (Barabási & Albert, 1999) can generate scale-free networks, identifies by the presence of few nodes with a very high degree and an overall power-law degree distribution, that are often seen in natural and man-made networks, including the Internet, citation networks, and some social networks (Barabási et al., 2000). The algorithm starts with n_0 nodes, the links between which are chosen arbitrarily. At each iteration i a new node is added, with $n(\leq n_0)$ edges that connect to nodes already in the network. The probability p_i that an edge is formed between the new node and node n_i is dependant on degree k_i of node n_i as:

$$p_i = \frac{k_i}{\sum_k k_j}, \tag{4.4.6}$$

where $\sum_k k_j$ is the total degree of the entire network. After i iterations the model generates a network with $N = i + n_0$ nodes and $E = n_0 + ni$ edges. Within the SNAP implementation of this model the graph is initialised as a single pair of nodes, $n_0 = 2$, with one edge connecting them.

4.4.2 Erdős-Rényi Model (ER)

The Erdős-Rényi model, one of the oldest models for random graph generation, (Erdős & Renyi, 1984) it produces a graph using a fixed node set with a fixed number of edges. Given user-specified values for nodes n and edges m the graph $G(n, m)$ is constructed. All edges are chosen uniformly randomly from the set of all possible edges.

4.4.3 Forest Fire Model (FF)

The Forest Fire model (Leskovec et al., 2007) requires three input parameters: The number of desired nodes n, a Forward burning probability P_f and a backward burning ratio P_b. Using these parameters an initial graph is constructed which consists of a single node, then for each additional node n_i that joins the graph the following steps are repeated:

1. n_i selects another node n_j uniformly at random, from the set of existing nodes, and forms an edge to it.

2. A random number x that is geometrically distributed with the forward burn probability mean $P_f/(1 - P_f)$ and a random number y that is geometrically distributed with the backwards ratio mean $P_f P_b/(1 - P_f P_b)$ are evaluated. x out-edges and y in-edges for n_j are selected, the corresponding nodes of which encompass the set $v_1 ... v_x$.

3. n_i forms edges to each node in $v_1 ... v_x$ and then repeats step 2 for each node in the set.

As the process continues, nodes cannot be 'burned' a second time, preventing the algorithm entering an infinite loop.

4.4.4 Watts-Strogatz Small World Model (WS)

The creation of a Watts-Strogatz (Watts & Strogatz, 1998) small-world random graph has two basic steps. Initially, a ring lattice is constructed by arranging n nodes in a circle and connecting each node to its neighbours nn 'hops' away. Subsequently, each edge in the graph is rewired, such that the edge will move to a different, randomly selected, destination node with k nearest neighbours, based on probability β. The rewired edge cannot be a duplicate or self-loop. Following the initial step, the graph represents a perfect ring lattice. As $\beta \rightarrow 1$ the graph becomes more random as a greater proportion of the edges are rewired.

4.4.5 Data Generation

Table 4.1 presents the generated data, including the parameters used to generate each set of graphs. Where N is the number of nodes the algorithm needs to generate and E is the number of required edges. β is the edge probability used in the Watts-Strogatz Small World Model. k is the average degree used in the Barabási-Albert and Watts-Strogatz Small World models. P_f and P_b are the forward and backwards burn probabilities used the the Forest Fire Model. A value of $-$ denotes that the parameter is not used for a particular method. A single set represents 50 graphs, all generated with an identical method and parameters.

Set	Model	N	E	β	k	P_f	P_b
1	ER	100000	100000	-	-	-	-
2	ER	100000	200000	-	-	-	-
3	ER	100000	300000	-	-	-	-
4	ER	100000	400000	-	-	-	-
5	ER	100000	500000	-	-	-	-
6	BA	100000	-	-	1	-	-
7	BA	100000	-	-	2	-	-
8	BA	100000	-	-	3	-	-
9	BA	100000	-	-	4	-	-
10	BA	100000	-	-	5	-	-
11	WS	100000	-	0.1	4	-	-
12	WS	100000	-	0.3	4	-	-
13	WS	100000	-	0.5	4	-	-
14	WS	100000	-	0.7	4	-	-
15	WS	100000	-	0.9	4	-	-
16	FF	100000	-	-	-	0.1	0.15
17	FF	100000	-	-	-	0.15	0.2
18	FF	100000	-	-	-	0.2	0.25
19	FF	100000	-	-	-	0.25	0.3
20	FF	100000	-	-	-	0.3	0.35

Table 4.1: Algorithms and parameters used to generate datasets

4.5 Results

Each of the clustering methods considered in this work will be evaluated on two metrics of good clustering, accuracy and stability (Schaeffer, 2007), along with time

taken to evaluate results, where each algorithm is given the entire set of GFPs to cluster. The intention is to identify an accurate algorithm for clustering which can be performed in 'reasonable' time.

Accuracy: An algorithm will be considered as accurate if it can successfully group together the Graph FingerPrints (GFP)s, described in Section 4.2, of graphs that are generated using the same generation method and identical parameters. Such that accuracy is reported as the percentage of identically generated graphs that are assigned to the largest single cluster.

Stability: An algorithm will be considered as stable if it shows consistent results over 100 runs of the entire algorithm. The value of 100 was chosen in order to give acceptable statistical precision (Carling & Meng, 2015) of the results.

4.5.1 Experimental Setup

All the experiments presented in this work were performed on a system with 2 × 10C 2.3GHz Intel Xeon E5-2650 v3, 64GB RAM, CentOS 7.2, GCC 4.8.5, Boost 1.56, Python 2.7.5, Graph-Tool 2.8 and R 3.3.1. A summary of the datasets used can be seen in Table 4.1, for each of which the relevant parameters are discussed in Section 4.4.

4.5.2 Hierarchical Clustering (HC)

Multiple distance calculation methods were evaluated for this clustering algorithm, these were: Ward, single, complete, average, mcquitty, median and centroid (Müllner, 2011), all using an underlying Agglomerative (top-down) method. Overall hierarchical clustering was not very accurate for correctly identifying clusters, with all accuracy values being under 35%, performing only slightly better than K-means. Therefore only the most promising results provided by the Ward method are re-

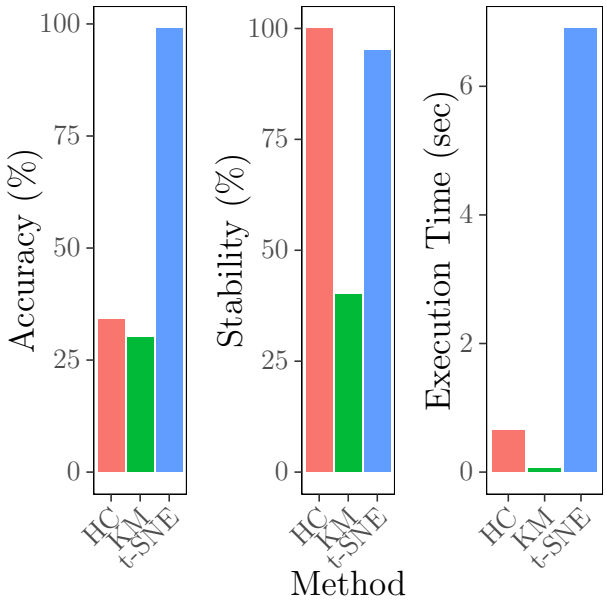


Figure 4.1: Comparative Results

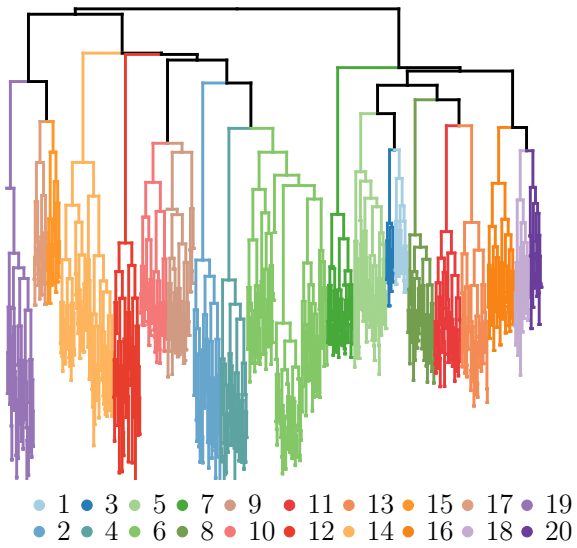


Figure 4.2: Hierarchical Clustering Dendrogram (Ward Algorithm)

ported here. The dendrogram shown in Figure 4.1 depicts how each cluster is defined by drawing a 'U' link between the children of a non-singleton cluster and their parent. The top of the 'U' indicates the point at which a cluster merges, while the legs of the link show which clusters were merged. The length of each leg represents the euclidean distance between the child clusters. Further the colours represent the

cluster number to which an entity was deemed to belong. Hierarchical clustering provided the most stable results, shown in Figure 4.1, demonstrating no difference in output over multiple iterations. Overall results showed that this approach was only able to correctly group similar datasets with an accuracy of 34%. This is largely attributed to the reliance of this method on Euclidean distance values, which are not ideally suited to high dimensional data. Figure 4.2 is a dendrogram representation of the result, showing a tree of branches and leaves, with colouring representing the clusters assigned by the algorithm. A branch is any intersection of a tree that has child nodes and a leaf has no children. This shows the algorithm fails to accurately cluster the data. Branches have an unbalanced number of leaves when they should fall into even groupings, denoting the desired cluster assignments.

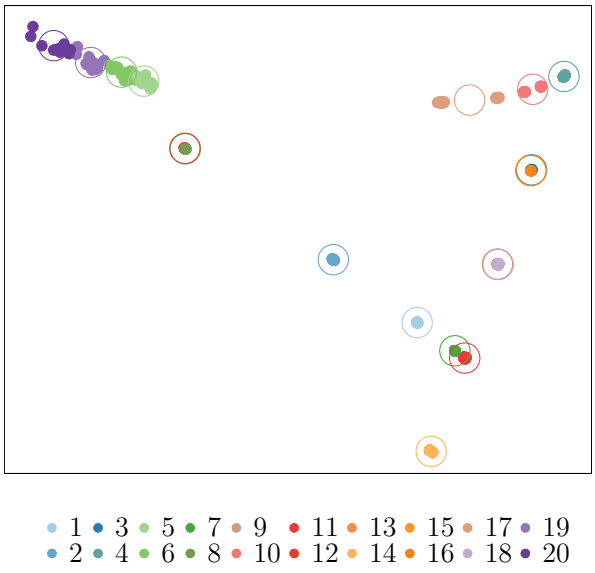


Figure 4.3: K-means Clustering

4.5.3 K-means++ (KM)

Although it may be assumed that K-means would provide reasonable clustering of the GFP data, this was found not to hold. The assumption was based around two factors: Using Principal Component Analysis to reduce the dimensionality of the

data and that due to the nature of the data the value of k is already known and does not need to be discovered. Figure 4.1 shows K-means is by far the fastest algorithm delivering an average runtime of ≈ 0.1 seconds which is five times faster than the next fastest approach. However, it can be seen from the accuracy and stability reported in Figure 4.3 that kMeans struggles to cluster properly. Some groups have multiple centres while others share one. K-means only managed to correctly cluster, on average, 30% of the data, with the best performance of 39%. K-means also demonstrates a high degree of variation in the produced results, over multiple runs, providing a very unstable solution to the clustering problem. This is primarily because K-means is significantly impacted by the initial placement of cluster centres, meaning centres can get caught in a local minimum of a group that should be members of another cluster.

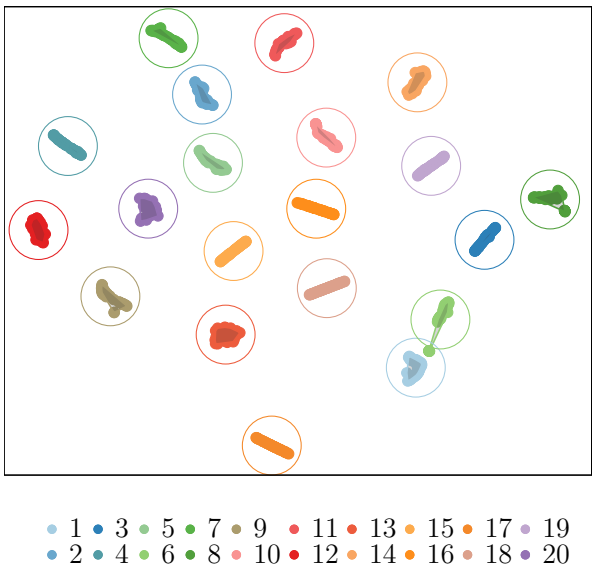


Figure 4.4: t-SNE Clustering

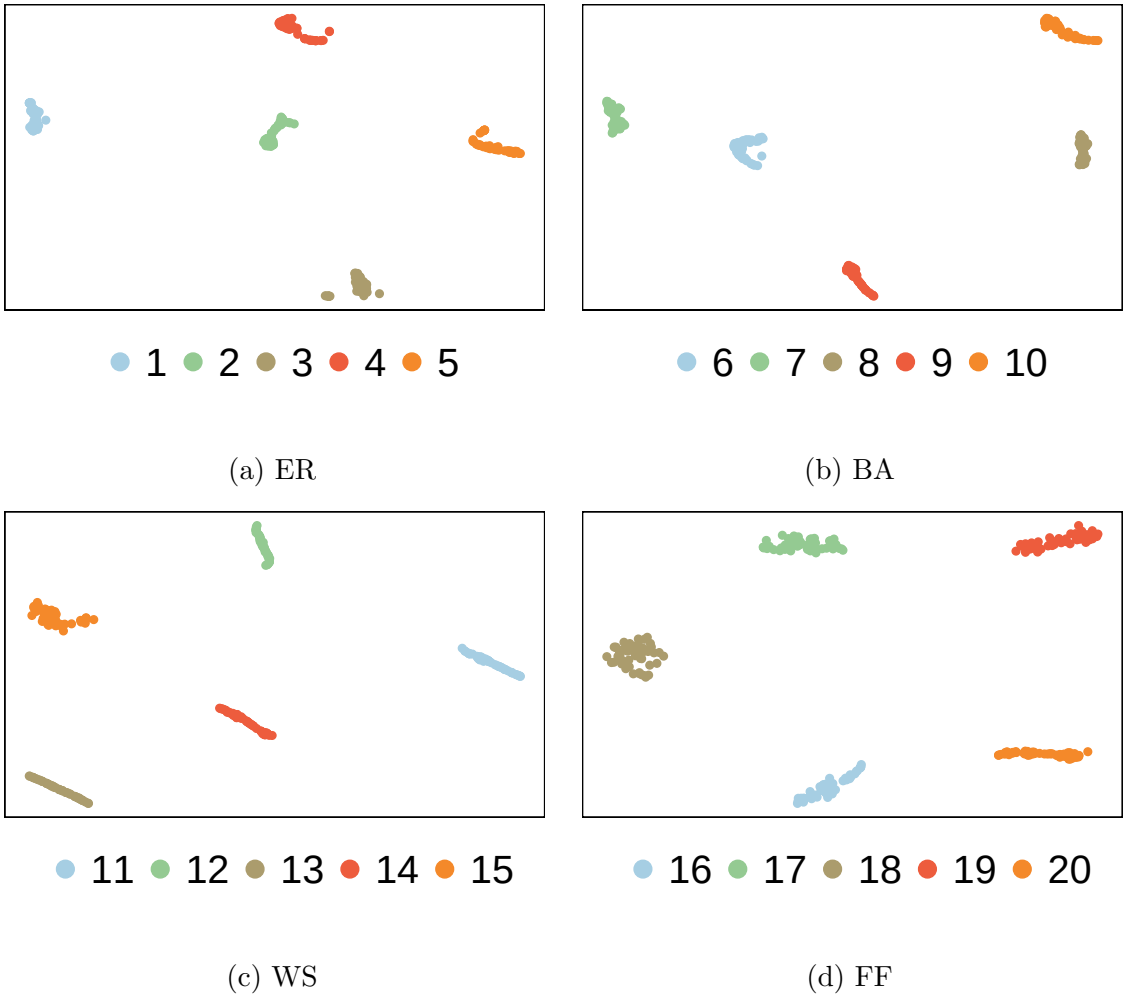
4.5.4 t-SNE

Clustering GFP data using the t-SNE method provided significantly better results than the other methods considered. The overall accuracy of 99% was observed, with

only 10 data points showing a normalised distance of greater than 0.07 from the other points within the cluster. Each of these data points was a member of data set #6, in Table 4.1. This is because using the SNAP implementation of the BA Model with $k = 1$ is capable of producing graphs where the production of highly connected hubs is heavily dissuaded. Therefore the degree distribution of these graphs tends to be closer to those observed with the ER model than a power-law distribution. Figure 4.1 shows that even though t-SNE is the slowest approach, over a number of iterations t-SNE shows good stability, consistently producing accurate results with only minor variations in intra-cluster distances. The overall results can be seen in Figure 4.4. Each cluster, with the exception of set #6, is shown to be clearly defined within an easily identifiable space with minimal outliers. The t-SNE method is also capable of differentiating between graphs generated with the same algorithm but different parameters. Figure 4.5 shows the clustering achieved for each generation method when attempting to identify differing parameters within a common method. The clear separation of the clusters shows that GFP + t-SNE is sensitive to the relatively minor topological differences between graphs of a similar class.

4.6 Conclusion

In this work it has been demonstrated that using GFP along with t-SNE clustering is a good method for identifying graph classes, with an accuracy of 99% and excellent repeatability. While traditional methods, such as Hierarchical Clustering and K-Means++, have been shown, in the results presented here, to be inaccurate, the t-SNE algorithm demonstrated a significant ability to cluster the high dimensional data of the GFP. This provides an efficient method for classifying very large and complex graphs. GFP + t-SNE delivers an accurate and stable solution that, while



(a) ER

(b) BA

(c) WS

(d) FF

Figure 4.5: t-SNE by Generation Method

it is significantly slower than other methods, provides useful results in a timely manner.

Chapter 5

Using Neural Networks to Identify Communities

5.1 Introduction

The identification of communities within graph structures has been heavily studied for many years. However, in recent years the levels of digital interaction have increased by many orders of magnitude. This interaction, with services such as Facebook and LinkedIn, has produced datasets that can no longer be easily analysed as they are far too large to fit into system memory. These social networks are usually represented as graphs, consisting of vertices depicting individuals and edges depicting relationships. Additional features are generally associated with the nodes and are rarely applied to edges. Furthermore, as these networks have grown in size and complexity is has become more difficult to interpret exactly which features, or a combination thereof, represent the membership of a node in a particular community.

Detecting communities within these structures is very important in many fields to enable understanding and extraction of information from these complex systems.

A number of strategies have been developed using Neural Networks in order to gain a better understanding of these complex inter-node relationships. These are discussed in Section 2.6.

5.2 Ego-nets

Large graphs have issues fitting into typically limited GPU memory. A possible approach to mitigate this problem is the use of ego-nets. Splitting a graph up into individual ego-nets allows us to batch the training and then the only bound for the graphs that can be learnt on is7 the amount of available disk space. Ego-nets are the subgraphs which consist of the connections directly related to the 'ego' node. The size of an ego-net is directly affected by the number of hops being considered. An ego-net of one-hop will consist of the ego vertex and all the vertices that can be reached with a maximum path length of one.

Given a variable number of hops h an ego-net, $egonet$, can be defined as $egonet_h \subset G$ where:

$$egonet_h = \{node_n \mid shortest\_path(ego\_node, node_n) <= h\} \tag{5.2.1}$$

Ego-nets can be easily constructed from an edge list without relying on random walks. This means there is no requirement to construct a graph structure in memory in order to determine the membership of ego-nets. This process is described in Algorithm 1 where *node_set* is a function that takes a list of edges, represented as pairs of nodes, and returns a set of all unique nodes contained within the list of pairs.

Algorithm 1: Ego-net Construction

Input: $G(V, E), h$

Result: All ego-nets of size h in G

for *ego_node in* V **do**

  $node\_list \rightarrow \emptyset$;

  for i *in range(h)* **do**

    if $node\_list \equiv \emptyset$ **then**

      Add *ego_node* to *node_list*;

    end

    $curr\_edges = \{E_{i,j} \mid i \in node\_list \text{ or } j \in node\_list\}$;

    $node\_list = node\_set(curr\_edges) \cup node\_list$

  end

  Save *curr_edges* to disk

end

5.3 Method

As this work was concerned with correctly identifying the community membership of a node, only real-world datasets were used for evaluation during this work. These benchmark datasets are: Cora, Citeseer and Pubmed. Performance of the baseline approaches were compared with the ego-net approach. The only addition required to allow ego-nets to be used with these approaches was to add the ability to support batches. The rest of this section gives a brief overview of each architecture used in this work.

5.3.1 Graph Convolutional Networks

A Graph Convolutional Network (GCN) (Kipf & Welling, 2017) is a Convolutional Neural Network (CNN) which operates on graphs. A GCN takes a graph G repre-

sented as $\hat{\mathbf{A}}$ and an initial node level feature matrix $\mathbf{X}$ as input and computes a new matrix of node level features $\mathbf{H} = GCN(\hat{\mathbf{A}}, \mathbf{X})$. The operation performed by each layer of a GCN is described as follows. (Kipf & Welling, 2017):

$$GCN^{(l)}(\mathbf{H}^{(l)}, \hat{\mathbf{A}}) = \sigma_r(\hat{\mathbf{A}}\mathbf{H}^{(l-1)}\mathbf{W}^{(l)}), \tag{5.3.2}$$

where $\mathbf{H}^{(l-1)}$ is the features from the previous layer, l is the current layer and $H^{(l-1)}$ are the weights for the previous layer. $\mathbf{W}^{(l)}$ are the learnable weights for the current layer. σ_r is the non-linear ReLU activation function $f(x) = max(0, x)$ where x is the input value.

5.3.2 GraphSAGE

Inductive Representation Learning networks, also known as GraphSAGE (Hamilton et al., 2017a) in an inductive framework for generating vertex embeddings from features that can be further used for classification. A GraphSAGE model takes a graph G represented as $\hat{\mathbf{A}}$ and a matrix $\mathbf{H} = GraphSAGE(\hat{\mathbf{A}}, \mathbf{X})$ of vertex features as input. Each layer of a GraphSAGE model has two steps aggregate and update described as follows. The aggregate step:

$$a_v = f(h_u \mid u \in N(v)), \tag{5.3.3}$$

where a_v in the aggregated representation for vertex v, h_u is the embedding h for all vertices u in the immediate neighbourhood of v $N(v)$. The update step:

$$h_v = f(a_v, h_v - 1), \tag{5.3.4}$$

where h_v is the updated representation for vertex v and $h_v - 1$ is the representation from the previous iteration. In both of these steps f can be any differentiable function but a simple mean function is used by default.

5.3.3 Graph Attention Networks

A Graph Attention Network (GAT) Veličković et al. (2018) uses spacial information to learn vertex embeddings. The input to a GAT network is the same as that described for GCNs above. A GAT performs four distinct operations in each layer.

$$z_i^{(l)} = \mathbf{W}^{(l)} h_i^{(l)}, \tag{5.3.5}$$

$$e_{ij}^{(l)} = \text{LeakyReLU}(\vec{a}^{(l)^T}(z_i^{(l)} \mathbin{+\!\!+} z_j^{(l)})), \tag{5.3.6}$$

$$\alpha_{ij}^{(l)} = \frac{\exp(e_{ij}^{(l)})}{\sum_{k \in \mathcal{N}(i)} \exp(e_{ik}^{(l)})}, \tag{5.3.7}$$

$$h_i^{(l+1)} = \sigma\left(\sum_{j \in \mathcal{N}(i)} \alpha_{ij}^{(l)} z_j^{(l)}\right). \tag{5.3.8}$$

Equation 5.3.5 describes a linear transformation of the previous layer embedding $h_i^{(l)}$ where $\mathbf{W}^{(l)}$ is the learnable weight matrix and i is the current node. Equation 5.3.6 shows the computation of a pair-wise attention score between two neighbours (i, j). Initially, the z embeddings of the two vertices are concatenated, denoted by $\mathbin{+\!\!+}$, then the dot product of the result and a learnable weight vector $\vec{a}^{(l)}$ is taken and then a LeakyReLU activation function $f(x) = max(0, 0.01x)$ where x is the input value is applied. Equation 5.3.7 is a softmax activation providing normalisation across the attention scores on each vertices incoming edges. Finally in equation 5.3.8, the representations of neighbours are aggregated together and scaled by the attention scores. This operation is analogous to how a GCN operates using the same ReLU activation function σ.

5.4 Experimental Setup

5.4.1 Datasets

The data used for the experiments in this work are all real-world graphs, in the form of the popular benchmark Cora, Pubmed and Citeseer (Yang et al., 2016) datasets.

- The **Cora** dataset is a publication citation network that consists of 2078 vertices with 5429 edges, labelled by publication class.
- The **Pubmed** dataset is a publication citation network, taken from the Pubmed database, that consists of 19,717 vertices with 44,338 edges, labelled by the type of diabetes that they pertain to.
- The **Citeseer** dataset is a machine learning focussed citation network that consists of 3327 vertices with 4732 edges. Each vertex is a representation of a single paper from the citation network, labelled by the area of machine learning that the paper focusses on.

Each of these have been used for vertex classification and link prediction in the original GCN (Kipf & Welling, 2017) and GAE (Kipf & Welling, 2016) papers, so were an ideal choice for assessing any predictive performance changes due to the use of sub-graphs.

5.4.2 Experimental Environment

All the experiments presented in this work were performed on a system with an 8 core 3.4GHz i7-6700, 32GB RAM, NVIDIA GeForce GTX 1080 Ti, Arch Linux 4.20.13, GCC 8.3, Python 3.7.

5.4.3 Experiments

As a precursor to any experimental evaluation all ego-nets were constructed and saved to disk. Using the method outlined in Algorithm 1 each of the graphs were split into individual ego-net graphs over the range of [1..6]. These ego-nets were then analysed for distribution of sizes across each dataset. With the ego-nets constructed. Initial experiments are run in order to compare the performance of the ego-net approach with the baseline approaches. From this initial comparison, the best performing ego-net size is selected for further comparison. A hyperparameter search is performed for both the baseline approaches and the best performing hop-size, using the parameters outlined in Table 5.1, and further comparisons are drawn. For each parameter defined in Table 5.1 a search was performed across each of the values defined within the value range set for a total of $23,760,000$ possible parameter combinations.

Parameter	**Value Range**
Batch Size	$\{1, 2, 4, 8, 16, 32, 64, 128\}$.
Learning Rate	$\{x \mid 0.0005 < x < 0.0999 \equiv 0 \; mod \; 0.001\}$.
Weight Decay	$\{x \mid 5 \times 10^{-6} < x < 1 \times 10^{-3} \equiv 0 \; mod \; 0.00001\}$.
Dropout	$\{x \mid 0 < x < 0.99 \equiv 0 \; mod \; 0.01\}$.
Optimiser	{Adam, SGD, RMSprop}

Table 5.1: Hyperparameter Search Space

5.5 Results

This section presents the results of the experiments outlined in section 5.4, evaluating the performance of the ego-net based approach against the baseline approaches.

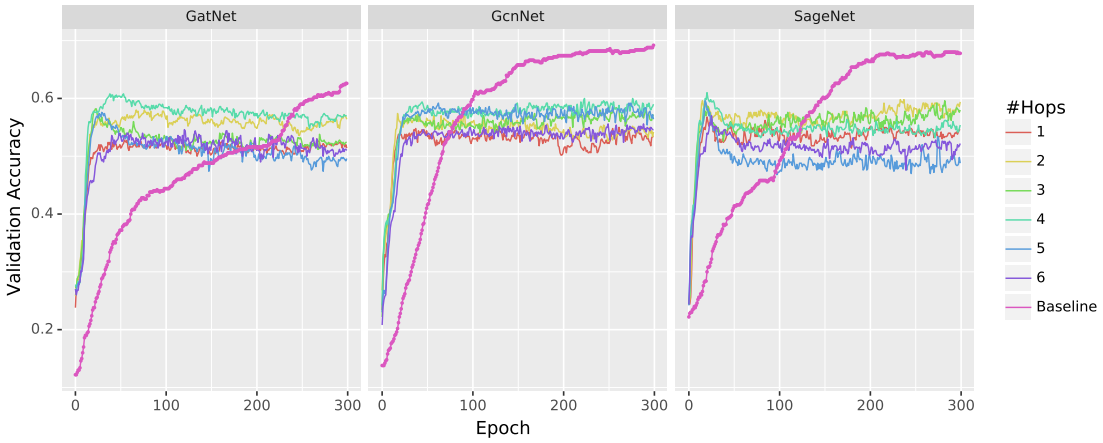
Table 5.2 shows initial comparison, across all ego-net sizes, to the baseline models. It can be seen here that the ego-net approach outperforms the baseline in almost all cases. The outliers from this trend are the SageNet baseline on the Cora dataset and the GatNet baseline on the Citeseer dataset. It is also worth noting the observation that an ego-net that has a size of 2 hops shows the best performance 50% of the time.

Network	Dataset	1 Hop	2 Hop	3 Hop	4 Hop	5 Hop	6 Hop	Baseline
SageNet	Cora	64.2	**69.2**	66.8	60.6	58.2	56.2	**69.6**
	Citeseer	53.2	**55.8**	55.2	55.4	48.2	52.2	44.4
	Pubmed	73.4	73.4	**76.6**	63.0	64.6	59.4	69.8
GatNet	Cora	57.8	65.8	62.8	63.6	**66.8**	55.6	58.4
	Citeseer	50.2	56.4	51.8	**58.4**	49.6	51.0	**64.4**
	Pubmed	74.4	**76.6**	76.4	70.2	68.0	65.6	70.8
GcnNet	Cora	69.0	**74.8**	71.0	68.6	64.6	56.2	40.6
	Citeseer	54.0	56.2	55.8	**59.2**	56.6	55.0	53.2
	Pubmed	74.6	76.2	**79.6**	75.6	68.0	59.0	66.8

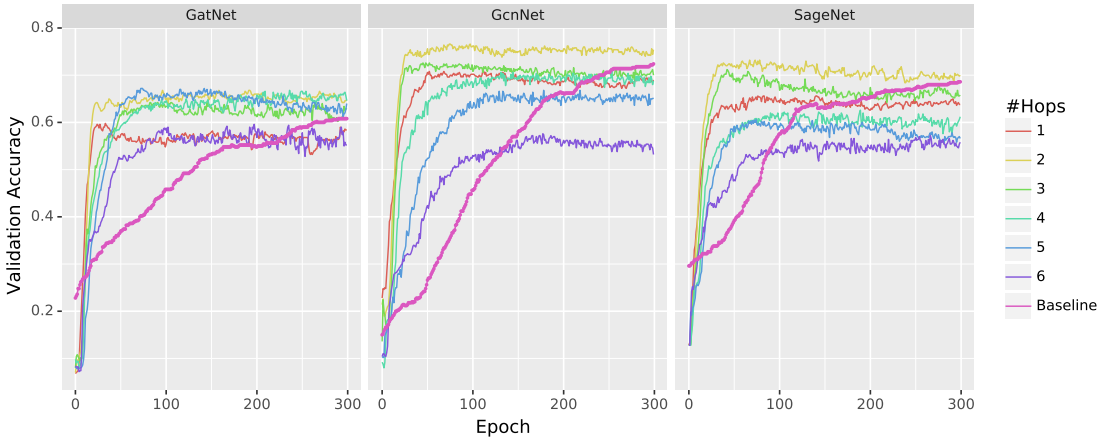
Table 5.2: Validation Accuracy for Varying Ego-Net Size

As shown in Figure 5.1 while the baseline approach often outperforms the ego-net approach, this can take a much larger number of epochs. The proposed approach provides a good level of accuracy after very few epochs compared to the baseline. Figure 5.1(a) shows that while the baseline always outperforms the ego-net approach given > 100 epochs the ego-net approach is able to produce a reasonable level of accuracy after only ≈ 10 epochs. That picture becomes much less clear in Figure 5.1(b) where the baseline approach never quite manages to outperform the proposed method. The ego-net approach still makes rapid gains on accuracy but the corresponding baseline approach never manages to beat this value, even after 300 epochs. This situation becomes even worse for the baseline in Figure 5.1(c) where

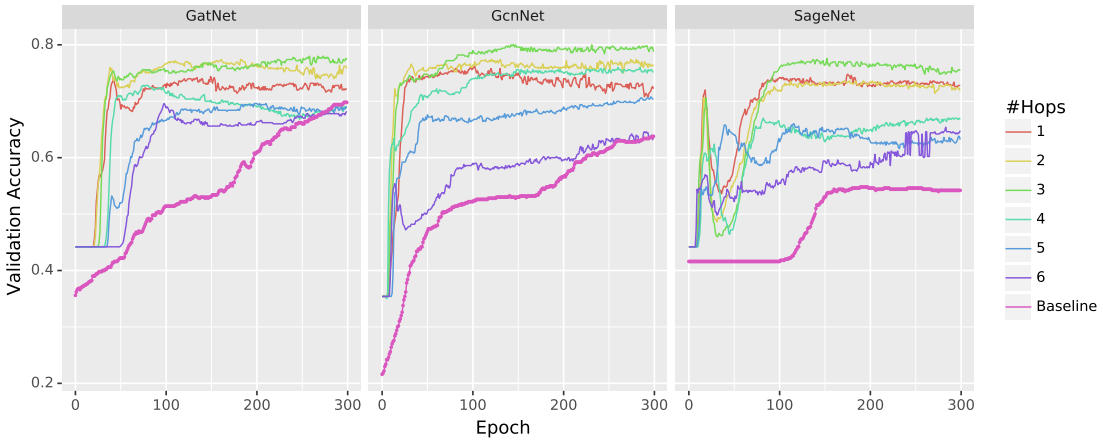
the largest dataset in this work is considered. In this instance, the baseline is outperformed across almost all ego-net sizes with only the GatNet baseline managing a higher accuracy than the three ego-net runs using the largest ego-net sizes.



(a) Citeseer



(b) Cora

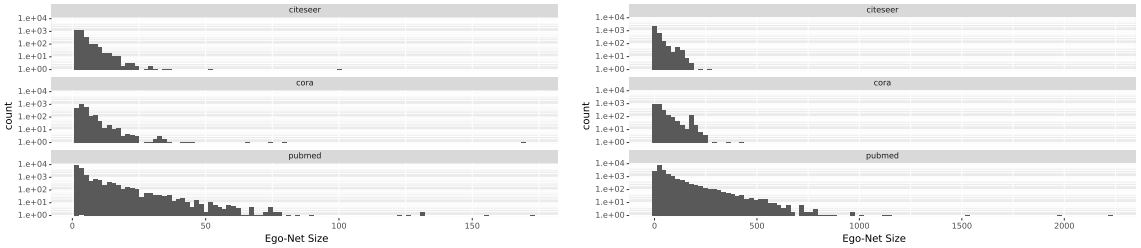


(c) Pubmed

Figure 5.1: Validation accuracy by number of hops

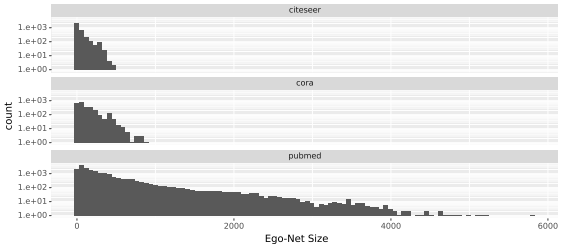
Figure 5.2 shows the distribution of ego-net sizes across each hop size. It can

be seen that greater hop sizes quickly increase the overall ego-net size to the point where a large proportion of the overall graph in present in each ego-net. Given this and the results from Table 5.2, 2 hops was selected as the fixed ego-net size for further comparison.

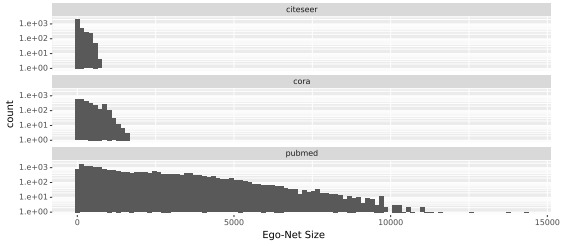


(a) 1 Hop

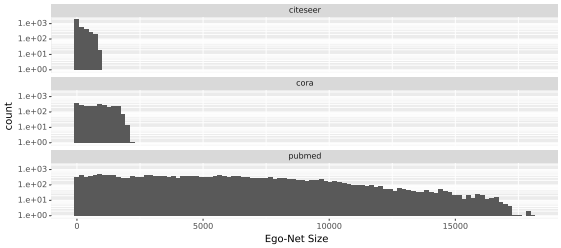
(b) 2 Hop



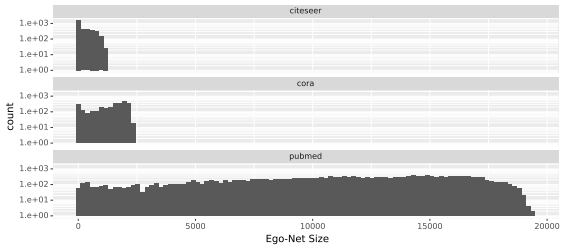
(c) 3 Hop



(d) 4 Hop



(e) 5 Hop



(f) 6 Hop

Figure 5.2: Distribution of egonet sizes by number of hops

Once the 2 hop ego-net had been selected as the most appropriate size for further consideration a hyperparameter search was performed across the hyperparameter ranges detailed in Table 5.1. Where Learning Rate (LR), Weight Decay (WD), Dropout and Accuracy are all within the ranges originally outlined in Table 5.1. Figure 5.3 compares the baseline to a 2 hop ego-net approach with tuned hyperparameters. From this, it can be repeatedly observed that the ego-net approach

gains good accuracy very early on while the baseline takes much longer to achieve these values. It can also be seen that after an extended number of epochs the only baseline methods that outperform the ego-net approach are GatNet on the Citeseer dataset and SageNet on the Pubmed dataset.

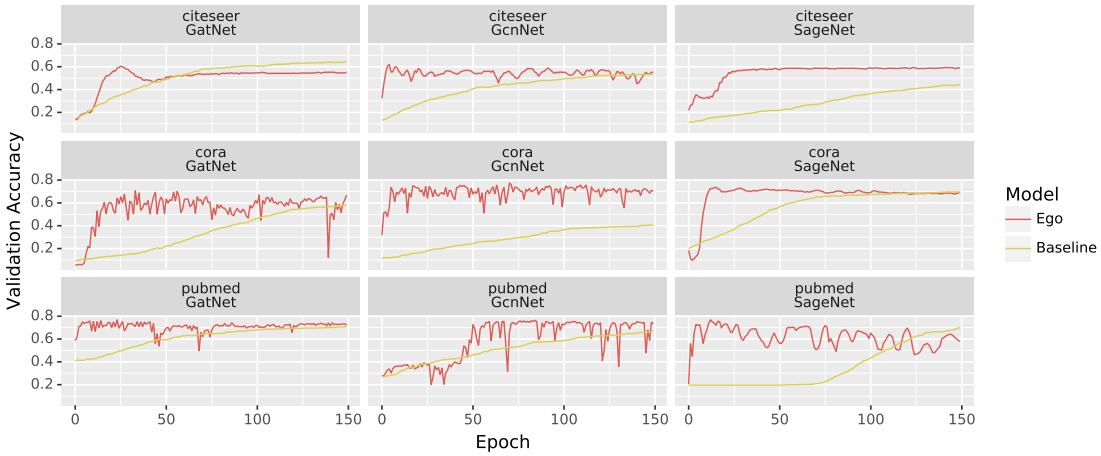


Figure 5.3: Best Hyperparameters

Table 5.3 shows the best hyperparameters that were found for each model. It is interesting to note that parameters that work well for the baseline would be unsuitable for the ego-net based approach.

Table 5.4 shows the min, max, mean, median and standard deviation of accuracy across all hyperparameters used for a given model. This shows that the ego-net approach is often outperformed in terms of pure accuracy. Such as in the citeseer GatNet example where the minimum and maximum accuracy for the ego-net approach figures were 0.074 and 0.158 below that of the baseline approach. However, the ego-net approach demonstrates better stability across multiple hyperparameters. This can be identified using the same example mentioned previously where for the ego-net result for standard deviation is 0.014 lower than the baseline. Given this reduction in standard deviation it can be inferred that any hyperparameter

Dataset	Model	LR	WD	Dropout	Accuracy
cora	SageNet	6.95e-02	4.55e-04	5.80e-01	7.26e-01
cora	GatNet	1.39e-02	4.65e-04	2.90e-01	7.32e-01
cora	GcnNet	1.68e-02	7.25e-04	2.70e-01	7.38e-01
citeseer	SageNet	9.72e-02	8.35e-04	8.00e-02	5.98e-01
citeseer	GatNet	8.91e-02	6.15e-04	1.00e-01	5.82e-01
citeseer	GcnNet	5.40e-03	7.05e-04	2.10e-01	6.04e-01
pubmed	SageNet	1.58e-02	2.15e-04	1.00e-02	7.46e-01
pubmed	GatNet	2.49e-02	5.75e-04	4.50e-01	7.70e-01
pubmed	GcnNet	9.72e-02	2.65e-04	2.70e-01	7.92e-01
cora	SageNetBaseline	4.11e-02	7.65e-04	5.80e-01	8.04e-01
cora	GatNetBaseline	6.73e-02	3.55e-04	7.50e-01	7.62e-01
cora	GcnNetBaseline	4.20e-02	4.95e-04	5.70e-01	7.96e-01
citeseer	SageNetBaseline	7.99e-02	9.75e-04	2.40e-01	7.10e-01
citeseer	GatNetBaseline	5.22e-02	3.75e-04	7.70e-01	7.40e-01
citeseer	GcnNetBaseline	9.75e-02	2.50e-05	3.10e-01	7.20e-01
pubmed	SageNetBaseline	9.08e-02	9.55e-04	7.10e-01	8.08e-01
pubmed	GatNetBaseline	7.80e-03	3.75e-04	4.40e-01	8.06e-01
pubmed	GcnNetBaseline	4.66e-02	8.95e-04	8.10e-01	8.08e-01

Table 5.3: Best Hyperparameters

Dataset	Model	Accuracy					
		count	min	max	mean	median	std
citeseer	**GatNet**	**18**	**1.00e-01**	**5.82e-01**	**4.05e-01**	**4.77e-01**	**1.62e-01**
citeseer	GatNetBaseline	15	1.74e-01	7.40e-01	5.70e-01	6.38e-01	1.76e-01
citeseer	**GcnNet**	**10**	**2.18e-01**	**6.04e-01**	**5.22e-01**	**5.79e-01**	**1.24e-01**
citeseer	GcnNetBaseline	19	1.42e-01	7.20e-01	5.66e-01	6.72e-01	1.92e-01
citeseer	**SageNet**	**22**	**6.80e-02**	**5.98e-01**	**4.66e-01**	**5.57e-01**	**1.61e-01**
citeseer	SageNetBaseline	16	1.48e-01	7.10e-01	5.56e-01	6.12e-01	1.84e-01
cora	**GatNet**	**17**	**1.74e-01**	**7.32e-01**	**4.72e-01**	**4.70e-01**	**2.20e-01**
cora	GatNetBaseline	16	1.18e-01	7.62e-01	5.63e-01	6.24e-01	2.17e-01
cora	**GcnNet**	**9**	**1.04e-01**	**7.38e-01**	**4.31e-01**	**5.32e-01**	**2.43e-01**
cora	GcnNetBaseline	17	4.12e-01	7.96e-01	6.14e-01	5.84e-01	1.40e-01
cora	**SageNet**	**24**	**1.62e-01**	**7.26e-01**	**5.13e-01**	**5.77e-01**	**1.89e-01**
cora	SageNetBaseline	17	2.08e-01	8.04e-01	6.61e-01	7.32e-01	1.72e-01
pubmed	**GatNet**	**14**	**2.04e-01**	**7.70e-01**	**5.90e-01**	**6.82e-01**	**1.85e-01**
pubmed	GatNetBaseline	20	2.14e-01	8.06e-01	6.36e-01	7.70e-01	1.85e-01
pubmed	**GcnNet**	**23**	**3.54e-01**	**7.92e-01**	**5.50e-01**	**4.62e-01**	**1.51e-01**
pubmed	GcnNetBaseline	17	1.96e-01	8.08e-01	6.02e-01	5.54e-01	1.89e-01
pubmed	**SageNet**	**13**	**4.18e-01**	**7.46e-01**	**6.08e-01**	**6.46e-01**	**1.12e-01**
pubmed	SageNetBaseline	13	2.08e-01	8.08e-01	5.43e-01	5.24e-01	1.73e-01

Table 5.4: Hyperparameter Search Stats

optimisation technique applied to the ego-net method should identify reasonable parameters using less computational resource than the baseline approach.

5.6 Conclusion

Overall, the approach presented in this chapter requires fewer epochs of training to get to a decent level of predictive performance, however the baseline approaches will eventual overtake it if allowed to train for extended periods of time. The lower epoch requirement is likely because when using the ego-net approach, the model is exposed to nodes and their structures multiple times per epoch. When using the baseline approach, the algorithm takes entire graph adjacency matrix as input for each epoch and operates on each node within that matrix. When using the ego-net approach, each input is a subgraph of the ego and its neighbours meaning that for a given graph, if node a is directly adjacent to node b, node a will appear in the ego-net for node b and vice versa. This subsampling is also the reason why peak accuracy is affected. As the model sees the same node multiple times but with slightly varying structure as those nodes outside the ego-net have been removed. This leads to a minor loss of information which is detrimental to the overall accuracy.

However, the ego-net approach is much better in the use of GPU memory – the primary goal of this work. All the baseline approaches require the whole graph to be loaded into memory where this approach only requires visibility of a single ego-net at a time. This means that much larger graphs can be processed within the constraints of GPU memory. Of course, as the ego-nets are precomputed beforehand and saved to disk, this approach also has the effect of requiring more disk storage but this is a much less precious resource than GPU memory.

Chapter 6

Half-Precision in Graph Convolutional Neural Networks

As in many other fields, deep learning is helping to revolutionise the area of graph analytics. Historically, graphs have been analysed through kernel-based methods (Kriege et al., 2020), however, recent advances in the area of Graph Convolutional Networks (GCN) have shown great promise for improved results. Although other approaches towards graph analysis have subsequently been developed, these have yet to diminish the need for GCNs (Shchur et al., 2018). A GCN layer is a learnable non-linear function of the vertex features from the previous layer (represented as a matrix) and the adjacency matrix[1] for the graph. As such, GCNs are almost entirely constructed from matrix operations and hence well amenable to GPU programming.

One of the main drawbacks to the use of GCNs is the memory footprint. Unlike other forms of Deep Learning where only a subset of the data ever needs to be processed at any given point in time, a GCN operates on the entire adjacency

[1] An adjacency matrix A is an n by n matrix, where n is the number of vertices in the graph. $[a, b] = 1$ indicates an edge between vertices a and b.

matrix – i.e. the entire graph – at each step. This limits the size of the graph which may be operated upon. This is compounded by the fact that most GCN functions comprise of a number of matrices of commensurate size to the adjacency matrix.

The use of reduced-precision computation, most often combining half and full precision floating-point values, has been demonstrated to be beneficial within other areas of deep learning, significantly reducing memory requirements and training time and improving performance (Micikevicius et al., 2018; Das et al., 2018; Kuchaiev et al., 2018). As memory requirements are the biggest limitation to GCNs, applying reduced-precision computations would seem an obvious line of attack in order to process larger and more complex graphs.

Recent advances in GPU technology have lead to the introduction of Tensor Cores, which enable dynamic adaptation of mixed-precision floating-point computations. These allow for acceleration of Deep Learning and, according to NVIDIA, can provide up to ten times speed up (*NVIDIA Tensor Cores: Unprecedented Acceleration for HPC and AI*, n.d.). This, again, could be most useful when training larger GCNs. In this work, four levels of optimisation are implemented, with respect to the operation precision levels used in the forward and backward passes of the network, on two types of GCNs – a standard GCN and a Graph Convolutional Auto-Encoder (GAE) – in order to evaluate the advantages and disadvantages of each optimisation level. Experiments range from using no optimisation (full precision for all operation) through to everything performed and stored in half-precision. In order to make use of the Tensor Cores, certain model parameters, including input size, need to be divisible by 8 due to hardware and software restrictions in the NVIDIA Tensor Core implementation (*NVIDIA Tensor Cores: Unprecedented Acceleration for HPC and AI*, n.d.). Therefore, the results are evaluated both with and without padding the data to be a multiple of 8. As a result, certain benchmark datasets used in the experiments need to be padded to meet this condition, thus the results are evaluated

both with and without padding.

Based on prior literature which used reduced precision in deep learning, such as computer vision and natural language processing, one would Naïvely assume that learning-based graph analysis tasks would see reduced memory requirements, faster training times and perhaps even improved performance – with the desire that memory reduction would be the most significant. As such, these are set as the hypotheses for this work and design the experiments accordingly to evaluate the validity of each point. Experiments are thus performed using two types of GCN with four optimisation levels on two datasets. The real-world Cora[2] dataset is used, and to allow the scaling of graphs to specific sizes, synthetic graphs are also generated using the Barabási-Albert Model (Barabási et al., 2000).

In short, this research attempts to answer the following important questions with respect to the effects of reduced-precision operations and Tensor Cores on graph-based neural networks:

- *Run-Time* - Will using reduced-precision operations and Tensor Cores lead to more efficient training time for graph convolutional neural networks?
- *Memory Usage* - Will using reduced-precision operations and Tensor Cores lead to reduced memory requirement for graph convolutional neural networks?
- *Predictive Performance* - Will using reduced-precision operations and Tensor Cores lead to an improvement, degradation or no significant change in the predictive performance of graph convolutional neural networks?

To the best of the author's knowledge, this is the first work to conduct a comprehensive analysis of the effects of reduced precision on graph convolutional neu-

[2] `https://relational.fit.cvut.cz/dataset/CORA`

ral networks. The source code for this work is available at `https://github.com/grossular/half-precision-gcn`.

6.1 Motivation

With the introduction of Tensor Cores, first featured in the Volta GPU microarchitecture developed by NVIDIA (*NVIDIA Volta: The Tensor Core GPU Architecture designed to Bring AI to Every Industry*, n.d.), improved deep learning performance was made possible compared to the conventional CUDA cores. Tensor Cores are meant to enable mixed-precision computing (combining features from both half and full-precision operations), dynamically adapting calculations to accelerate throughput while preserving the accuracy of all calculations. It has been claimed that this technology can provide up to 10× speed ups for training across a variety of workloads (*NVIDIA Tensor Cores: Unprecedented Acceleration for HPC and AI*, n.d.).

To take advantage of the capabilities of Tensor Cores, workloads must use *mixed-precision* computations. Deep learning training procedures traditionally use the full-precision (`FP32`) format but with the growing demands for larger datasets and thus network architectures, intensive compute and memory requirements make stable model training expensive or even intractable at times. Using half-precision (e.g. `FP16`) can address the issues of memory bandwidth and computation time. However, the `FP16` format has a narrower dynamic range than `FP32` and its use can lead to significant loss of accuracy to the point of making successful training impossible. To address this issue, mixed-precision operations are introduced to enable maintaining accuracy standards while taking advantage of reduced computation and memory bandwidth at the same time.

It is important to note that using mixed-precision operations is not the only requirement for using Tensor Cores and certain other conditions must also be met.

The use of Tensor Cores is predicated on certain layer parameters (e.g. batch size, input size, output size, number of channels) being *divisible by 8*. This requirement is based on how data is stored and accessed in memory (*NVIDIA Deep Learning Performance*, 2020). Tensor Cores primarily optimise GEMM (General Matrix Multiplications), a fundamental building block for many operations in neural networks, such as fully-connected layers, recurrent layers (e.g. RNN, LSTM, GRU) and convolutional layers. The use of Tensor Cores via mixed-precision operation has been extensively investigated for various tasks and data modalities, such as computer vision and natural language processing (Micikevicius et al., 2018; Kuchaiev et al., 2018). However, the important area of graph analysis has thus far been neglected in the existing literature. In this work, the capabilities and limitations of Tensor Cores and mixed-precision training of graph convolutional neural networks are explored.

The applicability of Automatic Mixed Precision (AMP) on two commonly-used graph network architectures (GCN and GAE) is investigated. NVIDIA Apex (*NVIDIA Apex: Tools for Easy Mixed-Precision Training in PyTorch*, n.d.) enables AMP via PyTorch (Paszke et al., 2019) – providing the primary framework for the experiments. Apex AMP provides the opportunity to easily experiment with different levels of precision, by selecting an "optimisation level". Four default AMP modes (optimisation levels) are provided, briefly described in the following:

O0: Enables full-precision `FP32` training. Neural network weights and their corresponding operations are `FP32`. As this makes no modifications it can be used to establish a baseline for the experiments.

O1: The most commonly-used AMP mode, places each operation into one of two lists: a whitelist for all *Tensor Core-friendly* operations (e.g. GEMM and convolutions), and a blacklist for all others (e.g. non-linear operations, normalisations). Whitelist operations are performed in `FP16` and blacklist in `FP32`. Dynamic loss scaling is also important, as activation gradient values during `FP16` training need to

be scaled to preserve values that could otherwise be lost to zero.

O2: ("Almost-`FP16` Mixed Precision"), casts the model weights to `FP16` but maintains a set of `FP32` master weights. The input data that is fed through the network is cast to `FP16` but the optimiser acts directly on the `FP32` weights. Dynamic loss scaling is implemented as in O1. O1 and O2 are essentially different implementations of mixed precision and their performance will depend on the type of data and the operations involved in the network architecture.

O3: Enables full `FP16` across all operations and as such does not achieve the stability of O1 and O2 and will lead to a loss of accuracy. Similar to O0, this mode is primarily used to provide a baseline for the evaluation of other levels.

These experiments will enable an accurate and detailed analysis of the effects of mixed-precision on GCNs in terms of model performance, memory footprint and run-time.

6.2 Methodology

The primary objective of this work is to investigate how graph-specific neural models are affected by the use of reduced-precision operations. In order to achieve this, experiments are run on real and synthetic graphs, over all the available optimisation levels, on hardware equipped both with and without Tensor Cores. In the remainder of this section, a brief overview of the neural network architectures used in this work is given, before detailing the changes made for this study.

6.2.1 Graph Convolutions

Here, the basics of Graph Convolutional Networks (GCN) (Kipf & Welling, 2017), as introduced in Section 5.3.1, and the details how they may be affected by the move to reduced precision are explored. GCNs can be thought of differentiable functions

for aggregating feature representations from the neighbourhood of a given vertex (Hamilton et al., 2017b). For initial input, a GCN-based model takes the normalised adjacency matrix $\hat{\mathbf{A}}$ representing a graph G, and a matrix of initial vertex level features $\mathbf{X}$, and computes a new matrix of vertex level features $\mathbf{H} = GCN(\hat{\mathbf{A}}, \mathbf{X})$. Whilst $\mathbf{X}$ can be initialized with pre-computed vertex features, it is common to initialize it with one-hot feature vectors when no prior knowledge is available (in which case $\mathbf{X}$ is the identity matrix $\mathbf{I}$). Each layer in a GCN performs the following operation (Kipf & Welling, 2017):

$$GCN^{(l)}(\mathbf{H}^{(l)}, \hat{\mathbf{A}}) = \sigma_r(\hat{\mathbf{A}}\mathbf{H}^{(l-1)}\mathbf{W}^{(l)}), \tag{6.2.1}$$

where l is the number of the current layer, $\mathbf{W}^{(l)}$ denotes the weight matrix of that layer, and $H^{(l-1)}$ refers to the features computed at the previous layer or is equal to $\mathbf{X}$ at $l = 0$.

A GCN function can be considered as performing a weighted average of the neighbourhood features for each vertex in the graph. Stacking multiple GCN layers has the effect of increasing the number of hops from which a vertex-level representation can aggregate information – a three-layer GCN will aggregate information from three-hops within the graph to create each representation.

One interesting thing to consider is the dimensionality of the matrices involved in the GCN operation in Equation 6.2.1:

- The adjacency matrix, $\mathbf{A}$, is of size $N_v \times N_v$, where $N_v = |V|$ is the number of vertices in the graph.
- The input features matrix, $\mathbf{X}$, is of size $N_v \times F_v$, where F_v is the number of features for each vertex. Where no vertex features are present and the identity matrix $\mathbf{I}$ is used, the dimensionality would again be $N_v \times N_v$.
- The parameter matrix, $\mathbf{W}$, is of size $F_v \times d$, where d is the number of units in

that layer.

One thing to note is that the number of model parameters is closely tied to the size of the input features and that the resulting output from each layer in a GCN is bound by the number of vertices, in contrast to computer vision models.

6.2.2 Graph Convolutional Auto-Encoders

GCNs are trained via supervised learning, where labels are provided for a specific task – commonly vertex classification (Hamilton et al., 2017b; Kipf & Welling, 2017). However, extensions have been made to allow for convolutional auto-encoders for graph datasets, called Graph Auto Encoders (GAE) (Kipf & Welling, 2016). Auto-encoders are a type of unsupervised neural network which compressed the input data to a low-dimensional space, and then reconstructs the original data from the learned representation. This is commonly performed in order to use the resulting embeddings for the task of link prediction (Kipf & Welling, 2016; Bonner et al., 2019; Bonner, Brennan, et al., 2018).

Here, a non-probabilistic version of the GAE is considered, where the goal is to learn a low-dimensional representation of $\mathbf{A}$ from G, via an encoding from a GCN $\mathbf{Z} = GCN(\mathbf{A}, \mathbf{X})$, such that it can be used to accurately reconstruct the graph via a product between $\mathbf{Z}$ and its transpose passed through an element-wise logistic function σ:

$$\mathbf{A}' = \sigma(\mathbf{Z}\mathbf{Z}^{\mathsf{T}}). \tag{6.2.2}$$

6.2.3 Reduced Precision Changes

For the models used in this work, the overall architecture from the prior works (Kipf & Welling, 2017, 2016) is replicated. However, some changes were required to ensure

suitability for processing using reduced precision. In order to take advantage of the Tensor Cores, available on Volta and subsequent NVIDIA GPU architectures, some padding of the input graph is required under certain conditions. This is because Tensor Cores are only activated when specific matrices involved in operations for the forward and backward passed are divisible by 8, for `FP16` matrices, or 16, for `INT8` matrices (*NVIDIA Deep Learning Performance*, 2020). Due to this, in the experiments presented, where an input matrix is observed to be indivisible by 8 it is padded with zeroes to make it adhere to this condition so that Tensor Cores could be fully utilised. This padding can be thought of as additional vertices added to the graph with no edges to its self of other vertices. These added vertices are removed before the loss computation is performed. An experimental evaluation of this is presented in Section 6.4 to assess if this process has any negative impact of predictive performance. Additionally, due to current limitations in PyTorch[3], all tensors needed to be cast to dense matrices in order to be able to use reduced precision modes.

6.3 Experimental Setup

6.3.1 Datasets

These experiments use two primary datasets: the real-world benchmark Cora dataset (Yang et al., 2016) and a synthetic dataset using the well-known Barabási-Albert Model graph generation model (Barabási et al., 2000). The Cora dataset has been used for vertex classification and link prediction in the original GCN (Kipf & Welling, 2017) and GAE (Kipf & Welling, 2016) papers, so was the ideal choice for assessing any predictive performance changes due to the reduced precision. For observing run

[3]https://github.com/pytorch/pytorch/issues/41069

time and memory-related metrics, synthetics Barabási-Albert graphs where used as they reflect the scale-free nature of many empirical graphs and allowed us to precisely control the number of vertices in the input graph (Barabási et al., 2000).

6.3.2 Experimental Environment

The performance of the models was measured on three different computer systems, with three different generations of NVIDIA GPU (Pascal, Volta and Turing). The V100 (Volta) and Titan RTX (Turing) both are equipped with Tensor cores, whilst the P100 (Pascal) has no dedicated 16-bit hardware. The three test system are as follows:

- *Pascal System* - NVIDIA Tesla P100 GPU (16GB), Intel(R) Xeon(R) CPU E5-2690 v4 @ 2.60GHz, 128GB RAM, with Ubuntu 16.04, Python 3.7, CUDA 10.1, CuDNN v7.6 and PyTorch 1.1.
- *Turing System* - NVIDIA Titan RTX GPU (24GB), Intel(R) i9-9820X, 64GB RAM, with Arch 5.7.9, Python 3.8, CUDA 10.2, CuDNN v7.6 and PyTorch 1.1.
- *Volta System* - NVIDIA Tesla V100 GPU (16GB), Intel(R) Xeon(R) CPU E5-2690 v4 @ 2.60GHz, 128GB RAM, with Ubuntu 16.04, Python 3.7, CUDA 10.1, CuDNN v7.6 and PyTorch 1.1.

6.3.3 Experiments

Two distinct sets of experiments are performed: firstly, measure any impact on the ability of the models to make predictions accurately. Secondly, assess the affect of half-precision on run-time and total GPU memory consumed.

For the predictive performance experiments, semi-supervised classification accuracy for the GCN, and Area Under the precision-recall Curve (AUC) and Average

Parameter	Value Range
Opt Level	O0, 01, 02, 03
Use Features	False, True
Model Size	16, 32, 64, 128, 256, 512, 1024, 2048
Num Vertices	$\{x \mid 2048 \leq x \leq 2^{15} \equiv 0 \; mod \; 1024\}$.

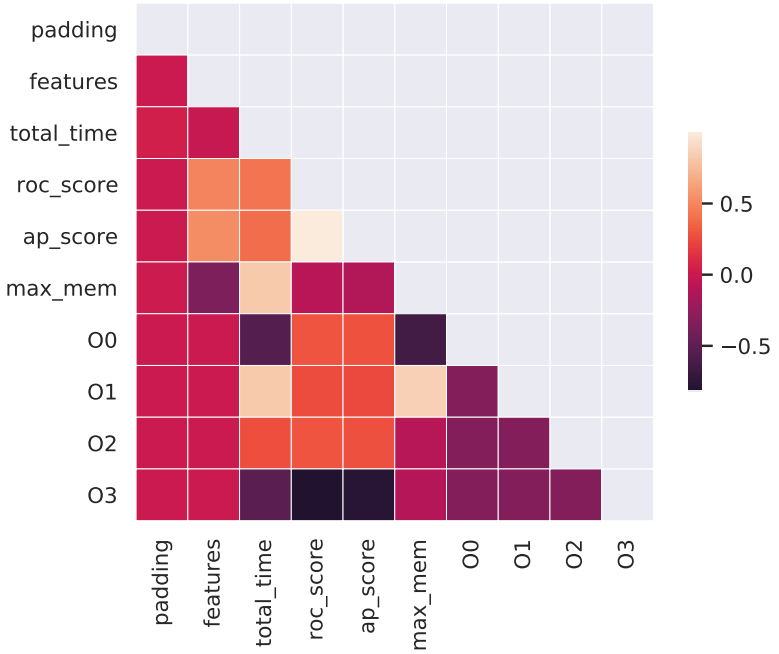
Table 6.1: Model and synthetic data parameter ranges.

Precision (AP) for the link prediction task are measured. These are the performance metrics used to measure the performance of the models when they were introduced (Kipf & Welling, 2017, 2016). The model architecture and primary hyperparameters are fixed and were also taken from the original work and were kept constant across all GPUs, padding use and optimisation levels.

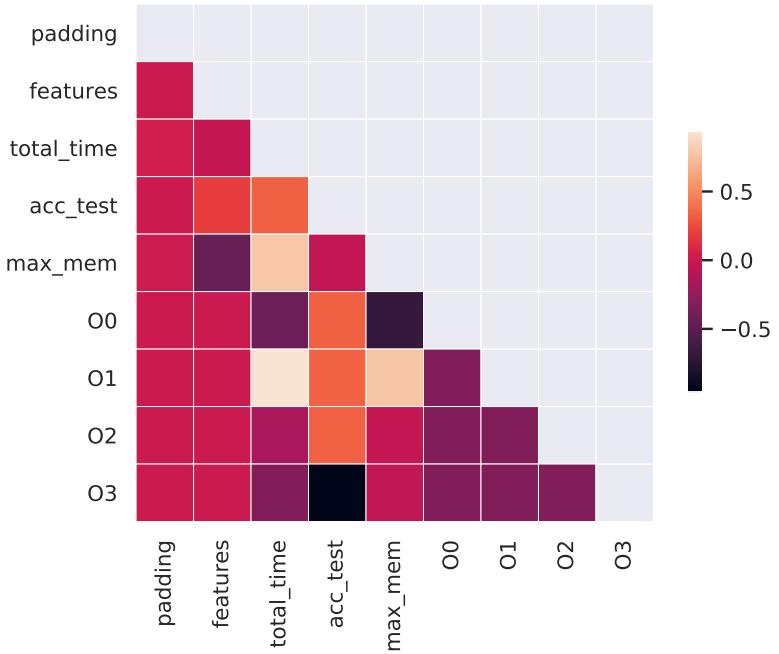
The run time performance results are taken using synthetic graphs so that the size can easily be controlled. This means that all graph sizes are divisible by 8, meaning no padding is required. For these experiments, models were trained using the various parameters detailed in Table 6.1 and measured both the memory consumption and the total training time. Each combination of parameters from Table 6.1 was repeated five times, each repeat with a different random seed.

6.4 Experimental Results

In this section, results of the experimental evaluation as discussed in Section 6.3 are presented. This begins with an assessment of the effects of mixed-precision training on the predictive performance of semi-supervised classification and link prediction. Then, the change in both run-time and the maximum memory consumed on the GPU is measured as the number of vertices in the input graph and the model size is increased for the various levels of reduced-precision optimisation.



(a) GAE



(b) GCN

Figure 6.1: Correlation of predictive performance values on the Cora dataset for the GCN and GAE models using the V100.

6.4.1 Assessing Model Predictive Performance

The first task here is to evaluate how predictive performance is affected by the move to reduced precision on the benchmark Cora dataset (Yang et al., 2016). To give a global view of the relationship between the variables, Figure 6.1 presents the correlation matrices for both the GCN and GAE approaches on the V100 GPU. It should be noted that very similar results were observed for all cards. The results demonstrating the performance of the various cards for the GCN model, with and without the use of padding, are presented in Table 6.2. The results in this table measure the classification accuracy on a holdout test set and are presented as the difference, Δ, to the normal full-precision mode, O0. It can be seen that the classification result deviates only very slightly, by a maximum of 0.5%, across all cards for opt levels O1 and O2 – meaning that both of these mixed-precision training modes can be used without adversely affecting predictive performance. However as expected, the use of O3, complete 16-bit mode, causes a significant drop in accuracy of $\approx 64\%$ across all GPUs tested. The results also show that the padding applied to the graph has no significant impact on model accuracy as the maximum difference in accuracy between the use and the exclusion of padding is 1%.

GPU	Opt Level	Δ Accuracy	
		w/ Padding	w/o Padding
V100	O1	+ 0.005	- 0.005
	O2	+ 0.005	- 0.003
	O3	- 0.642	- 0.648
Titan RTX	O1	+ 0.001	+ 0.002
	O2	+ 0.003	+ 0.004
	O3	- 0.647	- 0.646
P100	O1	+ 0.002	- 0.001
	O2	+ 0.004	0
	O3	- 0.642	- 0.637

Table 6.2: Comparison of the GCN classification results on the Cora dataset using vertex features across GPUs and optimisation levels. All elements indicate the difference Δ between the values from Ox and O0.

GPU	Opt Level	w/ Padding		w/o Padding	
		Δ AUC	Δ AP	Δ AUC	Δ AP
V100	O1	- 0.005	- 0.006	+ 0.001	- 0.005
	O2	0	0	+ 0.001	0
	O3	- 0.277	- 0.195	- 0.276	- 0.194
Titan RTX	O1	- 0.005	- 0.007	- 0.005	- 0.006
	O2	0	0	0	0
	O3	- 0.277	- 0.195	- 0.276	- 0.194
P100	O1	- 0.005	- 0.007	- 0.005	- 0.006
	O2	0	0	0	0
	O3	- 0.277	- 0.195	- 0.276	- 0.194

Table 6.3: Comparison of GAE edge prediction results on the Cora dataset using vertex features across GPUs and optimisation levels. All elements indicate the difference Δ between the values from Ox and O0.

The results for the task of link prediction using the GAE model are presented in Table 6.3. The results largely conform to those presented for the GCN, with the use of O1 and O2 having no significant impact on predictive performance versus the full precision baseline, and O3 causing significant degradation. Again, it can be seen that the use of padding does not impact performance.

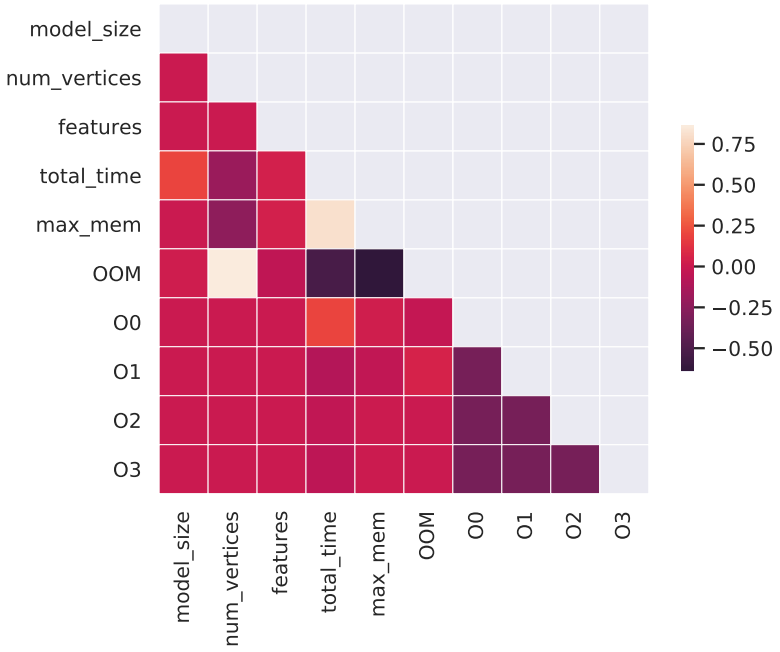
6.4.2 Run-Time and Memory Usage Analysis

This section presents the results measuring how run-time and memory usage change as both the size of the input Barabási-Albert graph and the model size are altered. For all results in this section, unless otherwise stated, the identity matrix of the

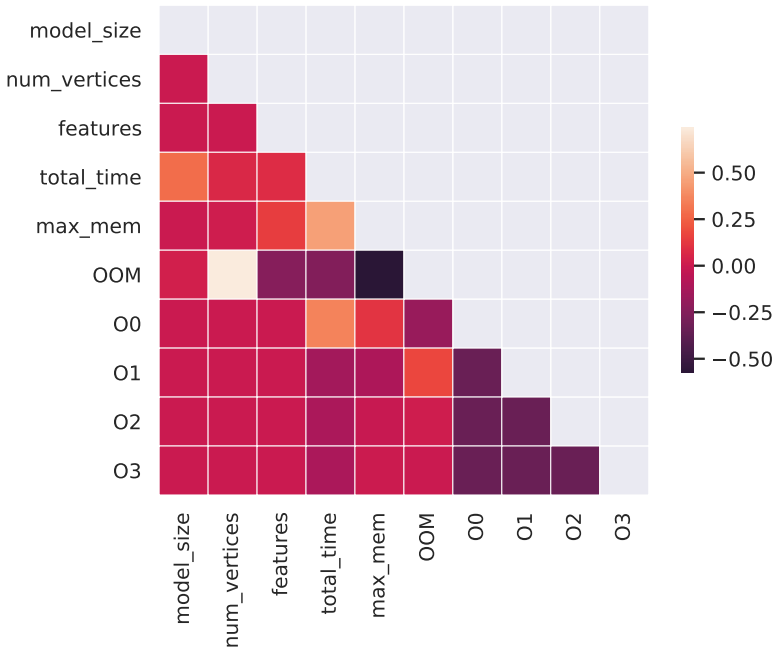
graph is used as the input features. Additionally, the figures are presented as the mean over five model seeds and either all model sizes, for the case of the graph size plots, or all graph sizes, for the case of the model size plots. Error bars are presented as the standard deviation of these.

To give an overview of how the various factors are related, Figure 6.2 presents a correlation matrix for the run-time experiments for both the GCN and GAE models running on the V100 GPU. Some unsurprising observations can be made across both model types, for example, the positive correlation between the model size and the total training time, ≈ 0.5 for GAE and ≈ 0.25 for GCN. However, some perhaps unexpected ones also arise - there is a clear positive correlation, for the GCN approach, of ≈ 0.25 between the training run being killed because of an Out Of Memory (OOM) error and the use of opt level O1.

Measuring Run-Time and Memory Usage Versus Graph Size

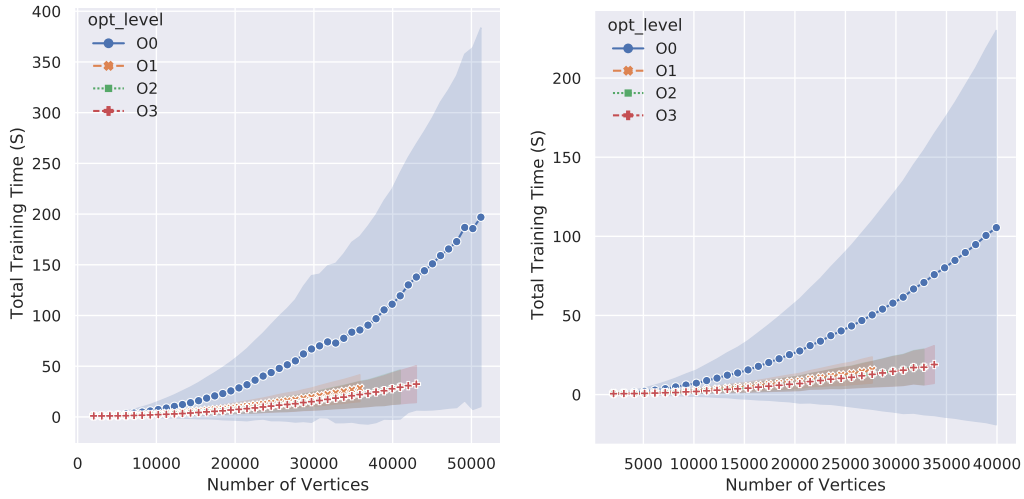


(a) GAE



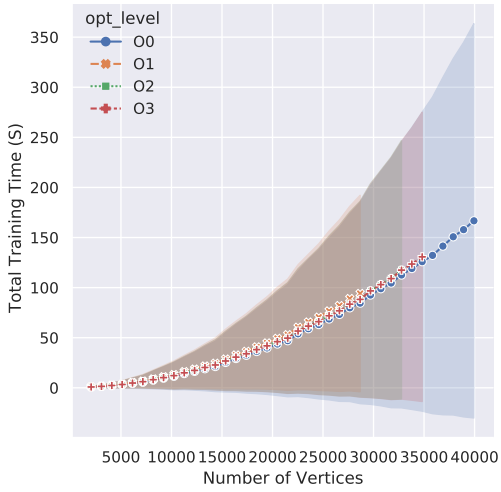
(b) GCN

Figure 6.2: Correlation of run-time experiment on the GCN and GAE models using the V100.



(a) Titan RTX

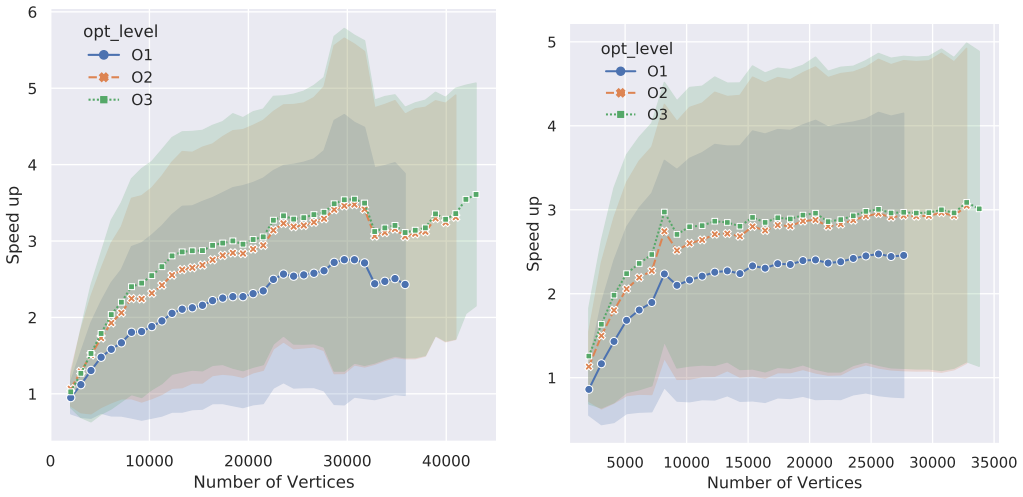
(b) V100



(c) P100

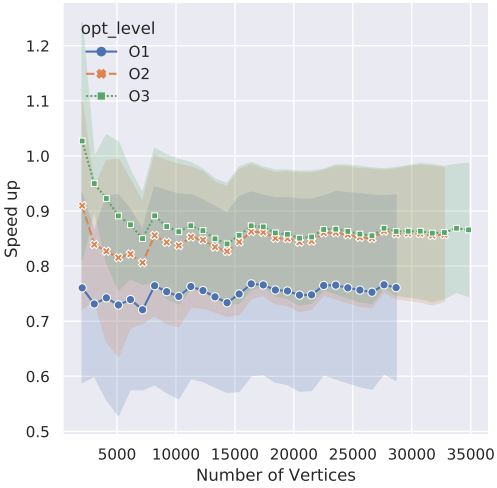
Figure 6.3: GCN total training time versus increases in graph size.

Now the focus shifts to measuring how the run-time and memory usage are affected as the number of vertices in the input training graph is changed. Figure 6.3 demonstrates how the run-time of the GCN model responds as the number of vertices in the input graph is increased. Firstly, by comparing between the two cards with Tensor Cores (Figures 6.3a and 6.3b) and the one without dedicated 16-bit hardware (Figure 6.3c), the effect at opt levels O1-O3 is immediately obvious, with respect to the observed total training time and variation across multiple runs. The P100 results in Figure 6.3c show the opt level to have almost no impact on the training time, whereas the other cards show a clear decrease in run-time whenever a 16-bit mode is enabled. One interesting observation is that the standard deviation (error bar) of each point is much lower for the 16-bit opt levels. As each point is presented as the mean over seeds and all the model sizes for that graph size, this would indicate that opt levels O1-O3 are less sensitive to model size when compared to the graph size. This will be investigated further in the next section when changes with respect to model size are studied. Perhaps the most surprising result is that opt level O0, the full 32-bit training model, can scale to larger graph sizes than the other levels. This mode was able to process graphs of up to 50k vertices where the other modes never successfully processed an input with more than 42.5k vertices. This is of note as intuitively one would expect that the reduced-precision modes would use less memory, and thus to scale to a larger input dataset size – however, this is clearly not the case. Further evidence of this will be presented when memory usage is considered.



(a) Titan RTX

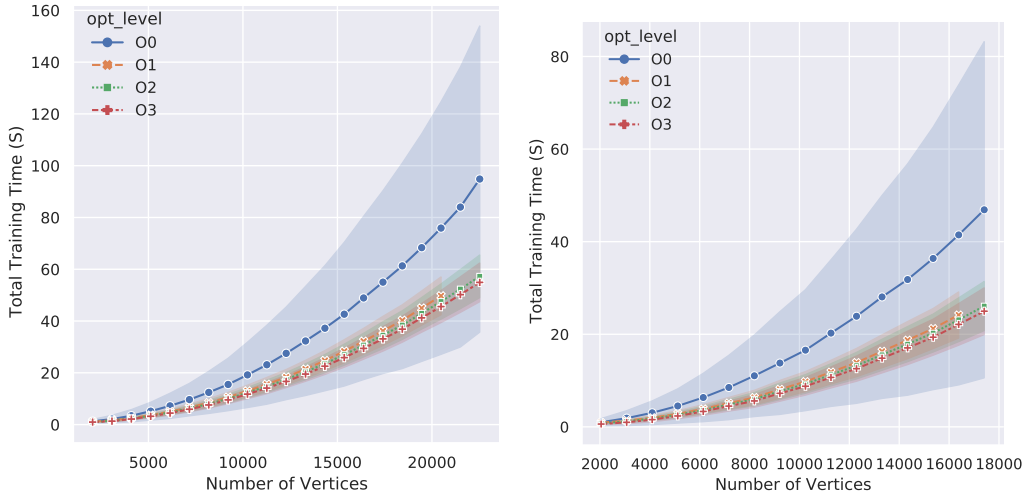
(b) V100



(c) P100

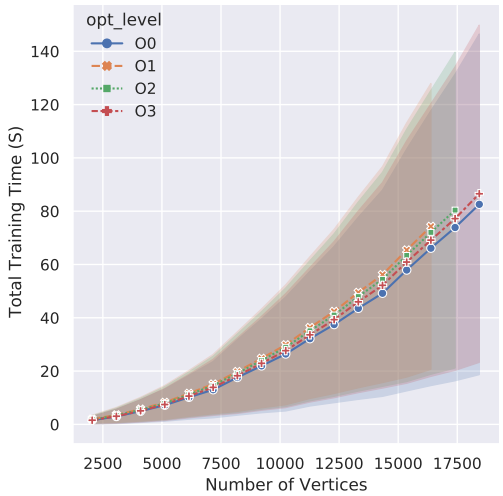
Figure 6.4: Speed up of the various opt levels versus O0 for the GCN approach.

To further investigate how various opt levels affect performance relative to the baseline of full-precision, the speed up of all opt levels relative to O0 for the GCN approach in Figure 6.4 are presented. The figure highlights how mixed-precision can offer large speed-ups for graph-based neural models, with the proviso that the GPU has dedicated hardware support for the operations. It can be seen that the best speed up figures are provided by opt levels O2 and O3, achieving maximum figures of ≈ 3.5 and ≈ 3 for the Titan RTX and the V100 respectively. However, opt level O1 trails behind these figures and also has a lower maximum number of vertices, as evidenced by the absence of data along the x-axis for this opt level. Figure 6.4c demonstrates that using mixed precision when the GPU is lacking the 16-bit specific hardware can actually result in a worse run-time overall, as illustrated by the speed up value of below 1 across all opt levels.



(a) Titan RTX

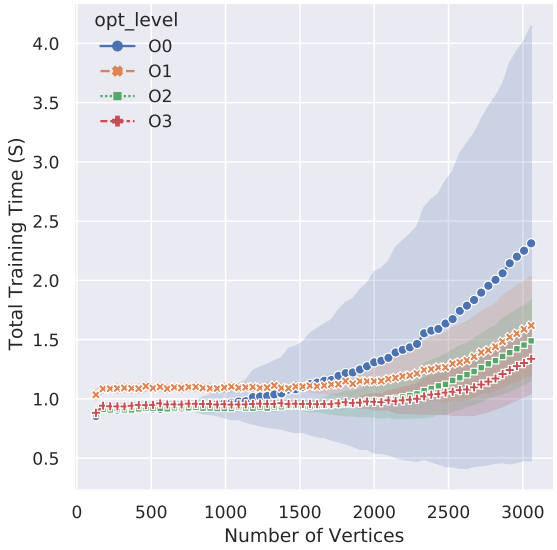
(b) V100



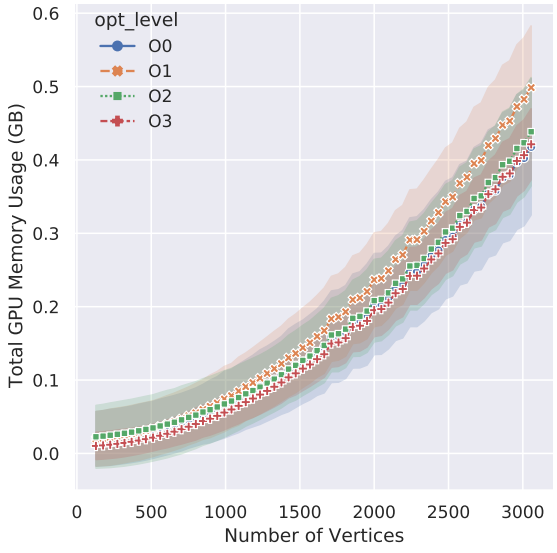
(c) P100

Figure 6.5: GAE total training time versus increases in graph size.

It is important to highlight how the GAE model is affected by the various optimisation levels, with Figure 6.5 showing the change in total training time versus the input graph size. It can again be seen in Figure 6.5c that using a mixed-precision training mode offers no run-time benefit if the GPU lacks dedicated hardware support. However, one clear trend is that, when compared to the GCN model, the GAE does not show the same level of decrease in run-time when mixed-precision training is utilised. This is demonstrated by opt level O0 being much closer to the mixed-precision modes, although it is still significantly higher. Another continuing trend, however, is the much lower run-time variance over model sizes for the mixed-precision approaches. To investigate at what graph size the mixed-precision modes start to outperform the baseline, a truncated view of the GAE results for the V100 is presented in Figure 6.6a. The figure shows that at a graph size of 1,000 vertices, opt levels O2 and O3 start to outperform O0, with O1 also outperforming it when a count of 1,500 vertices is reached.

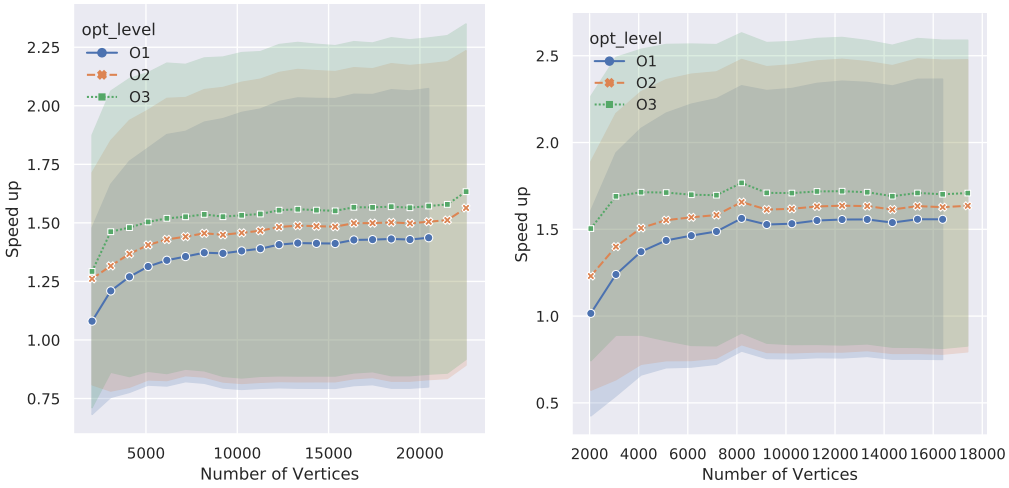


(a) Total Time



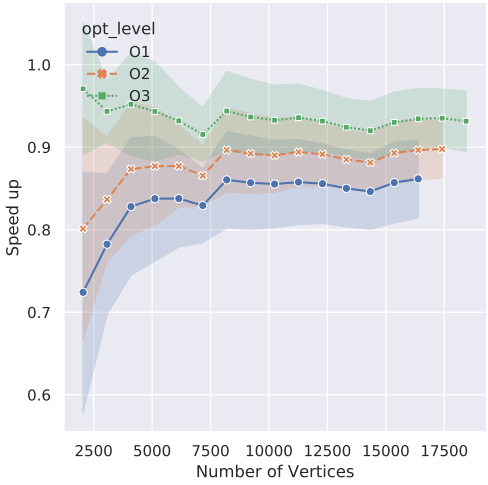
(b) Mem Usage

Figure 6.6: A truncated view of the GAE results for the V100.



(a) Titan RTX

(b) V100

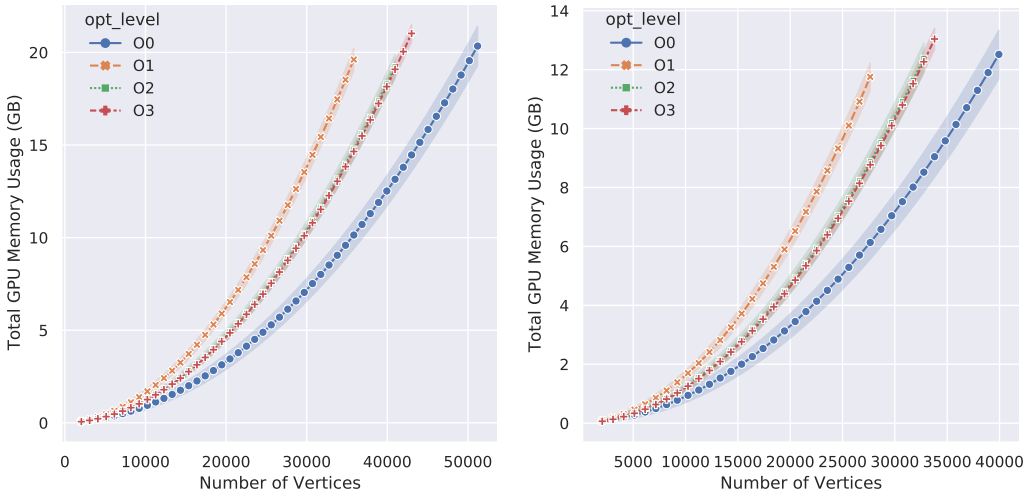


(c) P100

Figure 6.7: Speed up of the various opt levels versus O0 for the GAE approach.

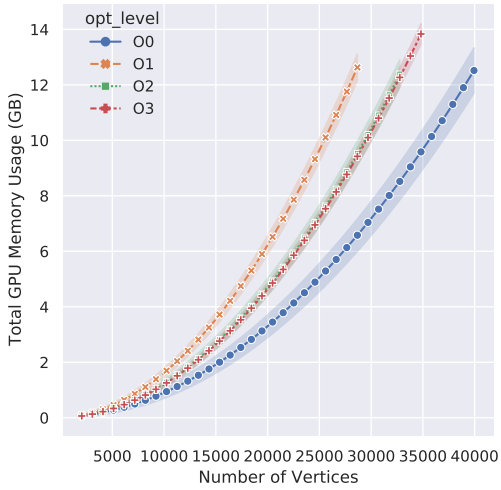
Figure 6.7 highlights the speed up from using mixed-precision modes with the GAE model. It shows that the speed-up is on average less than what was shown with the GCN approach. This is most likely due to the much more complex graph reconstruction loss function required by the GAE approach, as this may not benefit much from the use of reduced precision. Another interesting trend in the figure is that the speed-up is much more consistent across graph sizes, with the average speed up being almost identical from 4,000 to 18,000 vertices.

These experiments also include an analysis of how the change in the input graph size affects the maximum memory consumed on the GPU during the training process, with results across the three GPUs being presented in Figure 6.8. There is one clear and perhaps unintuitive result demonstrated across all cards - *using a reduced-precision mode of any kind results in more memory usage when compared to full precision for a given graph size.* The result explains the earlier observation that using a reduced-precision mode means that a smaller total graph size can run when compared to opt level O0 – they were running out of available memory sooner. It is interesting to note that the results are highly consistent across all cards, demonstrating that even when offering no performance benefit on the P100, the reduced-precision modes O1-O2 are still consuming the additional memory.



(a) Titan RTX

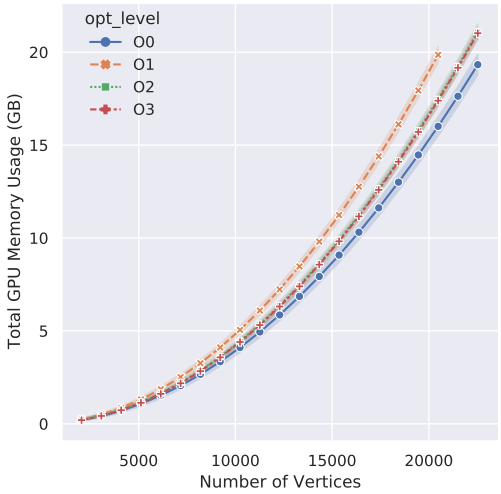
(b) V100



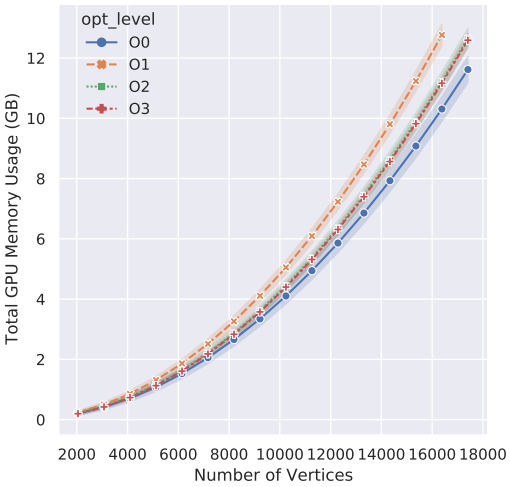
(c) P100

Figure 6.8: Maximum amount of GPU memory consumed during the training process across all cards for the GCN model.

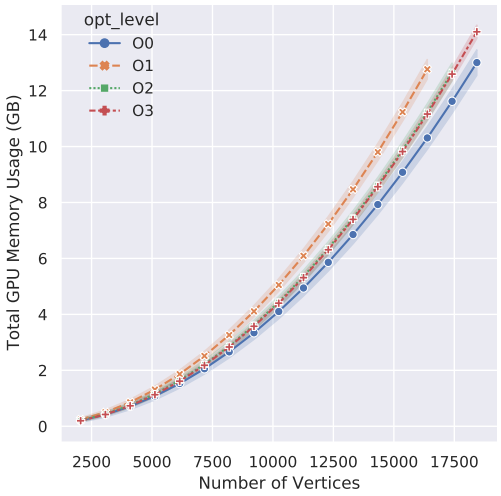
Figure 6.9 highlights how the maximum memory required for the various opt levels changes with respect to the number of vertices in the input graph. The figure highlights how, as was true for the run-time, the mixed-precision opt levels are much closer together with respect to the baseline here. It also re-emphasises how similar the pattern of memory usage is across the various cards – even on the P100 GPU. To analyse at what point the memory usage of the mixed-precision approaches starts to increase, Figure 6.6b presents a truncated view of the memory usage for the V100 GPU. The figure shows by a graph size of 1,000 vertices, opt level O1 is starting to demonstrate more memory usage than the other optimisation levels.



(a) Titan RTX



(b) V100

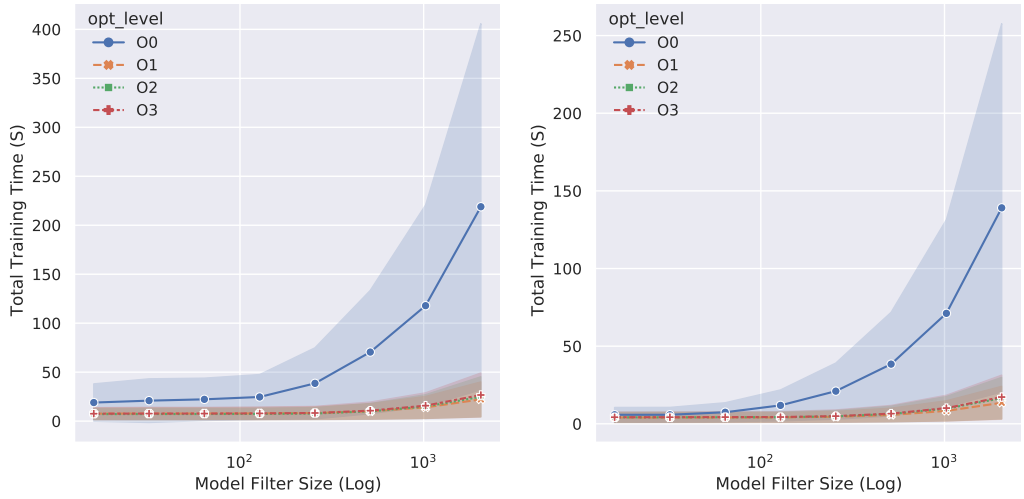


(c) P100

Figure 6.9: GAE max memory usage versus increases in graph size.

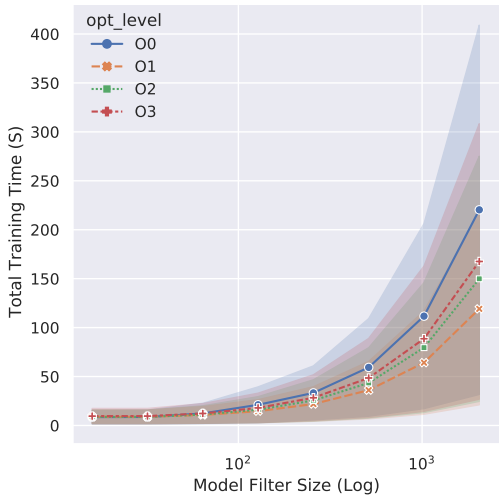
Measuring Run-Time and Memory Usage Versus Model Size

Results demonstrating how the performance and memory consumption is affected by the model size are presented here. Figure 6.10 shows the increases in model sizes against the total training time. Conforming to the trend established earlier, not using reduced precision results in large increases in run-time as larger model sizes are used. Conversely, any use of reduced precision means that the run-time is largely unaffected by increases in the number of model parameters – a very interesting observation. Figure 6.11 demonstrates the speed up of the various reduced-precision optimisation levels against the full-precision baseline. The figure shows that the potential speed up by using reduced precision continues to increase as large model sizes are used, indeed the speed-up has not plateaued even with the largest size used. This suggests that further experiments could be run to determine at what point the speed up no longer increases.



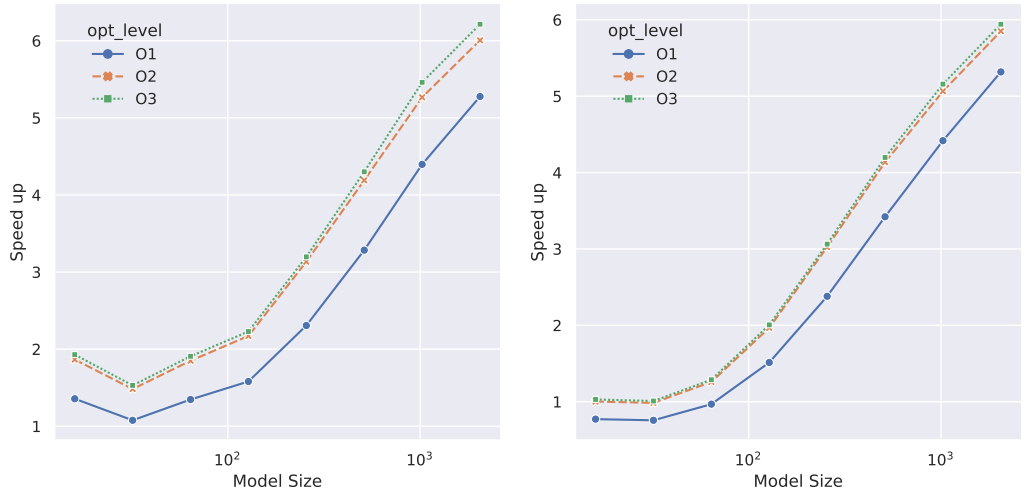
(a) Titan RTX

(b) V100

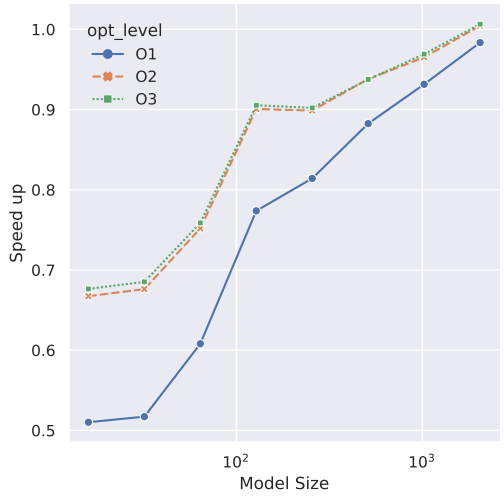


(c) P100

Figure 6.10: Total training time as the model size is increased for the GCN model.



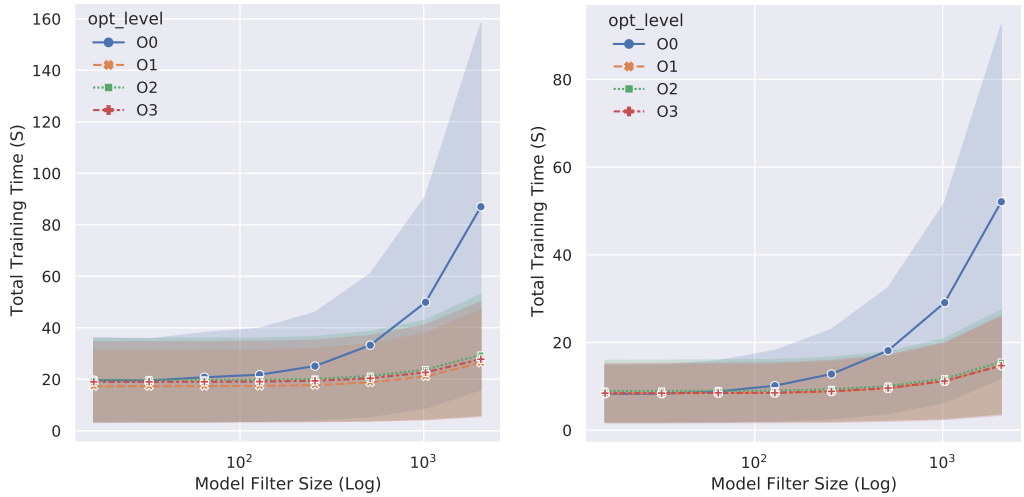
(a) Titan RTX (b) V100



(c) P100

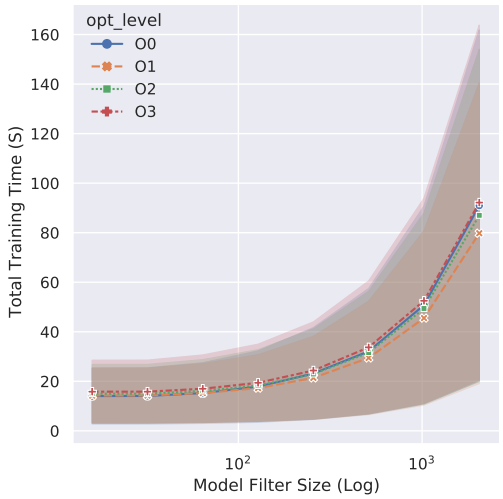
Figure 6.11: Speed up of the various opt levels versus O0 for the GCN approach. Results presented using the large graph size that was able to complete with all model sizes.

Figure 6.12 illustrates how various GPUs scale across model sizes in the GAE approach. The overall trend is similar to that of the GCN approach, with the full-precision mode demonstrating a large and sharp increase in run-time as larger model sizes are reached. However, one key difference is the larger variance displayed at each point, meaning the run-time for the GAE approach is more sensitive to the input graph size. Figure 6.13 shows the speed up versus the full-precision baseline for all cards. Two interesting trends can be observed from the figure: firstly, the difference in speed up between the various optimisation levels is reduced here versus the GCN results, secondly the speed up is overall less for a given model size than was shown for the GCN result. These results suggest that, due to the complex graph reconstruction method used for the model optimisation, GAE type models are more sensitive to the input graph size than the overall model complexity.



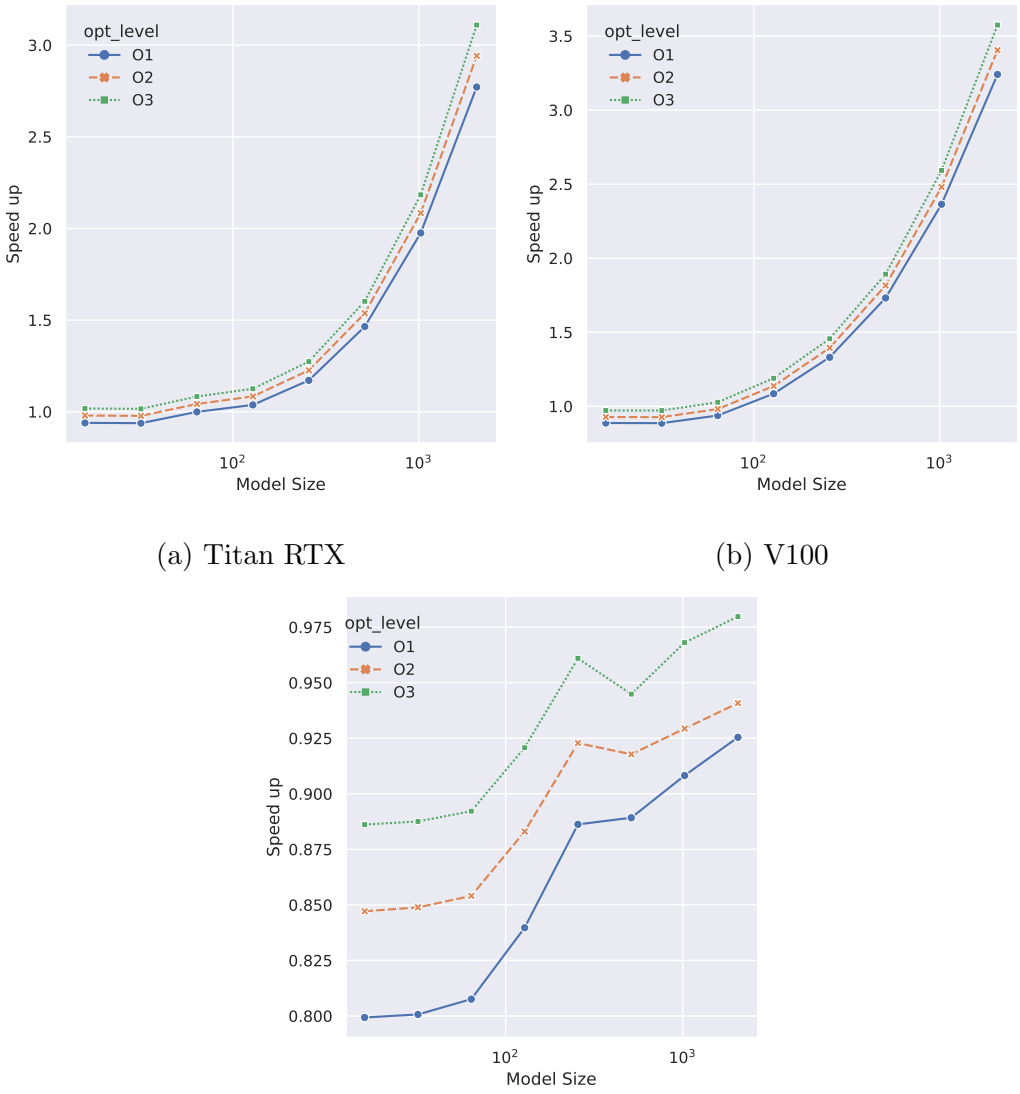
(a) Titan RTX

(b) V100



(c) P100

Figure 6.12: Total training time as the model size is increased for the GAE model.



(a) Titan RTX

(b) V100

(c) P100

Figure 6.13: Speed up of the various opt levels versus O0 for the GAE approach as model size increases.

6.5 Conclusion

This chapter provides a clear and detailed analysis of the impact of using reduced-precision computation and specialised GPU hardware designed for such operations on graph-based neural networks. Tensor Cores, introduced in modern NVIDIA GPU architectures, are capable of enabling mixed-precision operations by dynamically adapting calculations to accelerate throughput while preserving accuracy. While the effects of these improvements have been thoroughly explored in various facets of machine learning, such as computer vision and natural language processing, definitive literature on graph convolutional neural networks, which could theoretically benefit from reduced precision and Tensor Cores, is sparse. In this vein, comprehensive experiments are performed to evaluate the effects of using mixed-precision training on the predictive performance of both the semi-supervised classification and link prediction tasks in graph neural networks. The change in both run-time and the maximum memory consumed on the GPU is also measured as the number of vertices in the input graph and the model size is increased for the various levels of reduced-precision optimisation. As expected, the experiments demonstrate that using reduced-precision optimisation modes, taking advantage of Tensor Cores, reduce run-time for all models training by a significant margin. This points to the great advantage that reduced-precision and Tensor Cores can provide, considering certain layer parameters are divisible by 8. As for memory usage, the experiments indicate an adverse impact of using automatic mixed precision on graph convolutional networks. Using mixed-precision (O1,O2) increases memory usage compared to using full-precision (O0) or half-precision (O3) operations. In terms of the predictive performance of the models, it is observed that using complete half-precision (O3) fully hampers the learning process and unsurprisingly leads to a total model collapse in terms of accuracy, as shown in Table 6.2 and Table 6.3. Using mixed-

precision (O1,O2) indicates no significant change in performance compared to the full-precision mode (O0).

While this work has been primarily focused on NVIDIA Apex enabling automatic mixed-precision via PyTorch, other frameworks and libraries making use of various other forms of reduced precision need to be fully investigated, which would be an interesting trajectory for future work.

Chapter 7

Conclusions and Future Work

7.1 Conclusions

The use of graphs to provide a structured representation of data encompassing complex relationships is of interest across a range of domains. It has been demonstrated that many of the analysis methods used for extracting structural and spacial information from graphs struggle to scale well.

The remainder of this chapter will be presented as sections covering each of the original research objectives presented in Section 1.3.

7.1.1 Objective 1 - Development of a simple mechanism for ingesting graph data from its many forms.

The first piece of work described in this thesis, discussed in Chapter 3 describes SemNetCon. Providing an intuitive tool allowing network information to be extracted from disparate data sources and stored in a choice of data formats. This piece of work, which has been made available to the wider community via GitHub publishing as open source, has provided a simple mechanism for ingesting graph data from many different source formats. SemNetCon allows enables automatic construction

of graph datasets in a way that was not previously possible.

7.1.2 Objective 2 - An investigation of the optimal ways for comparing the similarity between graphs.

In Chapter 4 an investigation into how the comparison of similarity between graphs is conducted. The work presented in this chapter has demonstrated that the use of dimensionality reduction and clustering on derived graph fingerprints is an appropriate method for identifying similarity between graphs, providing a scalable comparison of graph structures. Delivering an overall accuracy of 99%, the graph fingerprint and t-SNE approach shows a significant improvement over traditional methods and provides an efficient method for clustering very large and complex graphs. While it is noted that this method has a significantly higher runtime than comparable methods, results are provided in a timely manner with more useful results than other methods. While t-SNE is very good at clustering data it does not actually classify the clusters. Further work in this area would be beneficial to extend the t-SNE algorithm to allow for the identification of cluster membership for each group.

7.1.3 Objective 3 - Exploration of the optimal ways for identifying sub-graphs with common features within a larger graph.

Further, Chapter 5 of this work has presented an exploration of the optimal ways for identifying sub-graphs with common features within a larger graph or, more simply, Community Detection. This has demonstrated that ego-nets extracted from graphs can de used to improve the scalability of graph focussed machine learning models. The presented method removes the need to store an entire graph in GPU memory

(which significantly reduces the size of graphs which can be handled), instead only requiring that a much smaller sub-graph, in the form of an ego-net, is available for each forward pass of the model. This means that any size of a graph can be processed on a GPU. It was also demonstrated that the use of an ego-net based approach performed favourably, in terms of accuracy, generally learning in far fewer epochs and producing similar final results to the baseline methods. The primary limitation of the approach presented here is the need to save every ego-net to disk. This could potentially be mitigated by constructing ego-nets as they are required, however, this is yet to be investigated. This work fulfils the original research objective of developing and evaluating methods for scalable machine learning methods for the identification of communities within graph structures.

7.1.4 Objective 4 - An evaluation of whether it is possible to reduce the amount of computing resources needed for processing large graphs.

Finally, in order to evaluate whether it is possible to reduce the amount of computing resources needed for processing large graphs, as discussed in Chapter 6. This work has provided an extensive analysis of the impact of using reduced-precision computation, leveraging specialised hardware in the form of Tensor cores. While this approach has been thoroughly evaluated in other domains, almost no work exists with respect to graph convolutional neural networks. The effects of reduced-precision training was evaluated at both vertex-level classification and link prediction tasks. It was demonstrated that using reduced-precision optimisation modes, taking advantage of Tensor Cores, was able reduce the training run-time for all approaches by a significant margin, generally providing a speed up value greater than 3. However, it was found that this approach is not without drawbacks. Experimentation

shows an adverse impact of using automatic mixed precision on graph convolutional neural networks. Mixed-precision has a higher memory consumption compared to full-precision or half-precision, due to duplicate matrices being held in memory. Regarding predictive performance it is observed that half-precision does not perform well but full-precision and mixed-precision provide very similar results.

7.2 Future Work

Whilst the work presented in this thesis has been successful in achieving the original research objectives, there is clear scope for future work to be undertaken.

1. A natural extension of the work presented in Chapter 4, taken together with the work in Chapter 5 is to investigate the use of ego-net derived fingerprints. Using this type of embedding at an ego-net level should allow for efficient community classification that generalises across graphs from different domains.

2. The work in Chapter 6 demonstrated that mixed-precision computation with graph neural networks yielded increased memory usage and much faster runtime while maintaining predictive accuracy. The use of an ego-net based approach in this context could mitigate the memory impact and allow mixed-precision to be used on much larger graphs than is currently possible.

3. A worthwhile extension to all the work presented here would be to investigate datasets beyond the citation networks used. Additional datasets from the Open Graph Benchmark [1] could provide a mechanism for a broader evaluation for this work.

[1] `https://ogb.stanford.edu/`

References

Akoglu, L., Tong, H., & Koutra, D. (2015). Graph-based Anomaly Detection and Description: A Survey. *Data Mining and Knowledge Discovery*, 626-688.

Antoniou, G., & Harmelen, F. V. (2008). *A semantic web primer, 2nd edition (cooperative information systems)* (2nd ed.). The MIT Press.

Arthur, D., & Vassilvitskii, S. (2007). k-means++: The advantages of careful seeding. In *Proceedings of the eighteenth annual acm-siam symposium on discrete algorithms* (pp. 1027—-1035). Society for Industrial and Applied Mathematics.

Barabási, A. L., & Albert, R. (1999). Emergence of scaling in random networks. *Science (New York, N.Y.)*, *286*(5439), 509–12.

Barabási, A. L., Albert, R., & Jeong, H. (2000). Scale-free characteristics of random networks: the topology of the world-wide web. *Physica A: Statistical Mechanics and its Applications*, *281*(1), 69–77.

Barabási, A. L., & Bonabeau, E. (2003). Scale-free networks. *Scientific American*(May), 50–59.

Barrat, A., Barthélemy, M., Pastor-Satorras, R., & Vespignani, A. (2004). The architecture of complex weighted networks. *Proceedings of the National Academy of Sciences of the United States of America*, *101*(11), 3747–3752.

Batagelj, V., & Mrvar, A. (2002). Pajek - Analysis and Visualization of Large Networks. *Lecture Notes in Computer Science*, *2265*, 477+.

Berkhin, P. (2006). A Survey of Clustering Data Mining Techniques. In *Grouping multidimensional data* (pp. 25–71). Berlin/Heidelberg: Springer-Verlag.

Berlingerio, M., Koutra, D., Eliassi-Rad, T., & Faloutsos, C. (2012). NetSimile: A Scalable Approach to Size-Independent Network Similarity. *CoRR*.

Berners-Lee, T., Hendler, J., & Lassila, O. (2001). The Semantic Web. *Scientific American*, *284*(5), 34–43.

Biggs, N., Lloyd, E. K., & Wilson, R. J. (1976). *Graph Theory, 1736-1936*. Oxford: Oxford University Press.

Blandford, D. K., Blelloch, G. E., & Kash, I. A. (2004). An experimental analysis of a compact graph representation. *Workshop on Algorithms Engineering and Experiments (ALENEX)*, 106.

Bollob'as, B. (1998). *Modern Graph Theory*. New York, New York, USA: Springer Science & Business Media.

Bonacich, P. (2007). Some unique properties of eigenvector centrality. *Social Networks*, *29*(4), 555–564.

Bonner, S., Atapour-Abarghouei, A., Jackson, P. T., Brennan, J., Kureshi, I., Theodoropoulos, G., ... Obara, B. (2019). Temporal neighbourhood aggregation: Predicting future links in temporal graphs via recurrent variational graph convolutions. In *2019 IEEE international conference on big data* (pp. 5336–5345).

Bonner, S., Brennan, J., Kureshi, I., Theodoropoulos, G., & McGough, A. S. (2016). Efficient Comparison of Massive Graphs Through The Use Of 'Graph Fingerprints'. In *Twelfth workshop on mining and learning with graphs (MLG) workshop at kdd'16*. ACM.

Bonner, S., Brennan, J., Kureshi, I., Theodoropoulos, G., McGough, A. S., & Obara, B. (2017). Evaluating the quality of graph embeddings via topological feature reconstruction. In *IEEE international conference on big data* (pp. 2691–2700).

Bonner, S., Brennan, J., Kureshi, I., Theodoropoulos, G., McGough, A. S., & Obara, B. (2018). Temporal graph offset reconstruction: Towards temporally robust graph representation learning. In *2018 IEEE international conference on big data* (pp. 3737–3746).

Bonner, S., Brennan, J., Theodoropoulos, G., Kureshi, I., & McGough, A. S. (2016). GFP-X: A parallel approach to massive graph comparison using SPARK. In *2016 IEEE international conference on big data (big data)* (p. 3298-3307).

Bonner, S., Brennan, J., Theodoropoulos, G., Kureshi, I., & McGough, A. S. (2017). Deep topology classification: A new approach for massive graph classification. *Proceedings - 2016 IEEE International Conference on Big Data, Big Data 2016*, 3290–3297.

Bonner, S., Kureshi, I., Brennan, J., & Theodoropoulos, G. (2017). Chapter 14. exploring the evolution of big data technologies. In I. Mistrik, R. Bahsoon, N. Ali, M. Heisel, & B. Maxim (Eds.), *Software architecture for big data and the cloud* (p. 253-283). Boston: Morgan Kaufmann.

Bonner, S., Kureshi, I., Brennan, J., Theodoropoulos, G., Mcgough, A. S., & Obara, B. (2018). Exploring the semantic content of unsupervised graph embeddings: An empirical study. *Data Science and Engineering*, *4*(3), 269-289.

Bonner, S., Mcgough, A. S., Kureshi, I., Brennan, J., Theodoropoulos, G., Moss, L., ... Antoniou, G. (2015). Data quality assessment and anomaly detection via map / reduce and linked data: A case study in the medical domain. In *2015 IEEE international conference on big data (Big Data)* (p. 737-746).

Brandes, U., Robins, G., McCranie, A., & Wasserman, S. (2013). What is network science? *Network Science*, *1*(01), 1–15.

Brennan, J., Bonner, S., Atapour-Abarghouei, A., Jackson, P. T., Obara, B., & McGough, A. S. (2020). Not half bad: Exploring half-precision in graph convolutional neural networks. In *2020 IEEE international conference on big*

data (Big Data). IEEE.

Bruna, J., Zaremba, W., Szlam, A., & LeCun, Y. (2013).

In *Spectral Networks and Locally Connected Networks on Graphs.*

Carling, K., & Meng, X. (2015). Confidence in heuristic solutions? *Journal of Global Optimization*, *63*, 381-399.

Clauset, A., Newman, M., & Moore, C. (2004). Finding community structure in very large networks. *Physical Review E*, *70*(6), 1–6.

Courbariaux, M., Bengio, Y., & David, J.-P. (2015). Training deep neural networks with low precision multiplications. In *International conference on learning representations, (ICLR).*

Das, D., Mellempudi, N., Mudigere, D., Kalamkar, D., Avancha, S., Banerjee, K., ... Pirogov, V. O. (2018). Mixed precision training of convolutional neural networks using integer operations. In *International conference on learning representations (ICLR).*

Defferrard, M., Bresson, X., & Vandergheynst, P. (2016). Convolutional neural networks on graphs with fast localized spectral filtering. In *Proceedings of the 30th international conference on neural information processing systems NeurIPS.*

Ding, C., & He, X. H. X. (2002). Cluster merging and splitting in hierarchical clustering algorithms. *2002 IEEE International Conference on Data Mining, 2002. Proceedings.*, 1–8.

Eiglsperger, M., Brandes, U., Lerner, J., & Pich, C. (2013). Graph Markup Language (GraphML). *Handbook of Graph Drawing and Visualization*, 517–541.

Erdős, P., & Renyi, A. (1984). The Evolution of Random Graphs. *Transactions of the American Mathematical Society*, *286*(1), 257.

Euler, L. (1741). Solutio problematis ad geometriam situs pertinentis. *Commentarii academiae scientiarum Petropolitanae*, *8*, 128–140.

Foltz, P. W., Lavoie, N., & Oberbreckling, Robert J Rosenstein, M. B. (2005). *Network science*. Washington, D.C.: The National Acedemies Press.

Ginsburg, B., Nikolaev, S., & Micikevicius, P. (2017). Training of deep networks with half precision float. In *Nvidia gpu tech conf.*

Goodfellow, I., Bengio, Y., & Courville, A. (2016). *Deep learning*. MIT Press. (`http://www.deeplearningbook.org`)

Gupta, S., Agrawal, A., Gopalakrishnan, K., & Narayanan, P. (2015). Deep learning with limited numerical precision. In *International conference on machine learning* (pp. 1737–1746).

Hamilton, W., Ying, Z., & Leskovec, J. (2017a). Inductive representation learning on large graphs. In *Advances in neural information processing systems* (Vol. 30, pp. 1–19). Curran Associates, Inc.

Hamilton, W., Ying, Z., & Leskovec, J. (2017b). Inductive representation learning on large graphs. In *Advances in neural information processing systems* (pp. 1024–1034).

Himsolt, M., & Passau, U. (1996). GML : A portable Graph File Format. *Syntax*, 1–11.

Hinton, G. E., & Roweis, S. T. (2002). Stochastic neighbor embedding. *Advances in neural information processing systems*, 833–840.

Hubara, I., Courbariaux, M., Soudry, D., El-Yaniv, R., & Bengio, Y. (2017). Quantized neural networks: Training neural networks with low precision weights and activations. *The Journal of Machine Learning Research*, *18*(1), 6869–6898.

Jolliffe, I. (n.d.). Principal Component Analysis. In *Wiley statsref: Statistics reference online*. Chichester, UK: John Wiley & Sons, Ltd.

Jolliffe, I., Pelleg, D., Pelleg, D., Moore, A. W., Moore, A. W., Ward, J. H., ... Cai, Z. (2011). Modern hierarchical, agglomerative clustering algorithms. *IEEE Transactions on Cybernetics*, *45*(3), 430–443.

Justus, D., Brennan, J., Bonner, S., & Mcgough, A. S. (2018). Predicting the computational cost of deep learning models. In *IEEE international conference on big data* (p. 3873-3882).

Karp, R. M. (1972). Reducibility among combinatorial problems. In R. E. Miller, J. W. Thatcher, & J. D. Bohlinger (Eds.), *Complexity of computer computations* (pp. 85–103). Boston, MA: Springer US.

Kernighan, B. W., & Lin, S. (1970). An efficient heuristic procedure for partitioning graphs. *The Bell System Technical Journal*, *49*(2), 291-307.

Kipf, T. N., & Welling, M. (2016). Variational graph auto-encoders. *NIPS Workshop on Bayesian Deep Learning*.

Kipf, T. N., & Welling, M. (2017). Semi-supervised classification with graph convolutional networks. In *International conference on learning representations (ICLR)*.

Köster, U., Webb, T., Wang, X., Nassar, M., Bansal, A. K., Constable, W., ... Rao, N. (2017). Flexpoint: An adaptive numerical format for efficient training of deep neural networks. In *Adv Neural Inf Process Syst* (pp. 1742–1752).

Koutra, D., Parikh, A., Ramdas, A., & Xiang, J. (2011). Algorithms for Graph Similarity and Subgraph Matching. In *Proc. ecol. inference conf 17*.

Kriege, N. M., Johansson, F. D., & Morris, C. (2020). A survey on graph kernels. *Applied Network Science*, *5*, 1-42.

Kuchaiev, O., Ginsburg, B., Gitman, I., Lavrukhin, V., Li, J., Nguyen, H., ... Micikevicius, P. (2018). Mixed-precision training for NLP and speech recognition with OpenSeq2Seq. *arXiv:1805.10387*.

Kullback, S., & Leibler, R. A. (1951). On information and sufficiency. *The Annals of Mathematical Statistics*, *22*(1), 79–86.

Leskovec, J., Kleinberg, J., & Faloutsos, C. (2007). Graph evolution: Densification and shrinking diameters. *ACM Transactions on Knowledge Discovery from*

Data, *1*(1), 2–43.

Leskovec, J., & Krevl, A. (2014). *SNAP Datasets: Stanford Large Network Dataset Collection.* Retrieved from `http://snap.stanford.edu/data` (Accessed June 12, 2017)

Leskovec, J., & Sosič, R. (2016). SNAP. *ACM Transactions on Intelligent Systems and Technology*, *8*(1), 1–20.

Li, G., Semerci, M., & Yener, B. (2011). Graph Classification via Topological and Label Attributes. *MLG*.

Lloyd, S. P. (1982). Least Squares Quantization in PCM. *IEEE Transactions on Information Theory*, *28*(2), 129–137.

Madhulatha, T. (2012). An overview on clustering methods. *Journal Of Engineering*, *2*(4), 719–725.

McCune, R. R., Weninger, T., & Madey, G. (2015). Thinking Like a Vertex: a Survey of Vertex-Centric Frameworks for Distributed Graph Processing. *ACM Comput. Surv.*, *46556*.

Mcgough, A. S., Forshaw, M., Brennan, J., Al Moubayed, N., & Bonner, S. (2018). Using machine learning to reduce the energy wasted in volunteer computing environments. In *2018 ninth international green and sustainable computing conference (IGSC)* (p. 1-8).

Mellempudi, N., Srinivasan, S., Das, D., & Kaul, B. (2019). Mixed precision training with 8-bit floating point. In *Relational representation learning workshop, NeurIPS.*

Micikevicius, P., Narang, S., Alben, J., Diamos, G., Elsen, E., Garcia, D., ... Wu, H. (2018). Mixed precision training. In *International conference on learning representations (ICLR).*

Milenković, T., Lai, J., & Przulj, N. (2008). GraphCrunch: a tool for large network analyses. *BMC bioinformatics*, *9*, 70.

Müllner, D. (2011). Modern hierarchical, agglomerative clustering algorithms. *CoRR*.

Newman, M. E. J. (2003). The structure and function of complex networks. *Dialogues in clinical neuroscience*, *45*, 167–256.

Newman, M. E. J. (2004). Fast algorithm for detecting community structure in networks. *Physical Review E - Statistical, Nonlinear, and Soft Matter Physics*, *69*(6 2), 1–5. doi: 10.1103/PhysRevE.69.066133

Newman, M. E. J. (2010). *Networks An Introduction*. Oxford: Oxford University Press.

Niepert, M., Ahmed, M., & Kutzkov, K. (2016). Learning Convolutional Neural Networks for Graphs. *ICML*.

NVIDIA Apex: Tools for easy mixed-precision training in PyTorch. (n.d.). https://developer.nvidia.com/blog/apex-pytorch-easy-mixed-precision-training/. (Accessed August 15, 2020)

NVIDIA deep learning performance. (2020). https://docs.nvidia.com/deeplearning/performance/. (Accessed August 15, 2020)

NVIDIA Tensor Cores: Unprecedented acceleration for HPC and AI. (n.d.). https://www.nvidia.com/en-us/data-center/tensor-cores/. (Accessed August 15, 2020)

NVIDIA Volta: The Tensor Core GPU architecture designed to bring AI to every industry. (n.d.). https://www.nvidia.com/en-us/data-center/volta-gpu-architecture/. (Accessed August 15, 2020)

Page, L., Brin, S., Motwani, R., & Winograd, T. (1998). *Technical report, Stanford Digital Library Technologies Project.* Stanford InfoLab.

Palla, G., Derényi, I., Farkas, I., & Vicsek, T. (2005). Uncovering the overlapping community structure of complex networks in nature and society. *Nature*,

435(7043), 814–818.

Papadopoulos, S., Kompatsiaris, Y., Vakali, A., & Spyridonos, P. (2012). Community detection in social media performance and application considerations. *Data Mining and Knowledge Discovery*, *24*(3), 515–554.

Parés, F., Gasulla, D. G., Vilalta, A., Moreno, J., Ayguadé, E., Labarta, J., ... Suzumura, T. (2018). Fluid communities: A competitive, scalable and diverse community detection algorithm. In C. Cherifi, H. Cherifi, M. Karsai, & M. Musolesi (Eds.), *Complex networks & their applications vi* (pp. 229–240). Cham: Springer International Publishing.

Paszke, A., Gross, S., Massa, F., Lerer, A., Bradbury, J., Chanan, G., ... others (2019). PyTorch: An imperative style, high-performance deep learning library. In *Advances in neural information processing systems* (pp. 8026–8037).

Prat-Pérez, A., Dominguez-Sal, D., Brunat, J. M., & Larriba-Pey, J.-L. (2012). Shaping communities out of triangles. *Proceedings of the 21st ACM international conference on Information and knowledge management - CIKM '12*, 1677.

Raghavan, N., Albert, R., & Kumara, S. (2007). Near linear time algorithm to detect community structures in large-scale networks. *Physical review. E, Statistical, nonlinear, and soft matter physics*, *76*, 036106.

Rastegari, M., Ordonez, V., Redmon, J., & Farhadi, A. (2016). XNOR-Net: ImageNet classification using binary convolutional neural networks. In *European conference on computer vision* (pp. 525–542).

Saltz, M., Prat-Pérez, A., & Dominguez-Sal, D. (2015). Distributed community detection with the WCC metric. In *Proceedings of the 24th international conference on world wide web companion, WWW* (pp. 1095–1100). ACM.

Schaeffer, S. E. (2007). Graph clustering. *Computer Science Review*, *1*(1), 27–64.

Shchur, O., Mumme, M., Bojchevski, A., & Günnemann, S. (2018). Pitfalls of

graph neural network evaluation. *Relational Representation Learning Workshop, NeurIPS*.

Shiokawa, H., Fujiwara, Y., & Onizuka, M. (2013). Fast Algorithm for Modularity-Based Graph Clustering. *Proceeding of the Twenty-Seventh Conference on Artificial Intelligence*, 1170–1176.

Shun, J., Roosta-Khorasani, F., Fountoulakis, K., & Mahoney, M. W. (2016). Parallel local graph clustering. *Proc. VLDB Endow.*.

Strogatz, S. H. (2001). Exploring complex networks. *Nature*, *410*(6825), 268–276.

Tan, P.-N., Steinbach, M., & Kumar, V. (2005). Chap 8 : Cluster Analysis: Basic Concepts and Algorithms. *Introduction to Data Mining*, Chapter 8.

Tipping, M. E., & Bishop, C. M. (1999). Probabilistic Principal Component Analysis. *Journal of the Royal Statistical Society: Series B (Statistical Methodology)*, *61*(3), 611–622.

Van Der Maaten, L. J. P., & Hinton, G. E. (2008). Visualizing high-dimensional data using t-sne. *Journal of Machine Learning Research*, *9*, 2579–2605.

Veličković, P., Cucurull, G., Casanova, A., Romero, A., Liò, P., & Bengio, Y. (2018). Graph Attention Networks. *International Conference on Learning Representations (ICLR)*.

W3C. (2016a). *Resource description framework (RDF).* Retrieved from `https://www.w3.org/RDF/` (Accessed March 4, 2016)

W3C. (2016b). *SPARQL Query Language for RDF.* Retrieved from `http://www.w3.org/TR/rdf-sparql-query/` (Accessed March 4, 2016)

Wang, M., Wang, C., Yu, J. X., & Zhang, J. (2015). Community Detection in Social Networks : An In-depth Benchmarking Study with a Procedure-Oriented Framework. *Proceedings of the VLDB Endowment*, 998–1009.

Wang, N., Choi, J., Brand, D., Chen, C.-Y., & Gopalakrishnan, K. (2018). Training deep neural networks with 8-bit floating point numbers. In *Adv Neural Inf*

Process Syst (pp. 7675–7684).

Ward, J. H. (1963). Hierarchical grouping to optimize an objective function. *Journal of the American Statistical Association*, *58*(301), 236–244.

Watts, D. J., & Strogatz, S. H. (1998). Collective dynamics of 'small-world' networks. *Nature*, *393*(6684), 440–442.

Wilcox, R. H. (1961). Adaptive control processes — A guided tour. *Naval Research Logistics Quarterly*, *8*(3), 315–316.

Xu, R. (2005). Survey of clustering algorithms for MANET. *IEEE Transactions on Neural Networks*, *16*(3), 645–678.

Yang, Z., Cohen, W., & Salakhudinov, R. (2016). Revisiting semi-supervised learning with graph embeddings. In *International conference on machine learning* (pp. 40–48).

Yano, A., & Wadayama, T. (2011). *Probabilistic Analysis of the Network Reliability Problem on a Random Graph Ensemble.*

Zhao, Y., & Karypis, G. (2002). Evaluation of hierarchical clustering algorithms for document datasets. In *Proceedings of the eleventh international conference on information and knowledge management - cikm '02* (p. 515). New York, New York, USA: ACM Press.