Advances in Learning and Understanding with Graphs through Machine Learning

STEPHEN ARTHUR ROBERT BONNER

Advances in Learning and Understanding with Graphs through Machine Learning

Stephen Arthur Robert Bonner

A thesis presented for the degree of
Doctor of Philosophy at Durham University

Department of Computer Science
Durham University
United Kingdom

11th October 2020

Advances in Learning and Understanding with Graphs through Machine Learning

Stephen Arthur Robert Bonner

Submitted for the degree of Doctor of Philosophy

Graphs have increasingly become a crucial way of representing large, complex and disparate datasets from a range of domains, including many scientific disciplines. Graphs are particularly useful at capturing complex relationships or interdependencies within or even between datasets, and enable unique insights which are not possible with other data formats. Over recent years, significant improvements in the ability of machine learning approaches to automatically learn from and identify patterns in datasets have been made.

However due to the unique nature of graphs, and the data they are used to represent, employing machine learning with graphs has thus far proved challenging. A review of relevant literature has revealed that key challenges include issues arising with macro-scale graph learning, interpretability of machine learned representations and a failure to incorporate the temporal dimension present in many datasets. Thus, the work and contributions presented in this thesis primarily investigate how modern machine learning techniques can be adapted to tackle key graph mining tasks, with a particular focus on optimal macro-level representation, interpretability and incorporating temporal dynamics into the learning process. The majority of methods employed are novel approaches centred around attempting to use artificial neural networks in order to learn from graph datasets.

Firstly, by devising a novel graph fingerprint technique, it is demonstrated that this can successfully be applied to two different tasks whilst out-performing established baselines, namely graph comparison and classification. Secondly, it is shown that a mapping can be found between certain topological features and graph embeddings. This, for perhaps the the first time, suggests that it is possible that machines are learning something analogous to human knowledge acquisition, thus bringing interpretability to the graph embedding process. Thirdly, in exploring two new models for incorporating temporal information into the graph learning process, it is found

that including such information is crucial to predictive performance in certain key tasks, such as link prediction, where state-of-the-art baselines are out-performed.

The overall contribution of this work is to provide greater insight into and explanation of the ways in which machine learning with respect to graphs is emerging as a crucial set of techniques for understanding complex datasets. This is important as these techniques can potentially be applied to a broad range of scientific disciplines. The thesis concludes with an assessment of limitations and recommendations for future research.

Declaration

The work in this thesis is based on research carried out within the Innovative Computing Group at the Department of Computer Science, Durham University, UK. No part of this thesis has been submitted elsewhere for any other degree or qualification, and it is all the author's work unless referenced to the contrary below.

Note on Publications Included in this Thesis

At the time of submission, three chapters of this thesis contain content which has been published at peer-reviewed conferences or journals.

Content from Chapter 3 has been published as the following works:

Stephen Bonner, John Brennan, Ibad Kureshi, Andrew Stephen McGough, and Georgios Theodoropoulos. Efficient comparison of massive graphs through the use of 'graph fingerprints'. In *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. *IEEE International Conference on Big Data*, pages 3298–3307, 2016

Stephen Bonner, John Brennan, Georgios Theodoropoulos, Ibad Kureshi, and Andrew Stephen McGough. Deep topology classification: A new approach for massive graph classification. In *IEEE International Conference on Big Data*, pages 3290–3297. IEEE, 2016

Content from Chapter 4 has been published as the following works:

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 *IEEE International Conference on Big Data*, pages 2691–2700. IEEE, 2017

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. *Data Science and Engineering*, 4(3):269–289, 2019

Content from Chapter 5 has been published as the following works:

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 *IEEE International Conference on Big Data*, pages 3737–3746. IEEE, 2018

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 *IEEE International Conference on Big Data*, 2019

These chapters are presented largely as submitted, although additional experimentation has been performed, as well as alterations made to the referencing and notation to improve consistency.

Note on Publications Not Included in this Thesis

As well as the above papers, the following works have been published during the period of research for this thesis; however, these publications do not fit into the narrative of this thesis and have not been included in the text.

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 *IEEE International Conference on Big Data*, pages 737–746. IEEE, 2015

Stephen Bonner, Ibad Kureshi, John Brennan, and Georgios Theodoropoulos. Exploring the evolution of big data technologies. In *Software Architecture for Big Data and the Cloud*, pages 253–283. Elsevier, 2017

Stephen Bonner and Flavian Vasile. Causal embeddings for recommendation. In *Proceedings of the 12th ACM Conference on Recommender Systems*, pages 104–112. ACM, 2018

Daniel Justus, John Brennan, Stephen Bonner, and Andrew Stephen McGough. Predicting the computational cost of deep learning models. In *IEEE International Conference on Big Data*, pages 3873–3882. IEEE, 2018

Nik Khadijah Nik Aznan, Stephen Bonner, Jason Connolly, Noura Al Moubayed, and Toby Breckon. On the classification of ssvep-based dry-eeg signals via convolutional neural networks. In *IEEE International Conference on Systems, Man, and Cybernetics*, pages 3726–3731. IEEE, 2018

David Rohde, Stephen Bonner, Travis Dunlop, Flavian Vasile, and Alexandros Karatzoglou. Recogym: A reinforcement learning environment for the problem of product recommendation in online advertising. *REVAL Workshop at the 12th ACM Conference on Recommender Systems*, 2018

Philip T Jackson, Amir Atapour-Abarghouei, Stephen Bonner, Toby P Breckon, and Boguslaw Obara. Style augmentation: Data augmentation via style randomization. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition Workshops*, pages 83–92, 2019

Nik Khadijah Nik Aznan, Amir Atapour-Abarghouei, Stephen Bonner, Jason Connolly, Noura Al Moubayed, and Toby Breckon. Simulating brain signals: Creating synthetic eeg data via neural-based generative models for improved ssvep classification. In *International Joint Conference on Neural Networks*, pages 1–8, 2019

Stephen Bonner and David Rohde. Latent variable session-based recommendation. *Second Symposium on Advances in Approximate Bayesian Inference*, 2019

Otmane Sakhi, Stephen Bonner, David Rohde, and Flavian Vasile. Reconsidering analytical variational bounds for output layers of deep networks. *Bayesian Deep Learning - Neural Information Processing Systems workshop*, 2019

Acknowledgements

Throughout the course of this work, I have benefited massively from the support and input of my supervisory team including Dr Boguslaw Obara, Dr Stephen McGough and Professor Georgios Theodoropoulos – my sincere and enduring gratitude to you all.

None of this would have even been possible without the amazing help of my parents (Dr Louise and Mr Neil Bonner) and family. I will be forever indebted to you for all the wise council and unwavering support throughout the years. Additionally, I was lucky enough to meet my partner, Nik Khadijah Nik Aznan, during the course of my studies - thank you for always being there for me.

I would also like to acknowledge some of my colleagues at Durham University who have provided invaluable help and support - in many different ways, throughout the years - Dr Ibad Kureshi, John Brennan, Dr Amir Atapour-Abarghouei, Dr Philip Jackson, Professor Toby Breckon, and Dr Chas Nelson.

During the course of this degree I was fortunate enough to spend two consecutive summers (2017 and 2018) in Paris as a research intern at Criteo AI Labs. I would like to thank some of the wonderful people I met and worked with there including Flavian Vasile, Ugo Tanielian, David Rhode, Sebastian Prillo-Rey, Travis Dunlop, Mike Gartrell, Suju Rajan and many more. Thank you all for making me feel so welcome and sharing your knowledge.

Finally, I would like to thank the Engineering and Physical Sciences Council, UK (Award number 1444075) for sponsoring this research. Additionally, I acknowledge the support of NVIDIA Corporation with the donation of the GPU used for parts of this research.


EPSRC

Contents

List of Figures

List of Tables

Chapter 1

Introduction

The volume and increasingly heterogeneous nature of data being generated by all aspects of modern society has been growing exponentially in recent years [139]. This data explosion encompasses not only the obvious everyday areas of social media and online commerce [225], but also various scientific domains, from disciplines as disparate as healthcare [188] and astronomy [250]. Indeed, the identification of patterns and trends from these massive datasets has been crucial in many recent important scientific discoveries [191]. As such, there has been tremendous research interest in how best to store, process and, most relevant for this thesis, identify patterns in these large complex datasets.

There are two primary goals that must be considered when attempting to infer new insights from a certain dataset, those being: how best to represent the data to make any important relationships it may contain more evident, and how best to learn patterns from this data. It can be argued that these two goals perhaps do not always align, with one often having to take precedence over another. Although, one data representation that has the potential to be able to achieve both aims is that of the graph, which will be the focus of the work presented in this thesis.

The idea of representing complex relationships or interdependencies in data via the form of a graph or network[1] has long existed [72]. In its simplest form, a graph comprises just

[1] To avoid confusion with neural networks, through this thesis the term graph will be used without loss of generality.

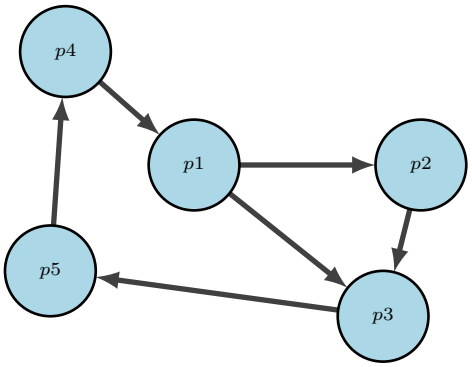


Figure 1.1: A graph of citation between five research papers, where directed edges represent a citation.

two primary components: vertices[2] and edges. When representing data as a graph, vertices frequently represent entities (a person or scientific paper), whilst edges capture the relationships between them (friendship or citation). As an example, it is common to represent the citations between different research papers as a graph, where the vertices represent the papers and an edge being present between two papers indicating the presence of citation. An illustrative graph of five papers $(p_1, ..., p_5)$ and their inter-citation can be seen in Figure 1.1. Edges are directed to indicate which paper cited the other. Even with this simple example, it can be seen that when compared to other data forms, graphs allow for inherent relationships in the data to be represented in a natural, semantically meaningful and interpretable manner [9].

Over the past decade, there has been a significant increase in the requirement for patterns in datasets to be automatically identified via computer programs [80]. The techniques, commonly emanating from the field of Machine Learning (ML), have grown in both complexity and capability, whilst finding applications in a broad range of domains. Commonly, these ML algorithms can be trained to perform a mapping from some input data to a target output In the case of classification this would be a mapping to a class label (for example, an image with an associated label indicating the presence of a cat). Applying machine learning algorithms to tasks as diverse as autonomous driving [86], automated language translation [226] and even medical diagnoses [62] has shown them to perform better than traditional approaches. However, the capability of a machine learning model to perform a certain task is ultimately bound by the quality of the dataset with which it has been trained [26], with recent models requiring

[2] Sometimes called nodes in the literture, but will be referred to as vertices throughout this thesis.

increasingly vast quantities of labelled data [137]. An additional facet of the algorithms used for machine learning is that the choice of algorithm will determine the representation that the input data must take, with conversely, the opposite statement also being true. As an example of this phenomenon, if tackling an image classification task using a Support Vector Machine (SVM) algorithm (explored more in Chapter 2), then the image must be represented as a numerical vector for input to the model, with this vector often comprising descriptive features extracted from the image. Conversely, if one wanted to tackle the problem using only the raw image as input, then the choice of algorithm would be limited to models which attempt image-based input, such as Convolutional Neural Network (CNN).

Despite the recent advances in developing new Machine Learning algorithms and techniques for image, video and text data, there has been comparatively little focus on developing graph specific approaches [42, 186]. This thesis will use this as motivation and as such, the majority of the work presented here will be directed towards addressing various issues and challenges that arise when learning from graphs via the use of machine learning.

1.1 Rationale and Motivations

In many domains[3], graphs have proven to be a good representation for capturing complex relationships in data [170]. As such, a large number of graph specific algorithms have been developed which are designed to capture and extract structural information from a given graph's topology. The results from such algorithms can then be used for a large number of tasks, often encapsulated by the term 'Graph Mining' [46]. As an example of such a task, one frequently desired metric to be extracted from a graph is that of vertex centrality [193], which defines how important a certain vertex is to the overall graph structure. Real-world applications are numerous, the classic exemplar being Google's use of a particular vertex centrality metric, entitled PageRank [178], to help decide the ranking of web pages in a user's search result. As another example, in many domains it is useful to partition the vertices into groups or communities using some measure of similarity [74]. Vertices which represent users on an e-commerce site and who belong to the same community can then be shown similar recommendations for example [240]

Recently, problems have emerged with this paradigm however, with arguably two primary issues being the continued increases in graph sizes and the complexity of questions which graph

[3] Newman (2010) provides interesting case-studies of graph use in the real world [170].

mining is being asked to answer. In common with all data sources, the size and number of graphs being stored and processed is increasing rapidly [123]. This increase has been so dramatic that the traditional approaches used in the graph mining field are struggling to perform well at these larger scales, either not running at all or having unacceptably large runtimes [10]. To help combat this, recent years have seen several software frameworks being launched which are specifically designed to help process graphs in parallel over a distributed set of compute nodes [61, 116, 159, 235, 248]. Additionally, even dedicated graph-specific processors have been developed [109]. Also, as graphs are being used ever more frequently for ever more complicated tasks, careful attention must be paid to what particular topological structure would be best suited to solve the given task. This process, which can be thought of as feature engineering, usually requires a domain expert to select the correct metric for a specific task, or even create a new one if the correct one cannot be found. This can be a costly process to undertake, both in terms of time and financial expenditure [140].

Machine learning offers many benefits over traditional approaches for data analysis in fields such as Computer Vision (CV) and Natural Language Processing (NLP), with one branch of machine learning, known as Deep Learning (DL) [80], demonstrating excellent performance on a diverse set of tasks. Deep Learning represents a family of models, most of which use many layered neural networks to learn from large datasets and make predictions [142]. It offers one particularly interesting aspect in that typically, data is fed into a model in its raw form, bypassing the feature extraction stage required by other models. Deep Learning models typically are understood to learn for themselves the best features to extract from the data, in order to minimise a certain training objective, for example learning to detect edges in images to perform classification [137].

Compared to other fields, there has been relatively little work undertaken in combing graph-based data with machine learning models. However, there is large scope for similar benefits to be brought to graph mining tasks via the use of machine learning. An important task which could be performed via the use of machine learning is that of graph classification, which can be considered both at the global and vertex/edge level. For example, being able to correctly classify chemical molecules being represented as graphs can aid in the discovery of new medicines [120], or the identification of malware infection in software programs captured as graphs [228]. An equally important graph mining task which could be achieved via the use of machine learning and graph data is that of missing edge prediction[4] [161]. Predicting that a new edge will form in a graph of a social network can be used for example to recommend that two users become

[4] More frequently know as link prediction.

friends [2]. Alternatively, in a graph of protein-protein interactions, predicting a new link would indicate that two proteins are likely to have some form of interaction [135].

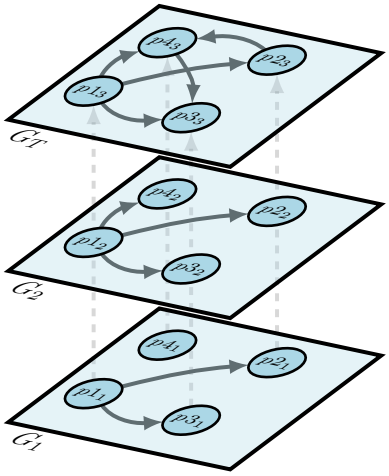


Figure 1.2: A snapshot of the evolution of a citation graph between four researchers, where directed edges represent a citation between two authors.

Despite the possibilities offered by combining graphs and machine learning, there are certain obstacles that arise, owing mostly to particular characteristics of graphs when compared to other data forms. Most existing machine learning algorithms are designed to accept input in the form of vectors or small matrices of numerical values and as such, cannot directly process graph data. Graph data is known to be extremely sparse, with the amount of existing edges in a graph typically being dwarfed by the total number of possible edges [56], which can cause issues for certain learning algorithms [252]. Additionally, many of the underlying datasets being represented as graphs are inherently temporal, meaning that many graphs naturally evolve over time. As an example of this, a temporal graph, representing citations between four researchers, is illustrated in Figure 1.2, showing how edges and vertices can change over time. Ideally then, any machine learning model being used on a graph would consider these temporal aspects in the learning process. A common concern with the use of a deep model is that interpretability of how a certain decision is arrived at is lost [77], as the rules determining the decision-making process are not explained by the model. For certain graph mining applications, in the medical or legal industries for example, this lack of interpretability could harm the uptake of graph-based machine learning as explainable decisions may be required.

1.2 Questions to be Addressed in this Research

The previous section discussed a broad range of issues that arise when performing graph mining using machine learning and highlighted that within this field there are some key issues still to be addressed. Therefore, the following specific questions will be addressed in this thesis:

- How to find a single representation encompassing the whole graph? An answer to this should enable processes, such as graph comparison and classification, to be performed at the macro level.
- How best to bring interpretability to automated graph representation learning? This would be crucial because it might allow graph-based techniques to be used in a broader range of fields and scenarios, whilst providing a vehicle for understanding how these approaches actually work.
- How to incorporate temporal information into graph-specific machine learning models? This is important because so much of the data/processes being represented as a graph is inherently temporal in nature, yet thus far, this has not been incorporated into models. To date, this has been a very challenging task.

1.3 Research Aim and Objectives

The work and contributions presented in this thesis will focus in particular on the three research questions identified in Section 1.2. The overall research aim of the thesis is to investigate how modern machine learning techniques can be adapted to tackle and improve key graph mining tasks. The following research objectives have been designed in order to achieve this aim:

1. To create an optimum global representation of a graph such that it is amenable for input into machine learning models.
2. To devise and evaluate new methods to begin to bring interpretability to graph-based machine learning models.
3. To produce and evaluate new models which are able to incorporate temporal dynamics into the learning process.

1.4 Thesis Scope

This thesis will explore various aspects of using machine learning to process data represented as a graph. However, it is important to note that the work presented here is limited by various factors including the availability of public datasets.The majority of the work explores the training of deep neural networks to solve various tasks within the field of graph mining. It is a well known phenomenon that such neural networks require large quantities of high quality training data [65]. Whilst other fields, such as Computer Vision with ImageNet for example, have established large and well understood benchmark datasets available for use by researchers, the field of graph mining is yet to establish such a resource. This is largely due to the heterogeneous, and often proprietary, nature of the data typically represented as graphs, making the collection of a representative resource by a singe person or institution extremely challenging and thus beyond the scope of this thesis.

The lack of a commonly agreed upon benchmark raises an additional issue of not easily being able to compare directly between competing approaches without reimplementation of the approach and reproducing the original experimental result. Some graph dataset repositories do exist, such as the Stanford Network Analysis Project (SNAP) [146], the Network Repository [196] and the Koblenz Network Collection (KONECT) [138], but they do not contain nearly enough data for many crucial tasks such as graph-level classification, let alone enough data with auxiliary information such as features, labels or temporal information.

As a direct consequence of this, all the chapters in this thesis will, to varying levels, employ graph data generated synthetically as a substitute for large graph datasets. Numerous graph generation approaches have been proposed in the literature, many of which attempt to emulate various aspects of topological features observed in empirical graphs. Any number of required graphs, matching any set of topological constraints, can be generated easily and reliably. However, there is the possibility that the generation methods might not reflect the scope, variation and noise seen in data from the real world and as such might be simpler for the machine learning models to make accurate predictions about. One interesting aspect to this is that such approaches could actually be used to improve the original generation methods, in order to more accurately reflect the real-world data, although this is beyond the scope of this thesis.

Additionally, other factors such as a lack of proper graph-specific model performance metrics can hinder truly rigorous evaluations, especially when attempting to measure the presence of topological structure in an embedding space.

1.5 Thesis Structure

In Chapter 2, a review of the relevant background material required for this thesis is presented. This includes an introduction to the fields of graph mining and machine learning.

In Chapter 3, work is undertaken to find the optimal global representation of a graph through the use of topological features and a neural network, with case studies presented for the tasks of graph classification and graph comparison. Evaluation shows the presented approaches to scale well to large graphs and beat current state-of-the-art approaches including Graph Kernels.

Chapter 4 shifts focus to investigate the newly emerging graph embedding techniques, with the aim of bringing interpretability to the models used. This is achieved by attempting to reconstruct known topological features from the embedding space.

In Chapter 5 two novel models are explored which are designed for incorporating temporal graph dynamics into the learning process. The approaches use deep graph-specific model architectures to produce temporally-aware vertex level representations optimised for predicting future links.

Finally, in Chapter 6 the thesis is drawn to a close by presenting a summary of the contributions made to the field and comparisons drawn with the original research aim and objectives. Additionally, potential future research which could be used to expand upon this work is identified.

Chapter 2

Background

2.1 Introduction to Graph Mining

Graph mining is an interdisciplinary field which allows for the studying of detailed real-world phenomena by viewing them as a series of connected components in an overall complex system. There are numerous examples of systems across the spectra of scientific, as well as other disciplines which are composed of individual elements linked together in some manner [170]. Some obvious 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 graph mining can be defined as the study of the collection, management, analysis, interpretation, and presentation of relational data [19].

A graph fundamentally comprises of a set of vertices, with pairs of vertices connected together via an edge. These edges can be undirected or directed, with implied directionality between two vertices creating a directed graph. Vertices and edges can have associated weights or attributes, often in the form of a numeric value. These graphs are known as weighted graphs and are used to embed a greater quantity of information within the structure of a graph.

2.2 Definitions

In this section, aspects and concepts explored in this thesis are formally introduced, as well as definitions of the notation used throughout the rest of the text.

The terms graph and network are often used interchangeably within the literature, however to avoid confusion with neural networks the term graph will be used throughout the remainder of this thesis without loss of generality. Mathematically a graph can be defined as $G = (V, E)$ where V is a finite set of vertices and E is a set of edges. The elements in E are unordered pairs $\{u, v\}$ of unique vertices $u, v \in V$. The number of vertices $s|V|$ and edges $|E|$ are often called the order and size of the graph G. A directed graph G can be represented where each edge in E displays an ordering to its vertices, so that $\{u, v\}$ is distinct from $\{v, u\}$. It is possible for a graph to have a set of labels associated with vertices, edges or both. In such cases we can define a graph $G = (V, E, L)$, where L is a set of weights or labels. A label contains additional information about an edge, vertex or the graph itself, for example a person's name or age within a social network.

Graph theory is the theoretical study of these graphs, their mathematical properties and their topological structure. Being well studied, graph theory provides a wide spectrum of mathematical tools for exploring and quantifying graphs. A graph can be represented in several forms, common ways being the adjacency, degree and laplacian matrices. It should be noted that, unless otherwise stated, the majority of the graphs used in this thesis are simple graphs. In graph theory, a simple graph can be defined as one which contains no self loops (edges which connect vertices with themselves) or parallel edges (multiple edges between two vertices).

An adjacency matrix $\mathbf{A}$ for a graph G is a $|V|$ x $|V|$ matrix, where the values are determined such that:

$$\mathrm{A}_{ij} = \begin{cases} 1 & \text{if node } i \text{ and } j \text{ are connected via an edge;} \\ 0 & \text{if no edge is present.} \end{cases} \tag{2.1}$$

This notation can also be adjusted for the case of weighted graphs such that:

$$\mathrm{A}_{ij} = \begin{cases} 1 & \text{if node } i \text{ and } j \text{ are connected via an edge with weight } w; \\ 0 & \text{if no edge is present.} \end{cases} \tag{2.2}$$

The degree matrix $\mathbf{D}$ for a graph is a diagonal matrix of size $|V|$ x $|V|$ where the diagonal elements are set such that:

$$\mathrm{D}_{ij} = \begin{cases} k_i & \text{if} \quad i = j; \\ 0 & \text{otherwise.} \end{cases} \tag{2.3}$$

Here, k_i would commonly be the total degree of node $i \in V$.

Finally the graph laplacian $\mathbf{L}_G$ is again a matrix of size $|V|$ x $|V|$. We can define the graph laplacian matrix as simply the degree matrix, subtracted by the adjacency matrix:

$$\mathbf{L}_G = \mathbf{D} - \mathbf{A}. \tag{2.4}$$

Whilst seemingly simple, the graph laplacian has many interesting properties which can be exploited to gain insights into graph structure [64].

2.2.1 Note on Mathematical Notation

The style of the mathematical notation used throughout this thesis, as well as some common definitions are presented in Table 2.1.

2.3 Extracting Graph Structure

One of the most compelling reasons to represent data as a graph is that there exists a wide range of measures to extract statistically important information about it's structure [171]. Such measures capture various aspects of patterns of connectivity within a graph and can allow unique insights into the data. Some of the most important measures of graph structure, used through the remainder of this thesis are outlined in this section. It should be noted that this is not a comprehensive list of all measures, instead it is limited to the ones directly relevant to the thesis.

Symbol	Definition
q	Style used to denote a scaler value (integer or real value).
$\mathbf{q}$	Style used to denote a vector.
q_i	Element i of vector $\mathbf{q}$.
$\mathbf{Q}$	Style used to denote a matrix.
$\mathrm{Q}_{i,j}$	Element i, j of matrix $\mathbf{Q}$.
Q	Style used to denote a set.
$\lvert Q \rvert$	The number of elements in set Q.
$F()$	Style used to denote a named function.
$f()$	Style used to denote a generic function.
$\mathbb{R}$	The set of all real numbers.
$P(\mathrm{a})$	A probability distribution over a variable.
$\mathrm{a} \sim P$	Random variable a has a distribution P.
$\mathbb{E}_{x \sim P}\Big[f(x)\Big]$	The expected value of $f(x)$ with respect to $P(\mathrm{x})$.
G	A graph with an associated set of vertices V and corresponding set of edges E.

Table 2.1: Definitions and Notations

2.3.1 Degree and Degree Distribution

One of the most frequently used measures is the degree of a vertex, which can be defined as the number of edges connected to it [170]. For a directed network, a vertex will have both an in and an out degree which can be calculated separately or summed together to give the total degree. Often the degree of vertex v is denoted by k_v (this can be considered the sum of the incoming edges k_v^- and outgoing edges k_v^+) and for a simple graph of size $|V|$, the degree in terms of an adjacency matrix, $A_{v,u}$, can be calculated as:

$$k_v = \sum_{u=1}^{|V|} \mathrm{A}_{v,u}. \tag{2.5}$$

To analyse the structure of complex graphs, the distribution of degree values is often used [189]. The degree distribution is used to calculate the probability that a randomly selected node will have a certain degree value. It provides a natural overview of connectivity within a graph and is often plotted as a histogram with a bin size of one [170], as will be done throughout this thesis.

2.3.2 Paths and Walks

Another common set of graph metrics to consider revolve around the concept of a path in a graph. A path is a route from one node to another through the graph, in such a way that every pair of vertices along the path are adjacent to one another. A path which contains no repeated vertices is known as a simple path. Additionally, a graph for which there exists a path between every pair of vertices is considered a connected graph [170]. An example path through a graph from two vertices v_5 and v_7 is displayed in Figure 2.1. Often there are many possible paths between two vertices, in which case the shortest possible path, which is the minimum number of edges needing to be traversed to connect two vertices, is often an important metric to consider as it forms the basis for more complex measures [103]. The path illustrated in Figure 2.1 happens to be the shortest path between the two aforementioned vertices. A graph can often be split into distinct groups if there are subsets of unique connected vertices for which there is no path linking them together. These individual pieces within a graph are called components [170]. However it is common to find that there exists a path between a large fraction of the vertices. This is called the giant connected component which has been observed in numerous graph datasets [106].

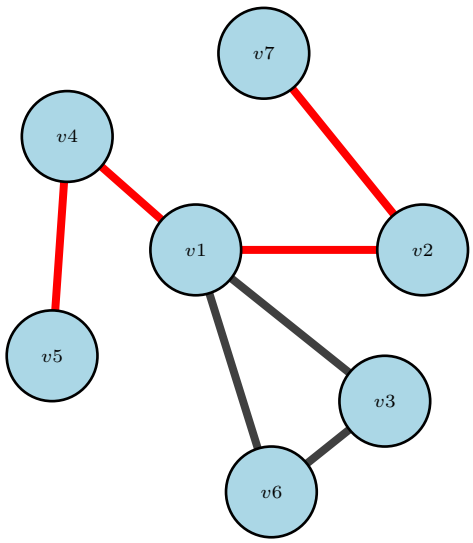


Figure 2.1: A path between vertex v_5 and v_7 (highlighted in red).

Linked to the concept of the path is that of a walk through a graph. A walk is defined as a finite list of vertices in which each vertex in the list is connected to the previous one via an edge

[170]. A walk from a given vertex can potentially visit the same vertices an unlimited number of times [16]. A special type of walk, known as the random walk, is commonly used in the graph mining literature, often as a way to sub-sample from a graph [93]. To perform a random walk from a given vertex, a neighbouring vertex connected to the first via an edge is chosen at random. From this new vertex, a neighbour connected to it is then chosen at random, with this process being recursive until the desired walk length is achieved. The probability with which a new vertex is chosen is often uniform [186], however it can be biased to alter the characteristic of the walk in some desired way [87].

2.3.3 Vertex Neighbourhoods, Clustering and Triangles

It is often useful to consider the neigbourhood of a vertex when measuring graph structure. The neigbourhood of a vertex can be defined as the set of vertices to which it is connected, with often the one-hop neighbourhood (the set of vertices with which it directly shared an edge) being used. The one-hop neigbourhood for a vertex is denoted as $N(v)$ and an example is displayed in Figure 2.2. However, the neigbourhood of a vertex can be defined to contain vertices which are multiple hops away from it [128]. As an example of this, the two-hop neigbourhood of vertex v_1 from Figure 2.2 would also include the vertices connected via a black edge, as they are neighbours of members of its one-hop neighbourhood.

Triangles

Within a vertex neighbourhood a commonly studied motif, a small and reoccurring local pattern of connectivity between vertices in a graph, is that of the triangle [170]. A triangle is a series of three vertices where an edge is present between three vertices. The graph in Figure 2.2 contains a triangle between the vertices v_1, v_3 and v_6. The number of triangles for a vertex v is the number of vertices in $N(v)$ which are also connected via an edge. Triangles are often considered a fundamental building block of graph structure, as in many graphs the process of triangles being formed over time has been observed – a process entitled triadic closure [68].

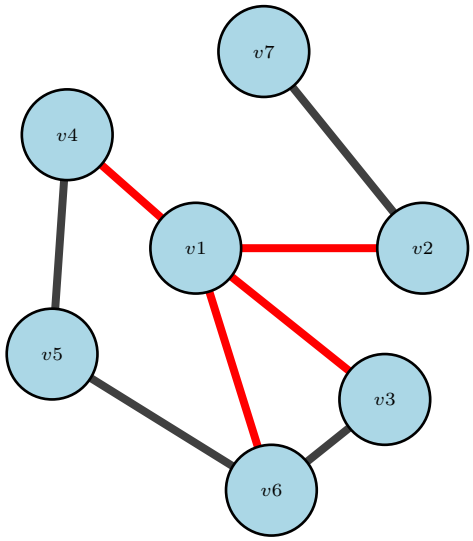


Figure 2.2: The one-hop neighbourhood of vertex v_1 (highlighted in red).

Local Clustering

A further measure of connectivity within a graph is that of the clustering coefficient. At the level of individual vertices, the clustering coefficient gives a measure of how connected that vertex's neighbourhood is within itself. More concretely, for a given vertex v, the clustering coefficient determines the fraction of one-hop neighbours of v which are themselves connected via an edge,

$$LC(v) = \frac{\text{number of complete triangles}}{\text{number of all triplets}} , \tag{2.6}$$

where triplets refers to all possible combinations of three vertices from $N(v)$, both open (meaning not complete triangles) and closed [170].

2.3.4 Vertex Centrality

There are many applications for which it would be beneficial to measure the relative importance of a given vertex within the overall graph structure, for example to find the most important web page or user of a social network. One such way of measuring this is vertex centrality, within which there are numerous methods proposed in the literature which measure different aspects

of the underlying graph structure. Many of these methods originate in the study of web and social networks, with the PageRank algorithm being a famous example as it formed a key part of the early Google search algorithm [178]. In addition to this, some of the other frequently used centrality measures include Degree, Eigenvector and Betweenness [129].

Degree Centrality

Perhaps the simplest measure of centrality is that of Degree centrality, which provides a normalised measure of vertex connectivity [115]. A graph where the vertices have been coloured in accordance with their respective Degree centrality value is presented in Figure 2.3. The Degree centrality value for a vertex v can be computed as:

$$DC(v) = \frac{1}{|V|} k_v. \tag{2.7}$$

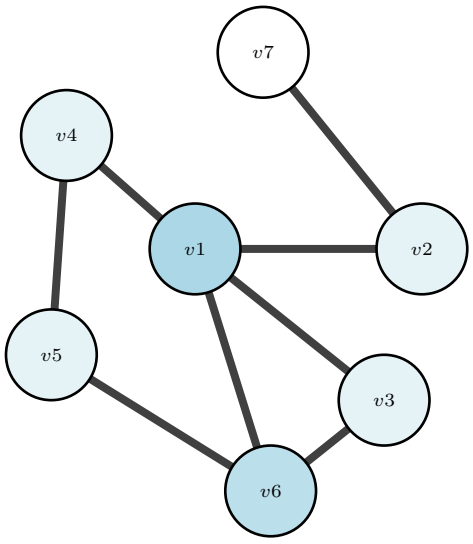


Figure 2.3: A graph where the vertices have been coloured approximately by their degree centrality value, where a darker shading indicates a higher value.

Betweenness Centrality

Betweenness centrality exploits the concept of shortest paths to argue that vertices through which a greater volume of shortest paths pass through, are of greater importance in the graph [75]. Therefore, vertices with a high value of Betweenness centrality can be seen as controlling the information flow between other vertices in the graph. The Betweenness centrality for a certain vertex v can be defined as:

$$BC(v) = \sum_{\substack{s \neq v \neq t \in V \\ s \neq t}} \frac{\sigma_{st}(v)}{\sigma_{st}}, \tag{2.8}$$

where σ_{st} is the total number of shortest paths from s to t and $\sigma_{st}(v)$ is the number of paths which contain v.

Eigenvector Centrality

A more complex measure is that of Eigenvector centrality, which assigns a value to a vertex based on high-scoring neighbouring vertices contributing more than lower scoring ones. Thus a high Eigenvector centrality value for a given vertex means that it is connected to other high scoring ones [27]. Formally the Eigenvector centrality can be written as an eigenvector equation using the adjacency matrix of a given graph:

$$\mathbf{A}\mathbf{x} = \lambda \mathbf{x}, \tag{2.9}$$

where λ is the largest eigenvalue, $\mathbf{A}$ is the graph G in adjacency matrix form and $\mathbf{x}$ is the corresponding eigenvector. Using the Perron–Frobenius theorem, there is a unique solution for $\mathbf{x}$ with all positive values when the largest eigenvalue is used [170]. The Eigenvector centrality for a certain vertex v is the v-th element indexed from the vector $\mathbf{x}$:

$$EC(v) = \mathbf{x}_v \tag{2.10}$$

PageRank Centrality

The PageRank centrality method was originally developed by Google to rank the importance of webpages, however it is now commonly used to measure the local influence of a vertex within

a graph [94, 178]. PageRank Centrality is closely related to the previously discussed Eigenvector Centrality, however crucially it can incorporate the additional information available in directed graphs. The PageRank centrality for a given vertex v can be defined as:

$$PR(v) = \frac{1-d}{|V|} + d \sum_{u \in N^-(v)} \frac{PR(u)}{k_u^+}, \tag{2.11}$$

where $N^-(v)$ is the set of incoming neighbours of v and d is a constant damping factor.

2.4 Graph Datasets

It has been strongly argued that many of the recent successes in the field of machine learning, especially the approaches exploiting deeper models, has been driven by the availability of large, high quality and importantly, labelled datasets [80]. For example, the Imagenet dataset has been key to dramatic advances in the ability of Computer Vision (CV) models by providing them with over 14 million human annotated images from which to learn [65]. In the field of Natural Language Processing (NLP), recent advances have also been driven by the availability of massive quantities of text data, with Wikipedia alone providing over 3.7 billion English language words [241].

However, the field of graph analysis does, to date, not have the same quantity of quality public datasets available for use by researchers. This has arguably made the same levels of progress in graph processing models more challenging when compared with other domains. The three main sources of public graph datasets in the field, and thus of ones used throughout this thesis, are the Stanford Network Analysis Project (SNAP) [146], the Network Repository [196] and the Koblenz Network Collection (KONECT) [138].

Whilst these data sources are useful, they do not contain the quantity and variety of data seen in datasets from other fields. For example, SNAP contains less than 200 unique graphs across 18 domains. Because of this lack of empirical data, the work presented in this thesis uses both synthetically generated graphs and graphs whose topological structure has been altered in some way.

2.4.1 Graph Generation Methods

There has long been an interest in developing methods which are able to generate synthetic and random graphs which conform to some structural constraints, thus replicating empirical data [113, 148]. Using such approaches means that an unlimited number of graphs can be generated, of varying sizes and structural complexities, thus helping to reduce the aforementioned data access issues. It has also been proposed that graphs generated from a known mathematical process could be used as a benchmark for a machine learning algorithm, as it could be tasked with uncovering the underlying generative process [8]. Some of the major synthetic graph generation methods utilised throughout this thesis are detailed in this section.

Random Graphs

In the generation of random graphs, as proposed by Erdős and Rényi [20], the probability of the existence of each edge is equal. Thus graphs generated using the Erdős-Rényi method have a degree distribution which looks to have been chosen uniformly at random. Such graphs would prove challenging for any machine learning model trained upon them, as there is no real structure to be learned.

Scale-Free Graphs

One of the mostly widely used models to study the formation of networks in the Barabási-Albert (BA) model [20]. It has been noted that actual real-world vertex degree distributions exhibit a fat-tailed, or power-law shape, meaning that a majority of vertices have a low degree value, whilst only a few vertices have a high value [20]. These graphs were entitled 'scale-free' graphs, due to their lack of natural scale [67]. Since this discovery, scale-free graphs have been reported in many other graph studies [98]. Although the prevalence of graphs which exhibit strict power-law distributions has been put under some doubt [130], generating graphs with this property can be a useful first approximation.

The BA model was designed to produce graphs which have a degree distribution which is approximately power-law, thus more closely replicating real-world data. The BA model functions

as follows: upon each new vertex being added to the graph, it has a probability p of forming an edge to an existing vertex v:

$$p(v) = \frac{k_v}{\sum_{i \in V} k_i}. \tag{2.12}$$

As the chance of new edges being formed is directly proportional to a vertex's degree value, hubs or densely connected vertices will appear.

Forest Fire Graphs

Whilst the BA model produces graphs which display the characteristic power-law degree distribution, it fails to capture other structural characteristics observed in graph data [145]. To address this issue, the *Forest Fire* model for synthetic graph generation has been proposed [145]. The proposed model is designed so that it captures the shrinking diameter and increasing densification characteristics highlighted in the study as being missing from other methods. The model functions in such a way that a new vertex v entering the graph attaches to a existing vertex w uniformly at random. Vertex v then begins to burn through a selection of in and out edges from w, creating links to the vertices it touches with a certain probability. The graphs created by the Forest Fire model conform to both shrinking diameter and increasing densification, as well as featuring a power-low degree distribution.

2.4.2 Graph Topology Random Rewire Process

Throughout the work presented in this thesis we will make use of the random rewire process to alter a given graph's topological structure. The random rewire process perturbs a given source graph's degree distribution by randomly altering the source and target of a set number of edges according to the Erdős-Rényi random model. This results in edges which are uniformly distributed among the vertices, instead of the more frequently observed power-law like distribution [20, 56]. The number of edges which are altered can be controlled to cause either major or minor changes to the graph's topology. During this rewire process, it is not guaranteed that the source or target of the edge will be altered, indeed it is not always possible due to the graph's topology. Also, it should be noted that the rewiring process does not change the total number of edges or vertices within the graph.

2.5 Introduction to Machine Learning

The field of Machine Learning refers to a range of techniques which are used to create computer programs which automatically learn and identify patterns in data. Unlike traditional programs, ML-based ones are trained rather than being explicitly programmed to perform a certain task. Machine Learning can be thought of as a series of three elements which together form an approach. These three being the learning task, the experience learned from the task (this encompasses both the ML model and dataset choice), and the performance measure used to assess the success of the task [166].

The remainder of this section will review these solely as they relate to this thesis. Hence, this should not be considered a complete review of the field of machine learning.

2.5.1 Machine Learning Tasks: Supervised and Unsupervised Learning

The task which can be performed by Machine Learning models is usually dictated by whether the chosen dataset has an associated set of labels or annotations available. If a dataset contains labels, then techniques for the family of supervised learning can be performed. If labels are not present, then techniques from the family of unsupervised learned must be employed. As models using both learning tasks will be utilised throughout this thesis, they are explored further below.

Supervised Learning

Supervised learning uses labelled or annotated data to help guide the learning process by providing models with pairs of data elements and associated labels [55]. Using the example of classifying a graph to belong to a certain class, the supervised learning process is detailed in more depth. It should be noted that the use of graphs could be replaced with any other data format (images or text) and the process would be identical. Also this example focuses on classification, but the task of regression – predicting numerical values from data – works in largely the same manner. In a supervised learning classification problem, we have a dataset D comprising n graphs $G_i \in D$, where $i = 1, ..., n$ and $G_i = (V_i, E_i)$ where a label might be present on the vertices or edges. Each graph in D has a corresponding class $y_i \in C$, where C is the set of l

categorical class labels, given as $C = 1, ..., l$. In the case of graphs, the categorical class label could correspond with a graph's domain, for example a social, biological or citation network, or the synthetic generation method used. The goal of this supervised learning task is to derive a mathematical formula to perform $f : D \rightarrow C$ which can accurately predict the class label of each graph in the dataset. When deriving f using a machine learning approach, the common pattern is to learn the function from a subset of D known as the training set for which labels are present. The function is then tested on the remaining examples from D, often called the test set. The accuracy of the function is assessed by comparing the predicted label $\hat{y}_i = f(G_i)$ with the ground truth label (y_i) for all graphs in D.

Unsupervised Learning

Unsupervised learning encompasses a range of techniques which attempt to learn from data without requiring the use of examples labelled via the use of human experts. Due to the varied nature of the tasks performed via unsupervised learning, it is hard to devise an exact definition of what is trying to be achieved. However, tasks such as grouping together data points which share some form of commonality (known as clustering [236]), or compressing the size of the input data by projecting it into a lower dimensional space (via Principal Component Analysis for example [110]) can be considered as unsupervised. A selection of unsupervised learning approaches are detailed in greater depth in Section 2.5.2.

2.5.2 Machine Learning Models

A machine learning model is used to learn from, and make predictions about, a particular input dataset. An important aspect to this learning process which can affect how the model is used is whether it can be considered interpretable or not. An interpretable model is one where the underlying decision process can be explained and understood by human observers, with models failing to meet this criteria being labelled as black box[1] [77].

Some of the major families of approaches relevant to this thesis are reviewed below.

[1] So called as only the inputs and outputs can be observed, with the internal components mapping between the two remaining opaque.

Traditional Supervised Models

Before the recent increase in the popularity of neural-based models, other forms of supervised machine learning models were prevalent. Whilst these approaches differ in the algorithms used, they almost all share one common trait, they require an n-Dimensional vector as input. This means that data which is not naturally represented in this format, including graphs, images and text, must be converted into a vector. Typically this vector represents descriptive features extracted from the data by domain experts, in a process know as feature extraction or feature engineering [122]. Once the input data has been converted into vector form it can, along with its associated set of labels, be used as input to a variety of models. Three of the most frequently used are detailed below:

- *Logistic Regression:* A supervised model for classification which is often used as a strong baseline approach is logistic regression [163]. Considering the binary case[2], logistic regression is a linear function that has a parameter per element in the input feature vector. The result of the multiplication between the input vector and the parameters is then passed through the logistic sigmoid function to ensure that the output is in the range 0 to 1 so that it can be interpreted as a prediction [80]. The parameters of the model are then tuned such that the model is more likely to produce the desired result using gradient descent [200] (a process introduced in greater depth in the following section).

- *Support Vector Machines:* A more complicated family of algorithms for supervised classification is that of Support Vector Machines (SVM) [58], which unlike logistic regression, directly map data points to predicted labels. Again considering the binary case, SVMs attempt to fit a decision boundary, in the form of a hyperplane, between the data points in a high dimensional space. This decision boundary is optimised such that it separates the data points belonging to the two classes [80]. Class predictions about any new data can then be made by measuring which side of the decision boundary they are. The mapping from the initial input vector to the new high dimensional space, through which an accurate decision boundary can be made, can be costly and computationally intractable [55]. To overcome this issue, SVMs exploit what is know as the kernel trick [206], which enable distances in a high dimensional space to be measured, without the need to actually perform the mapping process [55].

[2] Binary classification is where there are only two target classes to predict.

- *Random Forests:* More recently Random Forests have become one of the most widely used models for supervised learning. Random Forests are essentially ensembles, or collections, of individual decision tree models, combined together to perform a classification task [100]. As well as demonstrating excellent predictive performance, they are often favoured because their output can easily be considered as a series of decision rules, making for a more interpretable model [55]. Each decision tree model can be conceptualised as a tree-like structure, where the split at each node can be thought of as a test on a certain attribute or feature of the input data, for example, if a feature is below or above a certain value. Decision trees are trained using a two-step process: the induction process, where new rules are created and applied to the data, and the pruning process, where unnecessary structure is removed from the tree to help the model generalise better to unseen data [40].

Neural-based Models and Deep Learning

Artificial Neural Networks (ANNs) are a field within Machine Learning inspired by, but importantly not completely replicating, the functionality of a brain [142]. Whilst the origins of ANNs dates back to at least the 1960's, and perhaps earlier [195], they have recently experienced a dramatic increase in capability and thus popularity [142]. ANNs model problems via the use of connected layers of artificial neurons. Each ANN has an input layer of such neurons to which the data is passed, at least one hidden layer to transform the data in some way and an output layer where predictions are produced. In the traditional ANN concept, each neuron takes as input a weighted sum of the outputs of the neurons which are connected to it, with each layer containing a parameter matrix to enable this. Once the weighted sum has been performed, it is transformed using a pre-specified non-linear activation function. Commonly used examples of such functions including Sigmoid, Softmax and the Rectified Linear Unit (ReLU) [80]. Without the use of non-linear activation functions, a model would be limited to just learning linear (affine) transformations of the input data [80]. This would severely limit the learning capability of the model and make the use of multiple stacked layers redundant, as combing multiple linear layers would still result in a linear operation overall [55].

ANNs are modified to become better at a certain task using an iterative process, commonly referred to as training. This training process is performed as follows: Input data is passed into the network, transformed via the hidden layers and a prediction is produced at the output layer. Typically for ANNs, the correctness of this prediction is assessed via the use of a loss function.

A variety of functions can be utilised for this task and are specific to the type of learning which is being performed. For example, supervised problems use loss functions which exploit the availability of labels such as the cross-entropy function, a way to use the Kullback–Leibler (KL) divergence to measure the distance between the true and predicted output [108]. Once a loss value for the model has been computed, the parameters or weights are updated such that the probability of producing the desired outcome would increase if the same data was passed in a second time – a process know as back-propagation [201]. The back-propagation algorithm exploits the fact that all components of a neural network are differentiable and computes the gradient for the loss with respect to the model parameters, exploiting the chain rule for computational efficiency [55]. A separate family of algorithms, called optimisers, then takes this gradient and uses it to update the parameters directly. One of the most frequently used optimisers is Stochastic Gradient Decent (SGD), which uses randomly chosen sub-samples of the larger dataset to enable more efficient training [200].

Deep Learning is a term generally used to refer to ANN's which have multiple stacked hidden layers, so called Deep Feed Forward or Dense networks. In practice though the term encompasses an emerging field, including new model architectures, training procedures to allow for the use of massive datasets and even a philosophical shift in how data is represented as input to the models [80]. Traditionally Machine Learning has been performed upon features extracted from the data, which can be a cumbersome task performed by domain experts [186]. This manual process, known as feature selection [90] in the literature, has clear disadvantages as certain features may only be useful for a certain task. It could even negatively affect model performance if utilised in a task for which they are not well suited. Arguably, many of the recent exciting advances seen in the field of Deep Learning have been driven by the removal of this feature selection process [87], instead allowing models to learn the best data representations themselves [80]. This is often known as end-to-end learning as the model is learning the optimum feature representation, which is tuned to perform a certain task. An example of a deep model which exploits this setup is the family of Convolutional Neural Networks (CNNs) models, which have demonstrated state-of-the-art performance in image classification, among others [142]. CNNs take as input raw images, and exploit spatial locality patterns by sliding learnable filters over the images to both improve predictions and reduce the total number of parameters needed to perform a certain task [143]. However, such models have faced criticism for being black boxes and thus not possessing an interpretable decision process [77].

Unsupervised Models

As discussed in Section 2.5.1, unsupervised models are ones which do not require the use of labels to guide the learning process. One important unsupervised task, explored in detail in this thesis (See Chapter 4), is that of representation learning, more commonly know as embedding [186]. In the context of the machine learning literature, embedding models are used to map between a discrete entity, with no natural numerical representation, and a meaningful value for it in some vector space [165]. This can be formalised as performing the following function: $f : O \rightarrow \mathbb{R}^d$, where f learns to map a set of entities O to a vector of size d, importantly without requiring the use of labelled examples. Examples of entities which can be mapped this way include words [164], retail products [225] and graphs (see Chapter 4).

To perform this mapping function, a variety of unsupervised models can be used, with some traditional approaches using matrix factorization to learn the representation [149]. Increasingly however, neural networks are being utilised in place of such approaches, with one popular approach being the skip-gram model from Word2Vec [165]. Skip-gram is designed to transform words, taken from a sentence, into vector representations – crucially where some of the semantic and linguistic meaning of the word is preserved in the new embedding space. The skip-gram model is able to learn an embedding for a word by using surrounding words within a sentence as targets for a single hidden layer neural network model to predict. Due to the nature of this technique, words which frequently co-occur together in sentences will have positions which are close within the embedding space. However, it has been argued that such techniques should really be labelled as self-supervised learning, as they employ models and objective functions more commonly found in supervised learning, but generate the labels automatically from within the dataset [80]. The skip-gram model has subsequently been adapted to work on graph data [186].

2.5.3 Graphs and Machine Learning

As the primary focus of this thesis, it is important to consider how graphs can be used as input for machine learning models. It has been argued that graphs can be a particularly challenging format of data to process via the use of machine learning, owing to their unique properties [152]. Some of these properties include the heterogeneous nature of graphs themselves (they can be directional, can contain additional information on the vertices or edges and can be temporal),

be of differing sizes (with some graphs being of a massive size, causing scalability issues) and can be extremely sparse in regard to edges (many vertices in real-world graphs only contain a small number of edges, making a model trained to predict edges biased towards never predicting a edge). The task is further complicated by the lack of publicly available training data, or a standard set of benchmarks being available via which approaches can be compared.

Nevertheless, over recent years a growing number of methods and approaches have been presented in the literature combining graphs and machine learning [43, 92], which span the range of tasks and technique highlighted in this section. For example, there are methods for extracting representative features from graphs, which can then be passed to traditional supervised algorithms for classification (the relevant literature around this, as well as a novel approach is presented in Chapter 3). Additionally, using unsupervised techniques to automatically learn meaningful representations of graphs has begun to be explored (a survey of such approaches, as well as new techniques to bring interpretability to them is detailed in Chapter 4). Finally, graph specific models, which have been inspired by Deep Learning to allow for raw data input, have begun to be created (such approaches are explored in more depth in Chapter 5, with novel research presented on how to incorporate temporal evolution into the learning process).

Chapter 3

Graph Comparison and Classification Via Graph Fingerprints

Prologue

The fields of graph mining and machine learning were broadly introduced in the previous chapter. This chapter will explore how these fields can be combined to perform certain key tasks in graph mining, namely graph comparison and global graph classification, both of which are introduced in further detail in Section 3.1. Fundamentally this chapter explores questions that arise from considering how graphs can best be represented to make them amenable to being used as input for machine learning models. This work has been performed in response to research objective 1 (defined in Section 1.3).

This chapter explores the concept of the *graph fingerprint*[1], a feature vector representation of a graph which captures characteristics of the local neighbourhood structure of the graph's

[1] This term, introduced specifically for this research, is explored more in Section 3.3.

vertices, whilst also incorporating key global graph features. The graph fingerprint is shown to be a versatile representation as it can be used to input into graph comparison and classification tasks, requiring no changes, whilst out-performing competing approaches such as Graph Kernels.

The work presented in this chapter has been published as the following works:

Stephen Bonner, John Brennan, Ibad Kureshi, Andrew Stephen McGough, and Georgios Theodoropoulos. Efficient comparison of massive graphs through the use of 'graph fingerprints'. In *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. *IEEE International Conference on Big Data*, pages 3298–3307, 2016

Stephen Bonner, John Brennan, Georgios Theodoropoulos, Ibad Kureshi, and Andrew Stephen McGough. Deep topology classification: A new approach for massive graph classification. In *IEEE International Conference on Big Data*, pages 3290–3297. IEEE, 2016

3.1 Introduction

This chapter will explore the hypothesis that graphs can be accurately and efficiently represented by combining the aggregated characteristics of a vertex's local neighbourhood structure with simple global graph features. We thus extract a series of local and global graph features and assess their performance in the tasks of graph comparison and graph classification. Our hypothesis that neighbourhood features could be used as a unique fingerprint for a graph is, in part, driven by work which has shown that Graph Motifs and Graphlets (small sub-graph like patterns of vertex inter-connectivity within a larger graph [5]) can be used for a variety of tasks within graph analysis [88]. However, Graphlets are known to be challenging to compute efficiently and require some hand engineering, as the correct structure must be identified [198]. Here we explore whether a general set of vertex neighbourhood features can be used as a graph fingerprint, which could then be used successfully in multiple graph analysis tasks. This is in contrast to the work on Graph Kernels, detailed in greater in Section 3.8.5, which are often used to capture global graph properties – making scalable Graph Kernels approaches more challenging [209].

The two application domains utilised in this chapter, graph comparison and classification, in order to explore the ability of graph fingerprints to capture detailed topological information, are introduced in greater depth below.

3.1.1 Graph Comparison

In many scientific domains, being able to compute some measure of similarity between two graphs is an extremely valuable task. Such domains include: anomaly detection [6] [36], protein comparisons [244] [187] and the study of temporal graph evolution / link prediction [3]. Thus, graph comparison and specifically similarity measurement is an area of increasing research interest.

There are many definitions of similarity between graphs [25] [133] [181], however, they can be split into two categories – those which can only be applied to labelled graphs and those which can be applied to graphs irrespective of labelling. When labels are available, similarity can be based on such metrics as the number or similarity of labels appearing in both graphs. However when labels are not present similarity is based on topology comparison. In this chapter we focus on topology comparison of unlabelled graphs.

A number of considerations need to be addressed when computing the topological similarity between graphs to ensure accurate comparison. For example, two graphs might appear very similar when considering the individual edges between vertices, yet be of completely different graph sizes. Conversely two graphs which are of comparable size, might have vastly different degree distributions (the distribution of edges between the vertices within a graph).

Most importantly, any comparison approach should be able to scale to the so-called 'high volume' (massive) graphs (vertices and edges) seen in such areas as social networks. Graph processing techniques are being applied in a broader range of data driven fields, where data volumes are large and constantly increasing, resulting in more graphs of larger sizes [159]. This dramatic increase in the quantity of data means that ever larger graphs need comparing against one another. This has a significant impact upon graph similarity measures, as any such algorithm needs to produce accurate results, be computationally efficient in terms of resource usage and can be computed in realistic time-scales – suggesting that the use of parallel techniques could be required.

In this Chapter we present a new approach for extracting Graph Fingerprints, a compact but representative abstraction of a graph, with numerous potential applications within field such as machine learning. The new approach, entitled Graph Fingerprint Extract (GFP-X), utilises Apache Spark and GraphX to massively decrease feature extraction times through the use of parallel computing, whilst increasing the maximum size of processable datasets. We demonstrate an application of the fingerprint approach for the comparison of graphs, named Graph FingerPrint Comparison (GFP-C), that is label-independent as it exploits only the topology of a given graph in order to compare similarity.

3.1.2 Graph Classification

Representing data as graphs or networks has enabled researchers from across the scientific disciplines to not only understand the data itself but also any underlying relationships [170]. Being able to accurately match a graph, which may not have complete descriptive information, to its domain or application can help to identify unknown data. As such, there has been increasing interest in the literature on how best to develop models to classify these graph datasets [151] [186]. Two different branches of graph classification exist; classifying individual elements (vertices or edges) within a graph and classifying the entire graph itself. In this chapter, we are considering the second of these two problems; global graph classification. Global graph classification is required for a myriad of tasks within the field of network analysis (for example the identification of unique chemical compounds within Cheminformatics [152] or the identification of a unique social network user via a graph of their complete social circle [151]).

The volume of graph data, both in terms of size and complexity of individual graphs and the total number of available graphs, is increasing rapidly [159]. The current Facebook social network graph, for example, contains over one billion unique vertices (users) [66]. Traditionally, graph classification has been performed via graph kernels [227], but such methods can take a prohibitively long time to compute, even on comparatively small graphs of a few thousand vertices [152]. This lack of performance makes the applicability of graph kernels questionable on modern massive graphs. Thus, a new approach to massive graph classification is needed which does not require the use of graph kernel methods.

In this chapter, we thus also present a novel approach for both multi-class and binary classification of massive complex graphs entitled Deep Topology Classification (DTC). DTC,

unlike previous approaches, extracts both global and local topological features from each graph to transform it into n-Dimensional feature space. To perform the classification, a deep neural network is designed and trained. The approach is shown to be more accurate than the current state-of-the-art topological feature graph classification method.

3.1.3 Chapter Contributions

The key contributions of this chapter are as follows:

- Development of the graph fingerprint technique, a descriptive topological feature representation of a given graph, to validate the hypothesis that combining aggregated characteristics of vertex-level local neighbourhood structure, with simple global graph features can accurately represent graph structure.
- Introduction of a parallel approach using Apache Spark and GraphX to measure graph similarity – the first approach to explore the use of these systems. The approach is shown to scale sub-linearly to increases in dataset size and to be effective when processing graphs of over 100 million vertices, an order of magnitude greater than seen in the literature. The approach also scales from running on a single machine to a dedicated cluster.
- Demonstration that graph fingerprints are able to compare the topological structure of two graphs. The approach is shown to be more sensitive at detecting variations in graph size and topology than existing approaches. This is achieved by exploiting the combination of both global and local features when performing graph comparisons.
- Construction of a deep neural network for global graph classification using graph fingerprints to accurately predict the class in both a multi-class and binary setting. The Deep Topology Classification approach is shown to be more accurate than competing state-of-the-art methods and is, to the best of our knowledge, the first approach in the literature to make use of a deep neural network for feature-based global graph classification.

To aid in reproducibility of the results presented in this chapter, all of the associated code has been open-sourced and made available online. In addition, results are presented upon public benchmark datasets. The code for graph fingerprint extraction and comparison is available here - https://github.com/sbonner0/GFPX-GraphSimilarity and the code for performing classification of graphs via their fingerprints is available here - https://github.com/sbonner0/DeepTopologyClassification.

3.2 Previous Work

3.2.1 Graph Comparison

It has been argued [25] [133], that the various label dependant and independent methods for graph comparisons can be further categorised into three major cross cutting classes: graph-isomorphism based methods, iterative methods and feature extraction based methods. Prior work [25] [29] has shown feature extraction based methods to be more scalable and flexible, thus are the focus of this chapter.

Feature Extraction

A range of features can be extracted from a graph for comparison with other graphs — the more similar two graphs the more similar their features. Feature extraction based methods have advantages over other approaches as they can be highly scalable – thus have faster runtimes [133]. However, determining which features to extract to give the best, yet most compact, representation of a graph, is an area of active research [181].

One such feature extraction method presented by Roy et al. extracts a variety of centrality measures (used to rank the importance of a vertex within a graph [158]) and uses them for graph comparison [199]. This approach requires that the graphs are labelled and has only been validated on anything on small graphs, with the largest dataset having only 20,000 vertices. An alternative feature extraction method presented by Papadimitriou et al. has been used to measure the similarity between snapshots of a graph of links between webpages [181]. In this approach several similarity measures are tested on a time-series of graph data with the goal of detecting anomalies between time-steps. However, many of the methods tested rely on labelled data to compute similarity.

The NetSimile algorithm [25] relies upon extracting details about the EgoNet[2] for each vertex within a graph which is then compared, via a distance metric, with results from other graphs. In the presented results, NetSimile is shown to be independent of graph size when making the comparison and only considers the similarity of the underlying linking model, meaning that two

[2] A vertex's EgoNet is every other vertex which is connected to it in its local neighbourhood

graphs of vastly different scales could be identified as 'similar'. NetSimile does not run on a parallel graph analytic platform, thus limiting the size of graph it can compare.

Feature extraction has been explored outside of similarity measurement as a way of classifying graphs based on comparisons between global features and labels [152]. Additionally feature extraction has been explored by the anomaly detection community as a way of detecting unusual elements or events within static and temporal graphs [6].

Parallel Graph Similarity Measures

To date, there has been little work on comparing graphs in parallel or on how to efficiently compare graphs of millions of vertices or edges. A recent approach is entitled 'DeltaCon' [134] which compares the similarity of two graphs based upon common labelled vertices. Whilst the approach is stated to be scalable, only a dataset of 16M vertices is tested and a parallel version is only hypothesised, not implemented. A parallel approach for graph similarity using a Message Passing Interface (MPI) compute cluster has been created [131]. The approach is shown to scale to over 1000 compute cores and to a graph size of over one million vertices. However the approach does not produce a final similarity score for two graphs, instead the algorithm matches the similarity of each vertex in one graph to every vertex in the second, thus is very computationally expensive and cannot scale to truly massive graphs.

3.2.2 Graph Classification

The field of graph classification can be divided into two major categories; within graph classification and global graph classification. Within graph classification encompasses techniques designed to classify individual vertices or edges within a single graph and has been extensively explored by prior work [87]. Global graph classification techniques attempt to classify the type or domain of an entire graph. However, there is comparatively less research focusing on the classification of the entire graph, perhaps owing to the lack of sufficient quantity of publicly available datasets and the complexity of discovering an appropriate vector representation.

Within Graph Classification

Recent work on vertex classification has explored the use of a single hidden-layer neural network to learn features required for classification in an unsupervised manner. These approaches are inspired by the word2vec [165] or SkipGram [89] methods for automated feature learning from text documents, adapting the technique to fit graph data. The DeepWalk approach [186] uses a random walk to sample the structure of a vertices neighbourhood which is then fed into the SkipGram model, with the sequence of vertices replacing the sequence of words within a sentence. DeepWalk has been shown to be more accurate at classifying vertices in a variety of datasets than traditional methods like SpectralClustering [220] and EdgeClustering [219]. The node2vec [87] approach expands upon this method by having a flexible definition of a vertices neighbourhood. This is achieved by biasing the random walk to explore the vertices close or far from a given vertex, leading to a greater understanding of its local or global role within a graph.

Global Graph Classification

Graph Kernel Methods - A large body of work has been performed to classify graph datasets based upon graph kernels, a series of kernel functions which compute an inner product on graphs. In general, a kernel function $k(x, x')$ is a function which measures the similarity of two entities x and x' given two constraints: it must be symmetric and positive semi-definite [227]. Such kernels for graphs include random walks [76], shortest path [39] and discriminative subgraphs [221]. Subgraph kernels (sub-graphs which are found frequently in a given graph) are perhaps the most explored for performing graph classification [88, 121]. Subgraphs are conceptually very similar to Network Motifs [102] and Graphlets [187] commonly used in the biological sciences, which all represent ways of identifying small and significant repeating patterns of connectivity between vertices. In such approaches frequent discriminative sub-graphs are mined using a variety of kernel techniques and used as features for classification. Work has shown that larger subgraphs result in a more accurate classification but at the cost of a greatly increased runtime for feature extraction [88]. The use of sub-graph kernels for classification has been further explored when considering noisy and unbalanced datasets [179]. Graph kernels have been explored for multi-label classification of graph datasets in an approach entitled gMLC [132]. The gMLC approach uses an SVM to train a model to assign one or more labels from a set of possible labels to a range of medical and biological graphs. Graph kernels have also been utilised as a way to classify streams of massive time-series graph datasets in a memory efficient manner [239]. The approach

uses the Weisfeiler-Lehman graph kernel [209] and an SVM in an incremental manner. To reduce the large memory footprint inherent in graph streams, the support vectors from the previous time steps are used as training data for the model.

However, graph kernels are known to be prohibitively slow to extract from large graphs [87], thus are not suitable for our approach as we attempt to classify massive graphs.

Topological Feature Methods - There are a few approaches to graph classification which employ the extraction of topological features rather than the use of graph kernels. These approaches are designed to overcome the inherent problems of scalability and runtime efficiency required when extracting graph kernels [152]. There are numerous features which can be extracted from graphs and the technique has been used successfully for many graph mining tasks including graph similarity measurement [29], time series anomaly detection [6] and link prediction [3].

Work has been performed to explore the application of topological graph features to differentiate between graphs from different domains[117]. Although the work stops short of creating an actual classifier, it does conclude that both local and global features can be useful in identifying a graph's domain [117] which concurs with our ideas here.

Li et al. propose a novel method of classifying graphs into domains based on the extraction of global and label based features [151, 152]. The approach uses an SVM to classify the resulting feature vectors. The features are scaled using both range normalisation and z-normalisation to overcome the different scales of chosen features. The work presents results on the classification of three different graph datasets including chemical compound graphs, protein graphs and cell graphs. The approach is shown to be more accurate than state-of-the-art graph kernel based methods [151]. However, the approach is not extended to datasets in which multiple classes may be present and is missing the potentially rich descriptive features at the vertex level.

3.3 Generating Graph Fingerprints

This chapter explores the extraction of representative features from a graph, which can then be used for a variety of tasks including graph comparison and classification. In this section, we detail the features extracted at both the vertex and also the global graph level. For the local vertex features, we hypothesise that the distribution of vertex neighbourhood-based features is a powerful and distinctive feature of graphs, that could be used as an alternative to the more complex path and walk based features found in graph kernel approaches [136].

3.3.1 Vertex Features

For the Graph Fingerprint approach, we extract a variety of features from each vertex in the graph. Although a wide selection of vertex feature metrics exist, each exhibits different characteristics in terms of topological structure being measured and extraction runtime. This chapter is exploring the hypothesis that local neighbourhood features are able to accurately identify graphs. As such, many of the features extracted represent different properties of a vertices neighbourhood. Additionally, two vertex level centrality measures are also included in the feature vector. These were included as they also exploit local neighbourhood information to calculate vertex importance. However, it is important to note that other features could also be incorporated into this process if other characteristics of the graph are important. Below we detail the features extracted, where a value is computed for each vertex $v \in V$.

- **Total Degree** - The sum of both the in and out degree for a vertex v, denoted as k_v.
- **Two-Hop Away Neighbours** - The number of two-hop away neighbours from the current vertex v gives an indication of how connected, and thus how important, a vertex's neighbourhood is within the graph [25]. It can be defined as:

$$TH(v) = |N'(v)| \tag{3.1}$$

  where $N'(v)$ is the set of vertices two-hops away from the current vertex v.

- **Local Clustering Score** - The local clustering score for vertex v represents the probability of two neighbours of v also being neighbours of each other [231]. It is detailed more in Section 2.3.
- **Average Clustering of Neighbourhood** - The average clustering score of the neighbourhood is computed for each vertex by taking the mean of all the local clustering scores for the vertex's neighbourhood [25]. It can be defined as:

$$nc_v = \frac{1}{|N(v)|} \sum_{\forall j \in N(v)} LC(v), \tag{3.2}$$

  where $LC(j)$ is the local clustering score computing for vertex j.

- **Eigenvector Centrality Value** - The Eigenvector centrality is used to calculate the importance of each vertex within a graph by measuring neighbour importance [27]. More details on this as well as the equation for computing it can be found in Section 2.3.4.
- **PageRank Centrality Value** - The PageRank centrality method was originally developed by Google, however it is now commonly used to measure the local influence of a vertex within a graph [94, 178]. The equation for computing this value can be found in Section 2.3.4.

3.3.2 Graph Fingerprint Feature Vector Creation

After the extraction process detailed above is complete, a feature matrix $\mathbf{F} \in \mathbb{R}^{|V|,n}$ is created, where n is the number of vertex level features extracted – six in this case. In order to create the graph fingerprint, it is required to reduce the dimensionality of the feature matrix down to a single vector. To perform this transformation, a series of metrics are taken for each of the feature columns in the matrix. The metrics chosen are the mean, standard deviation, variance, skewness, kurtosis, minimum value and maximum value. These are frequently used and well understood methods to capture the numerical variation within a range of values [25]. After this has been completed, the resulting vertex feature vector $\mathbf{f}_G$ for graph G can be created. The vertex feature vector contains the eight aggregation scores for each column in the feature matrix $\mathbf{F}$ which are concatenated together:

$$\mathbf{f}_G = (\bar{x}_1, \sigma_1, \sigma_1^2, Skew[x]_1, Kurt[x]_1, x(1)_1, x(n)_1, ... \\ , \bar{x}_n, \sigma_n, \sigma_n^2, Skew[x]_n, Kurt[x]_n, x(1)_n, x(n)_n). \quad (3.3)$$

3.3.3 Global Features

In order to make the graph fingerprint approach sensitive to the global features of a given graph, a selection of global features are extracted in addition to the vertex ones discussed above. The global features which were chosen to represent each graph, were selected due to their ability to capture key elements of global graph topology, whilst also being efficient to compute in a

distributed environment. We focus on extracting a small quantity of these such features as we are primarily interested in neighbourhood level connectivity. A vector is used to represent these six global graph features which are then concatenated onto the aggregated vertex level feature created above, resulting in the final graph fingerprint.

- **Graph Order** - Defined as: $|V|$.
- **Number of Edges** - Defined as: $|E|$.
- **Number of Triangles** - The total number of triangles for a given graph is the number of vertices which form a triangle, with a triangle being a set of three vertices with an edge between every member.
- **Maximum Total Degree Value** - This represents the total number of edges the most connected vertex in the graph has to other vertices.
- **Number of Components** - This is the total number of components within the graph, with a component being a sub-graph in which there is a possible path between every vertex, whilst vertices in different components have no possible path between them.
- **Global Clustering Coefficient** - This feature is a representation of how connected the graph is overall, using the total number of possible vs complete triangles within a graph.

3.4 Graph Comparisons via Topological Structure

This section explores how the graph fingerprints can be used to allow for efficient and accurate comparisons between graphs based on topological structure. In this context, two graphs can be said to be similar if they share similar global and micro (vertex and edge) level topological features. The approach, entitled Graph FingerPrint Comparison (GFP-C), was required when it was found existing serial methods for graph comparison were unable to scale to massive scale graphs and slow when comparing even modest sized ones. Additionally, based on the literature presented in Section 3.2.1, it is clear there are gaps in the currents methods. Particularly, an approach which meets the following criteria is missing:

1. *Scalability* - Highly scalable to massive graphs of millions of vertices/edges, and capable of computing the similarity in a finite time.

2. *Sensitivity to Graph Size* - Taking the size and order of the graphs into consideration.
3. *Sensitivity to Similar Topologies* - Detecting the difference between graphs which are highly structurally and topologically similar.
4. *Label Free* - Able to perform comparisons without requiring labelled datasets, although the approach should still function when they are available.
5. *Low Number of User Defined Parameters* - A minimum number of user defined parameters should be required to measure graph similarity.

3.4.1 Graph Comparison Approach Overview

The approach comprises two distinct stages: the generation of a graph's fingerprint (GFP-X), as described in Section 3.3, and the comparison of these fingerprints (GFP-C). The GFP-X approach takes the high dimensionality inherent in complex graphs and reduces this down into two fixed length feature vectors. The GFP-X approach achieves this by extracting micro and macro features from the given graph, allowing it to capture both the micro and macro-level topological features. The decision to extract both vertex level and global level features was driven by the desire to make the comparison between graphs more sensitive to small variations in the underlying graph topology and the overall size of the graph than the current state-of-the-art methods [25].

During the process of GFP-X (detailed in Section 3.3), both the Vertex and Global generation produce a feature vector for each graph. Graphs can then be compared by computing the distance between their feature vectors - in this work we use the Canberra distance metric [141]. This results in two separate similarity scores, one comparing the vertex level topology and one the global level similarity. The last stage is to combine these two scores to produce the final similarity score between two graphs.

To help fulfil the scalability criteria established in Section 3.4, GFP-X and GFP-C have been written to make use of a distributed parallel processing framework called Apache Spark [245], which enables the processing of graphs to be performed across multiple machines. At the time of this work being performed, alternative parallelization approaches such as the use of GPUs, could not work with the size of graphs required, or scale past being run on a single machine [211]. The work performed to achieve the Apache Spark implementation, as well as other details, is documented in Appendix A.

3.4.2 Comparison of Graph Fingerprints

The GFP-C approach compares the fingerprints of two graphs in order to compute the similarity between them. In this work, the Canberra distance was selected to compare the numerical distance between the fingerprints, similar to [25]. Other distance metrics tested included the Bray, Correlation, Chebyshev, Cosine and Manhattan but these were found to be insensitive when the feature vectors were highly similar, or produced unintuitive results such as a high similarity score for highly dissimilar graphs.

The Canberra distance between two vectors $\mathbf{p}$ and $\mathbf{q}$ of n dimensions is defined as [141]:

$$CD(\mathbf{p}, \mathbf{q}) = \sum_{i=1}^{n} \frac{|\mathrm{p}_i - \mathrm{q}_i|}{|\mathrm{p}_i| + |\mathrm{q}_i|}. \tag{3.4}$$

It should be noted that when p_i and q_i are both equal to zero, there is no defined value for the distance and a score of zero is returned. Additionally, the maximum value returnable by the measure is equal to the number of dimensions in the two vectors being compared. For example, comparing two vectors of ten dimensions would result in a maximum possible Canberra distance of ten. Additionally, the Canberra distance is able to accurately detect changes close to zero, which makes it ideal for detecting small variations between graphs which might be highly topologically similar – one of the key goals for the GFP-C approach. The Canberra distance is used to compare both the distance between the vertex feature vectors and the global feature vectors. Two graphs are more 'similar' the closer the result of the Canberra distance is to zero, with a score of zero indicating that the graphs are 'fingerprint' identical.

3.4.3 Final Similarity Score Generation

The GFP-C approach returns two similarity scores, one for the distance between the vertex feature vectors $\mathbf{f}_v$ and one for the distance between global vectors $\mathbf{f}_g$ for the two graphs being compared. These two scores can be used independently to compare the global and local topological structure as separate entities. However, the GFP-C approach can produce a final similarity score between the two graphs, using the following aggregation - $FinalSimScore = \mathbf{f}_v + \gamma\mathbf{f}_g$. Where γ is a user controllable parameter to control the weighting of the difference between the global feature vectors in the final similarity score.

3.5 Graph Classification using Topological Structure

This section discusses the use of topological features as a means to perform global classification of complex graphs. Global graph classification can be considered a supervised problem in the context of Machine Learning (this was explored in Section 2.5.1), where individual graphs within a dataset D have associated class labels $y_i \in C$, where C is the set of l categorical class labels, given as $C = 1, ..., l$. The goal of the global graph classification task is to derive a reliable way to perform $f : D \rightarrow C$ which can accurately predicate the class label of each graph in the dataset. This work will explore the use of topological features combined with neural networks to learn the function f. Established models for performing classification, such as Support Vector Machines, Decision Trees or traditional Artificial Neural Networks (Often refereed to as Multilayer Perceptions (MLP) in the literature [80]) do not function directly with graphs since these models require an n-Dimensional vector as input. Therefore before any graph can be passed to the function, its inherent discrete nature must first be converted into a vector. Due to the size and complexity of modern graphs, this can be considered one of the most challenging aspects of global graph classification [152].

The approach presented in this section, designed to tackle this problem, is entitled Deep Topology Classification (DTC). DTC extracts both global and deep topological features from a given graph, rather than using a graph kernel method for feature representation. This approach takes inspiration from research showing how the use of global topological features can be used to outperform the classification accuracy of graph kernel based methods – the current state-of-the-art for tackling graph classification [152]. The DTC approach further improves upon this research by exploring the use of deep topological features extracted from the vertex level of the graph, rather than just global metrics. An additional benefit over the previous graph kernels based approaches is that the feature extraction procedure can be completed efficiently and in parallel. To classify the resulting vector representations, a deep feed-forward neural network is created and trained. The use of a deep neural network, rather than the traditional SVM utilised in the global graph classification literature, was inspired by recent advances in within-graph classification using neural networks [87].

3.5.1 Classification Model Design

The ANN created for the DTC approach follows the Deep Feed Forward (DFF) model. The size of the input layer is equal to the dimensionality of the extracted fingerprint vector and

the size of the output layer is equal to the number of unique categorical class labels. When designing a neural network, several key choices must be made in regards to the number of hidden layers, the random initialisation of the neuron's weights and the activation function they use. In addition a suitable loss function, a function which the ANN is trying to minimise must be chosen to ensure the most accurate model. To select the correct functions and parameters for the DTC network, a grid search was performed over a selection of well regarded options. For the initial random weights assigned to the neurons, the following functions were tested to generate the initial weights: Normal, Glorot Uniform, Lecun Uniform and He Normal [54]. For the neuron activation function the following were tested: Tanh and Rectified Linear Unit (ReLU) [95]. The grid search trained a series of networks with every possible combination of these functions and records the combination which resulted in the highest model accuracy. The network with the highest classification accuracy featured ReLU activation and initialisation via Glorot Uniform. Additionally the use of one, two and three hidden layers in the model was used to give some indication as to the complexity of the global graph classification task.

The ReLU function activates a neuron via $f(x) = \max(0, x)$, where x is the incoming signal to the neuron, which thresholds the activation to stop it going below zero and is designed to more accurately imitate biological activations [95]. ReLU has been shown to improve accuracy in many Deep ANN's, whilst also improving training times. Before the weights in an ANN are updated via back-propagation, they must be assigned some random value. This initial value has been shown to have a large impact on the overall network quality [78]. The Glorot Uniform initialisation method sets the initial value for a neuron to be sampled randomly from a uniform distribution and has been shown to improve accuracy [78]. For the loss function of the network categorical cross-entropy was used, commonly employed for multi-class classification tasks [79]. The RMSprop algorithm was used to update the model weights via back propagation [222]. Finally, small amounts of dropout (a dropout probability of 0.2) were used on each hidden layer, this is a regularisation strategy for ANNs which functions by randomly dropping neurons in an effort to prevent over-fitting [214]. An overview of the complete network, describing the size of each layer, the initialisation and activation functions used and the application of any dropout, is given in Table 3.1.

To ensure that the approach can also classify binary datasets (datasets for which only two unique labels are present), a second version of the DTC model was created. This version employs an alternative output layer with a single output neuron activated via a Sigmoid function, commonly used for binary classification tasks [60]. In addition, this network used binary cross-entropy for the loss function [79]. The alternative output layer can be seen in Table 3.1.

Table 3.1: The Deep Topology Classification model architectures.

Layer	Size	Initialisation	Activation	Dropout
Input	n	-	-	-
First Hidden	256	Glorot-Uniform	ReLU	0.2
Second Hidden	128	Glorot-Uniform	ReLU	0.2
Third Hidden	32	Glorot-Uniform	ReLU	0.2
Multi-Output	$\|C\|$	-	Softmax	-
Binary-Output	1	-	Sigmoid	-

3.5.2 Implementation

The code for the DTC approach has been written in Python programming language. The feature extraction code has been implemented using Graph-Tool [63]. The ANNs have been created using the TensorFlow and Keras [1] libraries, allowing exploitation of General Purpose Graphics Processing Unit (GPGPU) cards to decrease training times. The SVM models have been implemented using SciKit-Learn [185].

3.6 Experimental Evaluation

This section will detail the experimental evaluation and datasets used to assess the ability of Graph Fingerprints to be used for graph comparison and classification.

3.6.1 Comparision Datasets

The synthetic graphs used throughout the comparison results section (including Forest Fire [145] and Erdős-Rényi [70] random graphs) were generated using the SNAP graph analysis package [147]. The Forest Fire generation method was introduced by Leskovec, and produces more realistic synthetic graphs than the frequently used Barabási-Albert as it replicates more features seen in empirical graphs [145]. For all Forest Fire graphs used in the results section, the forward burning probability was set to 0.35 and the backwards burning probability set to 0.32. These values produce graphs which approximately follow $|E| = |V| * 4$. The empirical

Table 3.2: Empirical graph datasets used to assess graph comparisons using Graph Fingerprints

Dataset	$\|V\|$	$\|E\|$	$\%VinLCC$	$NumTriangles$
soc-Slashdot0902	82168	948464	100	602592
ca-HepPh	12008	118521	93.3	3358499
com-DBLP	317080	1049866	100	2224385
loc-Gowalla	196591	950327	100	2273138
wiki-Talk	2394385	5021410	99.8	9203519

data used was taken from the widely used SNAP datasets repository [146]. A summary of the datasets used can be seen in Table 3.2. The datasets are taken from a range of domains including collaboration, communication and social networks.

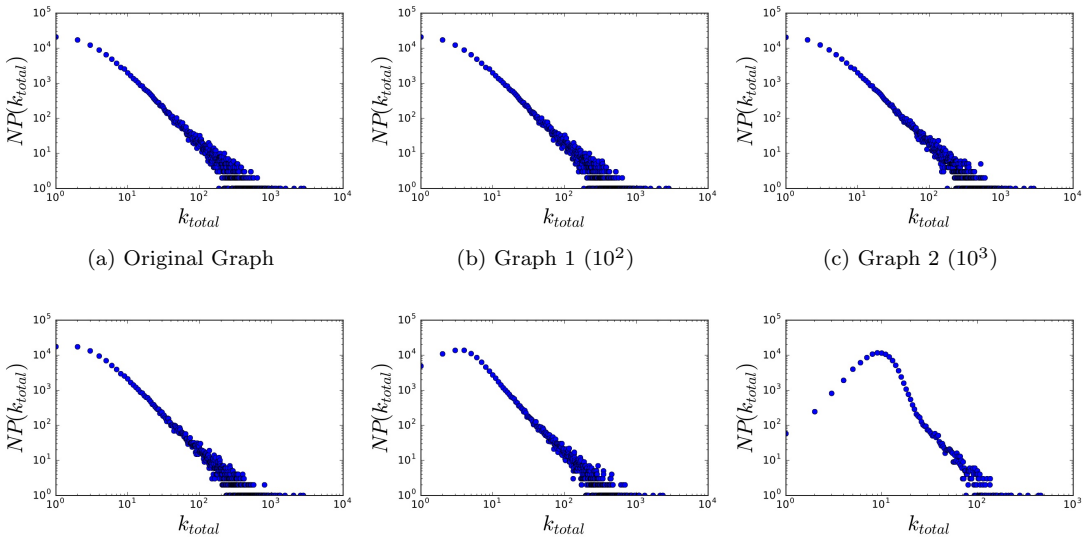
Random Rewire Graph Generation

To demonstrate that the GFP-C approach is highly sensitive to changes in the underlying topology of a given graph, the edges in a Forest Fire graph with 100,000 vertices were rewired in a random manner, as detailed in Section 2.4.2. Figure 3.1 shows how the degree distribution of the original graph was altered by the random rewiring process, where the number after the graph name indicates the quantity of edges rewired. The figure plots the number of vertices $NP(total)$ with a specified total degree value k_{total} and illustrate how the internal connectivity of the original Forest Fire graph is altered as more edges are rewired.

3.6.2 Comparision Testing Methodology and Environment

All the experiments for graph comparison presented in this chapter were performed upon a small development Hadoop cluster consisting of a head node with a 6 core Intel Xeon E5-2609v3, 64GB RAM and 1TB of SSD storage. In addition, the cluster contains 4 worker nodes each with 2 * 8 core Intel Xeon E5-2630v3, 64GB RAM and 1TB of SSD storage. All nodes in the cluster are connected via a dedicated SFP+ 10Gb network and run the same software stack of CentOS 7.2, Java 1.8, Scala 2.10.5, Apache YARN 2.7.1 and Apache Spark 1.6.1. All experiments using Spark were run using YARN to allocate cluster resources in the form of containers.

For all experiments, γ was set to 2 to increase the weightings of the global features in the final similarity score.



(a) Original Graph (b) Graph 1 (10^2) (c) Graph 2 (10^3)

(d) Graph 3 (10^4) (e) Graph 4 (10^5) (f) Graph 5 (10^6)

Figure 3.1: Change In Degree Distribution After Rewiring Process.

3.6.3 Classification Dataset Generation

As outlined in Section 2.4, large quantities of graph data are not readily available within the public domain. This makes the assessment of a global graph classification approach challenging as numerous (at least in the order of hundreds) examples of graphs from each of the classes being identified would need to be present, something that is not present in the standard benchmark datasets. Additionally, having data generated by a known algorithmic processes can serve as a very reliable source of ground truth. Due to these issues, two balanced synthetic datasets were created using a combination of five mathematically understood random graph generation methods from the SNAP graph library [147]. More details on the generation of random graphs can be found in Section 2.4.1. The two datasets created for the experimentation are detailed below:

Dataset One (Multi-Class) - Containing 10,000 graphs from each of the five generation methods, creating a final dataset of 50,000 graphs, with five balanced classes. This dataset was created to test the ability of the *DTC* approach at multi-class classification.

Dataset Two (Binary Classification) - Containing 10,000 forest fire graphs and 10,000 randomly rewired forest fire graphs. The goal of this dataset was to test the sensitivity of the *DTC* approach at classifying graphs which are highly topologically similar but of two different classes.

The forest fire graphs represent a normal distribution of graphs, whereas the rewired graphs represent anomalies where small changes have been made to their topologies. The random rewire process modifies a given source graph's topology by randomly altering the source and target of a set number of edges according to the Erdős-Rényi random model. The number of edges each graph was rewired by was chosen uniformly from a possible range of 100 to 10,000.

Many of the graph generation methods used require parameters to control aspects of the generation process. To avoid our models over fitting to a particular set of generation parameters, these we uniformly randomised by the amounts detailed below. Each graph was generated with 100,000 vertices and a varying number of edges controlled via the generation method.

The chosen generation methods cover a broad range of possible graph topological structures, with a particular focus on those found in the social, web and citation domains. However the type of structures found in many biological graphs (particularly graphs of brain connectivity) are not typically covered by these generation approaches. Such graphs tend to exhibit hierarchical [190] or modular [112] structure, however we leave analysis over these types of graphs as possible future work. The final generation methods chosen were:

- *Forest Fire (FF)* [145] - The forward and backward burn probabilities were chosen uniformly between 0 and 0.5.
- *Barabási-Albert (BA)* [7] - The number of connections made by each new vertex joining the graph was chosen uniformly between two and six.
- *Erdős-Rényi* [70] - No parameters were randomised for this method as edges are made at random, with each vertex having a mean degree of two.
- *Small World (SW)* [231] - The Watts-Strogatz Small world model was designed to generate random graphs whilst accounting for, and replicating, features seen in real-world graphs – specifically, to maintain the low average shortest path lengths of the ER model whilst increasing local clustering coefficient. The rewire probability for the small world model was chosen uniformly between 0 and 0.5.

- *R-MAT (RM)* [47] - The R-MAT graph generator uses a recursive matrix technique to generate realistic graphs. It requires the probability that a certain edge will fit into one of three partitions within a 2×2 matrix. These probabilities are uniformly chosen to sum to less than one, with the mean degree being two.

3.6.4 Classification Testing Methodology and Environment

All the accuracy scores presented in the results section are the mean accuracy after k-fold cross validation, considered the gold standard for model testing [11]. For k-fold cross validation, the original dataset is partitioned into k equally sized partitions. $k-1$ partitions are used to train the model, with the remaining partition used for testing. The process is repeated k times using a unique partition for testing and a mean taken to produce the final result.

Experimentation was performed on a compute system with 2 Nvidia Tesla K40c's, 20C 2.3GHz Intel Xeon E5-2650 v3, 64GB RAM and the following software stack: CentOS 7.2, GCC 4.8.5, CUDA 7.5, CuDNN v4, TensorFlow 0.10.0, Keras 1.0.8, scikit-learn 0.17.1, Boost 1.56, Python 2.7.5 and Graph-Tool 2.8.

3.7 Results - Graph Comparison

In this section, the GFP-C approach is assessed against the criteria as discussed in section 3.4. In each experiment, GFP-C is compared to the current state-of-the-art feature extraction graph comparison method – NetSimile [25]. As both the GFP-C and NetSimile approaches generate their final similarity scores using the Canberra distance, their results are directly comparable. It is worth highlighting that other distance metrics, used in place of the Canberra distance, would produce similar disparities between the results of the two approaches. When using the Canberra distance metric to compare graph feature vectors the closer to zero the result, the more similar the graphs. Thus a larger Canberra distance score indicates that two given graphs have a more dissimilar topological structure.

3.7.1 Sensitivity to Variations in Topology

For the results presented here, an original Forest Fire graph (with 10^5 vertices) was compared to each of the rewired graphs (discussed in section 3.6.1) to measure the similarity between them. Figure 3.2 shows that GFP-C is sensitive to the changes in the topology of the graph, with an increase in the percentage of the graph rewiring always being detected as more dissimilar to the source graph. The result shows that, not only is GFP-C comparable to NetSimile (NS), but it is more sensitive to topological change indicated by the higher value of the Canberra distance.

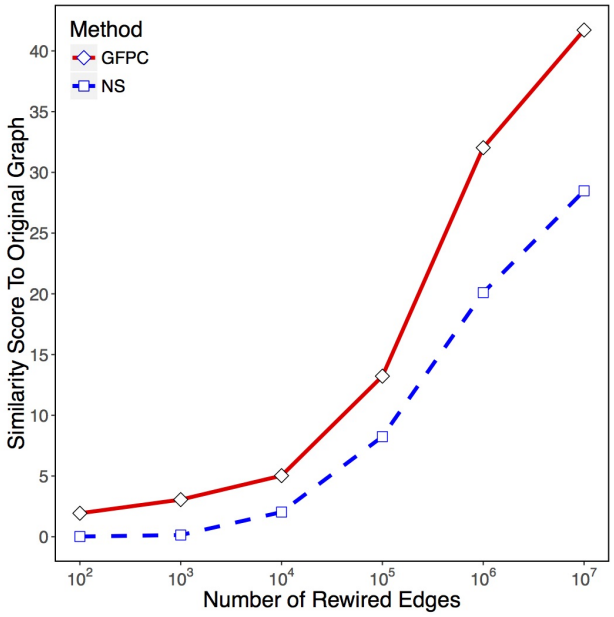


Figure 3.2: Measuring sensitivity to changes in topological structure after random rewiring using Graph Fingerprint Comparison (GFP-C) and NetSimile (NS).

3.7.2 Sensitivity to Variations in Size

The GFP-C approach was tested for its sensitivity at detecting variations in global graph size. For this experiment, a random Forest Fire graph G_o was generated with $|V| = 10^4$ and $|E| = 10^{4.6}$. To compare with the source graph, six new graphs were generated again using the Forest Fire method each with varying numbers of vertices and edges. As the Forest Fire method

was used to generate all graphs, they will be highly structurally similar in their topologies. The results comparing the GFP-C and NetSimile method for sensitivity to variations in graph size are displayed in Figure 3.3. In the figure, graphs of varying sizes were compared to the original graph G_o to generate the similarity score.

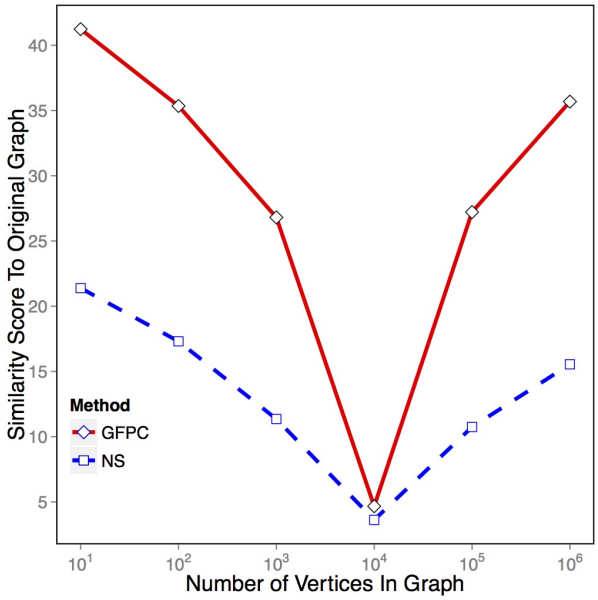


Figure 3.3: Measuring sensitivity to changes in the size of the graph using Graph Fingerprint Comparison (GFP-C) and NetSimile (NS). Note that scores are presented when comparing against an ordinal graph G_o with $|V| = 10^4$.

Figure 3.3 highlights that the GFP-C approach is more sensitive to variations in graph size than the NetSimile method, with a change in size of the graph always detected as more dissimilar to the source graph. It is interesting to note that GFP-C detects the graph of the same size as the source graph as being highly similar, showing that it is strongly affected by global graph size when making comparisons.

3.7.3 Runtime Analysis

The final criteria evaluated was the the runtime of the GFP-X feature extraction algorithm across a range of empirical data sources, as well as comparing it to NetSimile. This is an

interesting experiment as the reason for implementing GFP-X using Spark and GraphX was to improve both runtime performance and the size of graphs that can be compared. For this comparison, an implementation of the NetSimile approach in Graph-Tool, a highly efficient C++ graph analysis library which uses OpenMP to scale across multiple cores in a shared memory system [63], was used. All the measures of runtime presented incorporate reading the graph data into memory from HDFS or Disk as well as the YARN scheduling and allocation decision times. As such, the presented runtimes are the total time taken to produce a final result from the initial job submission. As NetSimile is not a distributed approach, its timings were obtained by running it upon a single node from within the cluster. For fair comparison, GFP-X was also run upon a single node in addition to the full cluster.

Figure 3.4 shows the runtime of the feature extraction stages for both GFP-X (Running on 1 ($1E$) and 12 ($12E$) Spark executors on the cluster) and NetSimile, across the datasets in Table 3.2, with the results being the average of five experiments and the error bars being one unit of standard deviation. Whilst a direct comparison is difficult, due to GFP-X and NetSimile being implemented in different languages, the figure does highlight some interesting results. Firstly, it is clear that when running upon a single compute node GFP-X is significantly, often by over an order of magnitude, faster than the C++ based NetSimile. Secondly, due to the comparatively small size of datasets used, running across all nodes in the Spark cluster does not always result in a decrease in runtime. It's only when running on the largest dataset, wiki-Talk, that the inherent costs associated with distributing data across the network become worthwhile.

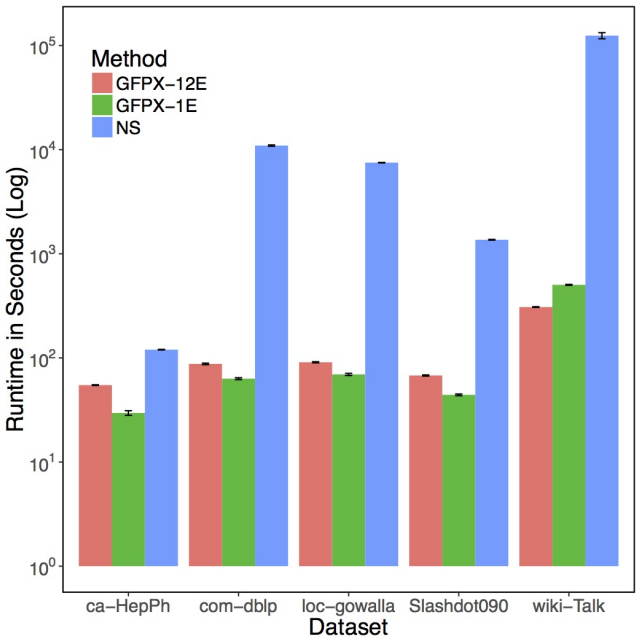


Figure 3.4: Runtime performance for the Graph Fingerprint Extract (GFP-X) and NetSimile (NS) approaches across empirical datasets.

Synthetic Data

In addition to testing on empirical datasets, the runtime of generating a single fingerprint using the GFP-X approach was evaluated across a range of synthetic Forest Fire and Erdős-Rényi graphs when running across the full five node Spark cluster. As the number of vertices was increased in the generated data, the number of edges was kept such that $|E| = |V| * 2$. These experiments were performed to assess the relationship between number of vertices within a range of topologically varying random graphs and the runtime of GFP-X. Again, all experiments were repeated five times and the error bars presented as one unit of standard deviation. The runtime of Apache Spark and GraphX jobs are significantly affected by several key user configurable parameters which control how resources are allocated to the job and how many partitions the data is stored across. For a fair comparison the number of containers, cores, partitions and memory was kept constant across each dataset size. Due to this, the presented runtimes are not the lowest achievable and could have been improved with optimal parameter selection for each dataset size. However, it should be noted that the implicit algorithm for counting connected components in GraphX currently contains an error in the code when scaling to massive graphs, so for all the runtimes measured below this global feature has not been extracted.

Figure 3.5 shows how the runtime of the GFP-X approach responds to increases in the number of vertices within a Forest-Fire graph. The additional line shows a linear relationship between dataset size and runtime. This figure shows that GFP-X responds in close to a sub-linear fashion to increases in the number of vertices within a graph. It can be seen that an increase of an order of magnitude in the number of vertices, never corresponds with same increase in runtime. It is interesting to note that at smaller graph sizes there is little variation in runtime, as it is likely that Spark has a fixed initialisation time (JVM initialisation time, YARN scheduling delay and data distribution) for a job of any dataset size.

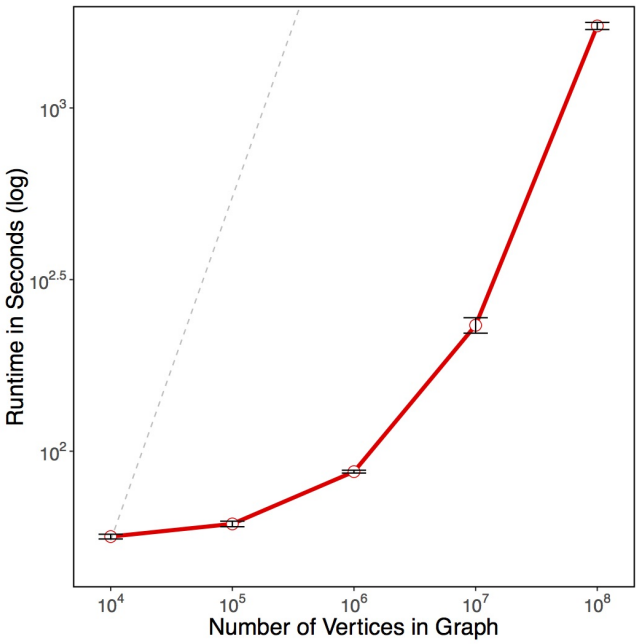


Figure 3.5: Runtime of the Graph Fingerprint Extract (GFP-X) across a range of Forest-Fire graph sizes. The dotted line indicates a linear increase in runtime.

Figure 3.6 shows how GFP-X responds to increases in the number of vertices within an Erdős-Rényi graph. Again it can be seen that the GFP-X approach scales approximately sub-linearly to increases in dataset size. Certainly below 10^7 vertices, the increase in runtime can be considered sub-linear. However the increase from 10^7 to 10^8 requires moderately more than linear time perhaps owning to the random nature of the topologies of Erdős-Rényi graphs not parallelising well. However below 10^8 vertices, the profile of the runtime performance of the Erdős-Rényi run is very similar to the profile of the runtime for the Forest Fire graphs. This suggests that the runtime of the GFP-X is largely independent of the topological structure of the graph being fingerprinted, a desirable quality for a graph mining algorithm.

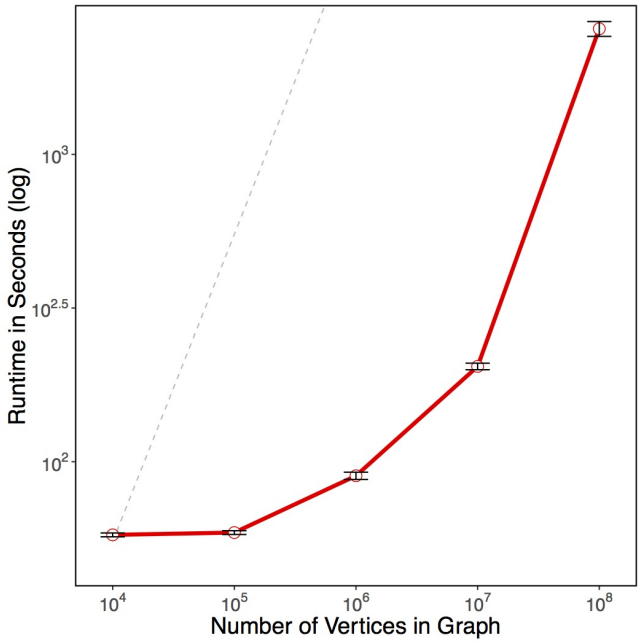


Figure 3.6: Runtime of the Graph Fingerprint Extract (GFP-X) across a range of Erdős-Rényi graph sizes. The dotted line indicates a linear increase in runtime.

3.7.4 Discussion

The GFP-C approach outperforms the current state-of-the-art feature based extraction methods, displaying excellent runtime and can scale to previously unmanageable graph sizes. The GFP-C approach is sensitive to detecting small variations in graph topology and overall graph size. Due to the nature of the features extracted, the GFP-X approach requires no labels with the graph datasets. However, perhaps the most promising result to arise is the sub-linear runtime of the approach when increasing dataset size up-to 10^8 vertices on a modest 4 node Spark cluster.

3.8 Results - Graph Classification

The ability of DTC to perform accurate classification of graphs was assessed via the use of the two datasets described in Section 3.6.3. Previous work has shown global graph features classified via an SVM to be more accurate than state-of-the-art graph kernel methods [151] [152]. As such, DTC is compared with an SVM to act as a baseline approach. To match with the approaches found in the literature, it is trained upon the global features detailed in Section 3.3. Additionally, comparison is made with an SVM trained on the full feature vector to directly assess the suitability of ANNs for graph classification. The SVM model parameters were chosen via a grid search which found a third order polynomial kernel to be the most accurate on average. Finally, all approaches are compared with and without the feature vectors being scaled to have a zero mean and unit variance across each feature. Many machine learning models benefit from the use of features that are standardised to the same range to aid the learning process [105].

For both the multi-class and binary classification results, six different methods are compared:

- *DTC-Scaled*: The DTC model trained on scaled full topological feature vectors.
- *DTC-Unscaled*: The DTC model trained on unscaled full topological feature vectors.
- *SVM-Scaled*: The SVM model trained on scaled full topological feature vectors.
- *SVM-Unscaled*: The SVM model trained on unscaled full topological feature vectors.
- *SVM-Global-Scaled*: The SVM model trained on scaled global only topological feature vectors.

- *SVM-Global-Unscaled*: The SVM model trained on unscaled global only topological feature vectors.

It should be noted that for this dataset, attempts were made to compare performance with several graph kernel approaches, but were unsuccessful as they could not fit in system memory. Due to this issue, a second smaller dataset was generated to allow for the graph kernel methods to run on them and for comparisons to be made with the DTC approach. The comparisons with graph kernel approaches is detailed in Section 3.8.5.

3.8.1 Multi-Class Classification

To assess the accuracy of the DTC approach at performing multi-class classification, Dataset One (detailed in Section 3.6.3) was used. The reported results, displayed in Table 3.3, are the mean accuracy as a percentage of the k-fold cross validation along with the 95% confidence interval. The table shows that the DTC approach has a very high accuracy across the k-fold cross validation run and is over 10% more accurate than the best SVM approach. It can also be seen that using the full feature vector with the SVM is much more accurate than using global features alone. The table also highlights how beneficial feature scaling is to the overall accuracy of both models. Figures 3.7 and 3.8 show the error matrices for the SVM-Scaled and the DTC-Scaled methods respectively. These figures show the predicted against the true labels. The SVM-Scaled approach has difficulty correctly classifying the ER, FF and SW classes, with the ER class more frequently being classified as BA than its true class. However, Figure 3.8 shows that DTC-Scaled is consistently accurate across all classes.

Table 3.3: Multi-Class Classification Results

Method	**Accuracy** (%)	**Recall**	**Precision**	**F1 Score**
DTC (Scaled)	$\mathbf{99.958 \pm 0.074}$	$\mathbf{0.99998 \pm 0.00004}$	$\mathbf{0.99998 \pm 0.00004}$	$\mathbf{0.99998 \pm 0.00004}$
DTC (Unscaled)	70.443 ± 7.819	0.70497 ± 0.07782	0.71247 ± 0.07862	0.70870 ± 0.012931
SVM (Full-Scaled)	88.432 ± 1.100	0.88396 ± 0.00867	0.88426 ± 0.00867	0.884261 ± 0.01097
SVM (Full-Unscaled)	26.113 ± 0.501	0.26079 ± 0.00948	0.25925 ± 0.00501	0.25897 ± 0.00721
SVM (Global-Scaled)	54.483 ± 1.252	0.54451 ± 0.01378	0.54483 ± 0.01252	0.54487 ± 0.01401
SVM (Global-Unscaled)	50.673 ± 1.092	0.50631 ± 0.01301	0.50673 ± 0.01092	0.50681 ± 0.01418

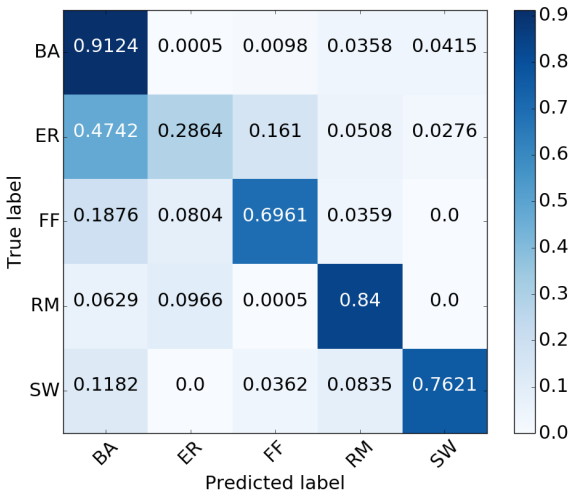


Figure 3.7: Normalized Error Matrix For SVM (Scaled)

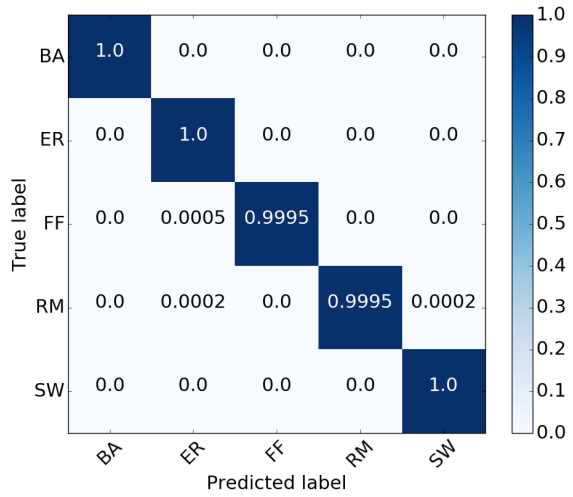


Figure 3.8: Normalized Error Matrix For DTC (Scaled)

3.8.2 Binary Classification

To assess the accuracy of the DTC approach at performing binary classification, Dataset Two (detailed in Section 3.6.3) was used. Here we assess the sensitivity of DTC when classifying

graphs which are highly topologically similar, so good performance in this task would indicate that graph fingerprints are very sensitive to topological structure. Table 3.4 shows the results for the binary classification. DTC achieves a very high accuracy when detecting binary classes, with the DTC-Scaled approach beating the best SVM approach by over 30%. The accuracy of the DTC in this dataset is very encouraging, as the topological distribution of the two classes represented in this dataset are very close.

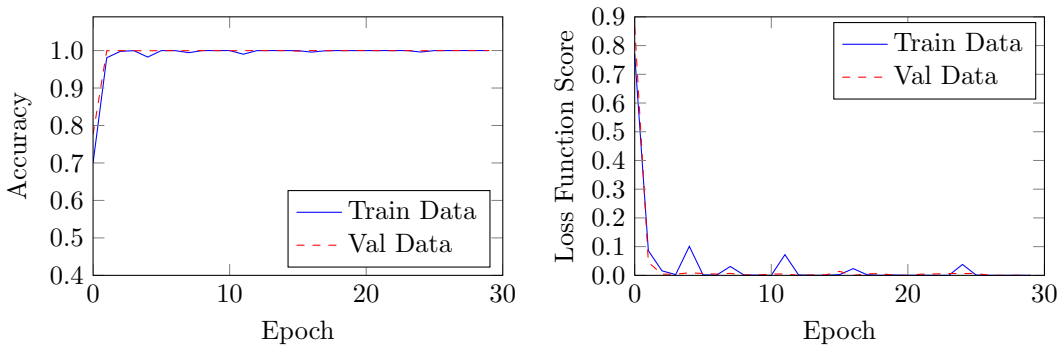
Table 3.4: Binary Classification Results

Method	**Accuracy** (%)	**Recall**	**Precision**	**F1 Score**
DTC (Scaled)	$\mathbf{99.980 \pm 0.049}$	$\mathbf{0.99995 \pm 0.00015}$	$\mathbf{0.99995 \pm 0.00015}$	$\mathbf{0.99995 \pm 0.00015}$
DTC (Unscaled)	51.435 ± 8.793	0.48850 ± 0.00983	0.52710 ± 0.00983	0.507066 ± 0.33614
SVM (Full-Scaled)	68.034 ± 8.821	0.70012 ± 0.31304	0.68034 ± 0.28739	0.71509 ± 0.33614
SVM (Full-Unscaled)	49.045 ± 1.141	0.48910 ± 0.01566	0.49045 ± 0.01141	0.49145 ± 0.00929
SVM (Global-Scaled)	56.482 ± 13.435	0.56780 ± 0.13913	0.57834 ± 0.14034	0.57302 ± 0.13024
SVM (Global-Unscaled)	42.546 ± 2.914	0.42916 ± 0.02959	0.43813 ± 0.03152	0.43359 ± 0.03102

3.8.3 Model Training Dynamics

When training Neural Network based models, the complete set of training data is passed through the network multiple times, with one epoch being a complete pass through the training data. Investigating the changes in model performance over this training process can give insights into how well the model is learning. For example, how quickly the loss curve for a model begins to plateau can give some indication of how complicated the given task is for the model to learn. Additionally, how the performance of the model on both the training and validation sets changes over time can be a useful indication of whether the model is over-fitting to the training data, and thus hurting performance on the validation set.

Figure 3.9 highlights how the accuracy and loss value change as the multi-class model is trained across thirty epochs. The figure shows that for the multi-class dataset, the DTC model learns very quickly to distinguish between the different graph generation methods, as within one epoch the loss curve has already plateaued close to zero. This suggests that the topological information contained within the graph fingerprints is highly representative of the graph's domain. Additionally, Figure 3.9 shows that the model does not exhibit any signs of over-fitting to the training set, as both the training and validations sets exhibit highly similar curves in both the accuracy and loss scores.

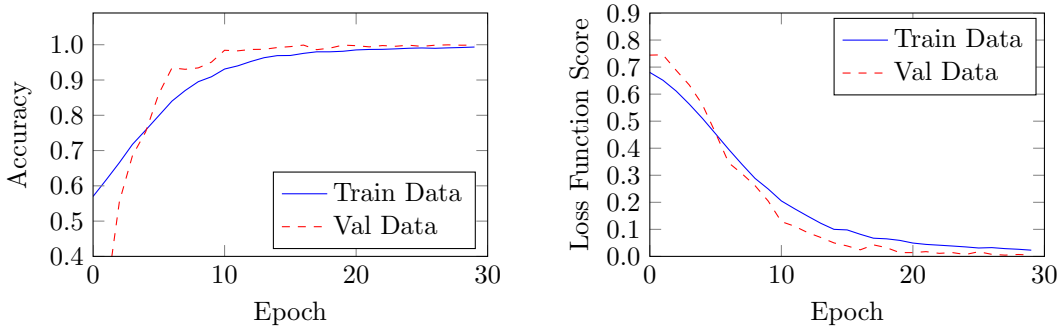


(a) Multi-Class Accuracy

(b) Multi-Class Loss

Figure 3.9: Multi Class Model Accuracy and Loss Score Over Epochs

Figure 3.10 shows how the performance of the model trained to perform binary classification varies over time. Compared with the multi-class model, it is interesting to note that this model takes over ten epochs longer until the improvements in accuracy begin to plateau. This demonstrates that the classification task required by the binary dataset is indeed more complicated for the model to learn correctly as the two classes have a similar topological structure. The Figure also suggests that the model is again not over-fitting to the training set due to the similar curves in the training and validation. This is promising as it shows that the model is able to generalise well to unseen data, even in this more challenging task.



(a) Binary Accuracy

(b) Binary Loss

Figure 3.10: Binary Model Accuracy and Loss Score Over Epochs

3.8.4 Measuring Feature Importance

In this section, experiments are performed to attempt to judge the importance of the various features that make up the graph fingerprint in the overall classification result. This can be interesting as it allows for deeper insight into which topological structures are most useful for the model during the classification process.

To achieve this, a concept similar to occlusion mapping from computer vision is used to test how the model performance changes when input features are removed [207]. To achieve this, a DTC model is trained to convergence as normal on the full training set. However, during the testing phase, features are removed from the input before being passed into the model. The predictive performance of the model with the features missing is then assessed to give some indication of how the performance changes and thus how important the features were. The results of this experiment are presented in Table 3.5. The first thing to note from the table is there is always a reduction in accuracy on the test set when features are removed, although this is quite small in many cases. It is also clear that the model is quite robust to the loss of features, with many of the vertex level features being able to be removed with less than a 1% drop in overall accuracy. This perhaps suggests that there is some redundancy in the features and that some could be removed without sacrificing much performance. It is also interesting to note that seemingly the feature whose removal affects the model the most is that of Eigenvector centrality, perhaps indicating that is a measure which is easily able to distinguish between graph classes.

To perform further analysis of feature importance, the fingerprint vectors were used to train a Random Forest model [40, 100]. Random Forest models allow for the importance of the input features for the final classification result to be measured using techniques such as the gini importance technique [41], which will be used here.

Table 3.6 shows the results of the gini feature importance test performed on the Random Forest model. Here the score presented for each feature group is summed to give the final value presented in the table. The results from this experiment seem to correlate with some of the observations which were made from the previous experiments, specifically that the vertex level features are more important than the global ones in the classification result. However, the results here differ from the earlier ones as the Random Forest seems to place less importance on certain features, specifically Eigenvector centrality. This is interesting as it suggests that the Random Forest is finding different features to be useful when compared with DTC's neural network.

Features Used	**Accuracy** (%)
All	99.99 ± 0.01
Vertex level only	99.92 ± 0.11
Global level only	69.34 ± 9.61
All but total degree	99.90 ± 0.20
All but two-hop away neighbours	99.90 ± 0.06
All but local clustering score	99.67 ± 0.30
All but average clustering of neighbourhood	99.47 ± 0.18
All but Eigenvector centrality	82.97 ± 5.48
All but Pagerank centrality	99.91 ± 0.09
All but centrality measures	75.57 ± 2.61
All but local neighbourhood measures	79.18 ± 5.19

Table 3.5: Measuring feature importance by removing various elements from the input to a trained model and measuring the changes in accuracy.

Feature	**Gini Importance**
Total Vertex	0.901
Total Global	0.098
Degree	0.247
Local clustering score	0.046
Two-hop away neighbours	0.125
Average clustering of neighbourhood	0.225
Pagerank centrality	0.108
Eigenvector centrality	0.098

Table 3.6: Measuring feature importance in a trained Random Forest model via gini importance.

3.8.5 Comparisons with Graph Kernels

Graph kernels are one of the most widely applied techniques for performing global graph level classification - however as we have previously discussed they often have issues with runtime, scaling to large graphs and requiring that graphs contain labels on the vertices or edges [136]. However it is still important to demonstrate that topological features are able to be at least as discriminative as graph kernels. For this comparison, we needed a graph kernel approach which does not require vertex or edge level features to be present, which unfortunately rules

out the majority of the popular approaches including the Weisfeiler-lehman family [209]. The two graph kernels which match these requirements are the Shortest Path [39] and Random Walk [216] Kernels. We attempted to run both approaches on our dataset (discussed in Section 3.6.3), but neither could run on the number and size of the graphs it contained.

We thus created a smaller dataset using the same five random generation methods which contained only 1000 examples of each class, with each graph containing 1000 vertices. This is interesting, as it allows us to explore how well the DTC approach performs when there is less training data and smaller graphs available. However, even with this smaller dataset, the Random Walk approach was unable to complete in over two weeks of runtime and it is excluded from these results. The Shortest Path Graph Kernel (SP-GK) we used was taken from the GraKel library [213], which was used to provide an optimised implementation in Python. The extracted Graph Kernels are then passed into an SVM to perform the final classification as is common in the literature [152].

Method	**Accuracy** (%)	**Recall**	**Precision**	**F1 Score**
DTC (Scaled)	$\mathbf{100. \pm 0.0}$	$\mathbf{1.0 \pm 0.0}$	$\mathbf{1.0 \pm 0.0}$	$\mathbf{1.0 \pm 0.0}$
SP-GK	99.96 ± 0.01	0.9996 ± 0.0001	0.9996 ± 0.0001	0.9996 ± 0.0001

Table 3.7: Comparing Deep Topology Classification versus the Shortest Path Graph Kernel with 10-fold cross-validation

The results from comparing DTC to SPGK on the smaller dataset are presented in Table 3.7. Again the results here are presented as the mean of 10-fold cross-validation. The table shows how on this reduced dataset, both approaches are able to predict the class of graphs exceptionally well, with DTC displaying complete ability to generalise to unseen examples across all test splits of the dataset. Additionally, this result demonstrates that the DTC approach does not need vast quantities of training data to perform well, with just 1000 examples of each class being used to generate these results. This is interesting as ANN based models are traditionally assumed to need many thousands of examples per class to perform well on unseen examples. This result seems to call this assumption into question when performing graph classification, perhaps even suggesting that less than 1000 examples per class could be used to train an accurate model.

3.9 Conclusion

This chapter has explored the Graph Fingerprint and detailed how it can be used for the tasks of graph comparison and global graph classification.

The Graph FingerPrint Comparison approach for assessing the similarity of two unlabelled graphs, based upon their macro and micro features, has been presented. The GFP-X fingerprint generation exploits Apache Spark and GraphX to extract powerful, neighbourhood based, features from a graph in parallel. When comparing two graphs, the GFP-C approach is shown to be sensitive to small variations in graph topology, graph size and function without the requirement of labelled datasets whilst also scaling nearly sub linearly with dataset size across a Spark cluster. Thus the GFP-C approach achieves all of the goals established for it in Section 3.4. The approach demonstrates promising results and the concept of a compact but accurate representation of a graph has numerous potential additional applications within machine learning.

Further, this chapter has presented a novel approach for global graph classification entitled Deep Topology Classification. The presented results show that the combination of extracting deep topological and global features from a graph and classifying these via a deep neural network is an effective approach to the problem of global graph classification. The approach is shown to have over 99% classification accuracy after k-fold cross validation across a multi-class and binary dataset. This compares very favourably with the current state-of-the-art approach which has an accuracy of just 88.4% for the multi-class and 68% for the binary datasets.

3.9.1 Current Limitations

Whilst the work presented in this chapter has been successful when compared with competing approaches, there are some limitations with the work which are worth considering:

Global graph only: Currently the work in this chapter has only considered applications that can be considered global graph tasks. There are however many important tasks in the field of graph mining which operate at the level of vertices and edges. The work presented thus far would not be applicable to such tasks.

Datasets used: Due to the highlighted issues around the lack of large, labelled and publicly available graph datasets, this chapter has made use of synthetically generated graphs as a proxy in many of the experiments. However, it remains to be seen if the high accuracy demonstrated by the approaches would be maintained if real-world data was to be used instead.

Hand-crafted features: The graph fingerprints comprise various topological features extracted from the graphs. Whilst they have proven to be effective across the two tasks and the datasets (both empirical and synthetic) used for evaluation, it is unknown if the same set of features would continue to work well across all domains and tasks. One clear trend in the machine learning literature is the move away from the use of hand-crafted features as input, and for models to automatically learn the best data representation for themselves [80].

Lack of interpretability: The DTC approach explored in this chapter uses a deep neural network to perform classification. However, concerns have been raised in the literature about how interpretable such models are [249]. Interpretability is covered in greater detail in later chapters, but briefly a model is said to be interpretable if the decisions made by it can be understood clearly [77]. The use of a deep network in this work could reduce the interpretability of the approach in the real-world. For example, limiting the ability of the model to 'explain' why a graph was classified as belonging to a certain domain.

3.9.2 Future Work

There is large scope for future research based upon the work presented in this chapter. Further work could be performed on incorporating other topological features into the graph fingerprint beyond those studied thus far, perhaps focusing on those which can exploit any auxiliary information available with the graphs. Additionally, steps could be taken to allow the DTC approach to be used on empirical datasets, which could be achieved via the use of data augmentation techniques to allow for model training upon limited amounts of input data.

Epilogue

This chapter has explored how best to represent a graph using only a set of topological features extracted from it. The features were shown to be useful for the tasks of graph comparison and

global classification, thereby achieving research objective 1. Additionally, the research presented in this chapter has, since its initial publication, been expanded by a number of works from other researchers which cite this work. For example, recent work has attempted to apply the concepts explored here to real-world datasets to show that empirical graphs can indeed be classified via their structural properties [197]. Other work has explored the use of a variation of the graph fingerprint vector as a way to increase the realism of synthetic graph generation methods by minimising the distance between generated and real graphs [169].

In the following chapter, focus will be shifting from exploring problems at the level of entire graphs to those at the constituent parts: vertices and edges. Additionally, study will turn to the emerging range of graph embedding techniques [84, 92, 124, 167], which learn graph representations automatically. Knowledge gained in this chapter about the ability of certain topological features to be able to represent a graph will be used to attempt to bring some level of interpretability to these new approaches.

Chapter 4

Exploring the Semantic Content of Unsupervised Graph Embeddings

Prologue

The work in Chapter 3 explored how a graph can be accurately represented by topological features extracted from them. The work in this chapter changes scale to focus upon learning representations at the level of vertices. In addition, focus will shift to explore recent methods, which unlike the hand-crafted and mathematically understood topological features explored thus far, attempt to automatically learn the best representations for a given problem. Such approaches are unsupervised machine learning models, commonly referred to as graph embeddings, which have recently emerged and demonstrated a more superior performance than traditional topological feature based approaches for a range of vertex centric tasks. These approaches attempt to learn a mapping from the vertices to a vector space, where certain key relationships present between vertices is maintained in the resulting vector space.

In order to investigate research objective 2 (see Section 1.3), this chapter will explore the possibility of bringing some level of interpretability to the new family of unsupervised graph

embedding models by investigating whether any known topological features are represented in the vector space. The experimental evidence presented in this chapter demonstrates that several of the known topological features, many of which were explored in Chapter 3, can be detected in the embedding space. This suggests that the type of topological structures being captured by the graph embedding techniques do approximate many of the same type of structural connectivity patterns used by human experts when representing graphs.

The work presented in this chapter has been published as the following works:

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 *IEEE International Conference on Big Data*, pages 2691–2700. IEEE, 2017

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. *Data Science and Engineering*, 4(3):269–289, 2019

4.1 Introduction

Representing the complex and inherent links and relationships between and within datasets in the form of a graph is a widely adopted practice across many scientific disciplines [170]. One reason for its popularity is that the structure or topology of the resulting graph can reveal important and unique insights into the data it represents. Recently, analysing and making predictions about graphs using machine learning has shown significant advances in a range of commonly performed tasks over traditional approaches [84]. Such tasks include predicting the formation of new edges within the graph and the classification of vertices [167]. However, graphs are inherently complex structures and do not naturally lend themselves as input into existing machine learning methods, most of which operate on vectors of real numbers.

Graph embeddings[1] are a family of machine learning models which learn latent representations for the vertices within a graph. The goal of all graph embedding techniques is broadly the same:

[1] In this work, focus is on vertex representation learning approaches.

to transform a complex graph, with no inherent representation in vector space, into a low-dimensional vector (often in the range of 50 to 300 dimensions) representation of the graph or its elements. More concretely, the objective of a graph embedding technique is to learn some function $f : V \rightarrow \mathbb{R}^d$ which is a mapping from the set of vertices V to a set of embeddings for the vertices, where d is the required dimensionality of the resulting embedding. This results in the mapping function f producing a matrix of dimensions $|V|$ by d, i.e. an embedding of size d for each vertex in the graph. It should be noted that this mapping is intended to capture the latent structure from a graph by mapping structurally similar vertices together in the embedding space. Many of the recent approaches are able to produce low-dimensional graph representations without the need for labelled datasets. These representations can then be used as input to secondary supervised models for downstream prediction tasks, including classification [186] or link prediction [87]. Thus, unsupervised graph embeddings are becoming a key area of research as they can be viewed as acting as a translation layer between the raw graph and some desired machine learning model.

However, to date, there has been little research undertaken into why graph embedding approaches have been so successful. They all aim to capture as much topological information as possible during the embedding process, but how this is achieved, or even exactly what structure is being captured, is currently unknown. In this work, focus is placed solely upon unsupervised graph embedding techniques as this work aims to explore what features the techniques learn from the topology alone, without the requirement for labels. This work attempts to provide insight into the graph embedding process itself, by exploring if the known and mathematically understood range of topological features [170] are being approximated in the embedding space. To achieve this, an investigation is performed to discover whether a mapping from the embedding space to a range of topological features is possible. The hypothesis of this work is that if such a mapping can be found, then the topological structure represented by that feature is thus approximately captured in the embedding space. Such a discovery could start to provide a way to interpret the graph embedding process, by experimentally demonstrating which topological structures are approximated to create the representations.

In summary, the work presented in this chapter is designed to tackle research objective 2 and the methodology employed uses a combination of supervised and unsupervised downstream models to predict topological features directly from the embeddings.

4.1.1 Chapter Contributions

The key contributions of this chapter are as follows:

- An investigation into whether unsupervised graph embeddings are learning something analogous to traditional vertex level graph features. If this is the case, is there a particular type of feature which is being approximated most commonly.
- Empirical evidence to show that several known topological features are demonstrated to be present in graph embeddings. This observation can be used to help bring interpretability to the graph embedding process by detailing which graph features are key in creating high quality representations.
- Detailed experimental evidence is presented, using five state-of-the-art unsupervised graph embedding approaches, across seven topological features and six empirical graph datasets to support these claims.

To aid in reproducibility of the results presented in this chapter, all of the associated code has been open-sourced and made available online. In addition, results are presented upon public benchmark datasets and key model parameters are reported. The code for extracting graph embeddings and performing experiments to measure topological features is available here - https://github.com/sbonner0/unsupervised-graph-embedding/

4.2 Previous Work

This section explores the prior research regarding graph embedding techniques and previous approaches measuring known features in embeddings. We first introduce the notion of graph embeddings, detail supervised and factorization based approaches, explore in detail state-of-the-art unsupervised approaches which will be used throughout the rest of this chapter and finally review past attempts to provide interpretability to embedding approaches.

4.2.1 Introduction to Graph Embeddings

The ability to automatically learn some descriptive numerical based representation for a given graph is an attractive goal, and could provide a timely solution to some common problems within the field of graph mining. Traditional approaches have relied upon extracting features – such as various measures of a vertex's centrality [178] – capturing the required information about a graph's topology, which could then be used in some down-stream prediction task [25, 152]. However, such a feature-extraction based approach relies solely upon the hand-crafted features being a good representation of the target graph. Often a user must use extensive domain knowledge to select the correct features for a given task, with a change in task often requiring the selection of new features [152].

Graph embedding models are a collection of machine learning techniques which attempt to learn key features from a graph's topology automatically, in either a supervised or un-supervised manner, removing the often cumbersome task of end users manually selecting representative graph features [186]. This manual process, known as feature selection [90] in the machine learning literature, has clear disadvantages as certain features may only be useful for a certain task. It could even negatively affect model performance if utilised in a task for which they are not well suited. Arguably, many of the recent exciting advances seen in the field of Deep Learning have been driven by the removal of this feature selection process [87], instead allowing models to learn the best data representations themselves [80]. For a selection of recent review papers covering the complete family of graph embedding techniques, readers are referred to [43, 59, 92, 247]. The work presented in this chapter focuses on neural network based approaches for graph embedding, as these have demonstrated superior performance compared with traditional approaches [84].

Supervised Approaches

Within the field of machine learning, approaches which are supervised are perhaps the most studied and understood [80]. 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 social network.

Perhaps the largest area of supervised graph embeddings is that of Graph Convolutional Neural Networks (GCNs) [42], both spectral [64, 128] and spatial [174] approaches. Such

approaches pass a sliding window filter over a graph, in a manner analogous with Convolutional Neural Networks from the computer vision field [80], but with the neighbourhood of a vertex replacing the sliding window. Current GCN approaches are supervised and thus require labels upon the vertices. This requirement has two significant disadvantages: Firstly, it limits the available graph data which can be used due to the requirement for labelled vertices. Secondly, it means that the resulting embeddings are specialised for one specific task and cannot be generalised for a different problem without costly retraining of the model for the new task.

Factorization Approaches

Before the recent interest in learning graph embeddings via the use of neural networks, a variety of other approaches were explored. Often these approaches took the form of adjacency matrix factorization, in a similar vein to classical dimensionality reduction techniques such as Principal Component Analysis (PCA) [92] [232]. Such approaches first calculate the pairwise similarity between the vertices of a graph, then find a mapping to a lower dimensional space, such that the relationships observed in the higher dimensions are preserved. An early example of such an approach is that of the Laplican eigenmaps, which attempt to directly factorize the Laplacian matrix of a given graph [23]. Other approaches, often using the adjacency matrix, define the relationship in low dimension space between two vertices in the graph as being determined by the dot product of their corresponding embeddings. Such approaches include Graph Factorization [4], GraGrep [44] and HOPE [177]. These dimensionality reduction based approaches are often quadratic in complexity [247] and the predictive performance of the embeddings has largely been superseded by the recent neural network based methods [84].

4.2.2 Unsupervised Stochastic Embeddings

DeepWalk [186] and Node2Vec [87] are the two main approaches for random walk based embedding. Both of these approaches borrow key ideas from a technique entitled Word2Vec [165] designed to embed words, taken from a sentence, into vector space. The Word2Vec model is able to learn an embedding for a word by using surrounding words within a sentence as targets for a single hidden layer neural network model to predict. Due to the nature of this technique, words which frequently co-occur together in sentences will have positions which are close within the embedding space. The approach of using a target word to predict neighbouring words is entitled Skip-Gram and has been shown to be very effective for language modelling tasks [164].

DeepWalk

The key insight of DeepWalk is to use random walks upon the graph, starting from each vertex, as the direct replacement for the sentences required by Word2Vec. A random walk can be defined as a traversal of the graph rooted at a vertex $v_t \in V$, where the next step in the walk is chosen uniformly at random from the vertices incident upon v_t [16], these walks are recorded as $w_0^t, ..., w_n^t$ (where t is the walk starting from v_t of length n, and $w_i^t \in V$), i.e. a sequence of the vertices visited along the random walk starting from $v_t = w_0^t$. DeepWalk is able to learn unsupervised representations of vertices by maximising the average log probability P over the set of vertices V:

$$\frac{1}{|V|} \sum_{t=1}^{|V|} \sum_{i=0}^{n} \sum_{-c \leq j \leq c, j \neq 0} \log P(w_{i+j}^t | w_i^t), \tag{4.1}$$

where c is the size of the training context of vertex w_n^t.[2]

The basic form of Skip-Gram used by DeepWalk defines the conditional probability $P(w_{i+j}^t | w_i^t)$ of observing a nearby vertex w_{i+j}^t, given the vertex w_i^t from the random walk t, can be defined via the softmax function over the dot-product between their features [186]:

$$P(w_{i+j}^t | w_i^t) = \frac{\exp\left(\mathrm{W}_{w_i^t}^{\intercal} \mathrm{W}'_{w_{i+j}^t}\right)}{\sum_{t=1}^{|V|} \exp\left(\mathrm{W}_{w_i^t}^{\intercal} \mathrm{W}'_{v_t}\right)}, \tag{4.2}$$

where $\mathbf{W}$ and $\mathbf{W}'$ are the hidden layer and output layer weights of the Skip-Gram neural network respectively.

Node2Vec

Whilst DeepWalk uses a uniform random transition probability to move from a vertex to one of its neighbours, Node2Vec biases the random walks by controlling which vertex will be visited next. This biasing introduces two user-controllable parameters which dictate how far from, or close to, the source vertex the walk progresses. This is done to capture either the vertex's role in its local neighbourhood (homophily), or alternatively its role in the global graph structure (structural equivalence) [87]. Changing the random walk means that Node2Vec has a higher accuracy over DeepWalk for a selection of vertex classification problems [87].

[2] Note if $i + j < 0$ then we skip these from the sum as we are past the start of the current walk.

4.2.3 Unsupervised Hyperbolic Embeddings

Recently, a new family of graph embedding approaches has been introduced which embed vertices into hyperbolic, rather than Euclidean space [48, 173]. Hyperbolic space has long been used to analyse graphs which exhibit high levels of hierarchical or community structure [168], but it also has properties which could make it an interesting space for embeddings [48]. Hyperbolic space can be considered "larger" than Euclidean with the same number of dimensions, as the space is curved, its total area grows exponentially with the radius [48]. For graph embeddings, this key property means that one effectively has a much larger range of possible points into which the vertices can be embedded. This property allows for closely correlated vertices to be embedded close together, whilst also maintaining more distance between disparate vertices, resulting in an embedding which has the potential to capture more of the latent community structure of a graph.

The hyperbolic approach we focus on was introduced by Chamberlain [48], and uses the Poincaré Disk model of 2D hyperbolic space [69]. This was chosen as it uses the same underlying skip-gram neural network so was directly comparable with the other models. In their model, the authors use polar coordinates $\mathbf{x} = (r, \theta)$, where $r \in [0, 1]$ and $\theta \in [0, 2\pi]$ to describe a point in space for each vertex v in the Poincaré Disk, which allows for the technique to be significantly simplified as only two values are required for a representation [48]. Similar to DeepWalk, an inner-product is used to define the similarity between two points within the space. The inner-product of two vectors in a Poincaré Disk can be defined as follows [48]:

$$< \mathbf{x}, \mathbf{y} >= ||x||||y|| \cos(\theta_x - \theta_y), \tag{4.3}$$

$$= 4 \text{ arctanh } r_x \text{ arctanh } r_y \cos(\theta_x - \theta_y), \tag{4.4}$$

where $\mathbf{x} = (r_x, \theta_x)$ and $\mathbf{y} = (r_y, \theta_y)$ are the two input vectors representing two vertices and *arctanh* is the inverse hyperbolic tangent function [48].

To create their hyperbolic graph embedding, the authors use the softmax function of Equation 4.2, common with DeepWalk and Node2Vec, but importantly replacing the Euclidean inner-products with the hyperbolic inner-products of Equation 4.3. Aside from this, hyperbolic approaches share many similarities with the stochastic approaches with regards to their input data and training procedure. For example, the hyperbolic approaches are still trained upon pairs of vertex IDs, taken from sequences of vertices generated via random walks on graphs.

4.2.4 Unsupervised Auto-Encoder Based Approaches

There is an alternative set of approaches for graph embeddings which do not rely upon random walks. Instead of adapting a technique based upon capturing the meaning of language, such models are designed specifically for creating graph embeddings using Deep Learning [80] – deep auto-encoders [99]. Auto-encoders are an un-supervised neural network, where the goal is to accurately reconstruct the input data through explicit encoder and decoder stages [203]. Two such approaches are Structural Deep Network Embedding (SDNE) [229] and Deep Neural Networks for Learning Graph Representations (DNGR) [45].

The authors of these approaches argue that a deep neural network, versus the shallow Skip-Gram model used by both DeepWalk and Node2Vec, is much more capable of capturing the complex structure of graphs. In addition the authors argue that for a successful embedding, it must capture both the first and second order proximity of vertices. Here the first order proximity measures the similarity of the vertices which are directly incident upon one another, whereas the second order proximity measures the similarly of vertices neighbourhoods. To capture both of these elements SDNE has a dual objective loss function for the model to optimise. The input data to SDNE is the adjacency matrix $\mathbf{A}$, where each row a represents the neighbourhood of a vertex.

The objective function for SDNE comprises two distinct terms, the first term captures the second order proximity of the vertices neighbourhood, whilst the second captures the first order proximity of the vertices by iterating over the set of edges E:

$$L_{SDNE} = \sum_{i=1}^{|V|} ||(\mathrm{q}_i' - \mathrm{q}_i) \odot b_i||_2^2 + \alpha \sum_{u,v=1}^{|E|} \mathrm{A}_{u,v} ||(\mathrm{W}_u^{(k)} - \mathrm{W}_v^{(k)})||_2^2, \tag{4.5}$$

where q_i and q_i' are the input and reconstructed representation of the input, $\odot$ is the element wise Hadamard product and b_i is a scaling factor to penalise the technique if it predicts zero too frequently, $\mathbf{W}^{(k)}$ is the weights of the k^{th} layer in the auto-encoder technique and α is a user-controllable parameter defining the importance of the second term in the final loss score [229].

To initialise the weights of the deep auto-encoder used for this approach, an additional neural network must be trained to find a good starting region for the parameters. This pre-training

neural network is called a Deep Belief Network, and is widely used within the literature to form the initialisation step of deeper models [71]. However, this pre-training step is not required by either the stochastic or hyperbolic approaches as random initialisation is used for the weights, and adds significant complexity.

In comparison with SDNE, instead of relying solely upon the raw adjacency matrix as input, DNGR creates a new denser representation to be passed to an auto-encoder [45]. The authors have the model reconstruct the pointwise mutual information matrix (PPMI) of the input graph, which captures vertex co-occurrence information in a sequence created via a random surfer model. Additionally, instead of passing this to a traditional auto-encoder, a stacked de-noising auto-encoder is used with the goal of creating a more robust vertex representation. This stacked de-noising auto-encoder adds a small quantity of noise to the input data, which the model must learn to disregard during the training process.

4.2.5 Observing Features Preserved in Embeddings

Graph Embeddings Features

To date, there has been little research performed exploring a theoretical basis as to why graph embeddings are able to demonstrate such good performance in graph analytic tasks, or to bring interpretability to the graph embeddings process. Goyal and Ferrar [84] presented an experimental review paper on a selection of graph embedding techniques. The authors use a range of tasks including vertex classification, link prediction and visualization to measure the quality of the embeddings. However the authors do not explore the use of topological structure as a way to provide interpretability of how the graph embedding process functions. In addition, the authors do not consider embeddings taken from promising unsupervised techniques – such as the family of hyperbolic approaches, nor do they explore performance across imbalanced classes during the classification.

Recent work has speculated on the use of a graph's topological features as a way to improve the quality of vertex embeddings by incorporating them into a supervised GCN based model [93]. They show how aggregating a vertex feature – even one as simple as its degree – can improve the performance of their model. Further, they present theoretical analysis to validate that their approach is able to learn the number of triangles a vertex is part of, arguing that

this demonstrates the model is able to learn topological structure. We take inspiration from this work, but consider unsupervised approaches as well as exploring whether richer and more complicated topological features are being captured in the embedding process. In a similar vein, an approach for generating supervised graph embeddings using heat-kernel based methods is validated by visualizing if a selection of topological features are present in a two-dimensional projection of the embedding space [150].

Research has investigated the use of a graph's topological features as a way of validating the accuracy of a neural network based graph generative model [156]. With the presented model, the authors aim to generate entirely new graph datasets which mimic the topological structure of a set of target graphs – a common task within the graph mining community [7]. To validate the quality of their model, they investigate if a new graph created from their generative procedure has a similar set of topological features to the original graph.

Perhaps most closely related to our present research is work exploring the use of random walk based graph embeddings as an approximation for more complex vertex level centrality measures on social network graphs [204]. The authors argue that graph embeddings could be used as a replacement for centrality measures as they potentially have a lower computational complexity. The work explores the use of linear regression to try to directly predict four centrality measures from the vertices of three graph datasets, with limited success [204].

Our own work differs significantly as we attempt to provide insight into what exactly graph embeddings are learning with a view to allow for greater interpretability, explore a wider range of embeddings approaches, use datasets from a wider range of domains, explore more topological features, use classification rather than regression as the basis for the analysis and address the inherent unbalanced nature of most graph datasets.

Feature Learning in Other Domains

A large number of the successful unsupervised graph embedding approaches have adapted models originally designed for language modelling [87, 186]. Some recent research has investigated how best to evaluate a variety of unsupervised approaches for embedding words into vectors [205]. They choose a variety of Natural Language Processing (NLP) tasks, which capture some known and understood aspects of the structure of language, and investigate how well the chosen

embedding models perform for these tasks. They conclude that no single word embedding model performs the best across all the tasks they investigated, suggesting there is not a single optimal vector representation for a word. What features are used to help word embeddings achieve compositionality – constructing the meaning of an entire sentence from the component words, has also been explored [153]. Further research has investigated the use of word embeddings to create representations for the entire sentence using word features [57]. The work suggests that word features learned by the embeddings for natural language inference can be transferred to other tasks in NLP, although fails to provide any real interpretability to them.

Outside of NLP, there has been work in the field of Computer Vision (CV) investigating what known features, already commonly used for image representation, are captured by deep convolutional neural network. These features can then be potentially used to provide interpretability to the models. For example, it has been shown that convolutional networks, when trained for image classification, often detect the presence of edges in the images [246]. The same work also shows how the complexity of the detected edges increases as the depth of the network increases.

In this present work, we take inspiration from these approaches and attempt to provide insight and a potential theoretical basis for the use of graph embeddings by exploring which known graph features can be reconstructed from the embedding space.

4.3 Semantic Content of Graph Embeddings

Despite extensive prior work in unsupervised graph embedding highlighting how they perform well for the tasks for which they were proposed (such as vertex classification and link prediction [84]), there has been little work exploring why these approaches are successful. This could allow for an increased level of interpretability to graph embeddings. The approach presented draws inspiration from recent work in Computer Vision and Natural Language Processing which examine if traditional features (the edges detected in images for example) are captured by deep models.

Topological features are one known and mathematically understood way to accurately identify graphs and vertices [152] (Also see the work detailed in Chapter 3). We hypothesise that if graph embeddings are shown to be learning approximations of existing features, this could begin to provide a theoretical basis for the interpretability of graph embeddings. This would suggest that

graph embeddings are automatically learning detailed and known graph structures in order to create the representations. This could explain how they have been so successful in a variety of graph mining tasks. Effectively the graph embedding techniques would be acting as an automated way of learning the most representative topological feature(s) for a given objective function.

If graph embeddings are shown to be learning topological features, then other interesting research questions arise. For example, do competing embedding approaches learn different topological structures, do different graph datasets each require different features to be approximated in order to create a good representation, what is the structural complexity of the features approximated by the embeddings or even are the embeddings capable of approximating multiple features simultaneously? These questions are explored more in the evaluation section of this chapter presented in Section 4.5.

In order to explore these questions, we attempt to predict a selection of topological features directly from graph embeddings computed from a range of state-of-the-art approaches across a series of empirical datasets. We suggest that if a second mapping function $f : \mathbb{R}^d \rightarrow \Lambda$ can be found which accurately maps the embedding space to a given topological feature Λ, then this is strong evidence that something approximating the structural information represented by Λ is indeed present in the embedding space. Here the mapping function could take the form of a linear regression, but for this work we investigate a range of classification algorithms – this is explored more in Section 4.3.3. We assess a range of known topological features, from simple to complex, to gain a better understanding of the expressive capabilities of the embedding techniques.

4.3.1 Predicting Topological Features

Numerous topological features have been identified in the literature, measuring various aspects of a graph's topology, at the vertex, edge and graph level [152]. As we are focusing our work here upon methods for creating vertex embedding, we will focus on features which are measured at the *vertex level* of a given graph. We have selected a range of vertex level features from the graph mining literature, which capture information about a vertex's local and global role within a graph [87]. These are similar features to those used to classify graph structure in the Chapter 3. This selection of features range from ones which are simple to compute from vertices directly adjacent to each other, to more complex features which can require information from

many hops[3] further along within the graph. This will allow us to explore whether embedding models learn complex topological features, or are they able to learn good representations of only simple features. The topological vertex level features we are predicting are detailed below, listed approximately by the complexity of structure being captured:

- **Total Degree** - The sum of both the in and out degree for a vertex v, denoted as k_v.
- **Degree Centrality** - A simple centrality score which provides a normalised measure of vertex connectivity [115]. The equation for computing this value can be found in Section 2.3.4.
- **Number Of Triangles** - The number of triangles containing the vertex v, detailed more in Section 2.3.3.
- **Local Clustering Score** - The local clustering score for vertex v represents the probability of two neighbours of v also being neighbours of each other [231]. It is detailed more in Section 2.3.
- **Eigenvector Centrality Value** - The Eigenvector centrality is used to calculate the importance of each vertex within a graph by measuring neighbour importance [27]. More details on this as well as the equation for computing it can be found in Section 2.3.4.
- **PageRank Centrality Value** - The PageRank centrality method was originally developed by Google, however it is now commonly used to measure the local influence of a vertex within a graph [94, 178]. The equation for computing this value can be found in Section 2.3.4.
- **Betweenness Centrality** - The Betweenness centrality of a vertex depends upon the frequency with which it acts as a bridge between two additional vertices [94]. The equation for computing this value can be found in Section 2.3.4.

4.3.2 Graph Feature Distribution

Many empirical graphs, especially those representing social, hyper-link and citation networks, have been shown to have an approximately power-law distribution of degree values, where most

[3] Hops represent the length of the sequences of vertices that must traversed to get from vertices i to j.

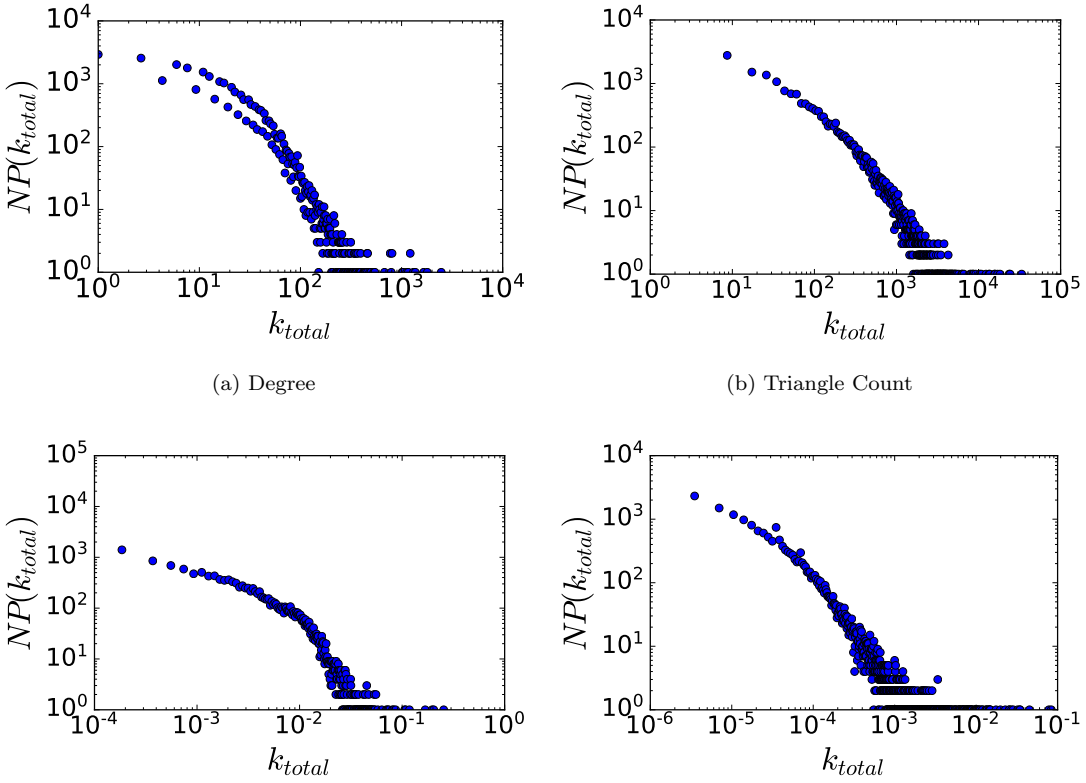
vertices only have a small number of edges and there are a few super-connected hubs [73][4]. This heavy-tailed distribution profile poses a challenge for machine learning models, as it means the features we are trying to predict are extremely unbalanced, with a heavy skew towards the lower range of features. Imbalanced class distribution creates difficulties for machine learning models, as there are fewer examples of the minority classes for the model to learn, which can often lead to poor predictive performance on these classes [80]. It has been shown that the distribution of other topological features can also follow a heavy-tailed distribution in many graphs [7]. To demonstrate this phenomenon, Figure 4.1 shows the distribution of a range of topological feature values for the cit-HepTh dataset [146]. The Figure shows that all the topological feature values tested largely follow an approximately heavy-tailed distribution. This fact has the potential to make predicting the value of a certain topological feature challenging, as the datasets will not be balanced and any model attempting to find the mapping $f : \mathbb{R}^d \rightarrow \Lambda$, will be prone to over-fitting to the majority classes. Our approach for tackling this issue is outlined in the following section.

4.3.3 Methodology

Unlike previous studies [204] we employ classification and visualization, instead of regression, as a way to explore the embedding space. We chose these approaches as predicting topological features directly via the use of regression has proven challenging in prior work [204], owing largely to the imbalance problem explored in Section 4.3.2. With such an imbalanced dataset, using a classification based approach is often advantageous [176] as techniques exist to over-sample minority examples. However, the features we are attempting to predict are continuous, so must go through some transformation stage before classification can be performed. For our transformation stage, we follow a procedure similar to that introduced by Oord *et al.*[176]. We bin the real-valued features into a series of classes via the use of a histogram, where the bin in which a particular feature is placed becomes its class label. One can consider each of these newly created classes as representing a range of possible values for a given feature. As an example, we could transform a vertex's continuous PageRank score [178] into a series of discrete classes via the use of a histogram with a bin size of three, where each of the newly created classes represented a low, medium or high PageRank score.

Although this binning process helps with the feature imbalance, it still produces a skew in the number of features assigned to each class. To further address this issue, we take the logarithm of

[4] That degree distributions in many domains are always truly power-law has become an area of disagreement within the community, as it is beyond the scope of this thesis, interested readers are refereed to [56, 215].



(a) Degree

(b) Triangle Count

(c) Eigenvector Cent

(d) Betweenness Cent

Figure 4.1: Distribution of topological feature values from the cit-HepTh dataset in log scale: (a) total vertex degree distribution, (b) distribution complete triangles for each vertex, (c) Eigenvector centrality distribution and (d) Betweenness centrality score distribution.

each feature value before it is passed to the binning function. This will mean that features within the same order of magnitude will be assigned the same class, for example vertices with degrees in the range of 0 to 10^1 would be assigned into one class, whilst degree values between 10^2 to 10^3 would be assigned to another class. This was performed as it dramatically improved the balance of the datasets, and as we are only attempting to discover if something approximating the topological features is present in the embedding space, we found that predicting the order of magnitude to be sufficient.

In order to allow for a good distribution of feature values in the datasets we are using, in our experiments we utilise a bin size of six for the histogram function, meaning that six discrete classes were created for each of the features. This value was chosen empirically from our datasets as it fully covered the numerical range of the topological features we measured. For example, we found that the centrality values in our datasets fell within a range of six orders of magnitude, which is what we used to set the number of bins. It should be noted that this value would need to be tuned depending upon the datasets and features being used.

In addition to the use of classification, we explore an additional method to bring interpretability to graph embeddings, that being a visualisation technique entitled t-Distributed Stochastic Neighbour Embedding (t-SNE) [157]. This technique allows relatively high dimensional data, such as graph embeddings, to be projected into a low dimensional space in such a way as to preserve the inter-spatial relationship between points that were present in the original space. Thus, we utilise t-SNE to project the embeddings down to two dimensions so they can be easily visualised. This process is performed without the need for any classification to be trained upon the embeddings, removing the issues associated with classifying unbalanced datasets. Once the projection has been performed, we can colour each point in accordance with its feature value, be that one that has been transformed via the binning process, or even the raw value itself.

4.3.4 Embedding Approaches Compared

In this chapter, five state-of-the-art unsupervised graph embedding approaches are evaluated as a way of exploring what semantic content is extracted from a graph to create the embeddings. The approaches are as follows: DeepWalk, Poincaré Disk, Structural Deep Network Embedding and Node2Vec [5], which are detailed in Table 4.1 and present the approach names, the year

[5] Please note, we explore two variations of Node2Vec, bringing the total number of approaches to five

and venue of publication, the primary technique used and the computational complexity of the embedding process. These approaches were chosen from the literature as they represent a good cross-section of the current competing methodologies and all either exploit a different method of sampling the graph, use different geometries for the embedding space or use competing methods of comparing vertices. This selection of approaches will allow exploration of interesting research questions. Such questions include whether any differences between the approaches can be explained by what graph structures they learn and do methods which promote local exploration around the target vertex only learn local structural information? To explore this second question in more detail, we created two versions of Node2Vec: Node2Vec-Structural, which biases the random walks used to create training pairs for the model to explore vertices further away from the target vertex and Node2Vec-Homophily, which biases the random walks to stay closer to the target vertex.

Approach	**Year**	**Type**	**Published**	**Complexity**
DeepWalk	2014	stochastic	KDD [186]	$O(\|V\|)$
Node2Vec	2016	stochastic	KDD [87]	$O(\|V\|)$
SDNE	2016	auto-encoder	KDD [229]	$O(\|V\|\|E\|)$
Poincaré Disk	2017	hyperbolic	MLG [48]	$O(\|V\|)$

Table 4.1: The Graph Embedding approaches used for experimentation.

4.4 Experimental Setup and Classification Algorithm Selection

4.4.1 Metrics

Presented Results

All the reported results are the mean of five replicated experiment runs along with confidence intervals. For the classification results, all the accuracy scores presented are the mean accuracy after k-fold cross validation – considered the gold standard for model testing [11]. For k-fold cross validation, the original dataset is partitioned into k equally sized partitions. $k-1$ partitions are used to train the model, with the remaining partition being used for testing. The process is repeated k times using a unique partition for each repetition and a mean taken to produce the final result.

Precision Metrics

For reporting the results of the vertex feature classification tasks, we report the macro-f1 and micro-f1 scores with varying percentages of labelled data available at training time. This is a similar setup to previous works [84] [87].

The micro-f1 score calculates the f1-score for the dataset globally by counting the total number of true positives (TP), false positives (FP) and false negatives (FN) across a labelled dataset L. Using the notation from [84], micro-f1 is defined as:

$$microf1 = 2 \cdot \frac{P_{micro} \cdot R_{micro}}{P_{micro} + R_{micro}}, \tag{4.6}$$

where:

$$\text{Micro Precision}(P_{micro}) = \frac{\sum_{l=1}^{|L|} TP(l)}{\sum_{l=1}^{|L|} TP(l) + FP(l)},$$

$$\text{Micro Recall}(R_{micro}) = \frac{\sum_{l=1}^{|L|} TP(l)}{\sum_{l=1}^{|L|} TP(l) + FN(l)},$$

and $TP(l)$ denotes the number of true positives the model predicts for a given label l, $FP(l)$ denotes the number of false positives and $FN(l)$ the number of false negatives.

The macro-f1 score, when performing multi-label classification, is defined as the average micro-f1 score over the whole set of labels L:

$$macrof1 = \frac{1}{|L|} \sum_{l \in L} f1(l), \tag{4.7}$$

where $f1(l)$ is the $f1$-score for the given label l.

4.4.2 Experimental Setup

Implementation Details

The graph embedding approaches used for experimentation were reimplemented in Tensorflow [1], as the author-provided versions were not all available using the same framework. We also

ensure the same Tensorflow-based optimisations were used across all the approaches wherever possible [210]. Neural Networks contain many hyper-parameters which a user can control to improve the performance, both of the predictive accuracy and the runtime, of a given dataset. This process can be extremely time consuming and often requires users to perform a grid search over a range of possible hyper-parameter values to find a combination which performs best [80]. For choosing appropriate values for the required hyper-parameters for the approaches, we used the default hyper-parameters as proposed by the authors in their original papers, keeping them constant across all datasets. The key hyper-parameters used for each approach are detailed in Table 4.2 which displays the otimiser choice, the learning rate used and other parameter choices specific to an approach. We have open sourced our implementations of these approaches and made them available online[6].

Experimental Environment

Experimentation was performed on a compute system with 2 NVIDIA Tesla K40c's, 2.3GHz Intel Xeon E5-2650 v3, 64GB RAM and the following software stack: Ubuntu Server 16.04 LTS, CUDA 9.0, CuDNN v7, TensorFlow 1.5, scikit-learn 0.19.0, Python 3.5 and NetworkX 2.0.

Experimental Datasets

The empirical datasets used for evaluation were taken from the Stanford Network Analysis Project (SNAP) data repository [146] and the Network Repository [196] and are detailed in Table 4.3, showing the dataset name, number of vertices and edges and the domain from which the data originates. This domain label is taken from the listings of the graphs domain provided by SNAP [146] and Network Repository [196].

[6] https://github.com/sbonner0/unsupervised-graph-embedding/

Approach	Optimiser	Learning Rate	Specific Parameters
SNDE	RMSProp	0.01	α=500, b=10, epochs=500
Node2Vec-S	SGD	0.1	p=0.5, q=2, epochs=15
Node2Vec-H	SGD	0.1	p=1.0, q=0.5, epochs=15
DeepWalk	SGD	0.1	epochs=15
Poincaré Disk (PD)	SGD	0.1	p=0.5, q=2, epochs=15

Table 4.2: Key hyper-parameters used when training the various graph embeddings models.

Dataset	$\|V\|$	$\|E\|$	Domain	Source
fly-drosophila-medulla	1,800	33,500	Biological	[196]
cit-HepTh	27,770	352,807	Citation	[146]
email-Eu-core	1,005	25,571	Communication	[146]
inf-openflights	2,900	30,500	Infrastructure	[196]
soc-sign-bitcoinotc	5,881	35,592	Blockchain	[146]
ego-Facebook	4,039	88,234	Social	[146]

Table 4.3: Empirical graph datasets used to assess the topological features approximated by unsupervised graph embedding techniques.

4.5 Results

This section presents both the supervised and unsupervised results for predicting topological features from graph embeddings.

4.5.1 Classification Algorithm Selection

As highlighted throughout this chapter, we are focusing our research on unsupervised graph embedding approaches. In order to be able to use the embeddings for a secondary task, they must be classified using a supervised classification model. Traditionally in the embedding literature, a simple Logistic Regression is used in any classification task [165, 186], with seemingly little work exploring the use of more sophisticated models to perform the classification.

In this section we explore the effectiveness of five different models at performing the classification of the different embedding approaches - Logistic Regression (LR), Support Vector Machine (SVM) (Linear Kernel), SVM (RBF Kernel), a single hidden layer Neural Network and finally a second more complex Neural Network with two hidden layers and a larger number of hidden units. All the classifiers utilised in this section were taken from the Scikit-Learn Python package [185]. Additionally, given that our datasets do not have an equal distribution among the classes, we also explore the effectiveness of weighting the loss function used by the model inversely proportional to the frequency of the class [118]. This use of a weighted loss function, although common in other areas of machine learning, has not hitherto been explored in regards to graph embeddings.

For the results in this section, we present the mean Macro and Micro F1 scores, introduced in Section 4.4.1, after 5-fold cross validation. To assess the performance of the classifiers against

Feature	Classifier	F1-Micro	F1-Macro	Uniform	Strat	Freq
DG	LR	0.336(±0.015)	0.190(±0.012)	+65.09%	+33.85%	+12.07%
	SVM(Lin)	**0.339(±0.017)**	0.164(±0.013)	**+66.57%**	**+35.03%**	**+13.07%**
	SVM(RBF)	0.336(±0.021)	0.158(±0.013)	+65.09%	+33.84%	+12.07%
	NN	0.329(±0.013)	**0.200(±0.018)**	+61.65%	+31.05%	+9.73%
	NN-2	0.326(±0.016)	0.192(±0.019)	+60.18%	+29.85%	+8.73%
TC	LR	0.340(±0.011)	0.154(±0.014)	+109.34%	+37.19%	+12.38%
	SVM(Lin)	**0.344(±0.015)**	0.139(±0.006)	**+111.8%**	**+38.8%**	**+13.7%**
	SVM(RBF)	0.335(±0.018)	0.130(±0.010)	+106.26%	+35.17%	+10.73%
	NN	0.331(±0.019)	0.157(±0.013)	+103.8%	+33.56%	+9.4%
	NN-2	0.326(±0.017)	**0.163(±0.015)**	+100.72%	+31.54%	+7.75%
EC	LR	0.590(±0.013)	0.474(±0.010)	+195.66%	+144.16%	+92.18%
	SVM(Lin)	0.591(±0.012)	0.480(±0.011)	+196.16%	+144.58%	+92.51%
	SVM(RBF)	0.552(±0.012)	0.446(±0.011)	+176.62%	+128.44%	+79.8%
	NN	0.629(±0.012)	0.512(±0.017)	+215.2%	+160.3%	+104.89%
	NN-2	**0.630(±0.019)**	**0.513(±0.021)**	**+215.7%**	**+160.72%**	**+105.21%**

Table 4.4: Degree (DG), Triangle Count (TC) and Eigenvector Centrality (EC) classification results for DeepWalk embeddings on the ego-Facebook dataset. Results for Micro and Macro-F1 scores are the mean after 5-fold cross validation, with standard deviations. Lift over Uniform, Stratified and Frequency predictors are presented as percentages.

the imbalance present in the datasets, we also display the performance lift in mean test set accuracy over three rule-based prediction methods to act as baselines. These methods are Uniform Prediction (where the classification of each item in the test is chosen uniformly at random from the possible classes), Stratified Prediction (where the classification follows the distribution of classes in the training set) and Frequent Class Prediction (where the classification is determined by the most frequent class in the training set). A positive lift across all metrics strongly suggests that a mapping from the embedding space to the topological features is being learned, as the classification algorithm is overcoming the biased distributions of classes in the dataset.

We performed this experiment for all combinations of datasets, embedding approaches and features, but due to the large quantity of results, we present only a subset here. Specifically we present the results for ego-Facebook dataset, using embeddings generated by DeepWalk and SDNE and classifying Degree, Triangle Count and Eigenvector Centrality. It should be noted that the patterns displayed here are representative of ones seen across all datasets.

Table 4.4 highlights the performance of the potential classifiers, when using the DeepWalk

embeddings taken from the ego-Facebook dataset. Results show that the choice of supervised classifier can have a large impact on the overall classification score. It can also be seen that the traditional choice of logistic regression does not produce the best results, with the neural network and SVM classifier often giving the best scores but no single classifier is best overall, suggesting that one needs to be chosen carefully for a given task.

Feature	Classifier	F1-Micro	F1-Macro	Uniform	Strat	Freq
DG	LR	0.284(±0.013)	0.177(±0.008)	+53.15%	+21.0%	−5.28%
	SVM(Lin)	**0.295**(±**0.020**)	0.167(±0.012)	**+59.08%**	+25.69%	**-1.61%**
	SVM(RBF)	0.289(±0.017)	0.142(±0.006)	+55.85%	**+23.13%**	−3.61%
	NN	0.253(±0.012)	0.187(±0.012)	+36.43%	+7.79%	−15.62%
	NN-2	0.247(±0.018)	**0.193**(±**0.019**)	+33.2%	+5.24%	−17.62%
TC	LR	0.284(±0.015)	0.138(±0.011)	+99.15%	+18.87%	−6.13%
	SVM(Lin)	0.296(±0.016)	0.125(±0.008)	+107.56%	+23.89%	−2.16%
	SVM(RBF)	**0.300**(±**0.018**)	0.124(±0.006)	+**110.37**%	**+25.57%**	**-0.84%**
	NN	0.264(±0.020)	0.161(±0.018)	+85.12%	+10.5%	−12.74%
	NN-2	0.247(±0.018)	**0.162**(±**0.016**)	+73.2%	+3.38%	−18.36%
EC	LR	0.297(±0.008)	0.166(±0.004)	+70.4%	+12.85%	−3.26%
	SVM(Lin)	**0.316**(±**0.010**)	0.156(±0.006)	+**81.3**%	+**20.07**%	+**2.93**%
	SVM(RBF)	0.309(±0.017)	0.149(±0.008)	+77.28%	+17.41%	+0.65%
	NN	0.286(±0.013)	0.198(±0.018)	+64.08%	+8.67%	−6.84%
	NN-2	0.272(±0.018)	**0.201**(±**0.014**)	+56.05%	+3.35%	−11.4%

Table 4.5: Degree (DG), Triangle Count (TC) and Eigenvector Centrality (EC) classification results for SDNE embeddings on the ego-Facebook dataset. Results for Micro and Macro-F1 scores are the mean after 5-fold cross validation, with standard deviations. Lift over Uniform, Stratified and Frequency predictors are presented as percentages.

Table 4.5 highlights the results for the potential classifiers, when using the SDNE embeddings taken from the ego-Facebook dataset. Again, the variation in classification score across the set of tested classification metrics is quite substantial, with the linear SVM and neural network approaches having perhaps a small margin of improvement over the others. It is interesting to note that the logistic regression frequently used in the literature, never produces the highest score in any metric. It can also be seen that, when compared with the DeepWalk results in Table 4.4, SDNE does less well at predicting all topological features which, although not the explicit purpose of this section, is interesting to note.

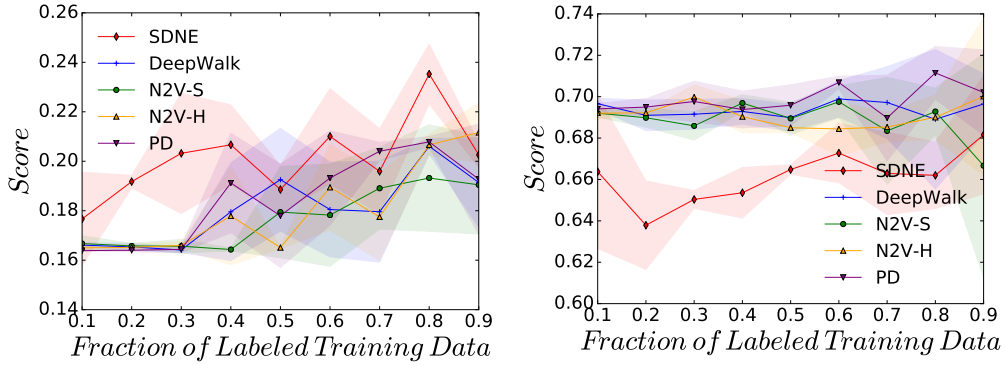
Using the results from this section, particularly the generally higher f1-macro scores which indicate a better prediction across all classes, all the classification results in the remainder of the

chapter are presented using a single hidden layer neural network.

4.5.2 Topological Feature Prediction

In this section, we present the experimental evaluation of the classification of topological features using the embeddings generated from the five approaches (DeepWalk, Node2Vec-H, Node2Vec-S, SDNE and PD) on the datasets detailed in Table 4.3. We present both the macro-f1 and micro-f1 scores plotted against a varying amount of labelled data available during the training process. Here, a higher score equates to a better classification result – with a score of one meaning a perfect classification of every example in the data. Each point of the line representing the mean result from 5-fold cross validation, and the coloured area around the line representing the standard deviation.

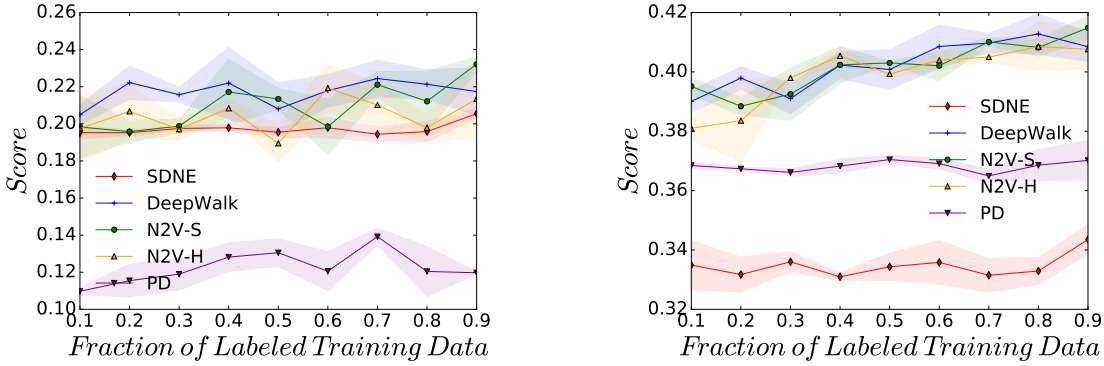
Figures 4.2 to 4.7 display the classification f1 scores for predicting the simplest feature we are measuring: the degree of the vertices. Interestingly we see a large spread of results across the datasets and between approaches, with no clear pattern emerging in this set of results. On certain datasets, it is possible to see a high micro-f1 score, for example in the bitcoinotc dataset, suggesting that an approximation of the degree value is present in the embedding. The set of figures also show that SDNE and PD often have a lower score when compared with the stochastic approaches. One interesting phenomenon is the characteristic saw-tooth pattern in the predicted value for the Eu-Core dataset, as seen in Figure 4.4. This can be attributed to the dataset being the smallest of the ones used for this set of experiments, as a small change in the prediction of the model (for example, making one prediction over another) can have a disproportionally large impact on performance. This pattern can be seen again with this dataset for many of the other topological features predicted in this section.



(a) Macro Drosophila

(b) Micro Drosophila

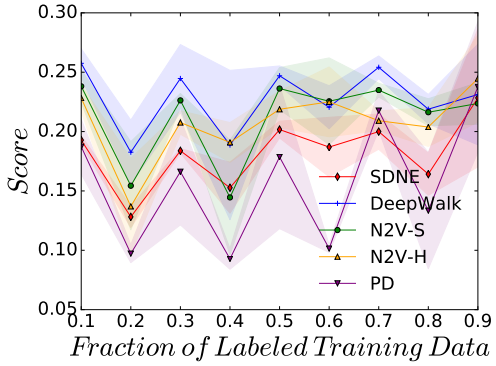
Figure 4.2: Micro and Macro F1 Scores, across a range of labelling fractions, for all approaches when predicting a vertex's Degree (DG) value on the fly-drosophila-medulla dataset.



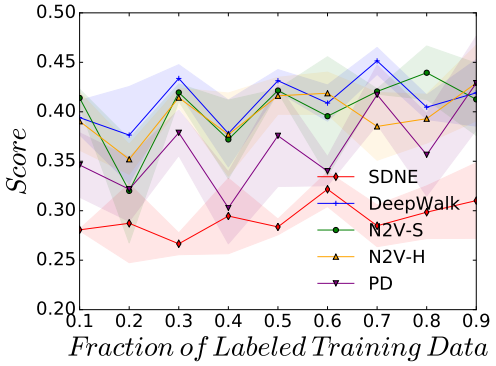
(a) Macro HepTh

(b) Micro HepTh

Figure 4.3: Micro and Macro F1 Scores, across a range of labelling fractions, for all approaches when predicting a vertex's Degree (DG) value on the cit-HepTh dataset.

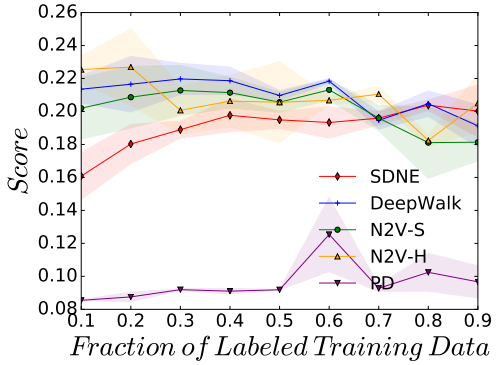


(a) Macro Email-EU

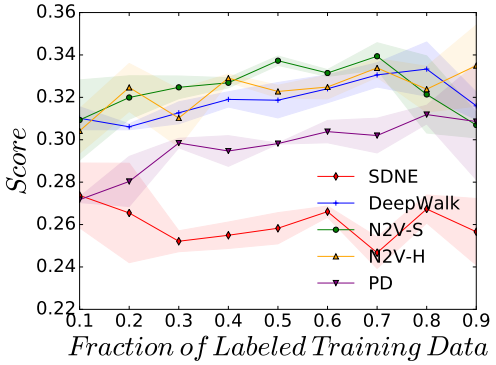


(b) Micro Email-EU

Figure 4.4: Micro and Macro F1 Scores, across a range of labelling fractions, for all approaches when predicting a vertex's Degree (DG) value on the email-Eu-core dataset.

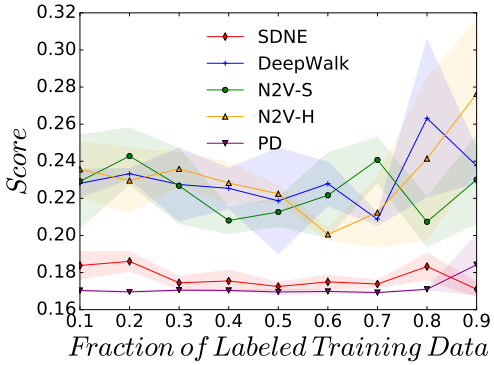


(a) Macro Facebook

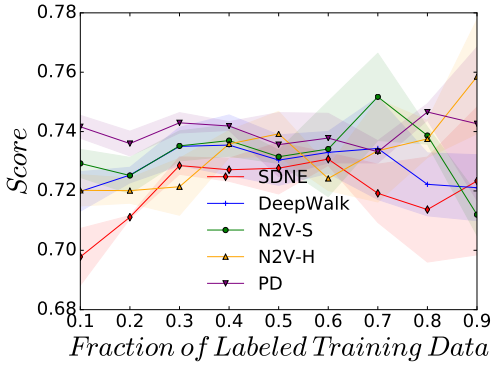


(b) Micro Facebook

Figure 4.5: Micro and Macro F1 Scores, across a range of labelling fractions, for all approaches when predicting a vertex's Degree (DG) value on the ego-Facebook dataset.

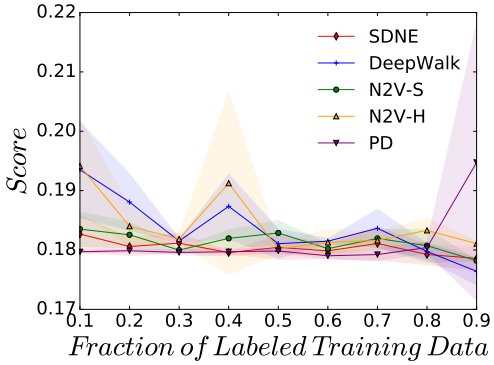


(a) Macro Openflights

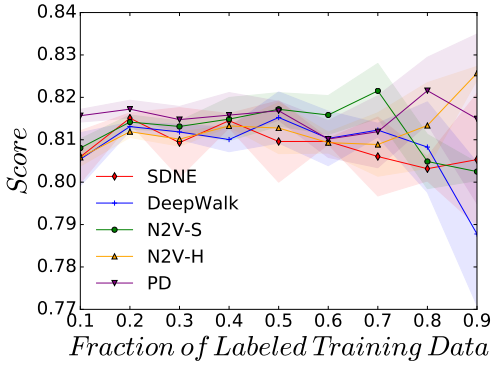


(b) Micro Openflights

Figure 4.6: Micro and Macro F1 Scores, across a range of labelling fractions, for all approaches when predicting a vertex's Degree (DG) value on the inf-openflights dataset.



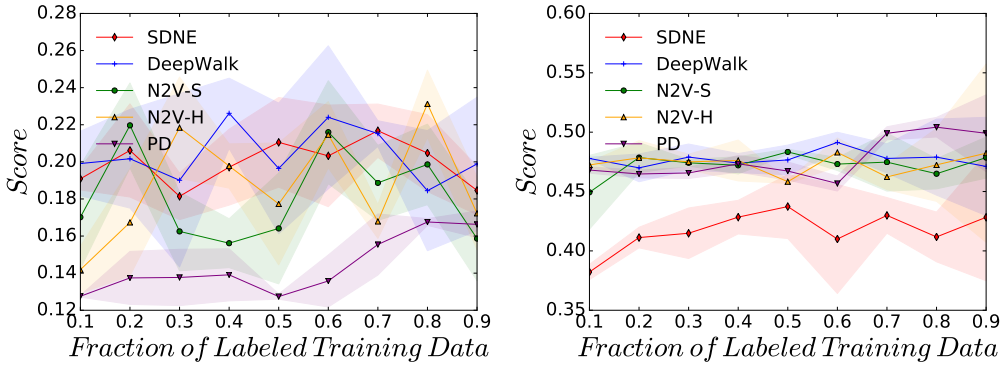
(a) Macro Bitcoinotc



(b) Micro Bitcoinotc

Figure 4.7: Micro and Macro F1 Scores, across a range of labelling fractions, for all approaches when predicting a vertex's Degree (DG) value on the soc-sign-bitcoinotc dataset.

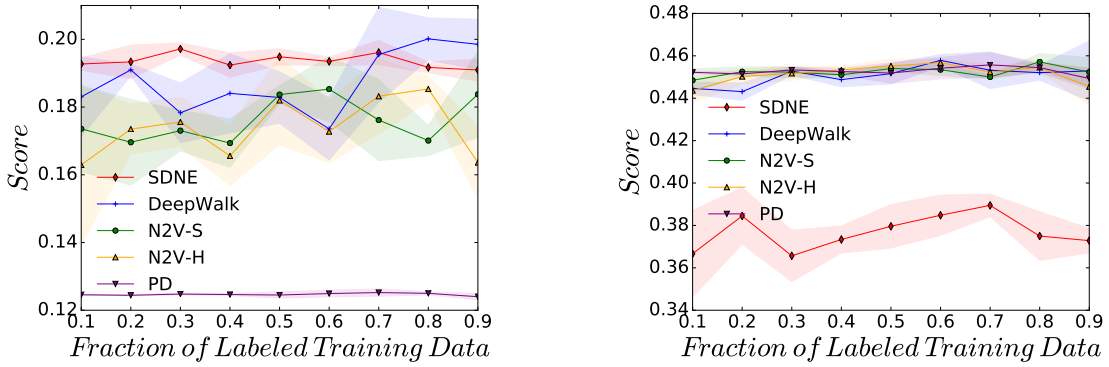
Figures 4.8 to 4.13 highlight the macro-f1 and micro-f1 scores for the classification of the Degree Centrality value. As the Degree Centrality of a given vertex is strongly influenced by its degree, it is perhaps unsurprising to observe largely similar patterns to those in the degree figures, which again shows the dataset bitcoinotc to be the dataset with the highest accuracies. As was seen in the set of degree figures, generally the three stochastic approaches have a similar score for both macro-f1 and micro-f1.



(a) Macro Drosophila

(b) Micro Drosophila

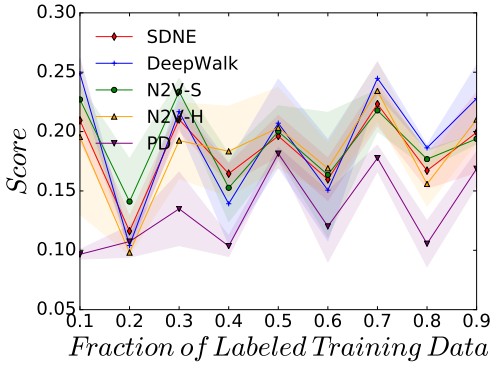
Figure 4.8: Micro and Macro F1 Scores, across a range of labelling fractions, for all approaches when predicting a vertex's Degree Centrality (DC) value on the fly-drosophila-medulla dataset.



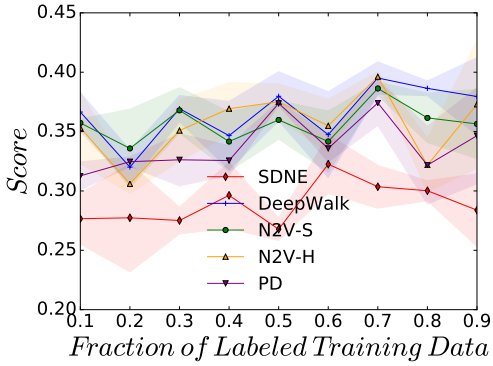
(a) Macro HepTh

(b) Micro HepTh

Figure 4.9: Micro and Macro F1 Scores, across a range of labelling fractions, for all approaches when predicting a vertex's Degree Centrality (DC) value on the cit-HepTh dataset.

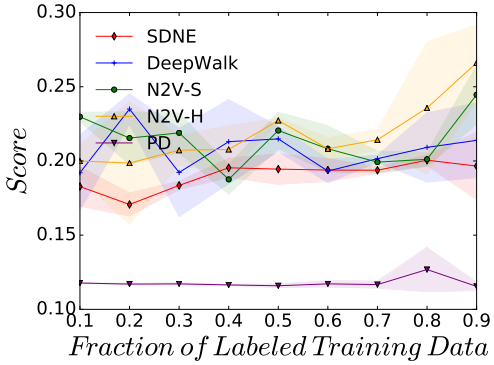


(a) Macro Email-EU

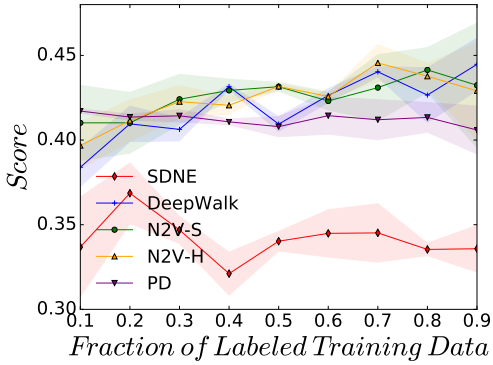


(b) Micro Email-EU

Figure 4.10: Micro and Macro F1 Scores, across a range of labelling fractions, for all approaches when predicting a vertex's Degree Centrality (DC) value on the email-Eu-core dataset.

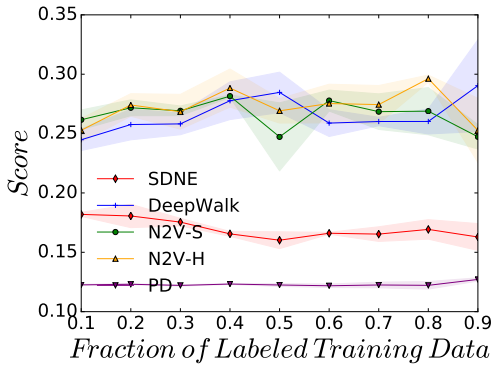


(a) Macro Facebook

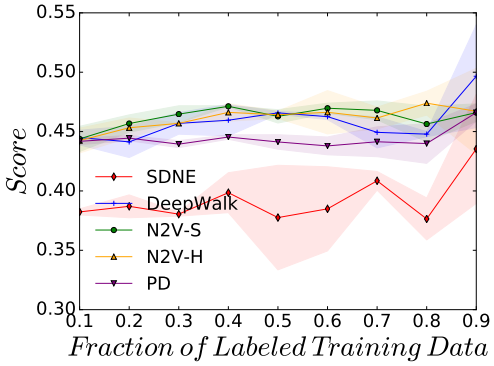


(b) Micro Facebook

Figure 4.11: Micro and Macro F1 Scores, across a range of labelling fractions, for all approaches when predicting a vertex's Degree Centrality (DC) value on the ego-Facebook dataset.

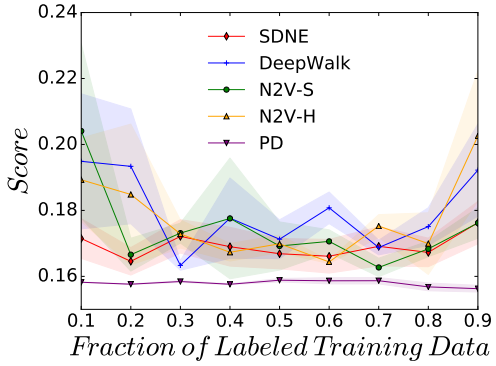


(a) Macro Openflights

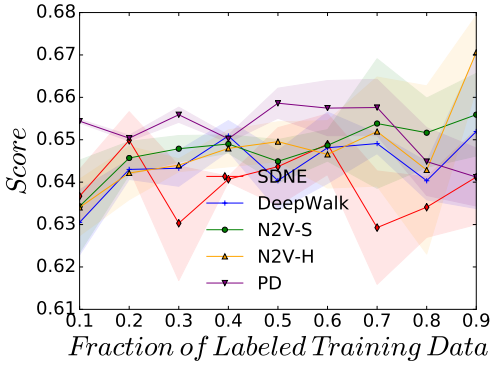


(b) Micro Openflights

Figure 4.12: Micro and Macro F1 Scores, across a range of labelling fractions, for all approaches when predicting a vertex's Degree Centrality (DC) value on the inf-openflights dataset.



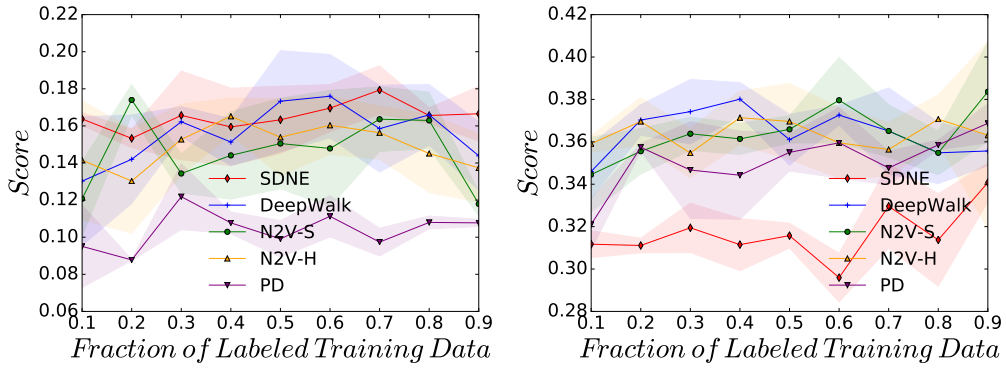
(a) Macro Bitcoinotc



(b) Micro Bitcoinotc

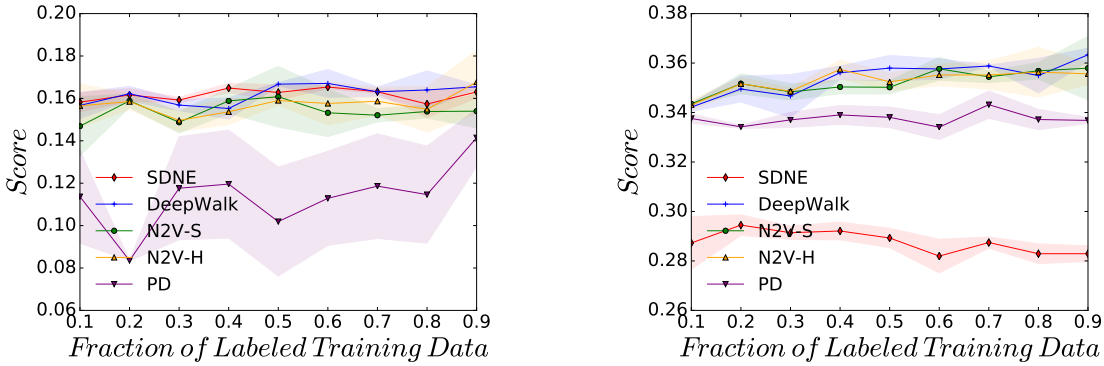
Figure 4.13: Micro and Macro F1 Scores, across a range of labelling fractions, for all approaches when predicting a vertex's Degree Centrality (DC) value on the soc-sign-bitcoinotc dataset.

The results for the classification of Triangle Counts for the vertices are presented in Figures 4.14 to 4.19. This is a more complex feature than the previous two, as it requires more information than is available from just the immediate neighbours of a given vertex. The figures show again that, to some degree of accuracy, the feature is able to be reconstructed from the embedding space, with bitcoinotc having the highest micro-f1 accuracy of all the datasets. SDNE and PD continue to have, on average, the lowest accuracies.



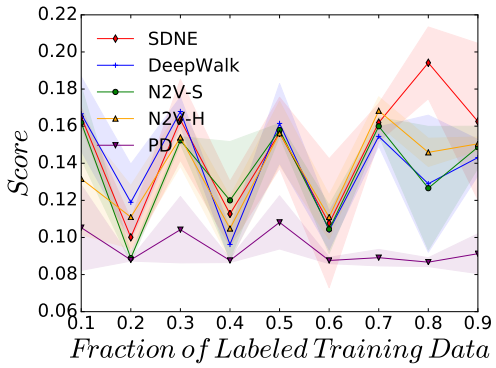
(a) Macro Drosophila (b) Micro Drosophila

Figure 4.14: Micro and Macro F1 Scores, across a range of labelling fractions, for all approaches when predicting a vertex's Triangle Count (TR) value on the fly-drosophila-medulla dataset.

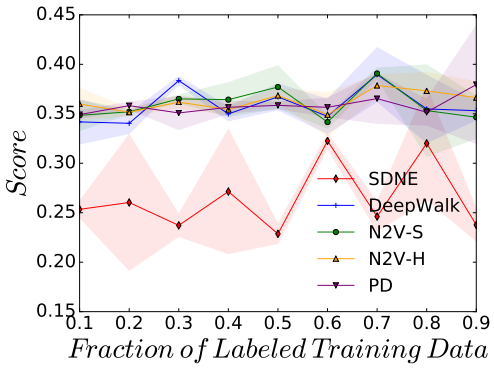


(a) Macro HepTh (b) Micro HepTh

Figure 4.15: Micro and Macro F1 Scores, across a range of labelling fractions, for all approaches when predicting a vertex's Triangle Count (TR) value on the cit-HepTh dataset.

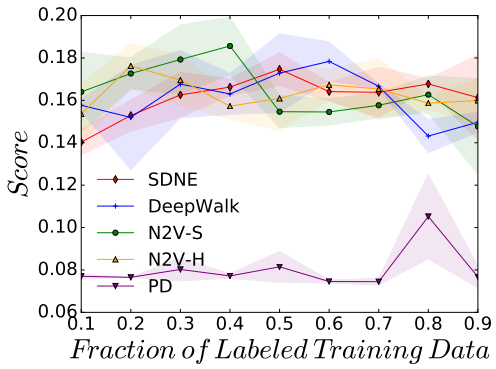


(a) Macro Email-EU

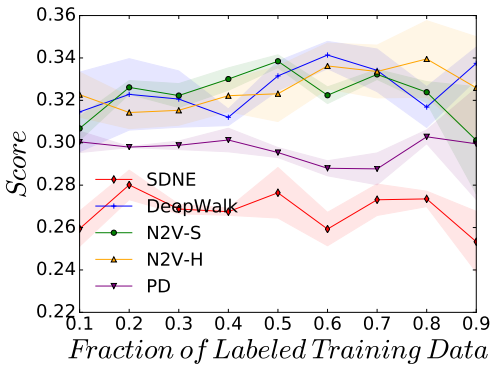


(b) Micro Email-EU

Figure 4.16: Micro and Macro F1 Scores, across a range of labelling fractions, for all approaches when predicting a vertex's Triangle Count (TR) value on the email-Eu-core dataset.

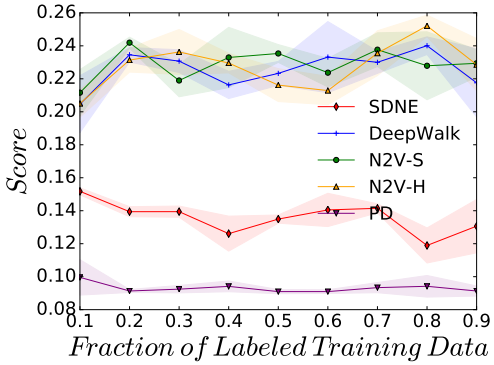


(a) Macro Facebook

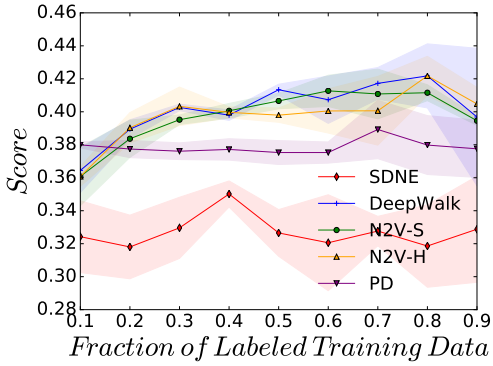


(b) Micro Facebook

Figure 4.17: Micro and Macro F1 Scores, across a range of labelling fractions, for all approaches when predicting a vertex's Triangle Count (TR) value on the ego-Facebook dataset.

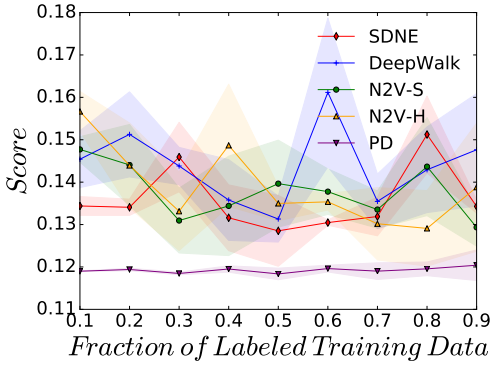


(a) Macro Openflights

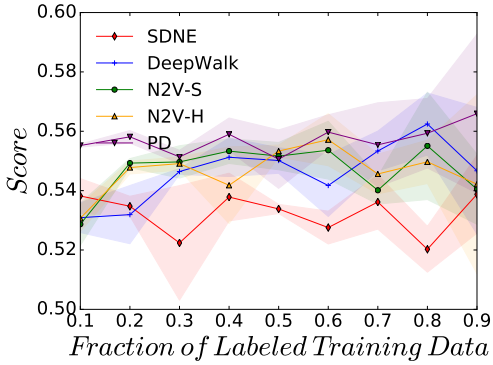


(b) Micro Openflights

Figure 4.18: Micro and Macro F1 Scores, across a range of labelling fractions, for all approaches when predicting a vertex's Triangle Count (TR) value on the inf-openflights dataset.



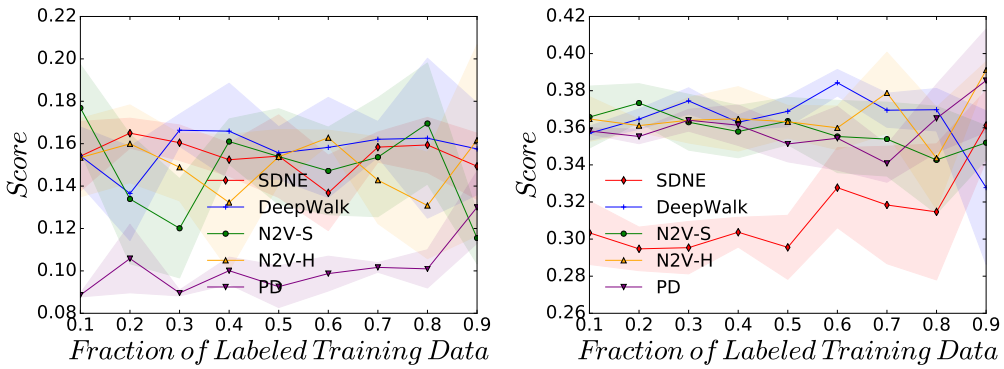
(a) Macro Bitcoinotc



(b) Micro Bitcoinotc

Figure 4.19: Micro and Macro F1 Scores, across a range of labelling fractions, for all approaches when predicting a vertex's Triangle Count (TR) value on the soc-sign-bitcoinotc dataset.

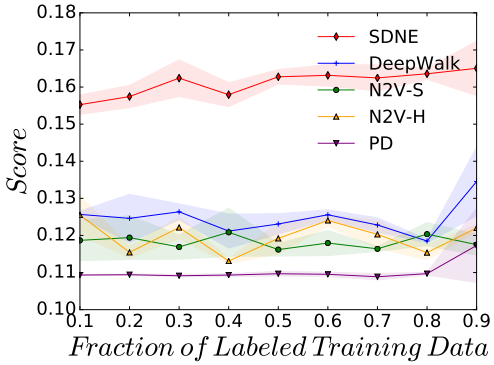
Classifying a vertex's local clustering score across the datasets is explored in Figures 4.20 to 4.25. The figures show that this feature, although more complicated to compute than a vertices triangle count, appears to be easier for a classifier to reconstruct from the embedding space. With this more complicated feature, some interesting results regarding SDNE can be seen in the Email-EU and HepTh datasets, where the approach has the highest macro-f1 score – perhaps indicating that the more complex model is better able to learn a good representation for this more complicated feature.



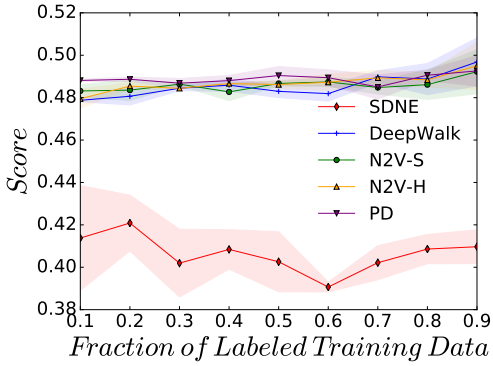
(a) Macro Drosophila

(b) Micro Drosophila

Figure 4.20: Micro and Macro F1 Scores, across a range of labelling fractions, for all approaches when predicting a vertex's Local Clustering Coefficient (CLU) value on the fly-drosophila-medulla dataset.

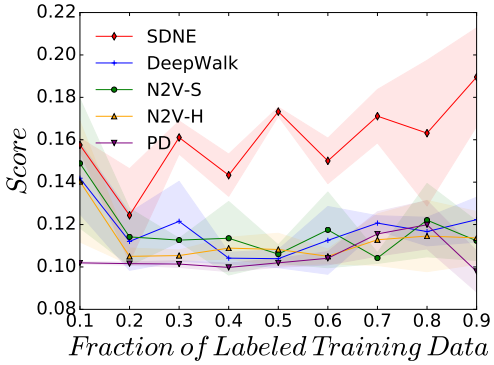


(a) Macro HepTh

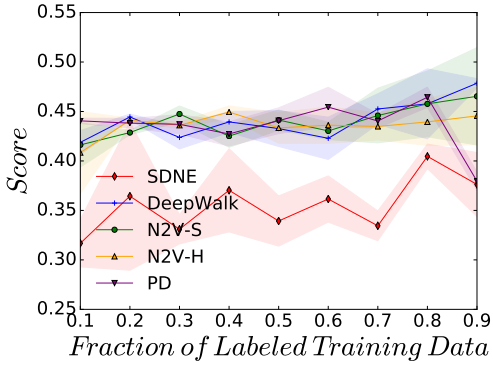


(b) Micro HepTh

Figure 4.21: Micro and Macro F1 Scores, across a range of labelling fractions, for all approaches when predicting a vertex's Local Clustering Coefficient (CLU) value on the cit-HepTh dataset.

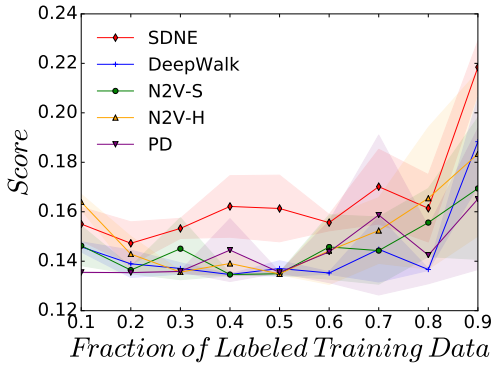


(a) Macro Email-EU

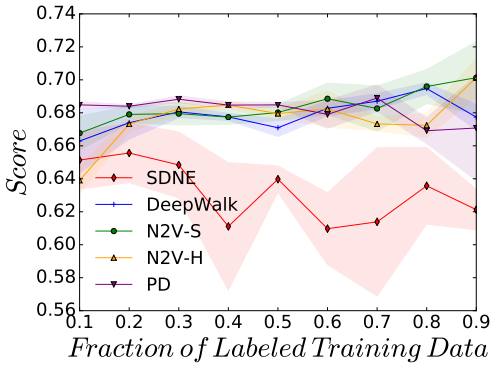


(b) Micro Email-EU

Figure 4.22: Micro and Macro F1 Scores, across a range of labelling fractions, for all approaches when predicting a vertex's Local Clustering Coefficient (CLU) value on the email-Eu-core dataset.

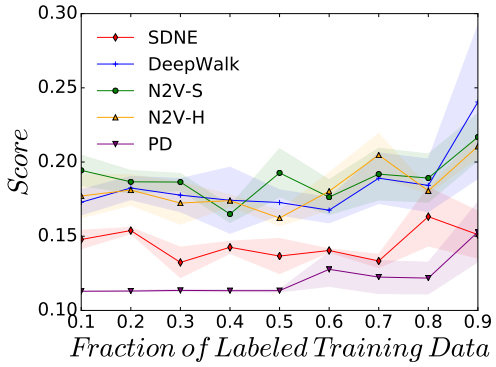


(a) Macro Facebook

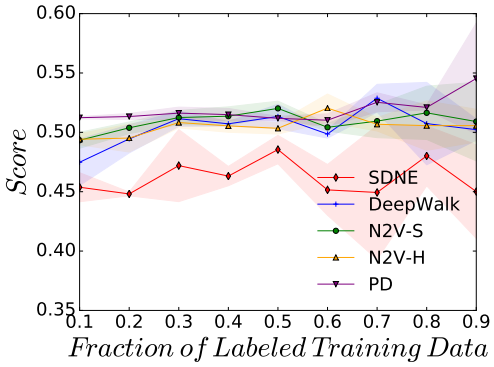


(b) Micro Facebook

Figure 4.23: Micro and Macro F1 Scores, across a range of labelling fractions, for all approaches when predicting a vertex's Local Clustering Coefficient (CLU) value on the ego-Facebook dataset.

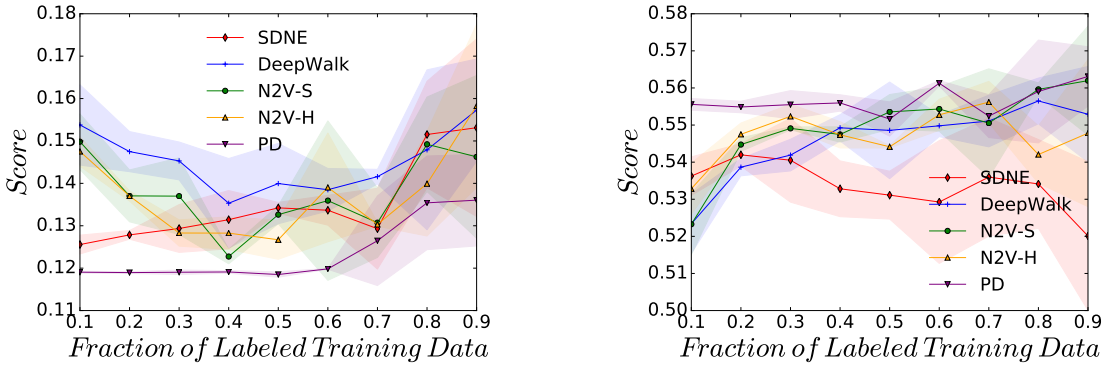


(a) Macro Openflights



(b) Micro Openflights

Figure 4.24: Micro and Macro F1 Scores, across a range of labelling fractions, for all approaches when predicting a vertex's Local Clustering Coefficient (CLU) value on the inf-openflights dataset.

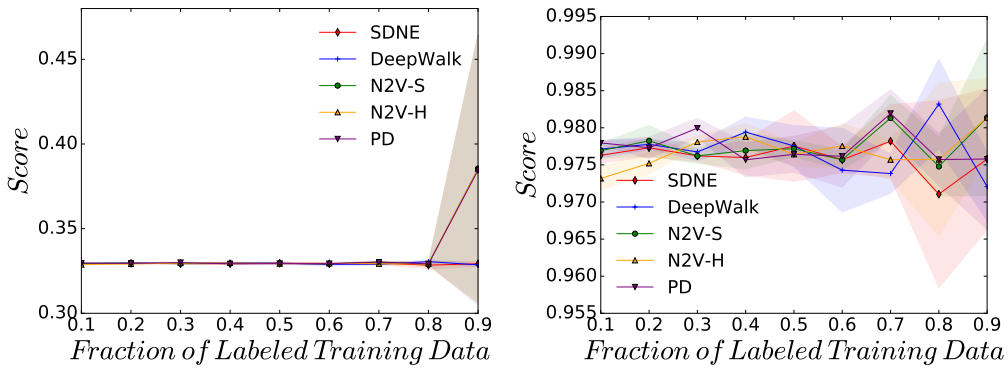


(a) Macro Bitcoinotc

(b) Micro Bitcoinotc

Figure 4.25: Micro and Macro F1 Scores, across a range of labelling fractions, for all approaches when predicting a vertex's Local Clustering Coefficient (CLU) value on the soc-sign-bitcoinotc dataset.

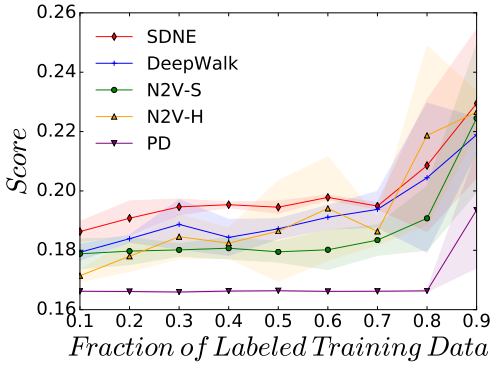
Figures 4.26 to 4.31 display the result for the classification of a vertex's Eigenvector centrality. This set of figures is perhaps the most interesting one so far as it shows high classification accuracies across many of the empirical datasets, even though this feature is of greater complexity than previous ones. They further support the results presented in Table 4.4, which showed Eigenvector centrality having not only the highest accuracies, but also the highest lifts in accuracy over the rule-based predictors. Interestingly SDNE does not demonstrate higher macro-f1 scores in this experiment.



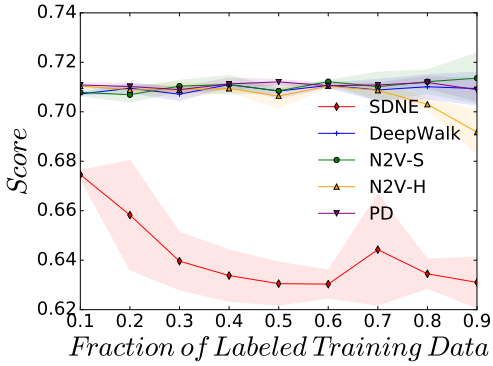
(a) Macro Drosophila

(b) Micro Drosophila

Figure 4.26: Micro and Macro F1 Scores, across a range of labelling fractions, for all approaches when predicting a vertex's Eigenvetor Centrality (EC) value on the fly-drosophila-medulla dataset.

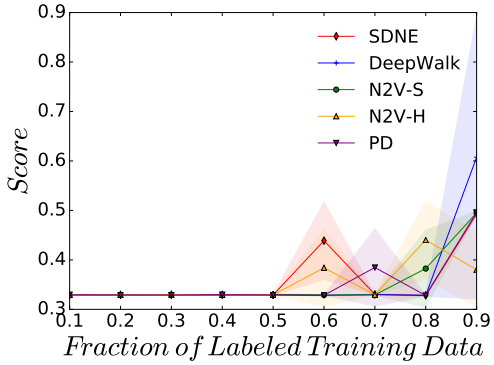


(a) Macro HepTh

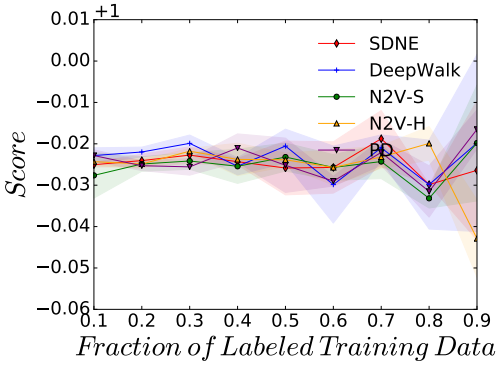


(b) Micro HepTh

Figure 4.27: Micro and Macro F1 Scores, across a range of labelling fractions, for all approaches when predicting a vertex's Eigenvetor Centrality (EC) value on the cit-HepTh dataset.

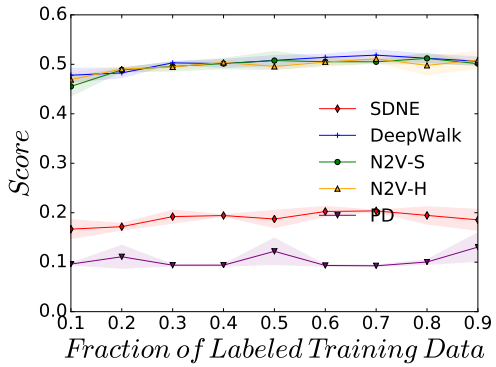


(a) Macro Email-EU

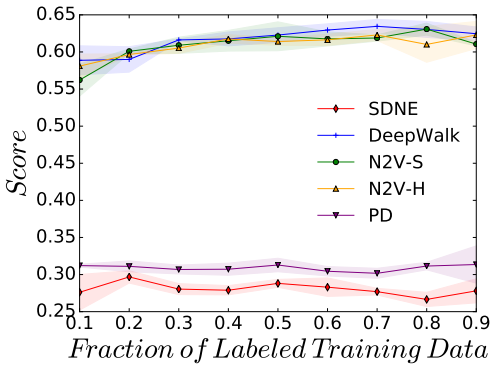


(b) Micro Email-EU

Figure 4.28: Micro and Macro F1 Scores, across a range of labelling fractions, for all approaches when predicting a vertex's Eigenvetor Centrality (EC) value on the email-Eu-core dataset.

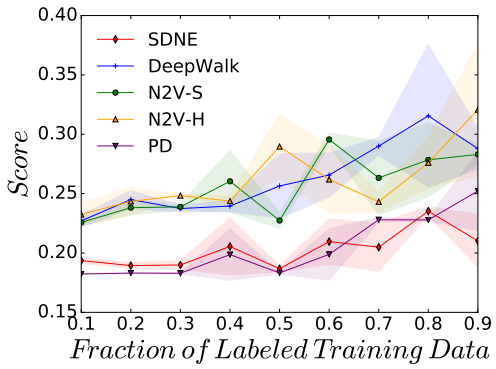


(a) Macro Facebook

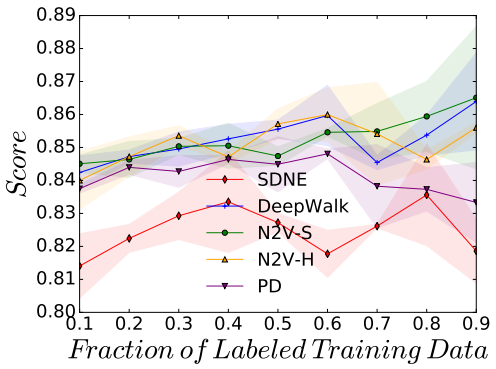


(b) Micro Facebook

Figure 4.29: Micro and Macro F1 Scores, across a range of labelling fractions, for all approaches when predicting a vertex's Eigenvetor Centrality (EC) value on the ego-Facebook dataset.

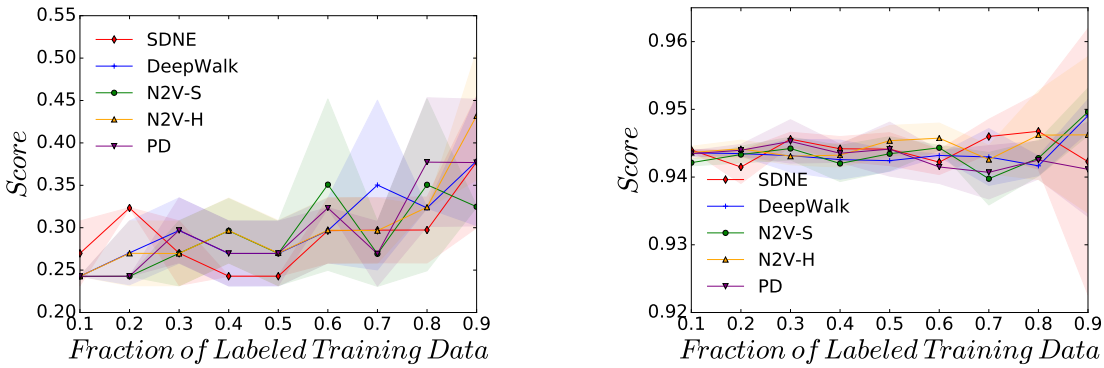


(a) Macro Openflights



(b) Micro Openflights

Figure 4.30: Micro and Macro F1 Scores, across a range of labelling fractions, for all approaches when predicting a vertex's Eigenvetor Centrality (EC) value on the inf-openflights dataset.

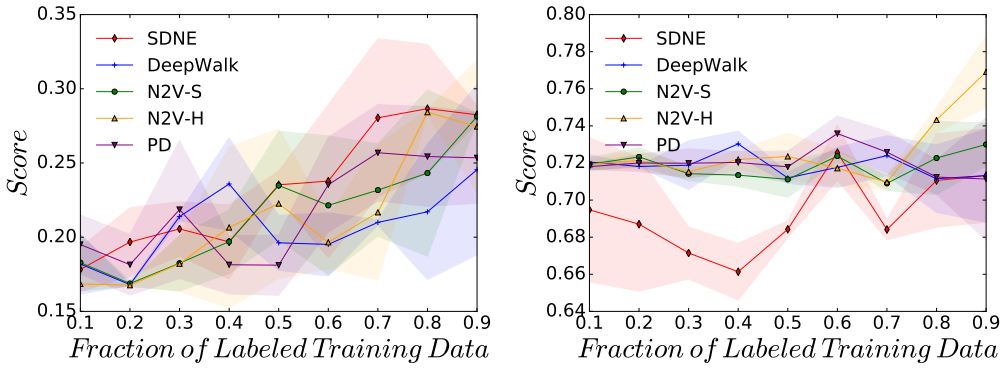


(a) Macro Bitcoinotc

(b) Micro Bitcoinotc

Figure 4.31: Micro and Macro F1 Scores, across a range of labelling fractions, for all approaches when predicting a vertex's Eigenvetor Centrality (EC) value on the soc-sign-bitcoinotc dataset.

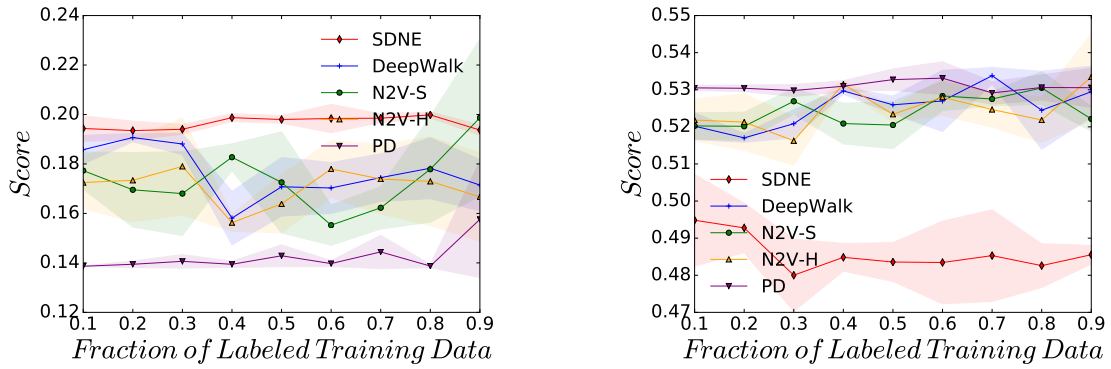
In Figures 4.32 to 4.37, the approaches' ability to correctly classify the PageRank score of the vertices is considered. Here we see generally lower classification accuracies than the last set of figures, perhaps owing to the more complicated nature of the PageRank algorithm, although high classification accuracies can still be seen, particularly on the on the Bitcoinotc and Drosophila datasets.



(a) Macro Drosophila

(b) Micro Drosophila

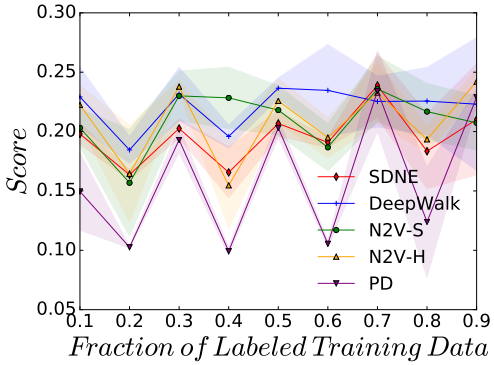
Figure 4.32: Micro and Macro F1 Scores, across a range of labelling fractions, for all approaches when predicting a vertex's PageRank (PR) value on the fly-drosophila-medulla dataset.



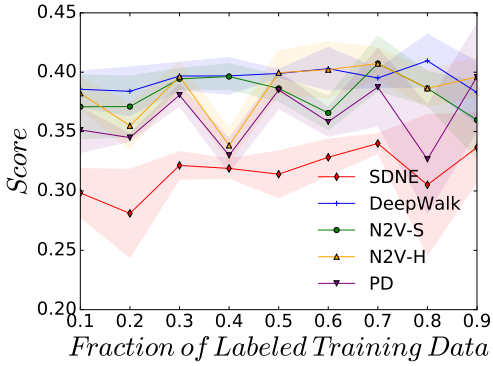
(a) Macro HepTh

(b) Micro HepTh

Figure 4.33: Micro and Macro F1 Scores, across a range of labelling fractions, for all approaches when predicting a vertex's PageRank (PR) value on the cit-HepTh dataset.

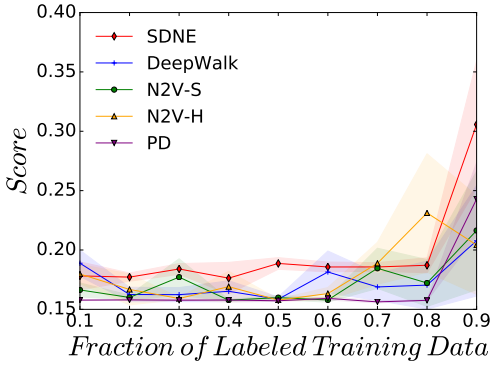


(a) Macro Email-EU

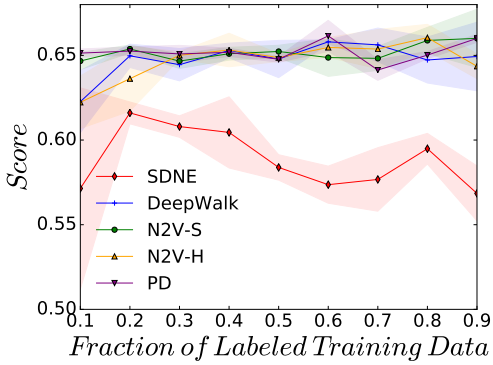


(b) Micro Email-EU

Figure 4.34: Micro and Macro F1 Scores, across a range of labelling fractions, for all approaches when predicting a vertex's PageRank (PR) value on the email-Eu-core dataset.

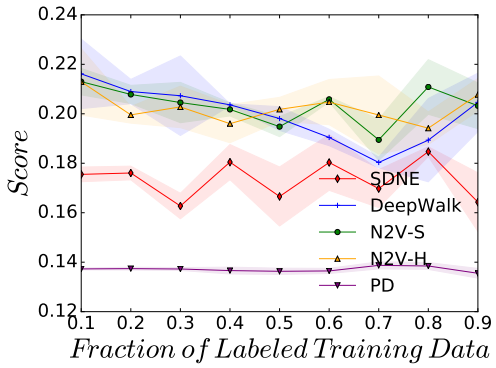


(a) Macro Facebook

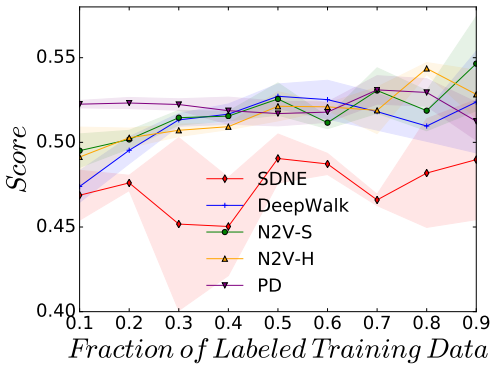


(b) Micro Facebook

Figure 4.35: Micro and Macro F1 Scores, across a range of labelling fractions, for all approaches when predicting a vertex's PageRank (PR) value on the ego-Facebook dataset.

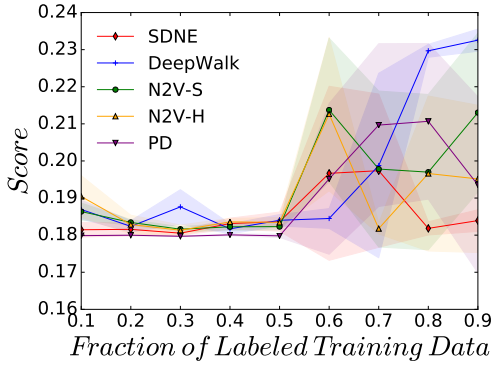


(a) Macro Openflights

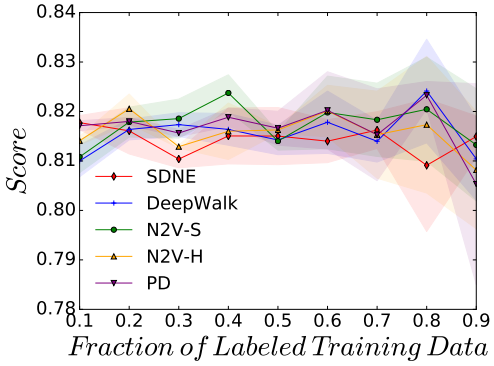


(b) Micro Openflights

Figure 4.36: Micro and Macro F1 Scores, across a range of labelling fractions, for all approaches when predicting a vertex's PageRank (PR) value on the inf-openflights dataset.



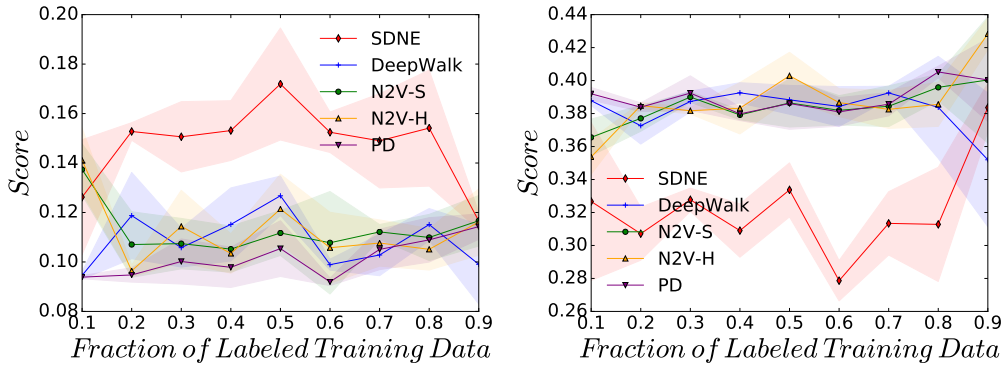
(a) Macro Bitcoinotc



(b) Micro Bitcoinotc

Figure 4.37: Micro and Macro F1 Scores, across a range of labelling fractions, for all approaches when predicting a vertex's PageRank (PR) value on the soc-sign-bitcoinotc dataset.

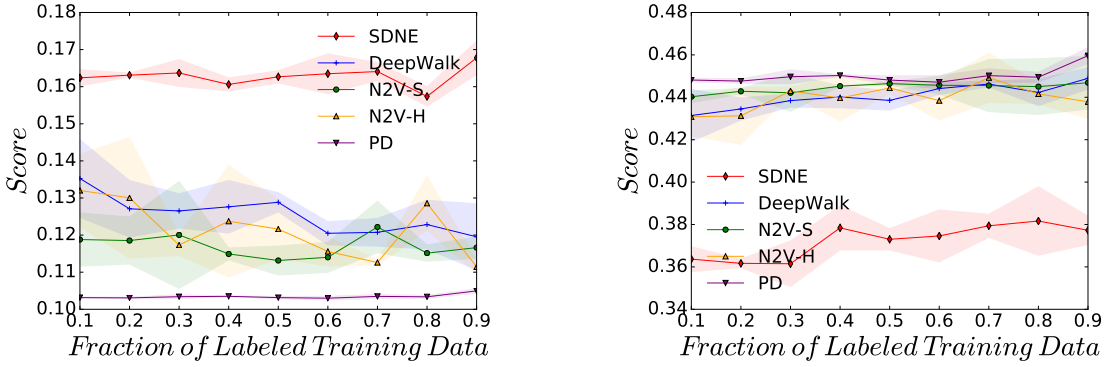
Finally, Figures 4.38 to 4.43 highlight the ability of the graph embeddings to predict betweenness centrality. Here, the figures show that this feature is on average, harder to predict from the embeddings than the previous two centrality measures as evidenced by the lower accuracies scores. Again SDNE shows the highest macro-f1 scores on the Drosophila and HepTh datasets, indicating its embedding capture something closer to this structural information more effectively than the other approaches.



(a) Macro Drosophila

(b) Micro Drosophila

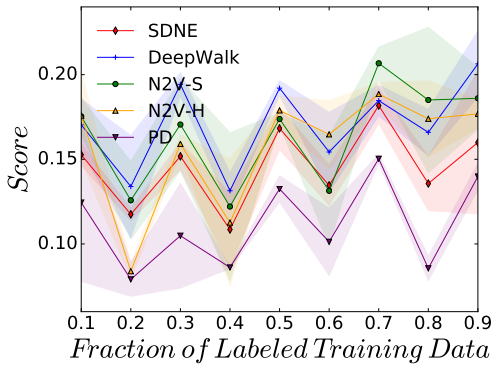
Figure 4.38: Micro and Macro F1 Scores, across a range of labelling fractions, for all approaches when predicting a vertex's Betweenness Centrality (BC) value on the fly-drosophila-medulla dataset.



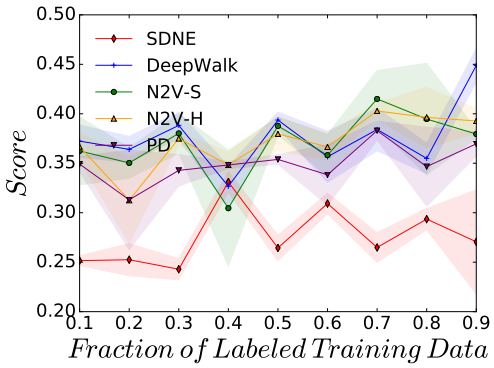
(a) Macro HepTh

(b) Micro HepTh

Figure 4.39: Micro and Macro F1 Scores, across a range of labelling fractions, for all approaches when predicting a vertex's Betweenness Centrality (BC) value on the cit-HepTh dataset.

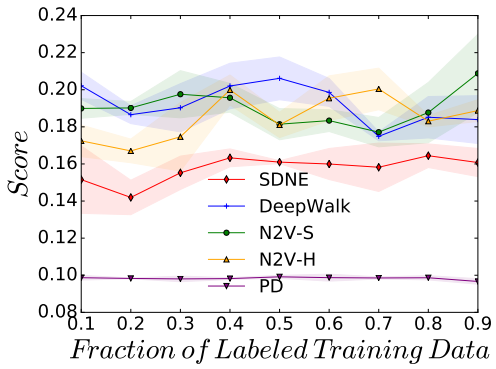


(a) Macro Email-EU

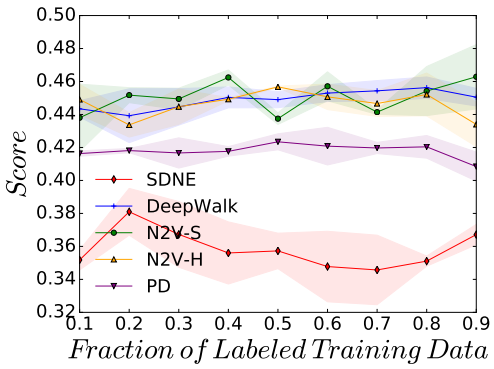


(b) Micro Email-EU

Figure 4.40: Micro and Macro F1 Scores, across a range of labelling fractions, for all approaches when predicting a vertex's Betweenness Centrality (BC) value on the email-Eu-core dataset.

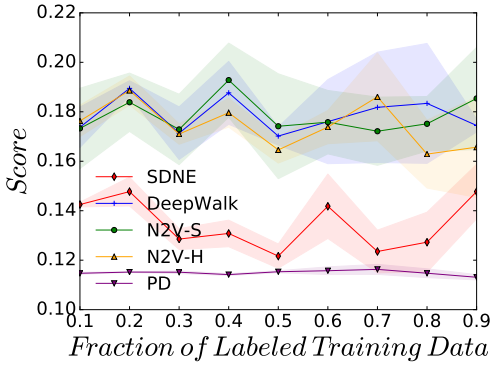


(a) Macro Facebook

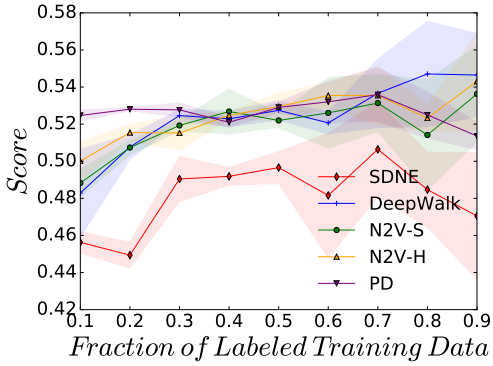


(b) Micro Facebook

Figure 4.41: Micro and Macro F1 Scores, across a range of labelling fractions, for all approaches when predicting a vertex's Betweenness Centrality (BC) value on the ego-Facebook dataset.

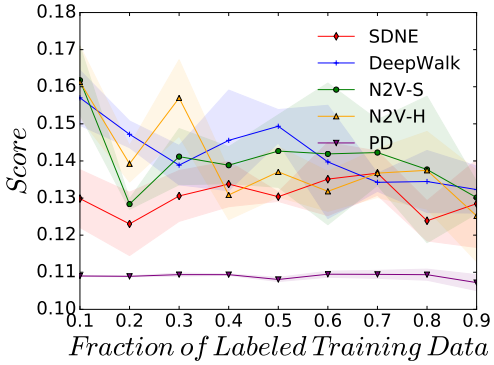


(a) Macro Openflights

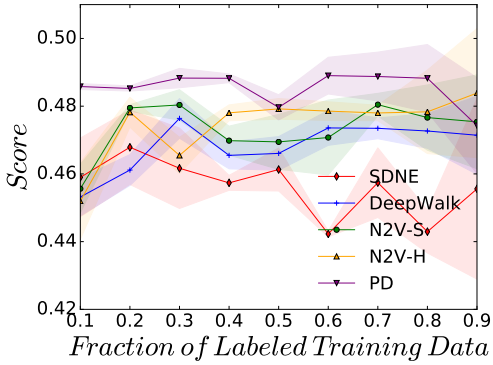


(b) Micro Openflights

Figure 4.42: Micro and Macro F1 Scores, across a range of labelling fractions, for all approaches when predicting a vertex's Betweenness Centrality (BC) value on the inf-openflights dataset.



(a) Macro Bitcoinotc



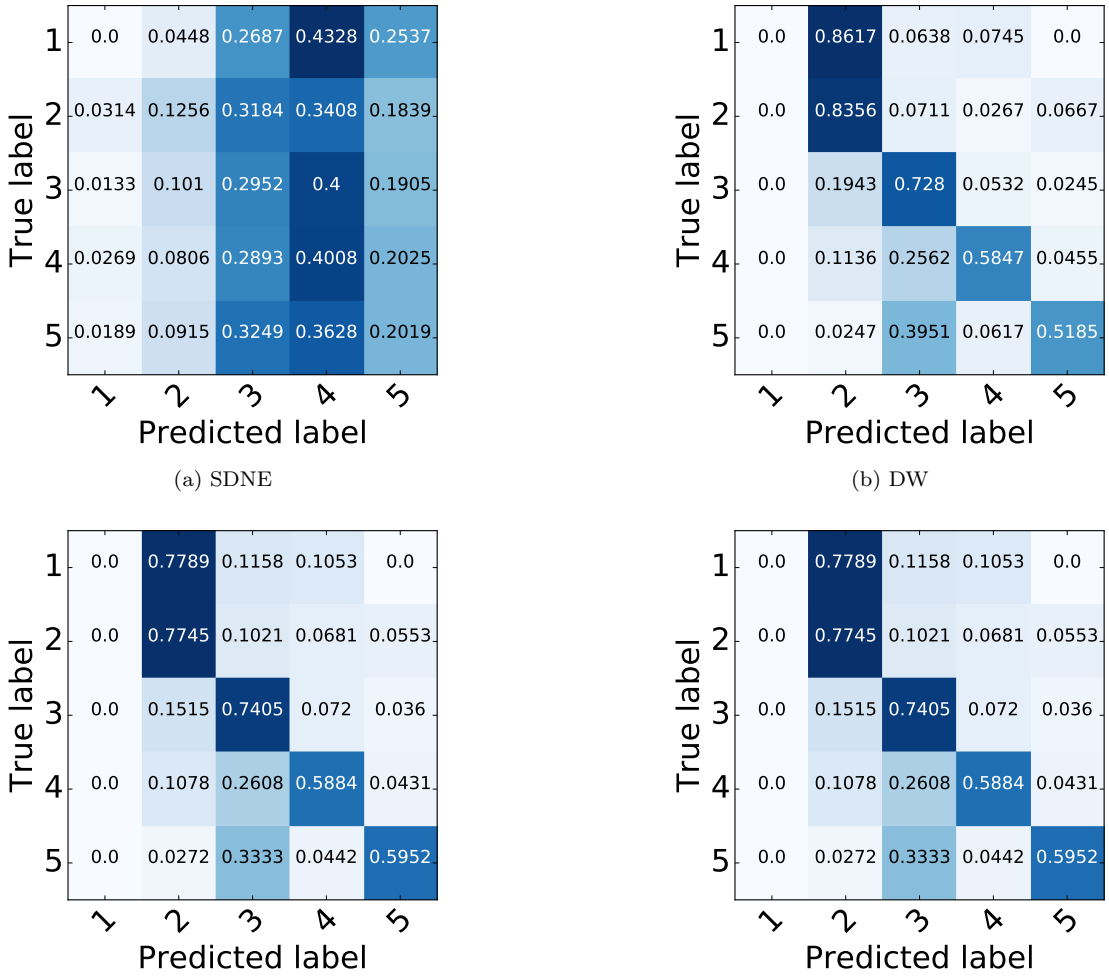
(b) Micro Bitcoinotc

Figure 4.43: Micro and Macro F1 Scores, across a range of labelling fractions, for all approaches when predicting a vertex's Betweenness Centrality (BC) value on the soc-sign-bitcoinotc dataset.

4.5.3 Confusion Matrices

One consideration that must be made is that the binning process, used to transform the features into targets for classification, removes the inherent ordering present in continuous values. As an example, a vertex with a degree of 8 would still be classified incorrectly if the prediction was 10 or 100, but clearly one is more incorrect than the other. To address this, we present a selection of error matrices, to explore how 'wrong' an incorrect prediction is. This is made possible as the labels used for classification have consecutive ordering, as a result of a histogram binning function, meaning that a prediction of 2 for a true label of 1, is more correct than a prediction of 5.

Figure 4.44 displays the error matrices for a selection of the tested embedding approaches when classifying Eigenvector Centrality in the ego-Facebook dataset, although similar patterns were found across all datasets. With error matrices, the diagonal values represent a correctly classified label, thus a good prediction will produce an error matrix with a higher concentration of diagonal values. Figure 4.44 shows that, for the stochastic walk approaches DeepWalk and Node2Vec, the error matrices have a higher clustering of values around the diagonals. Interestingly, when the classification is incorrect for these approaches, the incorrect prediction tends to be close to the true label. This phenomenon can clearly be seen in these approaches for labels 1 and 2, meaning that embeddings for vertices with this particular Eigenvector Centrality are similar. Figure 4.44 also shows that, for this particular vertex feature, the embeddings produced via SDNE seemingly do not contain the same topological information. This is highlighted by the lack of structure on the diagonals of its error matrix.



(a) SDNE

(b) DW

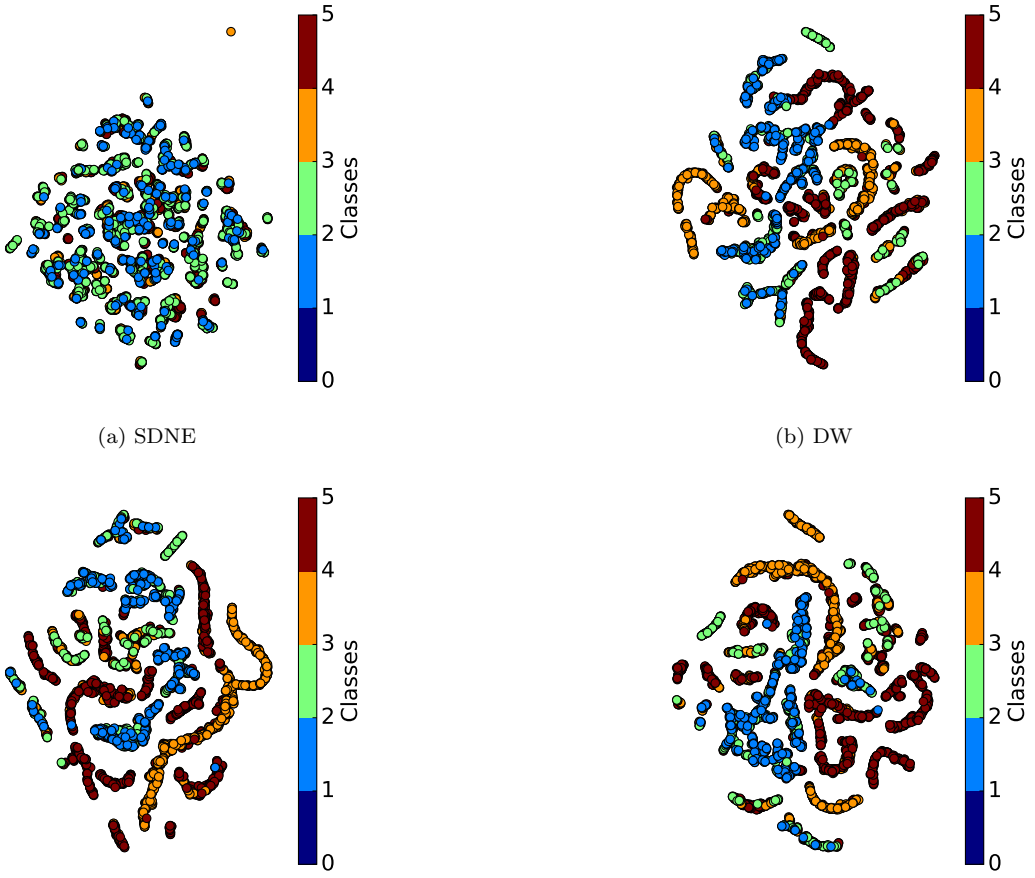
(c) N2V-H

(d) N2V-S

Figure 4.44: Error matrices for neural network classification of Eigenvector Centrality (EC) for the ego-Facebook dataset.

4.5.4 Unsupervised Low-Dimensional Projections

Figure 4.45 displays a selection of t-SNE plots taken from the ego-Facebook data, where the points are coloured according to the Eigenvector centrality value after being passed through the binning process. The figure shows that the SDNE embeddings seemingly have no clear structure in the low dimensional space which correlates strongly with the Eigenvector centrality, as points in the same class are not clustered together. However, with the other embedding approaches, it is possible to see a clear clustering of points belonging to the same class. For example, in both the Node2Vec approaches, there is very clear clustering of classes one, four and five. This result provides further evidence for our observation that, even when exploring the embeddings using an unsupervised method, it is possible to find correlations between known topological features and the embedding space.



(a) SDNE

(b) DW

(c) N2V-H

(d) N2V-S

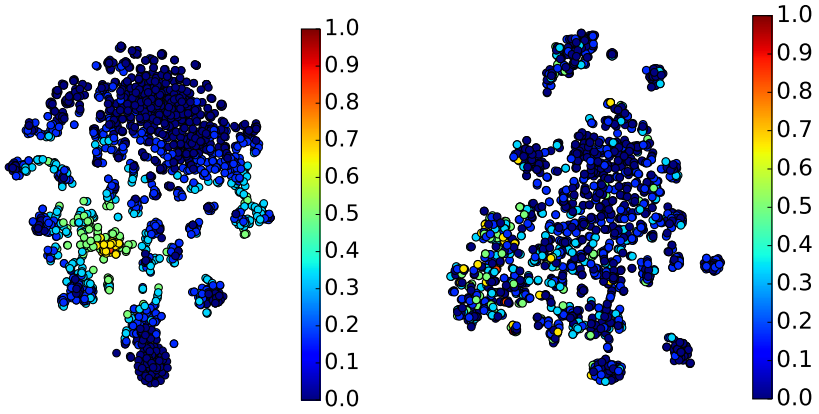
Figure 4.45: t-SNE plots of the embeddings taken from the ego-Facebook dataset, where the points are coloured according to their Eigenvector Centrality (EC) value.

4.5.5 Auto-Encoder Comparsion

The results presented thus far have shown that it can be comparatively challenging to recover evidence of topological features from the auto-encoder based SDNE approach. To investigate this further, we compare SDNE with another auto-encoder based approach entitled DNGR. Unlike the other approaches tested thus far, DNGR mandates the use of weighted graphs. However, from the empirical datasets we are using for this study, only the soc-sign-bitcoinotc dataset contains weighted edges, which represent the level of trust which users place in each other.

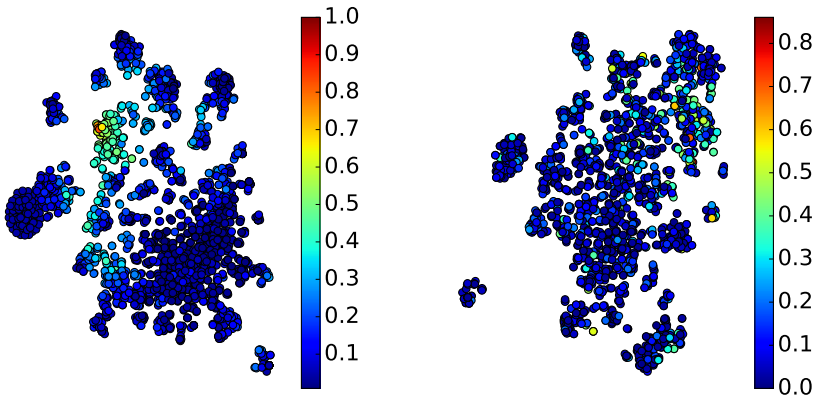
To investigate whether DNGR captures more recognisable topological structure in its embedding space, we will again use t-SNE. However, the soc-sign-bitcoinotc dataset has the lowest edge density of any of the graphs we are testing, resulting in a very unbalanced dataset (for example, the majority of the vertices have a very low degree value). To allow for greater insight, here we choose not to use the binning process to label each vertex embedding. Instead, we normalise the raw topological feature values to be between zero and one, we then use this value to directly colour the points on the t-SNE plots. Here we would expect to see points of a similar colour, and thus feature value, to be clustered together if vertices with similar topological features are close in the underlying embedding space. Due to soc-sign-bitcoinotc having a larger number of vertices than the dataset used for the previous t-SNE visualization, we plot only a randomly selected half of the vertices to allow for clearer figures.

Figures 4.46 to 4.49 display the t-SNE plots of the vertex embeddings for both SDNE and DNGR across four different topological features. The figures show that despite it being more challenging to recover topological features from SDNE in other experiments, there is still structure present in the embedding space correlating to several topological features. One can see SNDE embeddings with similar feature values being clustered together in these plots, for example there are clear clusters of vertices with a high and low degree, PageRank and Betweeness Centrality value visible. However, it is much harder to interpret any structure in the embedding space produced via DNGR. This could well be due to the fact that DNGR does not take as input the raw adjacency matrix, instead it is reconstructing the PPMI matrix, capturing vertex co-occurrence. Due to this transformed input, it is perhaps not surprising that normal topological features are present in the resulting representations.



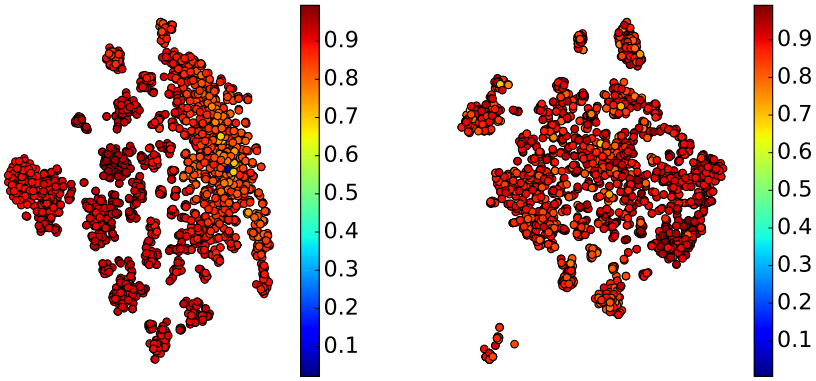
(a) SDNE DG (b) DNGR DG

Figure 4.46: t-SNE plots of SDNE and DNGR embeddings taken from the soc-sign-bitcoinitc dataset, where points are coloured according to the normalized degree value.



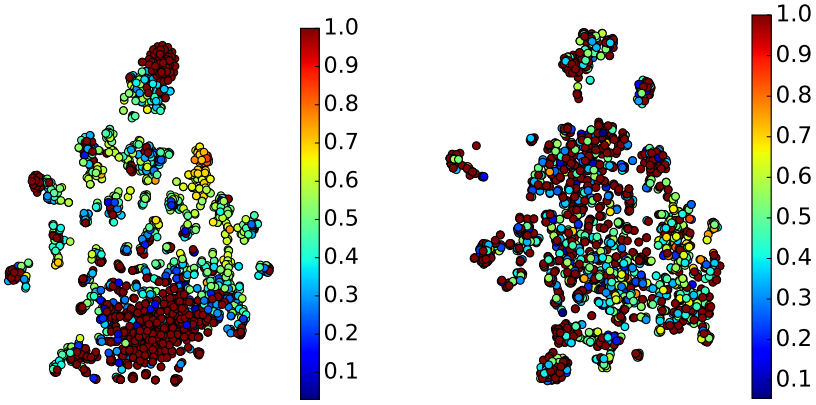
(a) SDNE PR (b) DNGR PR

Figure 4.47: t-SNE plots of SDNE and DNGR embeddings taken from the soc-sign-bitcoinitc dataset, where points are coloured according to the normalized pagerank value.



(a) SDNE EC (b) DNGR EC

Figure 4.48: t-SNE plots of SDNE and DNGR embeddings taken from the soc-sign-bitcoinitc dataset, where points are coloured according to the normalized Eigenvector centrality value.



(a) SDNE BC (b) DNGR BC

Figure 4.49: t-SNE plots of SDNE and DNGR embeddings taken from the soc-sign-bitcoinitc dataset, where points are coloured according to the normalized Betweeness centrality value.

4.5.6 Discussion

This section has provided extensive experimentation evaluation to explore topological feature presence in graph embeddings. Specifically, we investigated whether a broad range of topological features can be predicted from the embedding created from a range of unsupervised graph embedding techniques. Across all the features and datasets tested, it can be seen that many topological features can be approximated by the different embedding approaches, with varying degrees of accuracy. The results which show the increase in accuracy over the rule based predictions (Section 4.5.1) give strong indication that the approaches are able to overcome the inherent unbalanced nature of graph datasets and a mapping from the embedding space to features is present. It is also interesting to observe that numerous features can be approximated from the graph embeddings, suggesting that several structural properties are being automatically captured to create the best representation for a vertex. Of all the topological features measured in the experimentation section, the one which consistently gave the best results was Eigenvector centrality. Particularly for the stochastic approaches, Eigenvector centrality was predicted with a high degree of accuracy, suggesting that the topological structure represented by this feature is captured extremely well in the embedding space and indicates that this is a useful feature for minimising the objective functions of the approaches. This is further reinforced by the unsupervised projections (Figure 4.45), which shows clear and distinct clustering between classes, even without the use of a classification algorithm.

Another interesting observation from this study is that no one approach strongly outperforms the others when classifying a particular feature – seemingly all the approaches are approximating similar topological structures. The results overall show that the stochastic approaches (DeepWalk and Node2Vec) are the most consistent across all features and datasets, often having the highest macro-f1 and micro-f1 scores. SDNE demonstrates a more inconsistent performance profile for feature classification. This is in contrast to other studies which have found it to have the best performance in vertex labelling problems [84]. The performance of SDNE demonstrated in this work could be explained by it being the only deep model tested, meaning that it contains many more parameters. This increase in complexity means that SDNE could be very sensitive to the correct selection of hyper-parameters or possibly that more complex topological features are being approximated by the embeddings – or even that entirely novel features are being learned. Finally, it is interesting to note the performance of Hyperbolic (PD) approach, which has far fewer latent dimensions in which to capture topological information due to its limitation in modelling the space as a 2D disk. Empirically, PD shows largely similar performance to the other approaches

on most datasets, providing strong evidence that the hyperbolic space is an appropriate space in which to represent graphs.

4.6 Conclusion

Graph embeddings are increasingly becoming a key tool to solve numerous tasks within the field of graph mining. They have demonstrated state-of-the-art results by attempting to automatically learn a low dimensional, but highly expressive, representation of vertices, which captures the topological structure of the graph. However to date, there has been little work providing a theoretical grounding which would allow for greater interpretability. This chapter has begun to take a step in this direction by investigating which traditional topological graph features can be reconstructed from the embedding space. The hypothesis of this work being that if a mapping from the embedding space to a particular topological feature can be found, then the topological structure encapsulated by this feature is also captured by the embedding. The conclusions drawn in this chapter are supported by an extensive set of experiments exploring this issue across five unsupervised graph embedding techniques, classifying seven graph features, across a range of empirical datasets. The experiments find that a mapping from many topological features to the embedding space of the tested approaches is indeed possible, using both supervised and unsupervised techniques. This discovery suggests that graph embeddings are indeed learning approximations of known topological features, with the experiments showing Eigenvector centrality to be the best reconstructed by many of the approaches. This could allow key insights into how graph embedding techniques learn to create high quality representations, allowing for the embedding process to become more interpretable.

4.6.1 Current Limitations

Whilst the work presented in this chapter has allowed for some insights into what type of topological structures are being approximated in the embedding space, there are some limitations with the work:

Binning Process: In order to perform the supervised classification of vertex features from the embedding space, a binning process was used to transform the continuous values into discrete

labels. This process introduces an additional hyper parameter, that being the number of bins used in the transformation process. Whilst this process worked well for this research, careful attention needs to be paid to the number of bins used to ensure a balanced distribution of values between them. Additionally the features are further transformed by taking the logarithm of the original values, causing some precision to be lost.

Limited Features: The work presented in this chapter used a total of seven vertex level features to assess the semantic content of the graph embeddings. Whilst some interesting results were observed, a different set of features would perhaps result in a different picture emerging. In particular, the current set of features largely does not measure the presence of any small world-like, modular or hierarchical community structure within the graph.

Lack of Supervised Approaches: This work has focused solely on the family of unsupervised graph embedding techniques. However there is a large and increasing number of supervised approaches, including the majority of the Graph Neural Network (GNN) approaches [234, 253], which learn vertex level representations which have been fine-tuned to perform a specific task – vertex classification for example [128]. Currently the work presented here offers no insights into whether topological features are being approximated by these techniques.

4.6.2 Future Work

For future research, work could be performed to see if other Eigenvector based topological features, known to be representative of a graph's topology [152], are also captured as well by the embedding approaches. To help combat the heavy-tailed distribution of vertex feature values, more experimentation could be performed with synthetically created graphs with artificially balanced degree distributions. This will remove the unbalanced nature of empirical datasets, and allow us to explore the structure of the embeddings in more detail. Furthermore, it could be possible to use the research performed in this chapter to produce better embeddings which generalise more efficiently across other tasks.. This could be achieved by directly predicting topological features during the embedding training process, perhaps taking the form of an additional regularisation term in the loss function.

Epilogue

This chapter has explored the possibility of using topological features as a way to bring interpretability to unsupervised graph embedding techniques, thereby achieving research objective 2. The experimentation, using five state-of-the-art embedding approaches, has shown that a relationship can be found between a selection of known topological features and the embedding space.

In the following chapter, work will shift to studying how the temporal evolution present in many empirical graph datasets can be modelled and predicted via the use of machine learning. The research will continue the theme of unsupervised learning from the present chapter, but move attention to developing new models based around Graph Neural Networks [127, 128] in order to incorporate temporal dynamics in the learning process.

Chapter 5

Temporally Robust Graph Embeddings

Prologue

The work in Chapter 4 explored what topological features are being captured by a range of unsupervised techniques for learning representations on static graphs. However the majority of data whose relationships are being represented as a graph are dynamic in nature, a trait disregarded by all the techniques explored thus far in this thesis and many currently present in the literature.

The work in this chapter will explore new techniques for how best to incorporate the temporal evolution present in many graph datasets into the learning process - and ultimately explore how best to predict the changes in graphs over time. In order to address research objective 3 (defined in Section 1.3), this chapter introduces two new models for learning on temporal graphs and details how they can be used to tackle several key tasks within the field of graph mining including the challenging problem of temporal link prediction.

The work presented in this chapter has been published as the following works:

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 *IEEE International Conference on Big Data*, pages 3737–3746. IEEE, 2018

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 *IEEE International Conference on Big Data*, 2019

5.1 Introduction

Using graphs to represent relationships in large, complex and high-dimensional datasets has become a universal phenomenon across many scientific fields, encompassing not only computer scientists, interested in social and citation networks [128], but biologists, studying protein interaction graphs for associations with diseases [237] or modelling hierarchy in brain networks [114], chemists, who model molecule properties by treating them as graphs [233], and physicists, who use graphs to model a physical environment [22]. As such they provide a useful abstraction for how data is related. Graphs allow for complex analysis to be performed such as identifying the missing link within a graph (a person whom you might know or that paper you must read), however, to date almost all of the prediction work which has been performed on graphs has been focused on analysis in the topological domain as opposed to the temporal domain. This is interesting as almost all graphs change with time (making new friends or publishing new papers). An example of a temporal graph, where new edges are being formed between vertices is illustrated in Figure 5.1 which contains a four vertex graph evolving over three time-points.

The field of graph embedding has received significant attention as a means of analysing large, complex graphs via the use of machine learning (discussed in greater depth in Chapter 4). Graph representation learning, comprises a set of techniques that learn latent representations of a graph, which can then be used as the input to machine learning models for downstream prediction tasks [87]. The majority of graph representation learning techniques have focused upon learning vertex embeddings [84] and reconstructing missing edges [87]. The goal of graph representation learning is to learn some function $f : V \rightarrow \mathbb{R}^d$ which maps from the set of vertices V to a set of embeddings of the vertices, where d is the required dimensionality. This results in f being a mapping from

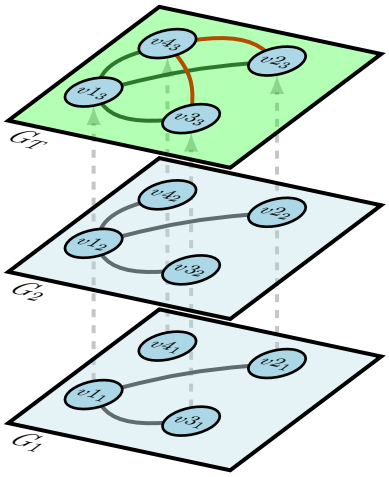


Figure 5.1: The temporal link prediction task is to predict the new edges (red) in the final graph snapshot G_T (green plane) given the previous graphs G_1 and G_2.

G to a representation matrix of dimensions $|V| \times d$, i.e. an embedding of size d for each vertex in the graph. However, the majority of graph representation learning approaches to date ignore the temporal aspect of dynamic graphs, resulting in models which perform poorly at predicting future change in a graph.

Additionally, there are many cases in the real-world where a machine learning model will be trained using historical data and then used for inference at a later point in time. A primary example of this is the recommender systems literature, where graphs can be used to model the relationship between users and items [24]. Here a model is trained to recommend items to users based on historically collected data, and then used to make predictions about newly arriving data. However, the underlying graph structure is dynamic and could undergo large changes between time points, leading to the existing model suffering from degraded predictive performance, forcing the model to be retrained.

The work presented in this chapter attempts to address some of these issues by creating graph processing models which explicitly incorporate temporal information. Two alternative model families are detailed in this chapter which incorporate temporal information using different and competing techniques, with trade-offs being made between model expressiveness and complexity. Specifically these two methods are:

- *Temporal Offset Reconstruction* - A new training procedure and associated model is first introduced, entitled Temporal Offset Reconstruction (TOR), which can be used to create vertex level representations which encode information about how the graph will evolve into the future, leading to a model which is more robust to temporal change. To achieve this, we introduce the temporal offset reconstruction method to create graph representations which are explicitly designed to predict the next time point for a dynamic graph. We show that this offset method results in vertex representations which perform better when used to make predictions on later time points. Further, we make use of graph convolutional neural networks [128], combined with our temporal offset reconstruction method to show that graph convolutions can be used to capture the dynamics of an evolving graph dataset. To the best of our knowledge, this is the first time this has been shown.
- *Temporal Neighbourhood Aggregation* - A more complex model is introduced, entitled Temporal Neighbourhood Aggregation (TNA), designed to learn vertex representations which capture both topological and temporal change by exploiting the rich information found in large dynamic graphs. To achieve this, we propose a novel model architecture combining graph convolutions with recurrent connections on the resulting vertex level representations to allow for powerful, hierarchical learning at multiple hops of a vertex's neighbourhoods. This approach means the model can explore at which neighbourhood depth the most useful temporal information can be learned. Further, we aggregate the temporal neighbourhood using tools from variational inference, resulting in a more robust and stable final representation for each vertex. Our TNA model is trained end to end on temporal graphs represented as time snapshots, where the objective is to directly and accurately predict the next graph in the sequence using the embeddings alone. This results in a model, which unlike many competing approaches, requires no explicitly parameterized decoder model.

5.1.1 Chapter Contributions

The primary contributions of this chapter are thus as follows:

- Exploration of two novel approaches for creating vertex level representations which contain temporal, in addition to structural, information about the vertex. Both approaches are unsupervised, requiring no additional labelling or features to be present at the vertex, edge or graph level.

- Presentation of evidence that Graph Convolution Networks can be used to capture temporal dynamics in graphs. This is achieved by using a novel training procedure where a model is taught to directly predict the future, rather than the current, state of the graph.
- Techniques are explored to incorporate elements of variational sampling, allowing for more robust temporal representations to be created. This consequently results in more accurate predictions of future graph states. It also allows for many synthetic temporal graphs to be generated given an input sequence.
- Producing models which are efficient and scalable, as they require significantly fewer parameters than competing approaches. This is partially achieved by the approaches requiring an explicitly parameterized decoder portion, leading to the models being scalable to larger graphs as a result of the memory efficiency.

To aid in reproducibility of the results presented in this chapter, all of the associated PyTorch [183] based source-code has been open-sourced and made available online. In addition, results are presented upon public benchmark datasets. The code for the Temporal Offset Reconstruction approach is available here - https://github.com/sbonner0/temporal-offset-reconstruction. Whilst the code for the Temporal Neighbourhood Aggregation approach is available here - https://github.com/sbonner0/temporal-neighbourhood-aggregation.

5.2 Related Works

5.2.1 Graph Representation Learning

As was discussed in greater detail in Chapter 4, traditionally graph representations were created via techniques based on matrix factorization, where a mapping to a lower dimensional space is found such that pair-wise relationships in the original graph are preserved. Examples of such approaches include Laplican eigenmaps [23], Graph Factorization [4], GraGrep [44] and HOPE [177]. More recently, models originally designed for Natural Language Processing (NLP) have been adapted to create graph embeddings. Such approaches use random walks to create 'sentences' which can be used as input to language inspired models such as Word2Vec [165]. NLP inspired graph embedding approaches include DeepWalk [186], Node2Vec [87] and Hyperbolic embeddings [48].

Recently graph specific neural network based models have been created which are inspired by Convolutional Neural Networks (CNNs) from the field of computer vision. Such approaches attempt to create a differential model for learning directly from graph structures. Many Graph CNN approaches operate in the spectral domain of the graph, using eigenvectors derived from the Laplacian matrix of a graph [128]. Early approaches to define convolution operators on graphs often had large memory and computation complexities and were thus unsuited to many real world graphs [42].

Later more efficient spectral methods were proposed which reduced the complexity of the filtering operations whilst still operating on the entire adjacency matrix [64, 128]. The Graph Convolutional Network (GCN) approach has proven to be particularly effective [128]. The GCN approaches uses a layer-wise propagation rule to aggregate information from a vertices 1-hop neighbourhood to create its representation. This layer-wise rule can be stacked k times to aggregate information from k-hops away from a given vertex. The requirement to have the whole adjacency matrix available in memory means that the GCN approach struggles to scale to massive graphs. To tackle this problem, Graph-Sage [93] learns to aggregate features from a fixed size *sample* of a vertices neighbourhood and as such can be applied to new vertices which have joined the graph. However, the approach mandates that all vertices in the graph have features available and the performance can vary depending upon the neighbourhood sampling strategy [49].

Thus far, all the graph specific models discussed have been supervised approaches, requiring the graphs to have labels. There have been a modest selection of unsupervised graph specific neural models, many of which are based on auto-encoders - a type of neural network whose task is to reconstruct the input data after being projected into a lower-dimension [17]. Structural Deep Network Embedding (SDNE) uses an auto-encoder to reconstruct each row in a graph's adjacency matrix [229]. A more recent auto-encoder employs a generative model to adversarially regularise the embeddings to help improve performance [243]. Work has also explored the use of GCNs as the basis of a convolutional auto-encoder model [127], producing state-of-the-art results for link-prediction in citation graphs.

5.2.2 Temporal Embeddings

All of the embedding approaches discussed so far have considered stationary non-evolving graphs. This section will review the literature regarding attempts to create temporal embeddings.

As many graph embedding techniques are taken from NLP, we briefly review the approaches here which consider the evolution of language, before looking at graph specific approaches.

Natural Language Temporal Embeddings

Some approaches from temporal language based models are created to model how word use evolves over time. For example, work has been performed to use cosine similarity to automatically measure how a word changes, relative to its neighbours, over time and to identify anomalies [125]. To overcome this non-convex problem, work has been performed in creating diachronic word embedding by aligning different embedding snapshots using orthogonal Procrustes, making the learning not end to end [91]. In later work, a dynamic Word2Vec model for word embedding is created which attempts to solve the non-convex problem common to embeddings via Bayesian variational black-box inference [18]. The approach creates embeddings which change smoothly over time and are better able to predict the change in context for a given word than previous methods.

Graph Specific Temporal Embeddings

To date, there have been few attempts to consider the temporal change of a graph when creating its embedding. However the existing approaches can broadly be split into two categories: Temporal Walk and Adjacency Matrix Factorisation based.

Temporal Walk-Based Approaches -

Many of the temporal graph embedding approaches which exist are based on data created via temporal walks, which are random walks over dynamic graphs. Perhaps the first such approach is that of STWalk [180]. In this work, the authors aim to learn *node trajectories* via the use of random walks which learn representations that consider all the previous time-steps of a temporal graph. In the best performing approach presented, the authors learn two representations for a given vertex simultaneously which are concatenated to create the final embedding. The first representation is a normal DeepWalk embedding designed to capture the spatial information for a vertex. The second representation is learned across a specially constructed graph structure, where each vertex is connected to its 1-hop neighbourhood from each previous time-step. However

the approach is not end-to-end and requires the user to manually chose how many time steps to consider.

Yu et al. [242], propose NetWalk, a vertex-level dynamic graph embedding model using random walks designed to facilitate anomaly detection in streaming graphs. The approach captures a collection of short random walks from the graph which are then passed into an auto-encoder based model to create the vertex representations. In addition to the usual reconstruction based loss term, an additional term is added to minimise the pair-wise distance between the representations of vertices occurring within the same walk. To apply this approach to the domain of streaming graphs, where changes to graph are being made online, the approach maintains for each vertex a list of neighbouring vertices which is updated as the graph changes. If changes in a vertices neighbour list occur, new random walks will be generated and the representations updated. The final anomaly detection is performed via a dynamic clustering model on the vertex representations. However, unlike the work presented in this chapter, the created embeddings are not capable of capturing temporal dynamics or are able to predict the future state of a graph.

Nguyen et al. [172], propose a model to incorporate temporal information when creating graph embeddings via random walks by capturing individual changes (edge addition/deletion for example) within a graph. The authors propose a temporal random walk to create the input data, however their approach creates more complex and rich temporal walks via a biasing process. The approach can be used to add temporal information into any embedding model which relies on random walks as input data, with the paper explicitly detailing a model based on the Skip-Gram architecture and shows the predictive performance increases over non-temporal baselines. The approaches presented in this chapter do not rely on the use of random walks and require only the raw graphs as input.

Adjacency Matrix Factorisation Approaches -

Goyal et al. [85], propose a model for creating dynamic graph embeddings, entitled DynGEM. In this approach they extend the auto-encoder graph embedding model of SDNE [229] to consider dynamic graphs. To do this, they use a method similar to Net2net [52], which is designed to transfer the learned knowledge from one neural network to a second model. This technique allows them to add more neurons to the auto-encoder, appropriate to the increasing graph size, via a heuristic approach entitled PropSize. The use of the Net2net technique means that the model can be expanded while ensuring the learned function is approximately preserved. The

process for training the model is as follows: at the fist time snapshot of the graph, a complete graph auto-encoder is trained as described in the SDNE paper [229]. For the next time step of the model, a new auto encoder is trained reusing the weights from the previous step, with any new neurons being added according to the PropSize heuristic. However the approach does not explicitly predict the future state of the graph, rather, it transfers knowledge from the previous timestamp to help the current auto-encoder better reconstruct the current time-step.

In a family of approaches entitled Dyngraph2vec*, comprising DynAE, DynRNN and DynAERNN, Goyal et al. [82] further extend an SDNE type approach to incorporate temporal information in a variety of ways. The best performing approach, DynAERNN, uses a combination of SDNE-like dense auto-encoders, with stacked recurrent layers to learn temporal information when creating vertex embeddings. However, they do not make use of graph convolutions and require a complex decoder model to predict the next graph.

There have been attempts to incorporate temporal aspects into GCNs. However, some [160, 208] focus upon supervised learning, but do not explicitly use the models to predict the future graph state or only have a single layer of recurrent connections. More recent approaches, such as GCN-GAN [144] and GC-LSTM [50] require large and complex decoder models, meaning they cannot scale to graphs of one-thousand vertices or more on current hardware, whilst also lacking the variational sampling of our approach. In comparison, EvolveGCN [182] uses recurrent layers to directly evolve the parameters of standard GCN layers which means it does not track vertex neighbourhood evolution explicitly.

One of the application areas most frequently learning temporal models on graphs is that of traffic modelling, where approaches like Spatial-Temporal Dynamic Network (STDN) [238] and Diffusion Convolutional Recurrent Neural Network (DCRNN) [155] combine graph learning with temporal models to predict traffic movement. However, unlike these approaches we focus on creating vertex level embeddings directly optimised to predict future edges and learn change at different hops of a vertices neighbourhood.

5.3 Methodologies

This section outlines the two approaches explored in this chapter, including the relevant background technologies, proposed model architectures and the training procedure. The two

Symbol	Definition
G	A graph with an associated set of vertices V and corresponding set of edges E.
$\mathbf{A}$	The adjacency matrix of graph G, a symmetric matrix of size $\|V\| \times \|V\|$, where $A_{i,j}$ is 1 if an edge is present and 0 otherwise.
$\hat{\mathbf{A}}$	$\mathbf{A}$ normalised by its degree matrix $\mathbf{D}$ and its identity matrix $\mathbf{I}$ such that $\hat{\mathbf{A}} = (\mathbf{D}^{-\frac{1}{2}}(\mathbf{A}+\mathbf{I})\mathbf{D}^{-\frac{1}{2}})$ [128].
$\mathbf{X}$	A matrix of features for each $v \in V$, set to the identity $\mathbf{I}$ of $\mathbf{A}$ for this work.
$\mathbf{H}$	The intermediate vertex representations in GCN and TNA layers.
$\mathbf{Z}$	The final variationally sampled representation matrix for each $v \in V$.
G'	A temporal graph comprised of snapshots $\{G_1, G_2, ..., G_T\}$.
T	The number of snapshots in G'.
G_t	A graph from G'.
σ_s	The sigmoid activation function.
σ_r	The rectified linear activation function (ReLU).
σ_{lr}	The leaky ReLU activation function.
l	A certain layer in the model.
$\mathbf{W}_g^{(l)}$	A weight matrix at layer l used in the GCN.
$\mathbf{W}_s^{(l)}$	A weight matrix at layer l used in the skip connection.
$\mathbf{W}_{\{r,u,h\}}^{(l)}$	Hidden transform matrices in the GRU.
$\mathbf{U}_{\{r,u,h\}}^{(l)}$	Input transform matrices in the GRU.
$\mathcal{N}(\mu, \sigma)$	A multi-dimensional Gaussian distribution parametrised by vectors μ and σ.
Θ	A trainable model containing a set of parameters.

Table 5.1: Definitions and Notations for Temporal Graph Learning

approaches are entitled Temporal Offset Reconstruction and Temporal Neighbourhood Aggregation. This section makes use of the notation detailed in Table 5.1, which lists the symbols used and an associated description.

5.3.1 Motivation

Many of the phenomena that are commonly represented via graph structures are known to evolve over time – Links between entities form and break in a constantly evolving stream of changes. We thus view graphs as a series of snapshots, with each graph snapshot containing the connections present at that particular moment in time. More formally, we can redefine a graph G to be a temporal graph $G' = \{G_1, G_2, ..., G_T\}$, where each graph snapshot $G_t \; \forall t \in [1, T]$ contains a corresponding vertex set V_t and edge set E_t.

In many real-world use cases of machine learning, a model is trained on historical data and then used to make predictions about new events at a future point in time. An example of where this practice is common is in the recommender systems industry where recent state-of-the-art systems, for recommending items to users, are based on graph convolutions [24, 240]. However, to date, the majority of models for creating graph representations do not consider how the graph evolves over time. This could potentially result in models which have good initial predictive capability, but whose performance will degrade as the graph continues to change over time.

Additionally, a common and vital task within the field of graph mining is that of future link prediction, where the goal is to accurately predict which vertices within a graph will form a connection in the future [83]. Figure 5.1 highlights this future link prediction task, where the goal is to predict the new edges, coloured in red, formed in G_T, given the previous graphs in the temporal history G_1 and G_2. Any model designed to accomplish this task must learn the evolution patterns present in edge formation, even though the number of edges changing at each time point is often a small fraction of the total number.

We propose to tackle this challenging problem of creating temporal robust graph embeddings by training a model to explicitly recreate a future time step of the graph. More concretely, a graph G_i is used as input to model $\theta(G_i)$ which learns a representation for each vertex in G_i such that its output can accurately predict the graph $G_{i+\delta}$. Ideally, we want to create a model $\theta(G_i)$ which can perform this temporal offset reconstruction using the graphs G_i and $G_{i+\delta}$ alone, $G_{i+\delta} = \theta(G_i)$, requiring no pre-processing steps which could affect the model's performance (e.g. random walk procedures), no pre-computed vertex features and no labels required or used.

The remainder of this section will detail the graph convolutions used to create the vertex representations, the models we explore to perform the temporal offset reconstruction and the training procedure.

5.3.2 Background Technologies

We first review the background technologies we are employing to make the presented approaches possible, namely Graph Convolutions [128] and Recurrent Neural Networks [53, 101].

Graph Convolutions

To perform the graph encoding required to create the initial vertex representations, we utilise the spectral Graph Convolution Networks (GCN) [128]. One can consider a GCN to be a differentiable function for aggregating information from the immediate neighbourhood of vertices [49, 93]. A GCN 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 $\mathbf{H} = GCN(\hat{\mathbf{A}}, \mathbf{X})$. $\mathbf{X}$ can be initialized with pre-computed vertex features, but it is sufficient to initialize it with one-hot feature vectors (in which case $\mathbf{X}$ is the identity matrix $\mathbf{I}$). A GCN can contain many layers which aggregate the data, where the operation performed at each layer by the GCN [128] is:

$$GCN^{(l)}(\mathbf{H}^{(l)}, \hat{\mathbf{A}}) = \sigma_r(\hat{\mathbf{A}}\mathbf{H}^{(l-1)}\mathbf{W}_g^{(l)}), \tag{5.1}$$

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

One can consider the GCN function to be aggregating 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.

The original methods presented in the literature required GCN based models to be trained via supervised learning, where the final vertex representation is tuned via provided labels for a specific task – classification as a common example [93, 128]. This is a key difference between GCNs and other graph embedding approaches, as these commonly require no labels and thus are applicable on a broader selection of graphs. Recently, extensions to the GCN framework have been made which allows for convolutional auto-encoders for graph datasets [127]. Auto-encoders are a type of un-supervised neural network model which attempt to compress input data to a low-dimensional space, and then reconstruct the original data directly from the learned representation.

Recurrent Neural Networks (RNN)

RNN are neural networks with circular dependencies between neurons. Activations of a recurrent layer are dependent on their own previous activations from a previous forward pass,

and therefore form a type of internal state that can store information across time steps. They are frequently used in sequence processing tasks where the response at one time step should depend in some way on previous observations. Long Short-Term Memory (LSTM) [101] and Gated Recurrent Units (GRU) [53] are RNNs with learned gating mechanisms, which mitigate the vanishing gradient problem when back-propagating errors over a sequence of inputs, allowing the model to learn longer-term dependencies. For this work, we employ the GRU cell, as it empirically offers similar performance to an LSTM, but with fewer overall parameters. The GRU computes the output $\mathbf{h}_t$, for the input vector $\mathbf{x}_t$ at time t in the following manner [53]:

$$\begin{aligned}
\mathbf{u}_t &= \sigma_s\Big(\mathbf{x}_t\mathbf{U}_u^{(l)} + \mathbf{h}_{t-1}\mathbf{W}_u^{(l)}\Big) \\
\mathbf{r}_t &= \sigma_s\Big(\mathbf{x}_t\mathbf{U}_r^{(l)} + \mathbf{h}_{t-1}\mathbf{W}_r^{(l)}\Big) \\
\tilde{\mathbf{h}}_t &= \tanh\Big(\mathbf{x}_t\mathbf{U}_h^{(l)} + (\mathbf{r}_t * \mathbf{h}_{t-1})\mathbf{W}_h^{(l)}\Big) \\
\mathbf{h}_t &= (1 - \mathbf{u}_t) \odot \mathbf{h}_{t-1} + \mathbf{u}_t \odot \tilde{\mathbf{h}}_t,
\end{aligned} \tag{5.2}$$

where $\odot$ is the Hadamard product, $\mathbf{r}$ and $\mathbf{u}$ are the rest and update gate values at time t, $\mathbf{U}^{(l)}$ and $\mathbf{W}^{(l)}$ are trainable parameter matrices at layer l and σ_s and tanh are the sigmoid and hyperbolic tangent activation functions.

5.4 Temporal Offset Reconstruction Model Overview

For creating our temporally offset graph embeddings, we will explore the use of both non-probabilistic and variational encoder models. These are related to the convolutional graph auto-encoders of Kipf [127]. However, we are exploring the creation of two models explicitly to reconstruct a future state of the graph, rather than just to capture the current graph. Below we detail the specifics of both the non-probabilistic Temporal Offset Graph Auto Encoder (TO-GAE) and Temporal Offset Graph Variational Auto Encoder (TO-GVAE) models used for temporal offset reconstruction. Owing to the similarities in sampling and objective function between this approach and the TNA model detailed in Section 5.5 and to avoid repetition, some equations for this approach are detailed in that section.

TO-GAE: TO-GAE is the non-probabilistic interpretation of the temporally offset graph auto-encoder concept, where the goal is to learn a low-dimensional representation of $\mathbf{A}_t$ from $\mathbf{G}_t$, via an encoding from a GCN $\mathbf{Z}_t = GCN(\mathbf{A}_t, \mathbf{X}_t)$, such that it can be used to predict

accurately the structure of some future time step δ of the graph via a product between Z_t and its transpose passed through a logistic sigmoid unit σ:

$$\mathbf{A}_{t+\delta} = \sigma(\mathbf{Z}_t \mathbf{Z}_t^{\mathsf{T}}). \tag{5.3}$$

For all the work presented in the chapter, the GCN model used to learn $\mathbf{Z}_t$ is a two layer model.

TO-GVAE: TO-GVAE is a variational interpretation of the temporally offset graph auto-encoder concept. Again the goal is to learn a vertex level representation for future graph reconstruction by using ideas from Bayesian inference [126]. This variational method differs from the non-probabilistic version outlined above as instead of directly learning the mapping Z, we instead learn a distribution from which Z is sampled. Using a variational approach to create the latent space has been shown to create more robust and meaningful embeddings, resulting in better performing models [126, 127].

As TO-GVAE is a Bayesian style model, we must define a model with which to perform inference. This again makes use of the GCN layers outlined in Section 5.3.2 to learn a mean μ and a variance γ vector used to parametrise the Gaussian distribution $\mathcal{N}$ from which Z_i is finally sampled, detailed in Equation 5.5.

Once the inference model has created the various required parameters, a generative model is created to predict the next time step in the graph. The generative model we use is again based on the inner-product between the latent representations, detailed in Equation 5.6.

To train the model, common for variational methods [126, 127], we directly optimise the lower bound $\mathcal{L}$ with regards to the model parameters, as detailed in Equation 5.7.

5.4.1 Model Parameters and Training Procedure

As with the original GAE approach [127], both TO-GAE and T0-GVAE make use of two layers of convolution, with the first layer comprising 32 filters, and the second having 16 filters. Results of grid-searches over possible parameter choices is presented in Section 5.7.1. For training

both the models we use full-batch gradient descent via the RMSProp algorithm with a learning rate of 0.001 for a total of 50 epochs. We also found the use of an additional term in the loss functions which penalises the model parameters for getting too large via L2 to help model performance. All of our models, as well as the comparative baselines, have been implemented in the PyTorch library [183].

5.5 Temporal Neighbourhood Aggregation Model Overview

We first detail the Temporal Neighbourhood Aggregation blocks which form the primary learning component, before describing the overall model topology and objective function.

5.5.1 TNA Block

One of the primary components of our model is the TNA block for topological and temporal learning from graphs. The overall structure of the block is illustrated in Figure 5.2. It is important to note that all the parameters in the block are shared through time. This allows complex temporal patterns to be learned, as well as allowing for a large reduction in the total number of parameters required by the model. Assuming that the TNA block is the first layer in the model, the flow for vertex $v \in V_t$ can be described as follows:

- The input is passed through the GCN layer, as detailed in Equation 5.1, which will learn to aggregate information for v from its one-hop neighbourhood to create its representation at this point in the block - $\mathbf{h}_t^{GCN}$. This is then normalised using Layer Norm [15], which will ensure that the representation for each vertex is of a similar scale. This has been shown to improve the training stability and convergence rate of deep models [15].
- This normalised representation is then passed into a GRU cell a row at a time, as detailed in Equation 5.2, where the output of the cell will be a function of the current input as well as all the previous inputs. This means that the cell can learn how much of the previous neighbourhood representation to use when creating the new representation for a given vertex $\mathbf{h}_t^{GRU}$. This is then passed through a second Layer Norm unit to ensure a normalised output.

- Finally, the $\mathbf{h}_t^{GCN}$ and $\mathbf{h}_t^{GRU}$ representations are concatenated together, before being passed through a linear layer and a leaky ReLU activation function to create the final representation for the vertex $\mathbf{h}_t^{TNA}$. Inspired by residual connections often used in computer vision networks [97], this enables the model to learn the optimum mix of topological and temporal information.

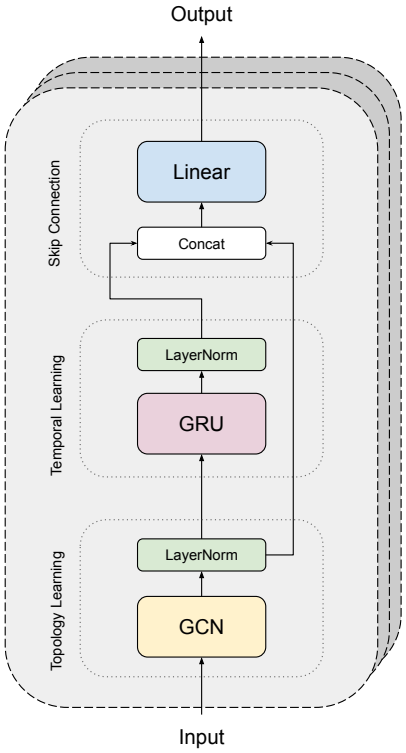


Figure 5.2: An overview of the Temporal Neighbourhood Aggregation (TNA) block, which comprises a Graph Convolutional Network (GCN) layer with a Gated Recurrent Unit (GRU). The combination of the topological and temporal learning is controlled via the final linear layer.

The layer-wise propagation rule of the TNA block at depth l can thus be summarised as follows for the entire graph $G_t \in G'$ with normalised adjacency matrix $\hat{\mathbf{A}}$:

$$
\begin{aligned}
\mathbf{H}_t^{GCN} &= GCN(\hat{\mathbf{A}}, \mathbf{H}_t^{(l-1)}) \\
\mathbf{H}_t^{GRU} &= GRU(\mathbf{H}_t^{GCN}, \mathbf{H}_{t-1}^{GRU}) \\
\mathbf{H}_t^{TNA^{(l)}} &= \sigma_{lr}\Big(\mathbf{W}_s^{(l)}\text{Concat}(\mathbf{H}_t^{GCN}, \mathbf{H}_t^{GRU})\Big) \\
TNA(\hat{\mathbf{A}}, \mathbf{H}_t^{(l)}) &= \mathbf{H}_t^{(l)} = \mathbf{H}_t^{TNA^{(l)}}
\end{aligned}
\tag{5.4}
$$

where $\mathbf{W}_s^{(l)}$ represents the weight matrix used to mix the topological and temporal representations, and σ_{lr} is the leaky ReLU activation function with a negative slope of 0.01.

5.5.2 Overall Model Architecture

As with normal GCN layers, TNA blocks can be stacked to aggregate information from greater depth within a graph, with each additional block adding one extra hop from which information can be aggregated for a certain vertex. However, as our TNA blocks are recurrent, information can also be aggregated from how connectivity within these hops has evolved over time, instead of just their present state. After extensive ablation studies (detailed in Section 5.8.1), we use the final configuration of the model detailed in Figure 5.3. Our model contains two stacked TNA blocks, to learn information from two hops within the temporal neighbourhood. This is then passed to two independent GCN layers which perform a final aggregation of this temporal representation. From these two layers, the final representation matrix Z_t is sampled using techniques from variational inference, specifically the reparametrisation trick [126].

Variational Sampling - To create the final representation matrix $\mathbf{Z}_t \in \mathbb{R}^{|V_t| \times d}$, the output from the two GCN layers GCN_μ and GCN_σ are used to parametrise a unit Gaussian distribution $\mathcal{N}$, from which $\mathbf{Z}_t$ is then sampled, rather than being explicitly drawn. This is the same concept used in Variational Auto-Encoders [126], and has previously been demonstrated to work well for creating more robust and meaningful vertex level representations [31, 127]. Our inference model used to create the vertex representations of graph G_t, with adjacency matrix $\mathbf{A}_t$ and identity matrix of $\mathbf{A}_t$, $\mathbf{X}_t$, can thus be described as :

$$
\text{q}(\mathbf{Z}_t|\mathbf{X}_t, \mathbf{A}_t) = \prod_{v=1}^{|V_t|} \mathcal{N}(\text{z}_v|GCN\mu_v,\, diag(GCN\sigma_v^2)),
\tag{5.5}
$$

where q is our approximation of the true and intractable distribution we are interested in capturing – $\mathrm{p}(\mathbf{A}_{t+1}|\mathbf{Z}_t)$. Here, both GCN_μ and GCN_σ take input from two stacked TNA layers as detailed in Figure 5.3.

Generative Model - To decode the information contained within $\mathbf{Z}_t$, a generative model is created to explicitly predict the new edges appearing in the next graph in the sequence. Here, the inner-product between the latent representation is used to directly predict $\mathbf{A}_{t+1}$:

$$\mathrm{p}(\mathbf{A}_{t+1}|\mathbf{Z}_t) = \prod_{i=1}^{|V|}\prod_{j=1}^{|V|} \mathrm{p}(\mathrm{A}_{t+1_{i,j}}|\sigma_s(\mathrm{z}_i\mathrm{z}_j^\mathsf{T})), \tag{5.6}$$

where $\mathrm{A}_{t+1_{i,j}}$ represents elements from $\mathbf{A}_{t+1}$ and z refers to the rows of each vertex taken from $\mathbf{Z}_t$.

This generative model is one of the key advantages of our approach, as it means that we have zero learnable parameters in the decoder portion of the model. This is in contrast to many competing approaches, which often require as many parameters as in the encoder to create a decoder with the desired functionality [82]. This results in our approach being able to scale to significantly larger graphs, with longer histories than some of the competing approaches, whilst also being less prone to over-fitting to non-changing edges.

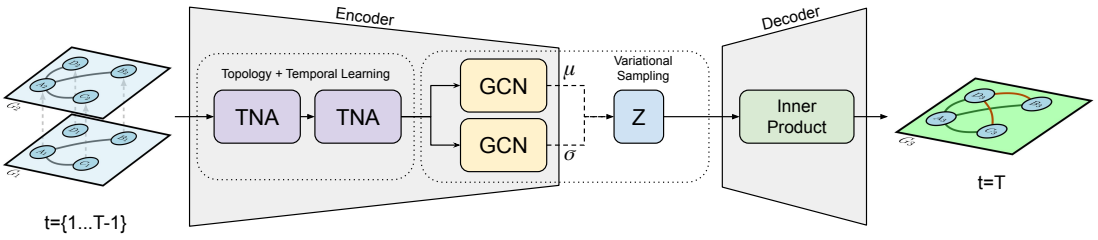


Figure 5.3: The overall Temporal Neighbourhood Aggregation Model: two stacked TNA blocks learning both topological and temporal information from the first and second hop neighbourhoods of a vertex. An embedding $\mathbf{z}_t$ is sampled for each vertex $v_t \in V_t$ using variational inference. The inner product is then used to directly predict the next graph in the sequence.

5.5.3 Objective Function

To train the TNA model, and as is common for variational methods [126, 127], we directly optimise the lower bound $\mathcal{L}$ with regards to the model parameters:

$$\mathcal{L} = \mathbb{E}_{\mathrm{q}(\mathbf{Z}_t|\mathbf{X}_t,\mathbf{A}_t)}\Big[\log\ \mathrm{p}(\mathbf{A}_{t+1}|\mathbf{Z}_t)\Big] - KL(\mathrm{q}(\mathbf{Z}_t|\mathbf{A}_t,\mathbf{X}_t)||\mathrm{p}(\mathbf{Z}_t)), \tag{5.7}$$

where $KL()$ is the Kullback-Leibler distance between p and q. We use a Gaussian prior as the distribution for $\mathrm{p}(\mathbf{Z}_t)$.

In addition, we apply L_2 regularization to the model parameters Θ to help with over-fitting, which is defined as:

$$\mathcal{L}_{reg} = \lambda \sum_{i=1}^{|\Theta|} \Theta_i^2, \tag{5.8}$$

where λ is a scaling factor, set to 10^{-5} for this work as used for other GCN-based approaches [128]. Consequently, the final objective function for our model is:

$$\mathcal{L}_{final} = \mathcal{L} + \mathcal{L}_{reg}. \tag{5.9}$$

5.5.4 Model Parameters and Training Procedure

After initial grid-searches, we empirically found two layers of Temporal Neighbourhood Aggregation, followed by variational sampling, to yield the optimal performance, and the first layer comprising 32 filters, whilst the second having 16 filters. For training the model, we empirically found using full-batch gradient descent with the RMSProp algorithm, a learning rate of 0.001 and 200 epochs to give the best results. Our model has been implemented in PyTorch [184].

5.6 Experimental Setup

This section will detail the setup for the experimental evaluation used to assess the performance of the two approaches explored in this chapter.

5.6.1 Temporal Offset Reconstruction: Evaluation Overview

As the primary goal of our approach is to create vertex representations which are better able to encode information about how the graph will evolve into the future, we will be using link prediction as our primary method of assessment. Formally the task of link prediction in the context of machine learning can be defined as follows: given a subset of edges $E_{train} \subset E$ from graph $G = (V, E)$, learn a model which can accurately predict the remaining edges $E_{test} = E - E_{train}$ [161]. Many recent methods attempt to solve this problem via vertex embedding similarity – i.e. vertices with more similar embeddings, according to some metric, are more likely to be connected via an edge [87, 127, 186].

During the evaluation, we will be investigating two ways in which a model trained on temporal graph data could be used for inference:

- *Evolution Pattern Prediction* – The original input graph G_0 is kept constant, whilst the rest of the time series $G_1, ..., G_t$ is used as the targets for prediction to measure the model's ability to predict the future graph changes.
- *Future Link Prediction* – The trained model is kept constant whilst each graph snapshot G_i is passed in. The model is then evaluated by making predictions on hold out edges not seen during training (E_{test}) from the same snapshot to test how well the model can predict edges in future snapshots of the graph.

Graph edges are predicted as follows: given the learned vertex embeddings, the adjacency matrix is reconstructed via a dot-product of the embedding matrix $A' = \sigma(ZZ^{\mathsf{T}})$. This reconstructed adjacency matrix is compared with the true graph to assess how well the embedding is able to reconstruct the future graph. For the future link prediction task, edges from the graph are randomly removed before the graph is fed into the model. The embeddings are then used to predict the hold-out set of edges.

5.6.2 Temporal Neighbourhood Aggregation: Evaluation Overview

As the primary goal of this approach is to create vertex representations which are better at encoding temporal change, we will be using the task of future link prediction as our primary

objective. More formally, we are trying to maximise the probability of $P(G_t|G_1...G_{t-1})$. In the context of machine learning, this can be defined as training a model from a temporal G' using $G_1...G_{t-1}$ such that it can predict the new edges in G_t, $E_t \setminus E_{t-1}$. The full training and evaluation process is detailed a stage at a time in Algorithm 5.1.

Similarly to the approach detailed in Section 5.6.1, graph edges are predicted as follows: given the learned vertex embeddings, the future adjacency matrix is reconstructed via the dot product of the embedding matrix $A'_{t+1} = \sigma(Z_t Z_t^{\mathsf{T}})$. This reconstructed adjacency matrix is compared with the true graph to assess how well the embedding is able to reconstruct the future graph.

Algorithm 5.1: New edge predicition procedure

Input : The temporal graph $G' = \{G_1, G_2, ..., G_T\}$
Output: Mean AUC and AP scores for predicting new edges for each graph in G'
for *all* $G_t \in G'$ *where* $t \geq 3$ **do**
  Load and pre-process the graphs $G_1, G_2, ..., G_T$
  Create new model Θ_i (as shown in Figure 5.3)
  Train Θ_i on sequence $G_1, G_2, ..., G_{t-1}$, where each graph is the input and used to predict the following one
  Predict new edges in G_t using $\Theta_i(G_{t-1})$: $E_t \setminus E_{t-1}$
  Store AUC and AP values
end
return *Mean AUC and AP values over* G'

5.6.3 Temporal Offset Reconstruction: Datasets

We make use of the following empirical graph datasets detailed in Table 5.2 when performing our experimental evaluation. The table details the dataset name, number of vertices and edges, and if the graph contains empirical of synthetic temporal information. For the two datasets which contain no inherent temporal information, we generate a synthetic evolutionary trajectory for the graph using one of the rewire processes. For the empirical cit-HepPh dataset, we create six snapshots based on a linear partitioning of the graph's timeline.

Random Rewire Process

In order to have access to large volumes of temporal graph data which has been evolved via a controllable process, we make use of the random rewire methodology detailed in Chapter 2. The

Dataset	$\|V\|$	$\|E\|$	Temporal	Reference
cit-HepPh	34,546	421,578	Snapshots	[146]
cora	2,708	5,429	Synthetic	[127]
citeseer	3,327	3,327	Synthetic	[127]

Table 5.2: Empirical graph datasets used to evaluate the temporal offset reconstruction approach.

rewire process alters a given source graph's degree distribution by randomly altering the source and target of a set number of edges. During this rewire process, it is not guaranteed that the source or target of the edge will be altered, indeed it is not always possible due to the graphs topology. Also, the rewiring process does not change the total number of edges or vertices within the graph. We employ two types of random rewire in this work:

- *Erdős* - The edges are rewired such that the resulting topology of the graph begins to resemble a Erdős-Rényi graph, where edges are uniformly distributed between vertices.
- *Configuration* - The edges are rewired in such a way that each vertex approximately preserves its associated number of edges, creating graphs with a similar degree distribution to the original.

5.6.4 Temporal Neighbourhood Aggregation: Datasets

When performing our experimental evaluation, we employ the empirical datasets detailed in Table 5.3. The table presents the dataset name, number of vertices and edges, the arrival date of the first and last edge, the number of temporal snapshots and the mean number of new edges added in each snapshot. The graphs used represent a range of application domains, sizes and temporal complexities.

Bitcoin-Alpha (Bitcoina) - Representing a trust network within a platform entitled Bitcoin Alpha, where edges are formed as users interact and rate each others' reputation. The graph covers a range of edges formed between 8th October 2010 and 22nd January 2016, which we partition into 62 monthly snapshots. The task of new edge prediction is thus analogous to predicting whether two users are going to interact within the next month.

Dataset	$\|V\|$	$\|E\|$	First Edge	Last Edge	# Snapshots	# New Edges	Reference
Bitcoin-Alpha (Bitcoina)	3,783	24,186	08/09/2010	22/01/2016	62	227	[146]
Wiki-Vote (Wiki)	7,115	103,689	28/02/2005	06/01/2008	34	2963	[146]
UC Irvine Messages (UCI)	1,899	20,296	15/04/2004	25/08/2004	27	513	[138]

Table 5.3: Empirical graph datasets used to assess the performance of the Temporal Neighbourhood Aggregation approach, where # New Edges is the average number of new edges added between time points.

Wiki-Vote (Wiki) - Representing a vote of escalating user privileges between users and administrators on the Wikipedia website. The graph covers a range of edges formed between 28th March 2004 and 6th January 2008, which we partition into 34 monthly snapshots. The task of new edge prediction within this data is analogous to predicting whether two users are going to vote for each other within the next week.

UCI-Messages (UCI) - Representing private messages sent between users on the University of California Irvine social network platform. The graph covers a range of edges formed between 15th April 2004 and 25th October 2004, which we partition into 27 weekly snapshots. The task of new edge prediction would represent the likelihood that two users will exchange messages with each other over the next week.

Synthetic Datasets

In addition, we use two synthetic datasets: a Stochastic Block Model (SBM) graph and a randomly perturbed version of the Cora dataset (R-Cora).

SBM - A random graph of 3,000 vertices, which evolves over 30 time points using the SBM algorithm [119]. The graph contains 3 communities and at each time point, 20 vertices will evolve by switching from one community to another.

R-Cora - To create this synthetic dataset, we take the original Cora dataset representing a citation network, and perturb the graph using the random rewire method (more details can be found about this in Chapter 2). The rewiring process alters a given source graph's degree distribution by randomly altering the source and target of a set number of edges. During this rewiring process, it is not guaranteed that the source or target of the edge will be altered, which

indeed is not always possible due to the topology of the graph. Also, the rewiring process does not change the total number of edges or vertices within the graph. We employ *Erdős* rewiring, i.e. the resulting topology of the graph begins to resemble a Erdős-Rényi graph, where the edges are uniformly distributed between vertices. Due to the random nature of the topological structure after this process, the predictive performance of the models should be lower.

5.6.5 Baseline Approaches

We compare against a variety of state-of-the-art graph representation learning techniques, both static and dynamic. We choose the baselines which compare most directly with our proposed approaches, meaning we opt for comparators which take advantage of deep neural networks to create vertex embeddings.

- *GAE*[127]: A non-probabilistic Graph Convolutional Auto-encoder (GAE), where the model is trained on G_{t-1} and then directly predicts new edges in G_t.
- *GVAE*[127]: A Graph Variational Convolutional Auto-encoder (GVAE), trained in the same manner as the GAE.
- *DynAE*[82]: A non-convolutional graph embedding model, similar to SDNE [229], extended to temporal graphs by concatenating the rows of the past graphs together before being passed into the model.
- *DynRNN*[82]: A non-convolutional graph embedding model, where stacked LSTM units are used to encode the temporal graph directly. The approach also requires a decoder model, also comprising stacked LSTM units, to reconstruct the next graph from the embedding.
- *DynAERNN*[82][1]: A combination of the previous two models, where a dense auto-encoder is used to learn a compressed representation which is passed to stacked LSTM units for temporal learning. It requires a large decoder, with both dense and LSTM layers, to predict the next graph. The E-LSTM-D approach [51] is also extremely similar to this model.

[1] For the Dyn* family of algorithms, we use the implementations as provided by the authors as part of their DynamicGEM package [83].

- *D-GCN:*[160, 208]: A dynamic GCN, similar to approaches proposed in [160] and [208]. Here, three stacked GCN layers are used to capture structural information with an LSTM unit used to learn temporal information and produce the final embeddings. To directly predict the next graph, we use an inner-product decoder on the embedding matrix.

Additionally, attempts were made to compare with GCN-GAN [144] and GC-LSTM [50], but we were unable to get them to scale to the size of graphs used for the experimentation.

5.6.6 Performance Metrics

As one can consider the task of link prediction to be that of a binary classification problem (an edge can only be present or not), we make use of two standard binary classification metrics to assess the performance of both approaches:

- *Area Under the Receiver Operating Characteristic Curve (AUC)* – The ratio between the True Positive Rate (TPR) and False Positive Rate (FPR) measured at various classification thresholds.
- *Mean Average Precision (AP)* – Across the set of test edges: $AP = \frac{TP}{TP+FP}$, where TP denotes the number of true positives the model predicts, and FP denotes the number of false positives.

For both of the chosen metrics, a larger value indicates more correctly predicted edges.

5.6.7 Experimental Environment

Experimentation was performed on a system with 2 * NVIDIA Titan Xp GPUs, 2.3GHz Intel Xeon E5-2650 v3, 64GB RAM, with Ubuntu Server 18.04 LTS, Python 3.7, CUDA 10.1, CuDNN v7.4 and PyTorch 1.1.

5.7 Temporal Offset Reconstruction: Results

This section will present the results of the experimental evaluation for the Temporal Offset Reconstruction approach. As outlined in Section 5.6, we are testing two ways to evaluate the models: evolution pattern prediction and future link prediction. We present results on the datasets introduced in Section 5.6.3, which contain both simulated and empirical evolving graph datasets. All the results presented are the mean, with the standard deviation, of ten repeats of the evaluation procedure, using a random train/test split for each repetition. When these results are presented as a figure, the mean value is represented as a point, with a standard deviation being represented as a shaded area of the same colour. For all figures and tables in this section, the TO-GAE and TO-GVAE approaches being proposed in this chapter are labelled beginning with the prefix *TO_*.

5.7.1 Parameter Selection

Before presenting the main results, we first detail a grid-search performed over possible parameters for the TO-GVAE model. Table 5.4 highlights results demonstrating how the model performance on the test set changes as the sizes of the first and second layers are altered when training on the Bitcoina and UCI datasets. To allow for fair comparison when altering the size of the first layer, the second was kept constant at 16 units. When altering the size of the second layer, the first was kept constant at a size of 32. One interesting aspect to note from the table is how close the model performance is at different layer sizes, indicating that after a certain size, there are limited performance gains to be made.

Table 5.5 demonstrates how the performance of the TO-GVAE model changes over three different optimisation algorithms. Each optimiser was tested with the same model configuration of 32 units in the first and 16 in the second layer. The table highlights that RMSProp demonstrates the best performance over the two different datasets, with ADAM being a relatively close second. SGD performs significantly worse than either, highlighting that GCN layers perform better with more complex optimisation strategies.

		Bitcoina		UCI	
Layer	**Filter Size**	**AUC**	**AP**	**AUC**	**AP**
First	4	0.662	0.661	0.608	0.591
	8	0.713	0.729	0.747	0.760
	16	0.818	0.847	0.754	0.756
	32	0.872	0.896	0.816	0.834
	64	0.891	0.907	0.794	0.811
Second	4	0.511	0.546	0.696	0.708
	8	0.691	0.706	0.766	0.771
	16	0.872	0.896	0.816	0.834
	32	0.908	0.924	0.842	0.857
	64	0.914	0.928	0.848	0.862

Table 5.4: Response in test set performance on the Bitcoina and UCI datasets as the layer sizes are altered.

	Bitcoina		UCI	
Optimiser	**AUC**	**AP**	**AUC**	**AP**
SGD	0.501	0.495	0.504	0.502
ADAM	0.872	0.896	0.842	0.857
RMSProp	0.926	0.937	0.863	0.877

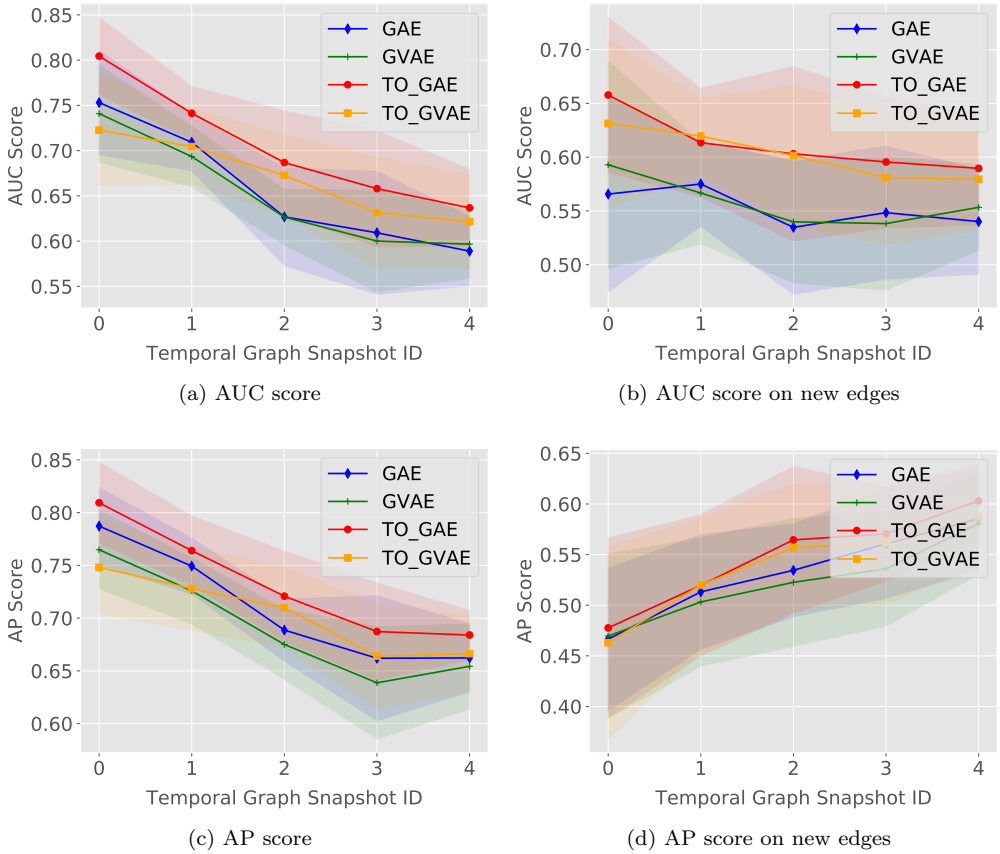
Table 5.5: Response in test set performance on the Bitcoina and UCI datasets as the optimiser is altered.

5.7.2 Simulated Graph Evolution

We first present results using time series simulated via the random rewire processes introduced in Section 5.6.3 on both the cora and citeseer datasets. For this experiment, we train using the original unaltered graph to reconstruct the next graph in the time series. We then measure the performance of the resulting embeddings at predicting edges in future graph snapshots using the two inference approaches outlined in Section 5.6.1. The points in the resulting figures are presented as the mean of the cross-validation, with the coloured areas highlighting the standard deviation.

Evolution Pattern Prediction

Figure 5.4 shows how well the models perform upon the Cora dataset with smaller possible perturbations introduced by the rewire process. The figure shows how both the temporally offset methods we introduce generally demonstrate superior performance over the baseline approaches. When considering the AUC score on new edges, we can see a large increase in performance. Our methods also show performance above the baselines when reconstructing the full graph, demonstrating that the temporal offset process does not harm the ability to predict edges which have not changed.



(a) AUC score

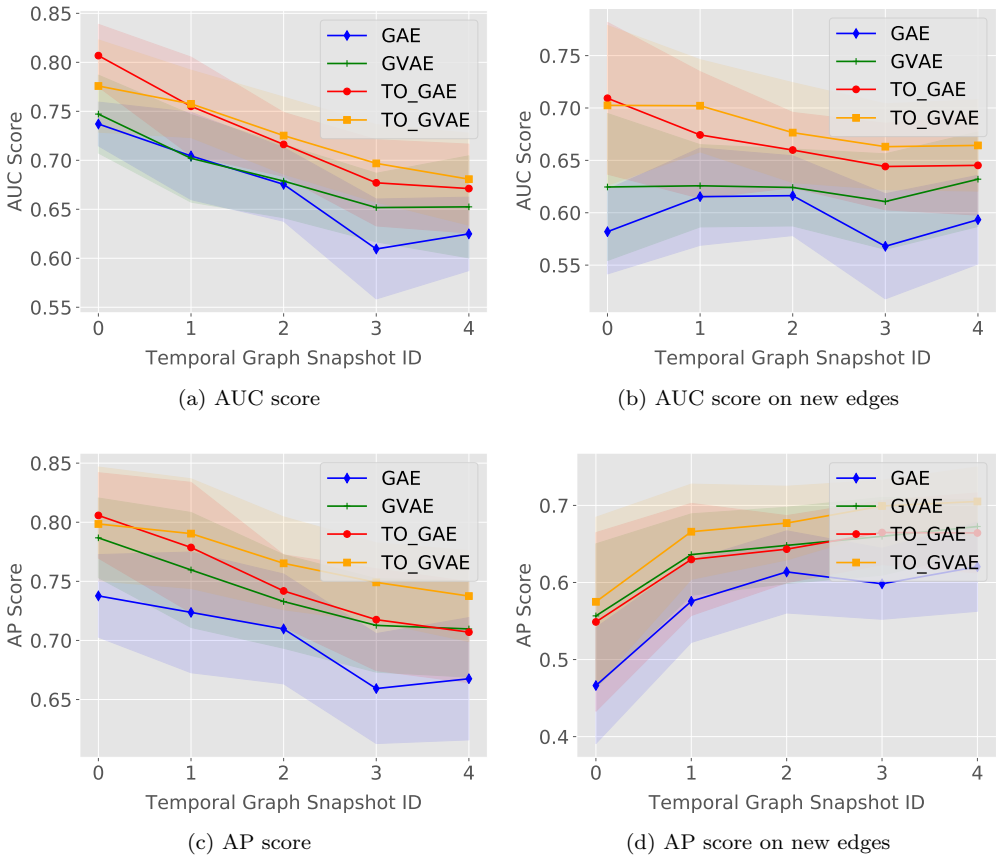
(b) AUC score on new edges

(c) AP score

(d) AP score on new edges

Figure 5.4: AUC and AP scores on the Cora dataset evolved via the configuration method with a 25% chance of edges being rewired per time step. Values are presented for the whole graph and only on new edges which have been altered since the graph used for training.

Figure 5.5 contains the results for the Citeseer dataset, again using the smaller rewire probability. This figure continues the trends established in the previous figures, with the temporally offset methods beating the baseline methods. However, this time it is the variational approach which often demonstrates the greater performance, especially when only new edges are considered.



(a) AUC score

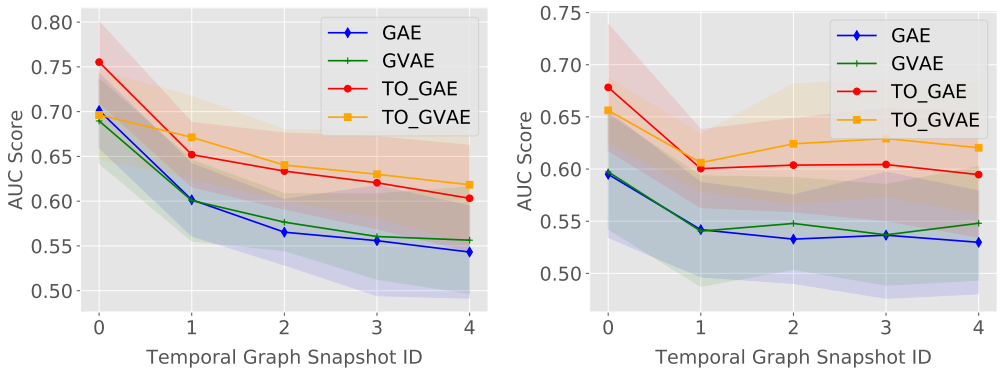
(b) AUC score on new edges

(c) AP score

(d) AP score on new edges

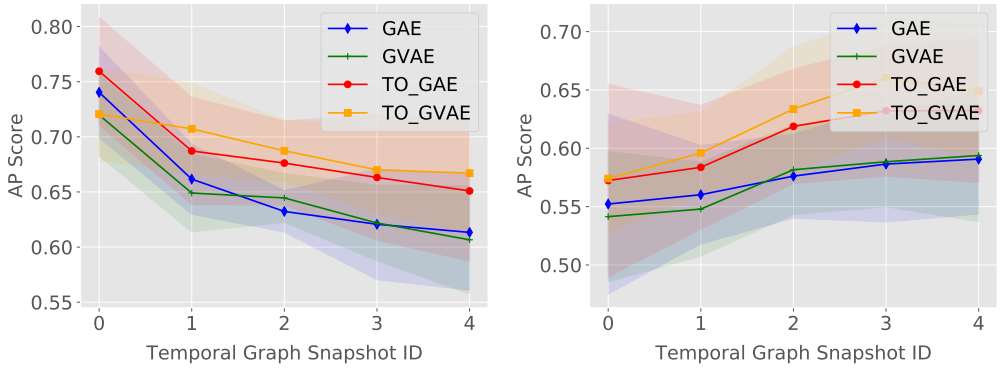
Figure 5.5: AUC and AP scores on the Citeseer dataset evolved via the configuration method with a 25% chance of edges being rewired per time step. Values are presented for the whole graph and only on new edges which have been altered since the graph used for training.

The next two sets of figures show results for when a large possible perturbation is made in-between each graph time step. Figure 5.6 demonstrates that, as was expected, when even large steps are made between graphs, the gap between our approaches and the baselines also increases. The temporally offset methods show a clear increase in performance, even when using the model to make predictions about graphs later in the time series, which will have a quite different topological structure to the graph used to train the model.



(a) AUC score

(b) AUC score on new edges

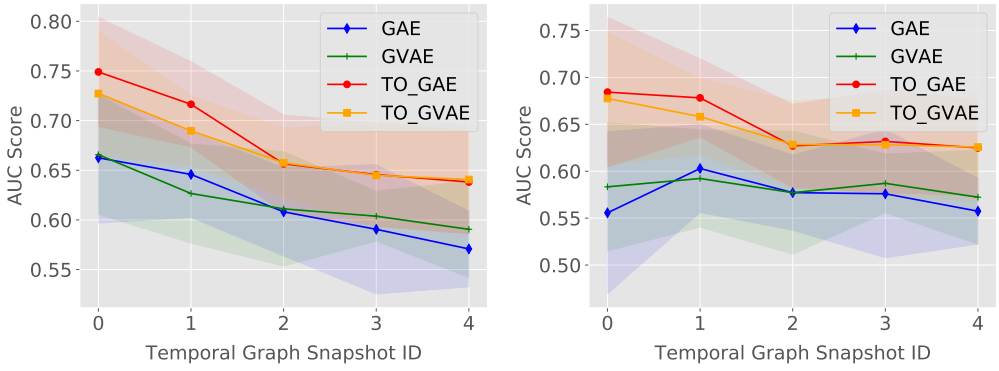


(c) AP score

(d) AP score on new edges

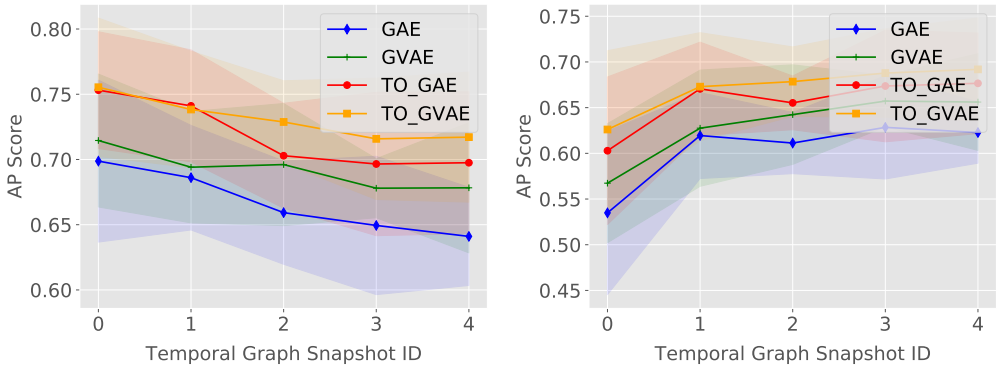
Figure 5.6: AUC and AP scores on the Cora dataset evolved via the configuration method with a 50% chance of edges being rewired per time step. Values are presented for the whole graph and only on new edges which have been altered since the graph used for training.

Figure 5.7 shows the results for the Citeseer dataset using the higher level of possible perturbation. The figure continues the trend of the previous results by showing the temporally offset models to be better at predicting future changes in the graph, even when considering both the unchanged and new edges.



(a) AUC score

(b) AUC score on new edges



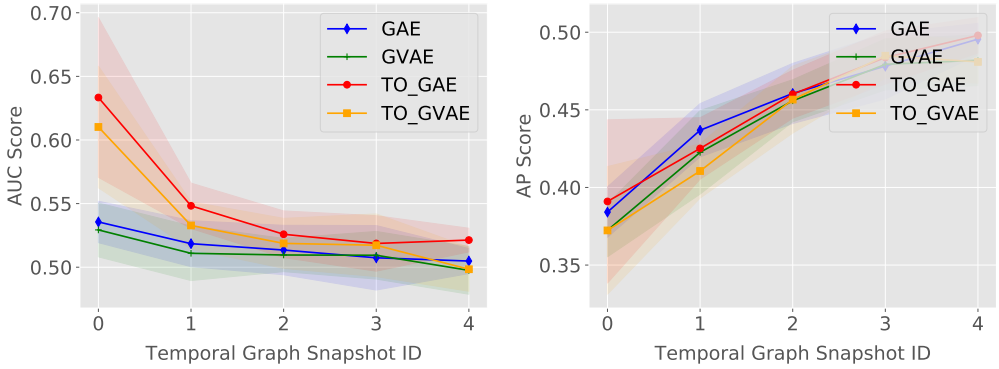
(c) AP score

(d) AP score on new edges

Figure 5.7: AUC and AP scores on the Citeseer dataset evolved via the configuration method with a 50% chance of edges being rewired per time step. Values are presented for the whole graph and only on new edges which have been altered since the graph used for training.

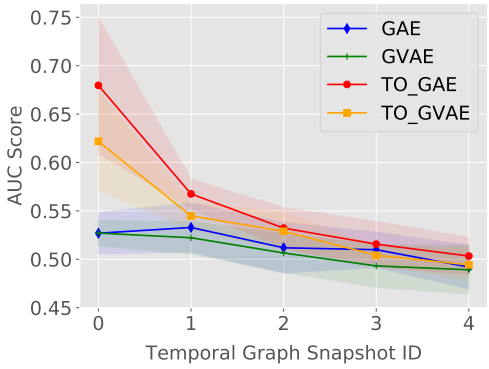
Future Link Prediction

For assessing the ability of the various models to continue to make accurate link predictions as the graph undergoes heavy topological change, we make use of the Erdős-based random rewire method. Figure 5.8 shows the results for only new edges on both the cora and citeseer datasets when 50% of the edges have the chance of being rewired in-between each graph snapshot. The figure shows that our temporally offset training method is more robust to the Erdős rewired edges, as it displays a higher level of predictive performance, particularly when regarding the AUC metric. However the performance of all approaches deteriorates to random chance as the graphs topology becomes increasingly random. Interestingly, of the two temporally offset models, it is the non-probabilistic approach which displays greater performance across both datasets.

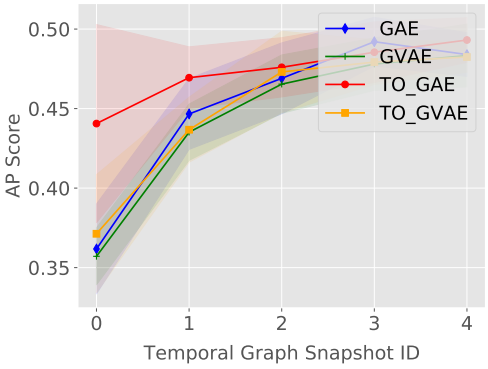


(a) AUC score on Cora

(b) AP score on Cora



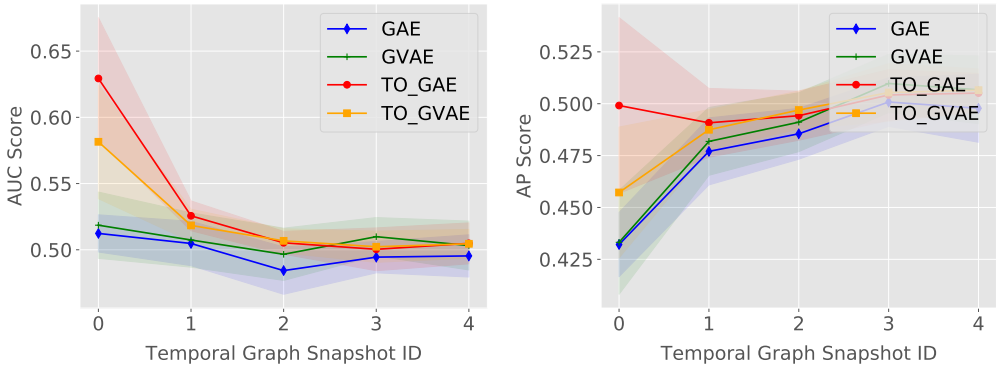
(c) AUC score on Citeseer



(d) AP score on Citeseer

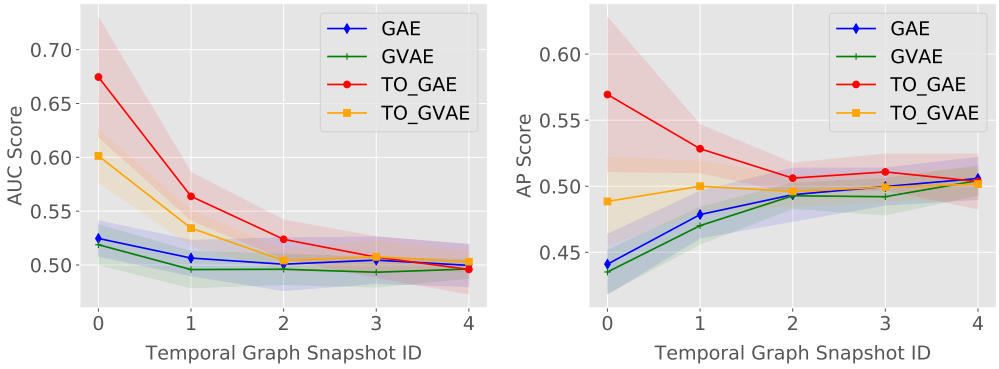
Figure 5.8: AUC and AP sscoes for the future link prediction task on both the Cora and Citeseer datasets evolved using the Erdős rewire method with $|E|/2$ edges having the chance of being rewired. The results presented are scores for predicting only new edges which have appeared after the original graph used for training the model.

Figure 5.9 highlights the performance on new edges when all the edges have the chance to be rewired between graph snapshots. The results show that the temporal offset approaches are generally more robust to the large change in graph topology between graph snapshots, with both demonstrating greater performance, especially at earlier points in the time series. Again it can be seen that all approaches tend towards a level of performance that could be achieved by random choice as the graphs themselves become increasingly random. Continuing the trend established in the previous experiment, the non-probabilistic temporally offset model out performs the variational approach. We hypothesise that as the variational approach is a more complex model, it is over-fitting more strongly to the non-rewired original graph edges, making it less able to learn the Erdős pattern of rewiring.



(a) AUC score on Cora

(b) AP score on Cora



(c) AUC score on Citeseer

(d) AP score on Citeseer

Figure 5.9: AUC and AP scores for the future link prediction task on both the Cora and Citeseer datasets evolved using the Erdős rewire method with the complete set of E having the chance of being rewired. The results presented are scores for predicting only new edges which have appeared after the original graph used for training the model.

5.7.3 Empirical Time-Series

In this section, the performance of the approaches when running on empirical temporal graph data will be assessed.

Evolution Pattern Prediction

Table 5.6 displays the results for the task of evolution pattern prediction for all models for the cit-HepPh dataset. The table shows that both the temporally offset methods significantly outperform the baseline approaches on both whole graph and new edges metrics at this particular task. The gap in performance between the temporally offset and normal approaches for this empirical dataset is larger than on the previous synthetic results, indicating that our approach is much better able to learn the temporal dynamics of real datasets. We can also see that the variational temporally offset approach is often the best performing of the two approaches, particularly at later time-points.

Model	**Metric**	G_1	G_2	G_3	G_4	G_5
GVAE	AUC	0.699(±0.0133)	0.6327(±0.0036)	0.5913(±0.0022)	0.5821(±0.0049)	0.5771(±0.0122)
	AP	0.8023(±0.0089)	0.7459(±0.0056)	0.7036(±0.001)	0.6853(±0.0035)	0.685(±0.0095)
	NE-AUC	0.5358(±0.0103)	0.513(±0.0036)	0.5012(±0.0098)	0.5034(±0.0067)	0.4979(±0.0112)
	NE-AP	0.5875(±0.0071)	0.5698(±0.0055)	0.5663(±0.0067)	0.5598(±0.0079)	0.552(±0.0111)
GAE	AUC	0.653(±0.0159)	0.5939(±0.0062)	0.5546(±0.0034)	0.5343(±0.0043)	0.5343(±0.0043)
	AP	0.7817(±0.0103)	0.7293(±0.0069)	0.6875(±0.0023)	0.6643(±0.006)	0.6643(±0.006)
	NE-AUC	0.4665(±0.0172)	0.4512(±0.0033)	0.4526(±0.0041)	0.4436(±0.0031)	0.4436(±0.0031)
	NE-AP	0.5541(±0.0131)	0.5416(±0.0055)	0.5434(±0.0031)	0.5286(±0.0077)	0.5286(±0.0077)
TO-GVAE	AUC	0.9943(±0.0004)	**0.873(±0.0022)**	**0.7728(±0.0031)**	**0.726(±0.0084)**	**0.7281(±0.0045)**
	AP	**0.9925(±0.0011)**	**0.9197(±0.0015)**	**0.8515(±0.0027)**	**0.8158(±0.0056)**	**0.8177(±0.0034)**
	NE-AUC	0.995(±0.001)	**0.8203(±0.0042)**	**0.7076(±0.0016)**	**0.6641(±0.0089)**	**0.6591(±0.008)**
	NE-AP	**0.989(±0.003)**	**0.8615(±0.0043)**	**0.776(±0.0023)**	**0.7396(±0.0055)**	**0.7367(±0.0044)**
TO-GAE	AUC	**0.9944(±0.0012)**	0.8702(±0.0029)	0.7629(±0.0062)	0.711(±0.0095)	0.711(±0.0095)
	AP	0.9915(±0.0028)	0.9176(±0.0022)	0.8461(±0.002)	0.8077(±0.0062)	0.8077(±0.0062)
	NE-AUC	**0.9955(±0.0009)**	0.8159(±0.0029)	0.6972(±0.0087)	0.6449(±0.0084)	0.6449(±0.0084)
	NE-AP	0.9882(±0.0043)	0.8588(±0.0025)	0.7711(±0.0035)	0.7285(±0.005)	0.7285(±0.005)

Table 5.6: Evolution pattern prediction results presented as mean values with standard deviation for both the whole graph and new edges on the cit-HepPh dataset across all models trained using G_0. A bold value indicates the highest score for that metric for the given graph snapshot.

Future Link Prediction

Table 5.7 highlights the results for all models at the task of future link prediction on the cit-HepPh dataset. We can see that compared with the previous task, all models are more closely matched. The results show all approaches to have a good predictive performance, even at later time points, however, the temporally offset approaches still outperform the baselines. Here it is interesting to note that the non-probabilistic model almost always outperforms the variational approach.

Model	**Metric**	G_1	G_2	G_3	G_4	G_5
GVAE	AUC	0.9702(±0.0009)	0.9336(±0.0026)	0.8796(±0.0018)	0.8503(±0.0018)	0.8499(±0.0009)
	AP	0.9759(±0.0006)	0.9461(±0.0021)	0.9016(±0.0012)	0.8762(±0.0013)	0.8761(±0.0007)
	NE-AUC	0.9586(±0.002)	0.9149(±0.0005)	0.8552(±0.0029)	0.8232(±0.002)	0.8231(±0.0039)
	NE-AP	0.9521(±0.0016)	0.9103(±0.0012)	0.8581(±0.0028)	0.8289(±0.0023)	0.8289(±0.0034)
GAE	AUC	0.9885(±0.0006)	0.9794(±0.0018)	0.9545(±0.0011)	0.9412(±0.0007)	0.9412(±0.0007)
	AP	0.9902(±0.0004)	0.9815(±0.0014)	0.9616(±0.0011)	0.9518(±0.0007)	0.9518(±0.0007)
	NE-AUC	0.9837(±0.0008)	0.9732(±0.0022)	0.9447(±0.0015)	0.9303(±0.0007)	0.9303(±0.0007)
	NE-AP	0.9803(±0.0008)	0.9685(±0.002)	0.9442(±0.002)	0.9332(±0.0012)	0.9332(±0.0012)
TO-GVAE	AUC	0.9957(±0.0004)	0.9871(±0.0006)	0.9651(±0.0014)	0.9534(±0.0009)	0.9524(±0.001)
	AP	**0.9944(±0.0007)**	0.9869(±0.0005)	0.9693(±0.0009)	0.9596(±0.0008)	0.9591(±0.0011)
	NE-AUC	0.9964(±0.0003)	0.9841(±0.0005)	0.9578(±0.0022)	0.9439(±0.0011)	0.9427(±0.0003)
	NE-AP	0.9921(±0.0008)	0.979(±0.0006)	0.9549(±0.0021)	0.9427(±0.0017)	0.9419(±0.0015)
TO-GAE	AUC	**0.9961(±0.0001)**	**0.99(±0.0001)**	**0.9751(±0.0009)**	**0.9645(±0.0003)**	**0.9645(±0.0003)**
	AP	0.9943(±0.0003)	**0.9887(±0.0001)**	**0.9757(±0.0008)**	**0.967(±0.0008)**	**0.967(±0.0008)**
	NE-AUC	**0.997(±0.0001)**	**0.9882(±0.0002)**	**0.9701(±0.0011)**	**0.9579(±0.0003)**	**0.9579(±0.0003)**
	NE-AP	**0.9922(±0.0006)**	**0.9825(±0.0)**	**0.9646(±0.0014)**	**0.9536(±0.0013)**	**0.9536(±0.0013)**

Table 5.7: Future link prediction results presented as mean values with standard deviation for both the whole graph and new edges on the cit-HepPh dataset across all models trained using G_0. A bold value indicates the highest score for that metric for the given graph snapshot.

5.8 Temporal Neighbourhood Aggregation: Results

This section presents results of the experimental evaluation of the Temporal Neighbourhood Aggregation approach.

5.8.1 Ablation Study

One of the major contributions of the work is highlighting how each component of our TNA model is crucial in producing good temporal embeddings. To highlight this, Table 5.8 shows how

adding components of the model sequentially affects the performance of predicting new edges in the final graph of the Bitcoina dataset. It is important to note that adding temporal information from both the first and second hop neighbourhood (Model TTV) lifts both AUC and AP scores by approximately 10% versus just first hop temporal information (Model TGV). This supports our hypothesis that a vertex requires temporal information from more than just its first-order neighbourhood in order to predict future edges. The ablation study also demonstrates that, with a modest increase in the number of parameters, the temporal models are able to exploit the rich information available in the graph's past evolution to much more accurately predict future edges.

Approach	**AUC**	**AP**	$\|\Theta\|$
GGG	0.574	0.747	$121K$
GGV	0.721	0.705	$122K$
TGV	0.772	0.809	$130K$
TTV	0.863	0.916	$132K$
TTV/LN	0.927	0.932	$132K$
TTV/LN/SC (TNA)	**0.977**	**0.976**	$133K$

Table 5.8: Ablation study results on the Bitcoina dataset. G is a GCN layer, V is a variational sampling layer, T is a GCN + GRU layer, LN is Layer Norm and SC is a skip-connection. $|\Theta|$ is the total number of learnable parameters in the model.

5.8.2 Next Graph Link Prediction

As one of the primary goals of the TNA model, we present results for predicting new edges in the next temporal graph, using the procedure detailed in Algorithm 5.1, in Table 5.9[2]. The table shows that TNA significantly outperforms the baseline approaches when predicting new edges in the next graph at all points along the time series. Compared with the Dyn* family of approaches, it is striking to note the significant number of parameters required by the models (often well over an order of magnitude more) and their poor performance in predicting new edges. We believe it is highly likely that this family of models is using the extra parameters to over-fit to the edges that do not change over time, resulting in bad predictive capability for the ones that do. It is also interesting to note that, compared with the D-GCN approach, TNA is better able to capture the dependences needed for good long-term prediction. For two datasets our model improves the past graph evolution data from which it has to learn. This is demonstrated by

[2] DynRNN is missing for the Wiki dataset as it could not fit in GPU memory.

the increasing AUC and AP scores for the Bitcoina and UCI datasets. However, all approaches struggle on the synthetic datasets due to their inherent random nature, as seen in Table 5.10.

Dataset	Approach	AUC			AP			$\lvert\Theta\rvert$
		25%	50%	100%	25%	50%	100%	
Bitcoina	GAE	0.466 ± 0.025	0.497 ± 0.042	0.531 ± 0.127	0.613 ± 0.031	0.643 ± 0.042	0.681 ± 0.093	$121K$
	GVAE	0.577 ± 0.048	0.602 ± 0.046	0.620 ± 0.083	0.634 ± 0.043	0.654 ± 0.040	0.670 ± 0.068	$122K$
	TO-GAE	0.551 ± 0.053	0.566 ± 0.053	0.576 ± 0.124	0.694 ± 0.038	0.701 ± 0.038	0.715 ± 0.085	$120K$
	TO-GVAE	0.598 ± 0.048	0.620 ± 0.045	0.631 ± 0.081	0.646 ± 0.044	0.665 ± 0.040	0.631 ± 0.081	$122K$
	DynAE	0.281 ± 0.080	0.247 ± 0.065	0.209 ± 0.071	0.435 ± 0.012	0.442 ± 0.012	0.439 ± 0.023	$4.16M$
	DynRNN	0.181 ± 0.081	0.170 ± 0.059	0.155 ± 0.066	0.388 ± 0.014	0.388 ± 0.011	0.393 ± 0.022	$69.9M$
	DynAERNN	0.093 ± 0.090	0.071 ± 0.066	0.048 ± 0.054	0.326 ± 0.022	0.320 ± 0.016	0.318 ± 0.012	$6.98M$
	D-GCN	0.622 ± 0.084	0.572 ± 0.080	0.519 ± 0.144	0.697 ± 0.058	0.661 ± 0.058	0.623 ± 0.107	$125K$
	TNA	$\mathbf{0.665 \pm 0.067}$	$\mathbf{0.698 \pm 0.075}$	$\mathbf{0.775 \pm 0.110}$	$\mathbf{0.762 \pm 0.048}$	$\mathbf{0.792 \pm 0.054}$	$\mathbf{0.849 \pm 0.079}$	$133K$
UCI	GAE	0.561 ± 0.075	0.600 ± 0.075	0.606 ± 0.092	0.661 ± 0.066	0.688 ± 0.060	0.689 ± 0.079	$61K$
	GVAE	0.571 ± 0.079	0.606 ± 0.074	0.619 ± 0.065	0.585 ± 0.059	0.621 ± 0.063	0.625 ± 0.060	$62K$
	TO-GAE	0.601 ± 0.059	0.633 ± 0.061	0.625 ± 0.087	0.682 ± 0.053	0.705 ± 0.050	0.699 ± 0.076	$61K$
	TO-GVAE	0.582 ± 0.072	0.614 ± 0.069	0.624 ± 0.062	0.590 ± 0.057	0.624 ± 0.062	0.627 ± 0.060	$62K$
	DynAE	0.234 ± 0.066	0.168 ± 0.076	0.128 ± 0.067	0.436 ± 0.019	0.435 ± 0.021	0.433 ± 0.017	$2.28M$
	DynRNN	0.161 ± 0.019	0.176 ± 0.024	0.159 ± 0.048	0.365 ± 0.016	0.370 ± 0.016	0.369 ± 0.029	$21.8M$
	DynAERNN	0.033 ± 0.032	0.021 ± 0.025	0.013 ± 0.019	0.314 ± 0.005	0.312 ± 0.004	0.312 ± 0.003	$4.15M$
	D-GCN	0.508 ± 0.041	0.555 ± 0.071	0.565 ± 0.068	0.605 ± 0.045	0.653 ± 0.066	0.656 ± 0.072	$64K$
	TNA	$\mathbf{0.694 \pm 0.077}$	$\mathbf{0.749 \pm 0.073}$	$\mathbf{0.764 \pm 0.071}$	$\mathbf{0.702 \pm 0.073}$	$\mathbf{0.763 \pm 0.075}$	$\mathbf{0.783 \pm 0.067}$	$72K$
Wiki	GAE	0.491 ± 0.035	0.487 ± 0.038	0.502 ± 0.040	0.642 ± 0.029	0.621 ± 0.033	0.617 ± 0.032	$228K$
	GVAE	0.580 ± 0.024	0.573 ± 0.018	0.563 ± 0.024	0.598 ± 0.032	0.589 ± 0.025	0.572 ± 0.029	$229K$
	TO-GAE	0.537 ± 0.052	0.556 ± 0.049	0.552 ± 0.048	0.700 ± 0.032	0.697 ± 0.027	0.668 ± 0.044	$228K$
	TO-GVAE	0.599 ± 0.028	0.595 ± 0.021	0.579 ± 0.029	0.613 ± 0.036	0.604 ± 0.029	0.583 ± 0.034	$229K$
	DynAE	0.354 ± 0.034	0.325 ± 0.041	0.244 ± 0.089	0.448 ± 0.009	0.463 ± 0.016	0.467 ± 0.013	$7.5M$
	DynAERNN	0.183 ± 0.024	0.179 ± 0.026	0.127 ± 0.056	0.342 ± 0.005	0.341 ± 0.006	0.329 ± 0.012	$11.9M$
	D-GCN	0.628 ± 0.160	0.591 ± 0.115	0.563 ± 0.087	0.745 ± 0.104	0.686 ± 0.094	0.629 ± 0.089	$231K$
	TNA	$\mathbf{0.674 \pm 0.034}$	$\mathbf{0.644 \pm 0.044}$	$\mathbf{0.634 \pm 0.050}$	$\mathbf{0.759 \pm 0.025}$	$\mathbf{0.740 \pm 0.032}$	$\mathbf{0.736 \pm 0.039}$	$239K$

Table 5.9: Next graph prediction results presented as mean values with standard deviation when predicting at various percentages of the length of the time-sequence. A bold value indicates the highest score for that metric. The number of parameters required by each model for the specific datasets are also included.

5.8.3 Full Graph Reconstruction

To measure the ability of the representations learned by the TNA model to be used as general purpose embeddings, we look at the problem of future graph reconstruction. Here, the performance of the model at predicting the presence of edges in the full graph G_t (given $G_1..G_{t-1}$) is measured – highlighting how we do not sacrifice performance at predicting existing edges. This will allow us to investigate the ability of the model to predict not only new edges, but that existing edges have not been removed. As before, a new model is trained to predict the final graph in the sequence given all previous time points, with the final results presented as the

Dataset	Approach	AUC	AP
SBM	GAE	$\mathbf{0.505 \pm 0.018}$	0.451 ± 0.009
	GVAE	0.500 ± 0.012	$\mathbf{0.503 \pm 0.011}$
	TO-GAE	0.504 ± 0.017	0.451 ± 0.008
	TO-GVAE	0.500 ± 0.012	0.503 ± 0.011
	DynAE	0.023 ± 0.003	0.431 ± 0.008
	DynRNN	0.039 ± 0.005	0.348 ± 0.009
	DynAERNN	0.008 ± 0.000	0.308 ± 0.000
	D-GCN	0.458 ± 0.017	0.458 ± 0.017
	TNA	0.502 ± 0.024	0.502 ± 0.017
R-Cora	GAE	0.501 ± 0.015	0.500 ± 0.0100
	GVAE	0.491 ± 0.011	0.494 ± 0.002
	TO-GAE	0.500 ± 0.013	$\mathbf{0.502 \pm 0.009}$
	TO-GVAE	0.490 ± 0.011	0.494 ± 0.011
	DynAE	0.356 ± 0.001	0.479 ± 0.003
	DynRNN	0.308 ± 0.011	0.381 ± 0.011
	DynAERNN	0.201 ± 0.000	0.346 ± 0.000
	D-GCN	$\mathbf{0.502 \pm 0.011}$	0.500 ± 0.008
	TNA	0.493 ± 0.012	0.493 ± 0.012

Table 5.10: Next graph prediction results on sythnetic graphs presented as mean values with standard deviation when predicting at each point in the time series.

mean over all graphs in the sequence. However, instead of predicting edges which have appeared since the last time point, here the results are for a balanced set of random sampled positive and negative edges in E_t which may or may not include ones formed since the previous time point.

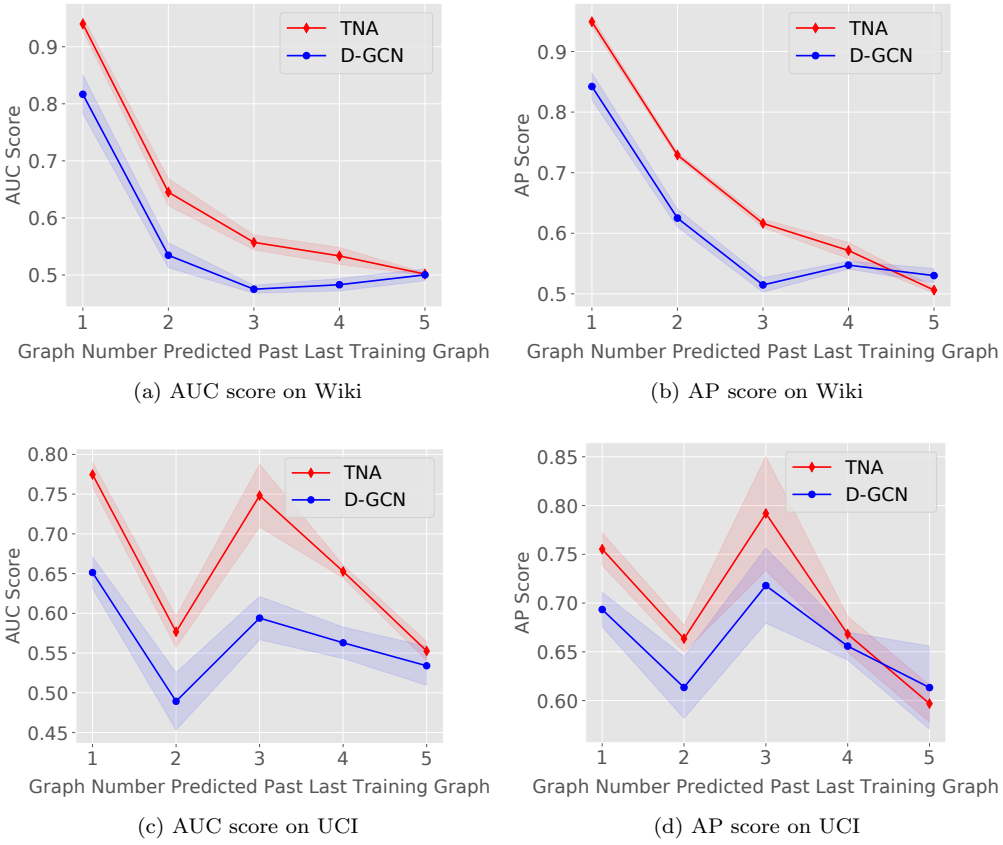
The results for this experiment are presented in Table 5.11 where for the sake of readability, we compare with only the temporal baselines. It is obvious that many of the baselines, especially the Dyn* family of approaches perform much better at predicting existing edges than new ones. This further suggests that they are utilising their larger set of parameters to, in some way, over-fit to edges which have been in the graph for a longer length of time, which form the vast majority. However despite this, our TNA approach still performs well at this task, displaying comparable performance with the baseline approaches and even outperforming them on the Wiki dataset. This further strengthens the argument that having recurrence at each hop in the neighbourhood aggregation produces a better representation, whilst requiring fewer parameters.

Dataset	Approach	AUC	AP
Bitcoina	DynAE	0.830 ± 0.068	0.844 ± 0.050
	DynRNN	0.922 ± 0.059	0.937 ± 0.039
	DynAERNN	$\mathbf{0.968 \pm 0.057}$	$\mathbf{0.981 \pm 0.034}$
	D-GCN	0.919 ± 0.021	0.934 ± 0.016
	TNA	0.932 ± 0.024	0.945 ± 0.018
UCI	DynAE	0.905 ± 0.061	0.908 ± 0.055
	DynRNN	0.957 ± 0.015	0.954 ± 0.010
	DynAERNN	$\mathbf{0.988 \pm 0.014}$	$\mathbf{0.993 \pm 0.009}$
	D-GCN	0.829 ± 0.019	0.862 ± 0.014
	TNA	0.821 ± 0.015	0.847 ± 0.012
Wiki	DynAE	0.765 ± 0.088	0.795 ± 0.062
	DynAERNN	0.882 ± 0.072	0.934 ± 0.037
	D-GCN	0.905 ± 0.019	0.936 ± 0.015
	TNA	$\mathbf{0.919 \pm 0.014}$	$\mathbf{0.945 \pm 0.007}$

Table 5.11: Results for predicting both new and old edges in the final graph in the sequence, presented as a mean and standard deviation over the whole time sequence. A bold value indicates the highest score for that metric. TNA remains competitive with, and even beats many baseline approaches with a much greater number of parameters.

5.8.4 Future Graph Evolution

For our final experiment, we investigate how TNA performs when predicting new edges further into the future than the next graph. We train the models on 70% of the available temporal history, then predict new edges and compare with the remaining ground truth data. To achieve this, we feed the graph predicted by the models as the next graph in the sequence back into the model, which is subsequently used to predict the next graph. This is similar to using RNNs as generative models to produce text data [217] and can be seen as a combination of both the previous tasks. Figure 5.10 displays the results for this task, where we compare with the closest baseline from Section 5.8.2. The results show how TNA is better able to predict new edges into the future, emphasising its capability to learn a good temporal representation for the vertices.



(a) AUC score on Wiki

(b) AP score on Wiki

(c) AUC score on UCI

(d) AP score on UCI

Figure 5.10: AUC and AP for the Wiki and UCI datasets when predicting new edges n number of time points away from the end of the training sequence. Results presented as the mean of three uniquely trained models, each with a different random seed.

5.9 Conclusion

Whilst a lot of focus has recently been placed on finding methods for learning graph representations which are accurately able to make predictions about the current state of the graph, few works have investigated how these models perform for temporal graphs. However, many real-world graph datasets have rich and complex temporal information available which is disregarded by the majority of the current approaches for creating vertex representations. This chapter has explored two new graph specific neural networks for incorporating temporal information in the

learning process: Temporal Offset Reconstruction and Temporal Neighbourhood Aggregation. The approaches demonstrate excellent performance through extensive experimental evaluation, beating several competing temporal and static models on a range of crucial dynamic graph tasks.

5.9.1 Current Limitations

Whilst the work presented in this chapter has been successful in creating temporal graph embeddings, there are some current limitations with the approach:

- *Fixed Input Graph Sizes:* Due to the use of GCN layers in both approaches, all graphs in the time series must have the same adjacency matrix size for the model parameters to be shared across the sequence. This means that graphs from earlier in the time series, which may contain a smaller number of vertices, must be padded with place-holder vertices, containing no edges, to ensure that all graphs have the same number of vertices as the largest. This leads to redundant computation being performed on the place-holder vertices, slowing training times and mandating a data pre-processing step.

- *Large Memory Requirements:* Although the approaches require fewer parameters than competing ones, the memory requirements for training are still large. This results in only relatively small graphs being able to be used as input, or larger graphs with shorter temporal evolutions. The high memory requirement of the approaches is primarily due to the GCN layers which require the whole graph to be in memory and do not support mini-batching of the data to alleviate this [128]. Additionally, RNNs grow linearly in memory requirement with the sequence length [154], meaning that every graph in the time series must fit in GPU memory.

- *Temporal Learning Component Choice:* The TNA approach uses an RNN to learn the temporal change in a graph. However, a new family of models have emerged from the field of natural language processing, entitled Transformers, which have resulted in large improvements in temporal tasks [226]. Transformers overcome some of the long term dependency issues with RNNs and are able to directly access any element in the time series.

- *Lack of Interpretability:* There has been little significant research on if and what graph topological features are being learned by GCN layers. Additionally, it is an open research question as to how one would actually measure what temporal change in graph topological structure is being learned by the RNNs. This means that when compared with the research performed in Chapter 4, it could be argued the models presented here are less interpretable.

5.9.2 Future Work

For future research, work could be performed to investigate whether replacing the GCN layers with alternative layers, perhaps ones designed for inductive learning [93], could allow for training on even larger graph datasets. Additionally this could allow for graphs containing a different number of vertices in each time step to be modelled. Work could also be performed to investigate whether incorporating select topological features as input features for each vertex could improve overall predictive performance. Some work has been performed incorporating the earth-movers, or Wasserstein distance into a Variational Auto-Encoder model [223] – altering the TNA model presented here to include this could result in better temporal representations. Finally, additional tasks could be explored to further assess the performance of the vertex representations, for example the task of anomaly detection could be performed by looking for abnormal reconstruction values, indicating a graph not commonly seen during training.

Epilogue

This chapter has introduced and rigorously tested two models and associated training procedures for creating representations of temporal graphs, thereby achieving research objective 3. The potential of these models is demonstrated by the superior performance on several key tasks in temporal graph mining.

Over the course of this thesis, various aspects of using machine learning to study graphs have been investigated. In the final Chapter, the work is drawn to a conclusion with final observations made and new research directions suggested.

Chapter 6

Conclusions

Representing large and complex datasets in the form of a graph has been demonstrated to be beneficial in numerous scientific domains, as it allows for unique insights to be gained by exploiting the graph's structure. Often this structural knowledge is captured via the use of hand-crafted graph-specific algorithms which can be complicated to create and difficult to even run on large graphs. However, in other fields, most notably Computer Vision, recent advances in machine learning have enabled significant improvements in performance in many key tasks which were traditionally tackled via the use of equivalent hand-crafted algorithms.

Consequently, if the same were to hold true for graphs, there is a large potential benefit for using machine learning to augment or replace traditional approaches for solving key issues in the field of graph mining. This thesis has explored the use of various techniques to improve the ability of machine learning to be performed on graphs. The primary research aim and objectives of the thesis were established in Chapter 1, and recapped here they were designed to address three primary concerns: global graph representation, increased interpretability of graph-based representation learning and the incorporation of temporal dynamics into graph-specific machine learning models.

6.1 Summary of Thesis Contributions

This thesis has made various contributions to the field of graph-based machine learning whilst fulfilling the original research aim and objectives. Whilst the contributions have been explored in full in the previous chapters, a summary is given in this section.

In Chapter 3, a new method, entitled Graph Fingerprints, for creating a numerical representation capturing the crucial elements of the topological structure of a given graph is detailed. The representation explored the hypothesis that combining aggregated features which captured the characteristics of vertex neighbourhood structure, with global graph topological features, can accurately represent graph structure. The resulting representation was demonstrated through experimentation on the crucial tasks of graph comparison and graph classification, using various real-world and synthetic datasets and compared against competing approaches. In order to accurately compare and measure the similarity of two graphs, a method was proposed that combined Graph Fingerprints with the Canberra distance metric. The incorporation of global features allowed more sensitivity when comparing graphs of different sizes. To allow for graphs to be classified with a high level of precision, a custom deep neural network was proposed which used Graph Fingerprints as input. Using this approach allowed both binary and multi-class classification to be performed with equal or greater accuracy than the state-of-the-art Graph Kernel-based methods. Further, evidence was provided that demonstrated the Graph Fingerprints can be extracted from a graph in less time than competing approaches and that this process can be performed in parallel across a compute cluster, allowing graphs of over 100 million vertices to be processed. Whilst the approach was successful overall, it considered only a single representation for a graph, and offers no way to represent individual vertices.

In Chapter 4 work moves to considering representation learning at the level of individual vertices within a graph. Recently, numerous unsupervised vertex-level graph embedding techniques have been proposed which do not require hand-crafted features, instead the representations are learned as part of the process. These techniques have demonstrated state-of-the-art performance in important tasks such as vertex classification. However, due to the unsupervised nature of the learning process, there is a lack of interpretability regarding which, if any, topological structure is being approximated. This could possibly hinder their use in crucial domains such as healthcare. Inspired by the work in the previous chapter showing how topological features can be an excellent representation for a graph, research in this chapter proposed to begin to bring interpretability to these approaches by investigating whether a mapping can be found from

the learned embeddings to known topological features. This proposed mapping was investigated using a combination of supervised and unsupervised methods, which yielded empirical evidence that several known topological features are approximated in the embedding space. The work resulted in the interesting discovery that Eigenvector centrality was consistently the feature most well approximated by the approaches, giving some insight into which topological structures are being captured.

Finally, Chapter 5 considers how best to incorporate the temporal dynamics present in many real-world graph datasets into models operating on graph data. The chapter proposed two novel alternative unsupervised methods for creating vertex level representations which contain temporal, in addition to, structural information. The first approach trains on offset pairs of snapshots from a graph's evolution, where the next time-point is directly predicted from the first. Despite being limited by only being able to consider a single previous graph, the approach was shown to create representations which are more robust over multiple future time-points. The second proposed approach addresses issues from the first by incorporating recurrence into a new model, meaning that all previous time-points in a graph's history can be recalled in order to improve performance. The approach achieved this by allowing vertices to learn their representations by aggregating information about how their neighbours had changed over time. Additionally, both proposed approaches demonstrated that sampling the vertex representations using variational approximations can create better ones overall. This was achieved whilst using a smaller number of model parameters than competing approaches, owing in part to not requiring a parameterized decoder. Further, the experimental evidence in the chapter highlighted that competing approaches can over-fit to older edges, resulting in poor performance at predicting newly arriving edges.

6.2 Review of Research Aim and Objectives

During the work performed for this thesis, the primary research aim and associated objectives, established in Chapter 1, have been successfully achieved:

- Firstly, a representation for a single graph using both local and global topological features was created (Chapter 3). Whilst this was largely achieved successfully, the approach still requires some hand-tuning as detailed in Section 3.9.1, and more recent end-to-end approaches may be able to alleviate this issue.

- Secondly, a family of methods to bring interpretability to graph embeddings via the use of topological features was proposed and thoroughly evaluated (Chapter 4), thus this research objective was successfully achieved. The outcome of the evaluation was that, even though a strong correlation was found between certain known topological features and the embedding space, attributing causality to this remains an issue (detailed further in Section 6.4.2).
- Thirdly, two new models were successfully created which incorporated temporal graph dynamics in alternative ways (Chapter 5). The experimental evaluation highlighted that the approaches performed well on certain datasets sizes, but did not always scale to larger sizes.

6.3 Evaluation and Analysis of Key Contributions

Sections 6.1 and 6.2 have described the contributions of the research and confirmed that the original research aim and objectives have been achieved. This section outlines a qualitative evaluation and analysis of the research contributions made during this thesis.

- This research has successfully identified a descriptive set of topological features that can be incorporated as a single numerical representation (here termed Graph Fingerprinting). This is significant because traditional ways of doing this have relied on slow and inefficient methods such as Graph Kernels.
- The thesis proposes new methods regarding the interpretability of existing graph embedding approaches and the associated code has also been made available publicly. This is important as the developed methods can begin to offer some explanation as to why unsupervised graph embedding approaches have proven so successful. In addition this might potentially allow graph embeddings to be used more broadly in sensitive fields, such as the medical or legal domains.
- The thesis provides two novel models (which have been made publicly available) allowing temporal aspects to be incorporated into graph based machine learning. This is important because the majority of existing approaches have typically ignored temporal information, whereas the reality is that many real-world problems in graph mining are dependant upon evolving graph dynamics.

- In a wider sense, this research has provided some insights and explanation into how machine learning is developing with specific respect to graphs. More specifically, that machine learning models appear to be 'thinking' about graphs in a similar way to that of human experts. This is a critical aspect of this research since this phenomenon has been observed in other sectors (Computer Vision for example) but hitherto not in the graph-based machine learning field.
- Overall, this research has highlighted that there are some potential ideological concerns regarding traditional human-driven versus machine-learned knowledge. This signposts that there should be caution in future research to the effect that human-driven knowledge is still valuable, especially with issues regarding scalability and large amounts of data and thus should not be ignored.
- This research has contributed to certain gaps in the literature, especially regarding interpretability and also the temporal nature of many graph datasets. Indeed, this research has arguably been among the first works to consider these particular aspects in depth.
- The experimental evaluation has demonstrated that the use of synthetic graph data is acceptable for measuring aspects of run-time performance, but perhaps not for measuring a model's predictive capability. This highlights the issue that the lack of availability of suitable public datasets is a major obstacle in the progression of this field.

6.4 Future Work

Whilst the approaches explored in this thesis have demonstrated, through experimental evaluation, that they are successful, there is clear scope for future work. This section is divided into work that could be undertaken to improve the current research and further novel research that could be conducted as a clear continuation of this research.

6.4.1 Improvements to Current Work

Despite the successes of the overall approaches explored in this thesis, with the benefit of hindsight, there are some areas in which the research could have been improved. This is

mostly laid out in the conclusion sections for Chapters 3 4 and 5, however the major items are summarised in this section.

All of the various approaches explored in this thesis have required the use of varying quantities of real-world and synthetic graph datasets as input. However, due mainly to the limited availability of large quantities of publicly available datasets, it was not possible to perform experimentation with graphs from all domains. Thus, perhaps the single most impactful improvement that could be made to the work presented in this thesis would be to expand the datasets upon which the various methods were evaluated. The global graph classification approach using Graph Fingerprints (detailed in Chapter 3) in particular was limited by only being able to be tested on synthetic data. It would therefore be a large validation of the work if the same patterns and conclusions could be drawn when real-world data was used.

Additionally, the work presented in this thesis focused primarily on undirected graphs with only a single edge type. Further work would be needed to confirm that the same approaches also worked on more complicated input data formats, such as directional, weighted or hyper graphs.

6.4.2 Expansions to the Work

Using the research undertaken in this thesis, there are several interesting ways in which work could continue, the most relevant of which are outlined below.

Creation of Benchmark Datasets: It can be argued that the release of large public benchmark datasets in the computer vision field have been partially responsible for the increases in predictive performance across a range of related tasks. Datasets such as MNIST [143] and ImageNet [65] not only get large amounts of data into the public domain, allowing for a greater amount of innovation, they can also act as a benchmark against which various competing approaches can be directly compared. The creation of such a resource for the field of graph mining would bring many potential benefits and allow for larger and more complex models to be created. This would be a challenging task however, as graphs typically represent data originating from a range of scientific disciplines and can contain temporal in addition to other auxiliary information. Complicating matters further, often the underlying data is of a proprietary nature and held across many different organisations, thus making the creation of a single representative benchmark dataset even harder.

Graph Generative Models: The work performed in this thesis employed the use of several graph generative models to create synthetic data (Forest-Fire for example [145]), the majority of which are based on simple statistical rules. There is large scope for machine learning to be used to generate random graphs that much more closely mimic the structure of a certain set of training graphs. This could not only result in more training data being made available for approaches like the ones introduced in this thesis, but it could be used to remove privacy concerns that might arise from releasing real-world data, as a synthetic copy which mimics the original could be released instead. One can view the TNA approach (introduced in Chapter 5) as a generative model, since an unlimited number of future graph states can be generated. However, other approaches, perhaps based on the framework of a Generative Adversarial Network (GAN) [81] could produce a broad range of synthetic graphs without requiring temporal data. Some early work has even been performed using GANs with graphs, however the focus of the work thus far has been in adapting GANs to perform link prediction, with GraphGAN being a representative example [230]. However, making such approaches learn from a large corpus of training graphs, and then produce examples which could plausibly come from the same distribution, would require a potentially exponential increase in their capacity and processing speed.

Graph Model Pre-training: A long line of research has shown that pre-training a deep neural network on some initial task or dataset can dramatically improve model performance on a secondary task [71]. One of the commonly accepted explanations for this is that the model is learning features which are able to be reused across datasets and tasks [96], making a model trained on a different task a better starting point than the randomly initialised weights typically used. In some fields, this process of using a model trained on a different dataset and fine-tuning on a different dataset or task is known as Transfer Learning [218] However, due to the nature of the architecture of graph-specific models being dataset dependant, particularly the GCN used in parts of this thesis [128], transferring knowledge across graph datasets is exceptionally challenging [104]. One possible solution for the pre-training of graph models, using the research presented in this thesis, would be to have the model pre-trained to predict certain key topological features from a graph, before then continuing on to the desired task. This could prime the model with knowledge about the graph's topological structure which has proven to be useful for the identification of graphs, as shown in Chapters 3 and 4.

Fully Explainable Graph Models: The work presented in Chapter 4 took some initial steps toward bringing interpretability to unsupervised graph embeddings. However the proposed approach was a secondary process which was used after the first model had been trained to

interpret the results, a common approach in the literature [192]. Further research would be needed to create a single model which was trained in an end-to-end manner to not only produce the desired predictions, but to also explain the rules and decision-making process that allowed the model to arrive at a certain decision. Such a hypothetical model could allow for interpretable decisions to be made, hopefully increasing the ability of graph-based models to be adopted in crucial industries such as healthcare and law.

Bibliography

[1] Martín Abadi, Paul Barham, Jianmin Chen, Zhifeng Chen, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Geoffrey Irving, Michael Isard, Manjunath Kudlur, Josh Levenberg, Rajat Monga, Sherry Moore, Derek G. Murray, Benoit Steiner, Paul Tucker, Vijay Vasudevan, Pete Warden, Martin Wicke, Yuan Yu, and Xiaoqiang Zheng. Tensorflow: a system for large-scale machine learning. *USENIX Symposium on Operating Systems Design and Implementation*, 16:265–283, 2016.

[2] Lada A Adamic and Eytan Adar. Friends and neighbors on the web. *Social networks*, 25(3):211–230, 2003.

[3] Charu Aggarwal and Karthik Subbian. Evolutionary network analysis: A survey. *ACM Computing Surveys (CSUR)*, 47(1):10, 2014.

[4] Amr Ahmed, Nino Shervashidze, Shravan Narayanamurthy, Vanja Josifovski, and Alexander J Smola. Distributed large-scale natural graph factorization. *International conference on World Wide Web*, pages 37–48, 2013.

[5] Nesreen K Ahmed, Theodore L Willke, and Ryan A Rossi. Estimation of local subgraph counts. In *2016 IEEE International Conference on Big Data (Big Data)*, pages 586–595. IEEE, 2016.

[6] Leman Akoglu, Hanghang Tong, and Danai Koutra. Graph based anomaly detection and description: a survey. *Data mining and knowledge discovery*, 29(3):626–688, 2015.

[7] Réka Albert and Albert-László Barabási. Statistical mechanics of complex networks. *Reviews of modern physics*, 74(1):47, 2002.

[8] Awrad Mohammed Ali, Hamidreza Alvari, Alireza Hajibagheri, Kiran Lakkaraju, and Gita Sukthankar. Synthetic generators for cloning social network data. *Proceedings of SocInfo*, 2014.

[9] Grigoris Antoniou and Frank Van Harmelen. *A semantic web primer*. MIT press, 2004.

[10] Sabeur Aridhi and Engelbert Mephu Nguifo. Big graph mining: Frameworks and techniques. *Big Data Research*, 6:1–10, 2016.

[11] Sylvain Arlot and Alain Celisse. A survey of cross-validation procedures for model selection *. *Statistics Surveys*, 2010.

[12] Michael Armbrust, Reynold S. Xin, Cheng Lian, Yin Huai, Davies Liu, Joseph K. Bradley, Xiangrui Meng, Tomer Kaftan, Michael J. Franklin, Ali Ghodsi, and Matei Zaharia. Spark sql: Relational data processing in spark. In *Proceedings of the 2015 ACM SIGMOD International Conference on Management of Data*, pages 1383–1394, 2015.

[13] Nik Khadijah Nik Aznan, Amir Atapour-Abarghouei, Stephen Bonner, Jason Connolly, Noura Al Moubayed, and Toby Breckon. Simulating brain signals: Creating synthetic eeg data via neural-based generative models for improved ssvep classification. In *International Joint Conference on Neural Networks*, pages 1–8, 2019.

[14] Nik Khadijah Nik Aznan, Stephen Bonner, Jason Connolly, Noura Al Moubayed, and Toby Breckon. On the classification of ssvep-based dry-eeg signals via convolutional neural networks. In *IEEE International Conference on Systems, Man, and Cybernetics*, pages 3726–3731. IEEE, 2018.

[15] Jimmy Lei Ba, Jamie Ryan Kiros, and Geoffrey E Hinton. Layer normalization. *arXiv preprint arXiv:1607.06450*, 2016.

[16] Lars Backstrom and Jure Leskovec. Supervised random walks: predicting and recommending links in social networks. In *Proceedings of the fourth ACM International Conference on Web Search and Data Mining*, pages 635–644, 2011.

[17] Pierre Baldi. Autoencoders, unsupervised learning, and deep architectures. In *Proceedings of ICML workshop on Unsupervised and Transfer Learning*, pages 37–49, 2012.

[18] Robert Bamler and Stephan Mandt. Dynamic word embeddings. *arXiv preprint arXiv:1702.08359*, 2017.

[19] Albert-László Barabási. *Network science*. Cambridge university press, 2016.

[20] Albert-László Barabási and Réka Albert. Emergence of scaling in random networks. *science*, 286(5439):509–512, 1999.

[21] Kajdanowicz T. Bartusiak R. Sparklinggraph: large scale, distributed graph processing made easy. Manuscript in preparation, 2016.

[22] Peter Battaglia, Razvan Pascanu, Matthew Lai, Danilo Jimenez Rezende, and Koray kavukcuoglu. Interaction networks for learning about objects, relations and physics. In *Advances in Neural Information Processing Systems*, pages 4502–4510, 2016.

[23] Mikhail Belkin and Partha Niyogi. Laplacian eigenmaps and spectral techniques for embedding and clustering. *Advances in neural information processing systems*, pages 585–591, 2002.

[24] Rianne van den Berg, Thomas N Kipf, and Max Welling. Graph convolutional matrix completion. *arXiv preprint arXiv:1706.02263*, 2017.

[25] Michele Berlingerio, Danai Koutra, Tina Eliassi-Rad, and Christos Faloutsos. Netsimile: A scalable approach to size-independent network similarity. *arXiv preprint arXiv:1209.2684*, 2012.

[26] Christopher M Bishop. *Pattern recognition and machine learning*. springer, 2006.

[27] Phillip Bonacich. Some unique properties of eigenvector centrality. *Social networks*, 29(4):555–564, 2007.

[28] 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 *IEEE International Conference on Big Data*, 2019.

[29] Stephen Bonner, John Brennan, Ibad Kureshi, Andrew Stephen McGough, and Georgios Theodoropoulos. Efficient comparison of massive graphs through the use of 'graph fingerprints'. In *KDD Workshop on Mining and Learning with Graphs (MLG)*, 2016.

[30] 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 *IEEE International Conference on Big Data*, pages 2691–2700. IEEE, 2017.

[31] 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 *IEEE International Conference on Big Data*, pages 3737–3746. IEEE, 2018.

[32] Stephen Bonner, John Brennan, Georgios Theodoropoulos, Ibad Kureshi, and Andrew Stephen McGough. Deep topology classification: A new approach for massive graph classification. In *IEEE International Conference on Big Data*, pages 3290–3297. IEEE, 2016.

[33] Stephen Bonner, John Brennan, Georgios Theodoropoulos, Ibad Kureshi, and Andrew Stephen McGough. Gfp-x: A parallel approach to massive graph comparison using spark. *IEEE International Conference on Big Data*, pages 3298–3307, 2016.

[34] Stephen Bonner, Ibad Kureshi, John Brennan, and Georgios Theodoropoulos. Exploring the evolution of big data technologies. In *Software Architecture for Big Data and the Cloud*, pages 253–283. Elsevier, 2017.

[35] 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. *Data Science and Engineering*, 4(3):269–289, 2019.

[36] 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 *IEEE International Conference on Big Data*, pages 737–746. IEEE, 2015.

[37] Stephen Bonner and David Rohde. Latent variable session-based recommendation. *Second Symposium on Advances in Approximate Bayesian Inference*, 2019.

[38] Stephen Bonner and Flavian Vasile. Causal embeddings for recommendation. In *Proceedings of the 12th ACM Conference on Recommender Systems*, pages 104–112. ACM, 2018.

[39] Karsten M Borgwardt and Hans-Peter Kriegel. Shortest-path kernels on graphs. In *IEEE International Conference on Data Mining*, pages 8–pp. IEEE, 2005.

[40] Leo Breiman. Random forests. *Machine learning*, 45(1):5–32, 2001.

[41] Leo Breiman. *Classification and regression trees*. Routledge, 2017.

[42] Joan Bruna, Wojciech Zaremba, Arthur Szlam, and Yann Lecun. Spectral networks and locally connected networks on graphs. In *International Conference on Learning Representations*, 2014.

[43] Hongyun Cai, Vincent W Zheng, and Kevin Chen-Chuan Chang. A comprehensive survey of graph embedding: Problems, techniques, and applications. *IEEE Transactions on Knowledge and Data Engineering*, 30(9):1616–1637, 2018.

[44] Shaosheng Cao, Wei Lu, and Qiongkai Xu. Grarep: Learning graph representations with global structural information. *ACM International on Conference on Information and Knowledge Management*, pages 891–900, 2015.

[45] Shaosheng Cao, Wei Lu, and Qiongkai Xu. Deep neural networks for learning graph representations. In *Thirtieth AAAI Conference on Artificial Intelligence*, 2016.

[46] Deepayan Chakrabarti and Christos Faloutsos. Graph mining: Laws, generators, and algorithms. *ACM computing surveys (CSUR)*, 38(1):2–es, 2006.

[47] Deepayan Chakrabarti, Yiping Zhan, and Christos Faloutsos. R-mat: A recursive model for graph mining. In *Proceedings of the 2004 SIAM International Conference on Data Mining*, pages 442–446. SIAM, 2004.

[48] Ben Chamberlain, Marc Deisenroth Clough, and James. Neural embeddings of graphs in hyperbolic space. *KDD Workshop on Mining and Learning with Graphs (MLG)*, 2017.

[49] Jie Chen, Tengfei Ma, and Cao Xiao. Fastgcn: fast learning with graph convolutional networks via importance sampling. *arXiv preprint arXiv:1801.10247*, 2018.

[50] Jinyin Chen, Xuanheng Xu, Yangyang Wu, and Haibin Zheng. Gc-lstm: Graph convolution embedded lstm for dynamic link prediction. *arXiv preprint arXiv:1812.04206*, 2018.

[51] Jinyin Chen, Jian Zhang, Xuanheng Xu, Chengbo Fu, Dan Zhang, Qingpeng Zhang, and Qi Xuan. E-lstm-d: A deep learning framework for dynamic network link prediction. *arXiv preprint arXiv:1902.08329*, 2019.

[52] Tianqi Chen, Ian Goodfellow, and Jonathon Shlens. Net2net: Accelerating learning via knowledge transfer. *arXiv preprint arXiv:1511.05641*, 2015.

[53] Kyunghyun Cho, Bart Van Merriënboer, Caglar Gulcehre, Dzmitry Bahdanau, Fethi Bougares, Holger Schwenk, and Yoshua Bengio. Learning phrase representations using rnn encoder-decoder for statistical machine translation. *arXiv preprint arXiv:1406.1078*, 2014.

[54] François Chollet. Keras. https://keras.io, 2015.

[55] Francois Chollet. *Deep Learning with Python*. Manning Publications Co., USA, 1st edition, 2017.

[56] Aaron Clauset, Cosma Rohilla Shalizi, and Mark EJ Newman. Power-law distributions in empirical data. *SIAM review*, 51(4):661–703, 2009.

[57] Alexis Conneau, Douwe Kiela, Holger Schwenk, Loic Barrault, and Antoine Bordes. Supervised learning of universal sentence representations from natural language inference data. *arXiv preprint arXiv:1705.02364*, 2017.

[58] Corinna Cortes and Vladimir Vapnik. Support-vector networks. *Machine learning*, 20(3):273–297, 1995.

[59] Peng Cui, Xiao Wang, Jian Pei, and Wenwu Zhu. A survey on network embedding. *arXiv preprint arXiv:1711.08752*, 2017.

[60] Gao Daqi and Ji Yan. Classification methodologies of multilayer perceptrons with sigmoid activation functions. *Pattern Recognition*, 38(10):1469–1482, 2005.

[61] Ankur Dave, Alekh Jindal, Li Erran Li, Reynold Xin, Joseph Gonzalez, and Matei Zaharia. Graphframes: an integrated api for mixing graph and relational queries. In *Proceedings of the Fourth International Workshop on Graph Data Management Experiences and Systems*, pages 1–8, 2016.

[62] Jeffrey De Fauw, Joseph R Ledsam, Bernardino Romera-Paredes, Stanislav Nikolov, Nenad Tomasev, Sam Blackwell, Harry Askham, Xavier Glorot, Brendan O'Donoghue, and Daniel Visentin. Clinically applicable deep learning for diagnosis and referral in retinal disease. *Nature medicine*, 24(9):1342–1350, 2018.

[63] Tiago de Paula Peixoto. graph-tool: An efficient python module for manipulation and statistical analysis of graphs. , 2016.

[64] Michaël Defferrard, Xavier Bresson, and Pierre Vandergheynst. Convolutional neural networks on graphs with fast localized spectral filtering. In *Advances in Neural Information Processing Systems*, pages 3844–3852, 2016.

[65] Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Li Fei-Fei. Imagenet: A large-scale hierarchical image database. In *2009 IEEE Conference on Computer Vision and Pattern Recognition*, pages 248–255. Ieee, 2009.

[66] Niels Doekemeijer and Ana Lucia Varbanescu. A survey of parallel graph processing frameworks. *Delft University of Technology*, page 21, 2014.

[67] Sergei N Dorogovtsev and José FF Mendes. *Evolution of networks: From biological nets to the Internet and WWW*. OUP Oxford, 2013.

[68] David Easley and Jon Kleinberg. *Networks, crowds, and markets*, volume 8. Cambridge university press Cambridge, 2010.

[69] David BA Epstein and Robert C Penner. Euclidean decompositions of noncompact hyperbolic manifolds. *Journal of Differential Geometry*, 27(1):67–80, 1988.

[70] P Erdős and A Rényi. On random graphs I. *Publicationes Mathematicae*, 1959.

[71] Dumitru Erhan, Yoshua Bengio, Aaron Courville, Pierre-Antoine Manzagol, Pascal Vincent, and Samy Bengio. Why does unsupervised pre-training help deep learning? *Journal of Machine Learning Research*, 2010.

[72] Leonhard Euler. Solutio problematis ad geometriam situs pertinentis. *Commentarii academiae scientiarum Petropolitanae*, pages 128–140, 1741.

[73] Michalis Faloutsos, Petros Faloutsos, and Christos Faloutsos. On power-law relationships of the internet topology. *ACM SIGCOMM Computer Communication Review*, 1999.

[74] Santo Fortunato. Community detection in graphs. *Physics reports*, 486(3-5):75–174, 2010.

[75] Linton C Freeman. A set of measures of centrality based on betweenness. *Sociometry*, pages 35–41, 1977.

[76] Thomas Gärtner, Peter Flach, and Stefan Wrobel. On graph kernels: Hardness results and efficient alternatives. In *Learning theory and kernel machines*, pages 129–143. Springer, 2003.

[77] Leilani H Gilpin, David Bau, Ben Z Yuan, Ayesha Bajwa, Michael Specter, and Lalana Kagal. Explaining explanations: An overview of interpretability of machine learning. In *2018 IEEE 5th International Conference on data science and advanced analytics (DSAA)*, pages 80–89. IEEE, 2018.

[78] Xavier Glorot and Yoshua Bengio. Understanding the difficulty of training deep feedforward neural networks. In *Proceedings of the thirteenth international conference on artificial intelligence and statistics*, pages 249–256, 2010.

[79] Pavel Golik, Patrick Doetsch, and Hermann Ney. Cross-entropy vs. squared error training: a theoretical and experimental comparison. In *Interspeech*, volume 13, pages 1756–1760, 2013.

[80] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep learning*. MIT press, 2016.

[81] Ian Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. Generative adversarial nets. In *Advances in neural information processing systems*, pages 2672–2680, 2014.

[82] Palash Goyal, Sujit Rokka Chhetri, and Arquimedes Canedo. dyngraph2vec: Capturing network dynamics using dynamic graph representation learning. *Knowledge-Based Systems*, 2019.

[83] Palash Goyal, Sujit Rokka Chhetri, Ninareh Mehrabi, Emilio Ferrara, and Arquimedes Canedo. Dynamicgem: A library for dynamic graph embedding methods. *arXiv preprint arXiv:1811.10734*, 2018.

[84] Palash Goyal and Emilio Ferrara. Graph embedding techniques, applications, and performance: A survey. *Knowledge-Based Systems*, 151:78–94, 2018.

[85] Palash Goyal, Nitin Kamra, Xinran He, and Yan Liu. Dyngem: Deep embedding method for dynamic graphs. *arXiv preprint arXiv:1805.11273*, 2018.

[86] Sorin Grigorescu, Bogdan Trasnea, Tiberiu Cocias, and Gigel Macesanu. A survey of deep learning techniques for autonomous driving. *Journal of Field Robotics*, 2019.

[87] Aditya Grover and Jure Leskovec. node2vec: Scalable feature learning for networks. In *Proceedings of the 22nd ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 855–864. ACM, 2016.

[88] Ting Guo and Xingquan Zhu. Understanding the roles of sub-graph features for graph classification: an empirical study perspective. In *Proceedings of the 22nd ACM international conference on Information & Knowledge Management*, pages 817–822. ACM, 2013.

[89] David Guthrie, Ben Allison, Wei Liu, Louise Guthrie, and Yorick Wilks. A closer look at skip-gram modelling. In *LREC*, pages 1222–1225, 2006.

[90] Isabelle Guyon and André Elisseeff. An introduction to variable and feature selection. *Journal of machine learning research*, 3(Mar):1157–1182, 2003.

[91] William L Hamilton, Jure Leskovec, and Dan Jurafsky. Diachronic word embeddings reveal statistical laws of semantic change. *arXiv preprint arXiv:1605.09096*, 2016.

[92] William L Hamilton, Rex Ying, and Jure Leskovec. Representation learning on graphs: Methods and applications. *arXiv preprint arXiv:1709.05584*, 2017.

[93] William L Hamilton, Zhitao Ying, and Jure Leskovec. Inductive representation learning on large graphs. In *Advances in Neural Information Processing Systems*, pages 1024–1034, 2017.

[94] Minyang Han, Khuzaima Daudjee, Khaled Ammar, M Tamer Özsu, Xingfang Wang, and Tianqi Jin. An experimental comparison of pregel-like graph processing systems. *Proceedings of the VLDB Endowment*, 7(12):1047–1058, 2014.

[95] Kazuyuki Hara, Daisuke Saito, and Hayaru Shouno. Analysis of function of rectified linear unit used in deep learning. In *2015 International Joint Conference on Neural Networks (IJCNN)*, pages 1–8. IEEE, 2015.

[96] Kaiming He, Ross Girshick, and Piotr Dollár. Rethinking imagenet pre-training. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 4918–4927, 2019.

[97] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778, 2016.

[98] Oliver Hein, Michael Schwind, and Wolfgang König. Scale-free networks. *Wirtschaftsinformatik*, 48(4):267–275, 2006.

[99] Geoffrey E Hinton, Alex Krizhevsky, and Sida D Wang. Transforming auto-encoders. In *International conference on artificial neural networks*, pages 44–51. Springer, 2011.

[100] Tin Kam Ho. Random decision forests. In *Proceedings of 3rd international conference on document analysis and recognition*, volume 1, pages 278–282. IEEE, 1995.

[101] Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural computation*, 9(8):1735–1780, 1997.

[102] Paul W Holland and Samuel Leinhardt. The statistical analysis of local structure in social networks. Technical report, National Bureau of Economic Research, 1974.

[103] Petter Holme and Jari Saramäki. Temporal networks. *Physics reports*, 519(3):97–125, 2012.

[104] Weihua Hu, Bowen Liu, Joseph Gomes, Marinka Zitnik, Percy Liang, Vijay Pande, and Jure Leskovec. Pre-training graph neural networks. *arXiv preprint arXiv:1905.12265*, 2019.

[105] Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. In *International Conference on Machine Learning*, pages 448–456, 2015.

[106] Matthew O Jackson. *Social and economic networks*. Princeton university press, 2010.

[107] Philip T Jackson, Amir Atapour-Abarghouei, Stephen Bonner, Toby P Breckon, and Boguslaw Obara. Style augmentation: Data augmentation via style randomization. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition Workshops*, pages 83–92, 2019.

[108] Katarzyna Janocha and Wojciech Marian Czarnecki. On loss functions for deep neural networks in classification. *arXiv preprint arXiv:1702.05659*, 2017.

[109] Zhe Jia, Blake Tillman, Marco Maggioni, and Daniele Paolo Scarpazza. Dissecting the graphcore ipu architecture via microbenchmarking. *arXiv preprint arXiv:1912.03413*, 2019.

[110] Ian T Jolliffe and Jorge Cadima. Principal component analysis: a review and recent developments. *Philosophical Transactions of the Royal Society A: Mathematical, Physical and Engineering Sciences*, 374(2065):20150202, 2016.

[111] Daniel Justus, John Brennan, Stephen Bonner, and Andrew Stephen McGough. Predicting the computational cost of deep learning models. In *IEEE International Conference on Big Data*, pages 3873–3882. IEEE, 2018.

[112] Marcus Kaiser, Matthias Goerner, and Claus C Hilgetag. Criticality of spreading dynamics in hierarchical cluster networks without inhibition. *New Journal of Physics*, 9(5):110, 2007.

[113] Marcus Kaiser and Claus C Hilgetag. Spatial growth of real-world networks. *Physical Review E*, 69(3):036103, 2004.

[114] Marcus Kaiser, Claus C Hilgetag, and Rolf Kötter. Hierarchy and dynamics of neural networks. *Frontiers in neuroinformatics*, 4:112, 2010.

[115] U Kang, Spiros Papadimitriou, Jimeng Sun, and Hanghang Tong. Centralities in large networks: Algorithms and observations. In *Proceedings of the 2011 SIAM international conference on data mining*, pages 119–130. SIAM, 2011.

[116] U Kang, Charalampos E Tsourakakis, and Christos Faloutsos. Pegasus: A peta-scale graph mining system implementation and observations. In *IEEE International Conference on Data Mining*, pages 229–238. IEEE, 2009.

[117] Burcu Kantarci and Vincent Labatut. Classification of complex networks based on topological properties. In *2013 International Conference on Cloud and Green Computing*, pages 297–304. IEEE, 2013.

[118] Grigoris I Karakoulas and John Shawe-Taylor. Optimizing classifers for imbalanced training sets. *Advances in Neural Information Processing Systems*, pages 253–259, 1999.

[119] Brian Karrer and Mark EJ Newman. Stochastic blockmodels and community structure in networks. *Physical review E*, 83(1):016107, 2011.

[120] Steven Kearnes, Kevin McCloskey, Marc Berndl, Vijay Pande, and Patrick Riley. Molecular graph convolutions: moving beyond fingerprints. *Journal of computer-aided molecular design*, 30(8):595–608, 2016.

[121] Nikhil S Ketkar, Lawrence B Holder, and Diane J Cook. Empirical comparison of graph classification algorithms. In *2009 IEEE Symposium on Computational Intelligence and Data Mining*, pages 259–266. IEEE, 2009.

[122] Samina Khalid, Tehmina Khalil, and Shamila Nasreen. A survey of feature selection and feature extraction techniques in machine learning. In *2014 Science and Information Conference*, pages 372–378. IEEE, 2014.

[123] Arijit Khan and Sameh Elnikety. Systems for big-graphs. *Proceedings of the VLDB Endowment*, 7(13):1709–1710, 2014.

[124] Megha Khosla, Avishek Anand, and Vinay Setty. A comprehensive comparison of unsupervised network representation learning methods. *arXiv preprint arXiv:1903.07902*, 2019.

[125] Yoon Kim, Yi-I Chiu, Kentaro Hanaki, Darshan Hegde, and Slav Petrov. Temporal analysis of language through neural language models. In *Proceedings of the ACL 2014 Workshop on Language Technologies and Computational Social Science*, pages 61–65, 2014.

[126] Diederik P Kingma and Max Welling. Auto-encoding variational bayes. *arXiv preprint arXiv:1312.6114*, 2013.

[127] Thomas N Kipf and Max Welling. Variational graph auto-encoders. *arXiv preprint arXiv:1611.07308*, 2016.

[128] Thomas N. Kipf and Max Welling. Semi-supervised classification with graph convolutional networks. *International Conference on Learning Representations (ICLR)*, 2017.

[129] DJ Klein. Centrality measure in graphs. *Journal of mathematical chemistry*, 47(4):1209–1223, 2010.

[130] Eric D. Kolaczyk. *Statistical Analysis of Network Data: Methods and Models*. Springer Publishing Company, Incorporated, 1st edition, 2009.

[131] Giorgos Kollias, Madan Sathe, Olaf Schenk, and Ananth Grama. Fast parallel algorithms for graph similarity and matching. *Journal of Parallel and Distributed Computing*, 74(5):2400–2410, 2014.

[132] Xiangnan Kong and S Yu Philip. gmlc: a multi-label feature selection framework for graph classification. *Knowledge and Information Systems*, 31(2):281–305, 2012.

[133] Danai Koutra, Ankur Parikh, Aaditya Ramdas, and Jing Xiang. Algorithms for graph similarity and subgraph matching. In *Proc. Ecol. Inference Conf*, volume 17, 2011.

[134] Danai Koutra, Joshua T Vogelstein, and Christos Faloutsos. Deltacon: A principled massive-graph similarity function. In *Proceedings of the 2013 SIAM International Conference on Data Mining*, pages 162–170. SIAM, 2013.

[135] István A. Kovács, Katja Luck, Kerstin Spirohn, Yang Wang, Carl Pollis, Sadie Schlabach, Wenting Bian, Dae-Kyum Kim, Nishka Kishore, Tong Hao, Michael A. Calderwood, Marc Vidal, and Albert-László Barabási. Network-based prediction of protein interactions. *Nature communications*, 10(1):1–8, 2019.

[136] Nils M Kriege, Fredrik D Johansson, and Christopher Morris. A survey on graph kernels. *arXiv preprint arXiv:1903.11835*, 2019.

[137] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. In *Advances in neural information processing systems*, pages 1097–1105, 2012.

[138] Jérôme Kunegis. Konect: the koblenz network collection. In *Proceedings of the 22nd International Conference on World Wide Web*, pages 1343–1350. ACM, 2013.

[139] Alexandros Labrinidis and Hosagrahar V Jagadish. Challenges and opportunities with big data. *Proceedings of the VLDB Endowment*, 5(12):2032–2033, 2012.

[140] Hoang Thanh Lam, Johann-Michael Thiebaut, Mathieu Sinn, Bei Chen, Tiep Mai, and Oznur Alkan. One button machine for automating feature engineering in relational databases. *arXiv preprint arXiv:1706.00327*, 2017.

[141] Godfrey N Lance and William T Williams. Mixed-data classificatory programs i - agglomerative systems. *Australian Computer Journal*, 1(1):15–20, 1967.

[142] Yann LeCun, Yoshua Bengio, and Geoffrey Hinton. Deep learning. *nature*, 521(7553):436–444, 2015.

[143] Yann LeCun, Léon Bottou, Yoshua Bengio, and Patrick Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998.

[144] Kai Lei, Meng Qin, Bo Bai, Gong Zhang, and Min Yang. Gcn-gan: A non-linear temporal link prediction model for weighted dynamic networks. In *IEEE INFOCOM 2019-IEEE Conference on Computer Communications*, pages 388–396. IEEE, 2019.

[145] Jure Leskovec, Jon Kleinberg, and Christos Faloutsos. Graph evolution: Densification and shrinking diameters. *ACM transactions on Knowledge Discovery from Data (TKDD)*, 1(1):2–es, 2007.

[146] Jure Leskovec and Andrej Krevl. SNAP Datasets: Stanford large network dataset collection. http://snap.stanford.edu/data, 2014.

[147] Jure Leskovec and Rok Sosič. Snap: A general-purpose network analysis and graph-mining library. *ACM Transactions on Intelligent Systems and Technology (TIST)*, 2016.

[148] Jurij Leskovec, Deepayan Chakrabarti, Jon Kleinberg, and Christos Faloutsos. Realistic, mathematically tractable graph generation and evolution, using kronecker multiplication. In *European conference on principles of data mining and knowledge discovery*, pages 133–145. Springer, 2005.

[149] Omer Levy and Yoav Goldberg. Neural word embedding as implicit matrix factorization. In *Advances in neural information processing systems*, pages 2177–2185, 2014.

[150] Cheng Li, Xiaoxiao Guo, and Qiaozhu Mei. Deepgraph: graph structure predicts network growth. *arXiv preprint arXiv:1610.06251*, 2016.

[151] Geng Li, Murat Semerci, Bulent Yener, and Mohammed J Zaki. Graph classification via topological and label attributes. In *Proceedings of the 9th international workshop on mining and learning with graphs (MLG), San Diego, USA*, volume 2, 2011.

[152] Geng Li, Murat Semerci, Bülent Yener, and Mohammed J Zaki. Effective graph classification based on topological and label attributes. *Statistical Analysis and Data Mining: The ASA Data Science Journal*, 5(4):265–283, 2012.

[153] Jiwei Li, Xinlei Chen, Eduard Hovy, and Dan Jurafsky. Visualizing and understanding neural models in nlp. *arXiv preprint arXiv:1506.01066*, 2015.

[154] Xiang Li, Tao Qin, Jian Yang, and Tie-Yan Liu. Lightrnn: Memory and computation-efficient recurrent neural networks. In *Advances in Neural Information Processing Systems*, pages 4385–4393, 2016.

[155] Yaguang Li, Rose Yu, Cyrus Shahabi, and Yan Liu. Diffusion convolutional recurrent neural network: Data-driven traffic forecasting. *arXiv preprint arXiv:1707.01926*, 2017.

[156] Weiyi Liu, Pin-Yu Chen, Fucai Yu, Toyotaro Suzumura, and Guangmin Hu. Learning graph topological features via gan. *IEEE Access*, 7:21834–21843, 2019.

[157] Laurens van der Maaten and Geoffrey Hinton. Visualizing data using t-sne. *Journal of Machine Learning Research*, 9(Nov):2579–2605, 2008.

[158] SM Mahantesh, Sudarshan Iyengar, M Vijesh, Shruthi R Nayak, and Nikitha Shenoy. Prediction of arrival of nodes in a scale free network. In *Proceedings of the 2012 International Conference on Advances in Social Networks Analysis and Mining (ASONAM 2012)*, pages 517–521. IEEE Computer Society, 2012.

[159] Grzegorz Malewicz, Matthew H Austern, Aart JC Bik, James C Dehnert, Ilan Horn, Naty Leiser, and Grzegorz Czajkowski. Pregel: a system for large-scale graph processing. In *Proceedings of the 2010 ACM SIGMOD International Conference on Management of data*, pages 135–146. ACM, 2010.

[160] Franco Manessi, Alessandro Rozza, and Mario Manzo. Dynamic graph convolutional networks. *Pattern Recognition*, page 107000, 2019.

[161] Víctor Martínez, Fernando Berzal, and Juan-Carlos Cubero. A survey of link prediction in complex networks. *ACM Computing Surveys*, 49(4):69, 2017.

[162] Robert Ryan McCune, Tim Weninger, and Greg Madey. Thinking like a vertex: a survey of vertex-centric frameworks for large-scale distributed graph processing. *ACM Computing Surveys (CSUR)*, 48(2):1–39, 2015.

[163] Scott Menard. *Applied logistic regression analysis*, volume 106. Sage, 2002.

[164] Tomas Mikolov, Kai Chen, Greg Corrado, and Jeffrey Dean. Efficient estimation of word representations in vector space. *International Conference on Learning Representations (ICLR)*, 2013.

[165] Tomas Mikolov, Ilya Sutskever, Kai Chen, Greg S Corrado, and Jeff Dean. Distributed representations of words and phrases and their compositionality. In *Advances in neural information processing systems*, pages 3111–3119, 2013.

[166] Tom M Mitchell. Machine learning, 1997.

[167] Luis G Moyano. Learning network representations. *The European Physical Journal Special Topics*, 226(3):499–518, 2017.

[168] Tamara Munzner. Exploring large graphs in 3d hyperbolic space. *IEEE Computer Graphics and Applications*, 1998.

[169] Marcell Nagy and Roland Molontay. On the structural properties of social networks and their measurement-calibrated synthetic counterparts. *arXiv preprint arXiv:1908.08429*, 2019.

[170] Mark Newman. *Networks: an introduction*. Oxford university press, 2010.

[171] Mark EJ Newman. The structure and function of complex networks. *SIAM review*, 45(2):167–256, 2003.

[172] Giang Hoang Nguyen, John Boaz Lee, Ryan A Rossi, Nesreen K Ahmed, Eunyee Koh, and Sungchul Kim. Continuous-time dynamic network embeddings. In *3rd International Workshop on Learning Representations for Big Networks (WWW BigNet)*, 2018.

[173] Maximillian Nickel and Douwe Kiela. Poincaré embeddings for learning hierarchical representations. In *Advances in neural information processing systems*, pages 6338–6347, 2017.

[174] Mathias Niepert, Mohamed Ahmed, and Konstantin Kutzkov. Learning convolutional neural networks for graphs. In *International conference on machine learning*, pages 2014–2023, 2016.

[175] Boguslaw Obara, Vicente Grau, and Mark D Fricker. A bioimage informatics approach to automatically extract complex fungal networks. *Bioinformatics*, 28(18):2374–2381, 2012.

[176] Aaron van den Oord, Sander Dieleman, Heiga Zen, Karen Simonyan, Oriol Vinyals, Alex Graves, Nal Kalchbrenner, Andrew Senior, and Koray Kavukcuoglu. Wavenet: A generative model for raw audio. *arXiv preprint arXiv:1609.03499*, 2016.

[177] Mingdong Ou, Peng Cui, Jian Pei, Ziwei Zhang, and Wenwu Zhu. Asymmetric transitivity preserving graph embedding. *International Conference on Knowledge Discovery and Data Mining*, pages 1105–1114, 2016.

[178] Lawrence Page, Sergey Brin, Rajeev Motwani, and Terry Winograd. The pagerank citation ranking: Bringing order to the web. Technical report, Stanford InfoLab, 1999.

[179] Shirui Pan and Xingquan Zhu. Graph classification with imbalanced class distributions and noise. In *Twenty-Third International Joint Conference on Artificial Intelligence*, 2013.

[180] Supriya Pandhre, Himangi Mittal, Manish Gupta, and Vineeth N Balasubramanian. Stwalk: learning trajectory representations in temporal graphs. In *Proceedings of the ACM India Joint International Conference on Data Science and Management of Data*, pages 210–219. ACM, 2018.

[181] Panagiotis Papadimitriou, Ali Dasdan, and Hector Garcia-Molina. Web graph similarity for anomaly detection. *Journal of Internet Services and Applications*, 1(1):19–30, 2010.

[182] Aldo Pareja, Giacomo Domeniconi, Jie Chen, Tengfei Ma, Toyotaro Suzumura, Hiroki Kanezashi, Tim Kaler, and Charles E Leisersen. Evolvegcn: Evolving graph convolutional networks for dynamic graphs. *arXiv preprint arXiv:1902.10191*, 2019.

[183] Adam Paszke, Sam Gross, Soumith Chintala, Gregory Chanan, Edward Yang, Zachary DeVito, Zeming Lin, Alban Desmaison, Luca Antiga, and Adam Lerer. Automatic differentiation in pytorch. 2017.

[184] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Köpf, Edward Yang, Zach DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, and Soumith Chintala. Pytorch: An imperative style, high-performance deep learning library. In *Advances in Neural Information Processing Systems*, pages 8024–8035, 2019.

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

[186] Bryan Perozzi, Rami Al-Rfou, and Steven Skiena. Deepwalk: Online learning of social representations. In *Proceedings of the 20th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 701–710. ACM, 2014.

[187] Nataša Pržulj. Biological network comparison using graphlet degree distribution. *Bioinformatics*, 23(2):e177–e183, 2007.

[188] Wullianallur Raghupathi and Viju Raghupathi. Big data analytics in healthcare: promise and potential. *Health information science and systems*, 2(1):3, 2014.

[189] Erzsébet Ravasz and Albert-László Barabási. Hierarchical organization in complex networks. *Physical review E*, 67(2):026112, 2003.

[190] Erzsébet Ravasz, Anna Lisa Somera, Dale A Mongru, Zoltán N Oltvai, and A-L Barabási. Hierarchical organization of modularity in metabolic networks. *science*, 297(5586):1551–1555, 2002.

[191] Daniel A Reed and Jack Dongarra. Exascale computing and big data. *Communications of the ACM*, 58(7):56–68, 2015.

[192] Marco Tulio Ribeiro, Sameer Singh, and Carlos Guestrin. " why should i trust you?" explaining the predictions of any classifier. In *Proceedings of the 22nd ACM SIGKDD international conference on knowledge discovery and data mining*, pages 1135–1144, 2016.

[193] Francisco Aparecido Rodrigues. Network centrality: an introduction. In *A Mathematical Modeling Approach from Nonlinear Dynamics to Complex Systems*, pages 177–196. Springer, 2019.

[194] David Rohde, Stephen Bonner, Travis Dunlop, Flavian Vasile, and Alexandros Karatzoglou. Recogym: A reinforcement learning environment for the problem of product recommendation in online advertising. *REVAL Workshop at the 12th ACM Conference on Recommender Systems*, 2018.

[195] Frank Rosenblatt. Principles of neurodynamics. perceptrons and the theory of brain mechanisms. Technical report, Cornell Aeronautical Lab Inc Buffalo NY, 1961.

[196] Ryan A. Rossi and Nesreen K. Ahmed. The network data repository with interactive graph analytics and visualization. *AAAI Conference on Artificial Intelligence*, 2015.

[197] Ryan A Rossi and Nesreen K Ahmed. Complex networks are structurally distinguishable by domain. *Social Network Analysis and Mining*, 9(1):51, 2019.

[198] Ryan A Rossi, Rong Zhou, and Nesreen K Ahmed. Estimation of graphlet counts in massive networks. *IEEE transactions on neural networks and learning systems*, 30(1):44–57, 2018.

[199] Matthieu Roy, Stefan Schmid, and Gilles Tredan. Modeling and measuring graph similarity: The case for centrality distance. In *Proceedings of the 10th ACM international workshop on Foundations of mobile computing*, pages 47–52. ACM, 2014.

[200] David E Rumelhart, Geoffrey E Hinton, and Ronald J Williams. Learning representations by back-propagating errors. *Nature*, 323(6088):533–536, 1986.

[201] David E Rumelhart, Geoffrey E Hinton, and Ronald J Williams. Learning representations by back-propagating errors. *Cognitive modeling*, 5(3):1, 1988.

[202] Otmane Sakhi, Stephen Bonner, David Rohde, and Flavian Vasile. Reconsidering analytical variational bounds for output layers of deep networks. *Bayesian Deep Learning - Neural Information Processing Systems workshop*, 2019.

[203] Ruslan Salakhutdinov and Geoffrey Hinton. Semantic hashing. *International Journal of Approximate Reasoning*, 50(7):969–978, 2009.

[204] Fatemeh Salehi Rizi and Michael Granitzer. Properties of vector embeddings in social networks. *Algorithms*, 10(4):109, 2017.

[205] Tobias Schnabel, Igor Labutov, David Mimno, and Thorsten Joachims. Evaluation methods for unsupervised word embeddings. *Conference on Empirical Methods in Natural Language Processing*, pages 298–307, 2015.

[206] Bernhard Schölkopf. The kernel trick for distances. In *Advances in neural information processing systems*, pages 301–307, 2001.

[207] Ramprasaath R Selvaraju, Michael Cogswell, Abhishek Das, Ramakrishna Vedantam, Devi Parikh, and Dhruv Batra. Grad-cam: Visual explanations from deep networks via gradient-based localization. In *Proceedings of the IEEE international conference on computer vision*, pages 618–626, 2017.

[208] Youngjoo Seo, Michaël Defferrard, Pierre Vandergheynst, and Xavier Bresson. Structured sequence modeling with graph convolutional recurrent networks. In *International Conference on Neural Information Processing*, pages 362–373. Springer, 2018.

[209] Nino Shervashidze, Pascal Schweitzer, Erik Jan van Leeuwen, Kurt Mehlhorn, and Karsten M Borgwardt. Weisfeiler-lehman graph kernels. *Journal of Machine Learning Research*, 12(Sep):2539–2561, 2011.

[210] Shaohuai Shi, Qiang Wang, Pengfei Xu, and Xiaowen Chu. Benchmarking state-of-the-art deep learning software tools. *arXiv preprint arXiv:1608.07249*, 2016.

[211] Xuanhua Shi, Zhigao Zheng, Yongluan Zhou, Hai Jin, Ligang He, Bo Liu, and Qiang-Sheng Hua. Graph processing on gpus: A survey. *ACM Computing Surveys (CSUR)*, 50(6):1–35, 2018.

[212] Jeremy G. Siek, Lie-Quan Lee, and Andrew Lumsdaine. The Boost Graph Library (BGL), 2016.

[213] Giannis Siglidis, Giannis Nikolentzos, Stratis Limnios, Christos Giatsidis, Konstantinos Skianis, and Michalis Vazirgiannis. Grakel: A graph kernel library in python. *arXiv preprint arXiv:1806.02193*, 2018.

[214] Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Dropout: a simple way to prevent neural networks from overfitting. *The journal of machine learning research*, 15(1):1929–1958, 2014.

[215] Michael PH Stumpf and Mason A Porter. Critical truths about power laws. *Science*, 335(6069):665–666, 2012.

[216] Mahito Sugiyama and Karsten Borgwardt. Halting in random walk kernels. In *Advances in neural information processing systems*, pages 1639–1647, 2015.

[217] Ilya Sutskever, James Martens, and Geoffrey E Hinton. Generating text with recurrent neural networks. In *Proceedings of the 28th International Conference on Machine Learning (ICML-11)*, pages 1017–1024, 2011.

[218] Chuanqi Tan, Fuchun Sun, Tao Kong, Wenchang Zhang, Chao Yang, and Chunfang Liu. A survey on deep transfer learning. In *International conference on artificial neural networks*, pages 270–279. Springer, 2018.

[219] Lei Tang and Huan Liu. Relational learning via latent social dimensions. In *Proceedings of the 15th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 817–826. ACM, 2009.

[220] Lei Tang and Huan Liu. Leveraging social media networks for classification. *Data Mining and Knowledge Discovery*, 23(3):447–478, 2011.

[221] Marisa Thoma, Hong Cheng, Arthur Gretton, Jiawei Han, Hans-Peter Kriegel, Alex Smola, Le Song, Philip S Yu, Xifeng Yan, and Karsten M Borgwardt. Discriminative frequent subgraph mining with optimality guarantees. *Statistical Analysis and Data Mining: The ASA Data Science Journal*, 3(5):302–318, 2010.

[222] Tijmen Tieleman and Geoffrey Hinton. Lecture 6.5-rmsprop: Divide the gradient by a running average of its recent magnitude. *COURSERA: Neural networks for machine learning*, 4(2):26–31, 2012.

[223] I Tolstikhin, O Bousquet, S Gelly, and B Schölkopf. Wasserstein auto-encoders. In *International Conference on Learning Representations*, 2018.

[224] Rakshit Trivedi, Mehrdad Farajtbar, Prasenjeet Biswal, and Hongyuan Zha. Representation learning over dynamic graphs. *arXiv preprint arXiv:1803.04051*, 2018.

[225] Flavian Vasile, Elena Smirnova, and Alexis Conneau. Meta-prod2vec: Product embeddings using side-information for recommendation. In *Proceedings of the 10th ACM Conference on Recommender Systems*, pages 225–232, 2016.

[226] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. In *Advances in neural information processing systems*, pages 5998–6008, 2017.

[227] S Vichy N Vishwanathan, Nicol N Schraudolph, Risi Kondor, and Karsten M Borgwardt. Graph kernels. *Journal of Machine Learning Research*, 11(Apr):1201–1242, 2010.

[228] Cynthia Wagner, Gerard Wagener, Radu State, and Thomas Engel. Malware analysis with graph kernels and support vector machines. In *2009 4th International Conference on Malicious and Unwanted Software (MALWARE)*, pages 63–68. IEEE, 2009.

[229] Daixin Wang, Peng Cui, and Wenwu Zhu. Structural deep network embedding. In *Proceedings of the 22nd ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 1225–1234, 2016.

[230] Hongwei Wang, Jia Wang, Jialin Wang, Miao Zhao, Weinan Zhang, Fuzheng Zhang, Xing Xie, and Minyi Guo. Graphgan: Graph representation learning with generative adversarial nets. In *Thirty-second AAAI conference on artificial intelligence*, 2018.

[231] Duncan J Watts and Steven H Strogatz. Collective dynamics of 'small-world'networks. *nature*, 393(6684):440, 1998.

[232] Svante Wold, Kim Esbensen, and Paul Geladi. Principal component analysis. *Chemometrics and intelligent laboratory systems*, 2(1-3):37–52, 1987.

[233] Zhenqin Wu, Bharath Ramsundar, Evan N Feinberg, Joseph Gomes, Caleb Geniesse, Aneesh S Pappu, Karl Leswing, and Vijay Pande. Moleculenet: a benchmark for molecular machine learning. *Chemical science*, 9(2):513–530, 2018.

[234] Zonghan Wu, Shirui Pan, Fengwen Chen, Guodong Long, Chengqi Zhang, and Philip S Yu. A comprehensive survey on graph neural networks. *arXiv preprint arXiv:1901.00596*, 2019.

[235] Reynold S Xin, Joseph E Gonzalez, Michael J Franklin, and Ion Stoica. Graphx: A resilient distributed graph system on spark. In *First international workshop on graph data management experiences and systems*, pages 1–6, 2013.

[236] Dongkuan Xu and Yingjie Tian. A comprehensive survey of clustering algorithms. *Annals of Data Science*, 2(2):165–193, 2015.

[237] Rendong Yang, Yun Bai, Zhaohui Qin, and Tianwei Yu. Egonet: identification of human disease ego-network modules. *BMC genomics*, 15(1):314, 2014.

[238] Huaxiu Yao, Xianfeng Tang, Hua Wei, Guanjie Zheng, and Zhenhui Li. Revisiting spatial-temporal similarity: A deep learning framework for traffic prediction. In *AAAI Conference on Artificial Intelligence*, 2019.

[239] Yibo Yao and Lawrence Holder. Scalable svm-based classification in dynamic graphs. In *2014 IEEE international conference on Data Mining*, pages 650–659. IEEE, 2014.

[240] Rex Ying, Ruining He, Kaifeng Chen, Pong Eksombatchai, William L Hamilton, and Jure Leskovec. Graph convolutional neural networks for web-scale recommender systems. *arXiv preprint arXiv:1806.01973*, 2018.

[241] Tom Young, Devamanyu Hazarika, Soujanya Poria, and Erik Cambria. Recent trends in deep learning based natural language processing. *ieee Computational intelligenCe magazine*, 13(3):55–75, 2018.

[242] Wenchao Yu, Wei Cheng, Charu C Aggarwal, Kai Zhang, Haifeng Chen, and Wei Wang. Netwalk: A flexible deep embedding approach for anomaly detection in dynamic networks. In *Iternational Conference on Knowledge Discovery & Data Mining*, pages 2672–2681. ACM, 2018.

[243] Wenchao Yu, Cheng Zheng, Wei Cheng, Charu C Aggarwal, Dongjin Song, Bo Zong, Haifeng Chen, and Wei Wang. Learning deep network representations with adversarially regularized autoencoders. In *International Conference on Knowledge Discovery & Data Mining*, pages 2663–2671. ACM, 2018.

[244] Laura A Zager and George C Verghese. Graph similarity scoring and matching. *Applied mathematics letters*, 21(1):86–94, 2008.

[245] Matei Zaharia, Mosharaf Chowdhury, Michael J Franklin, Scott Shenker, and Ion Stoica. Spark: Cluster computing with working sets. *HotCloud*, 10(10-10):95, 2010.

[246] Matthew D Zeiler and Rob Fergus. Visualizing and understanding convolutional networks. *European conference on computer vision*, pages 818–833, 2014.

[247] Daokun Zhang, Jie Yin, Xingquan Zhu, and Chengqi Zhang. Network representation learning: a survey. *arXiv preprint arXiv:1801.05852*, 2017.

[248] Qizhen Zhang, Da Yan, and James Cheng. Quegel: a general-purpose system for querying big graphs. In *Proceedings of the 2016 International Conference on Management of Data*, pages 2189–2192, 2016.

[249] Quan-shi Zhang and Song-Chun Zhu. Visual interpretability for deep learning: a survey. *Frontiers of Information Technology & Electronic Engineering*, 19(1):27–39, 2018.

[250] Yanxia Zhang and Yongheng Zhao. Astronomy in the big data era. *Data Science Journal*, 14, 2015.

[251] Ye Zhang, Md Mustafizur Rahman, Alex Braylan, Brandon Dang, Heng-Lu Chang, Henna Kim, Quinten McNamara, Aaron Angert, Edward Banner, Vivek Khetan, Tyler McDonnell, An Thanh Nguyen, Dan Xu, Byron C. Wallace, and Matthew Lease. Neural information retrieval: A literature review. *arXiv preprint arXiv:1611.06792*, 2016.

[252] Zheng Zhang, Yong Xu, Jian Yang, Xuelong Li, and David Zhang. A survey of sparse representation: algorithms and applications. *IEEE access*, 3:490–530, 2015.

[253] Jie Zhou, Ganqu Cui, Zhengyan Zhang, Cheng Yang, Zhiyuan Liu, Lifeng Wang, Changcheng Li, and Maosong Sun. Graph neural networks: A review of methods and applications. *arXiv preprint arXiv:1812.08434*, 2018.

[254] Lekui Zhou, Yang Yang, Xiang Ren, Fei Wu, and Yueting Zhuang. Dynamic network embedding by modeling triadic closure process. *Conference on Artificial Intelligence*, 2018.

Appendix A

GFP-X Parallel Implementation

A.1 Apache Spark and GraphX

Apache Spark is a general-purpose parallel computing framework for processing massive datasets [245], the core of which is the Resilient Distributed Dataset (RDD) abstraction. An RDD is a read only collection of data partitioned across a set of Spark cluster machines and cached in memory. The RDD concept has further been expanded via the higher-level DataFrames, which arrange the distributed collection of data into labelled columns similar to a traditional relational database [12].

GraphX is a system for processing of graph datasets using Spark [235]. It includes a variant of Google's Pregel API – the first of the 'Think Like A Vertex' (TLAV), designed to bring the scalability of a Map / Reduce like system to graph processing [162]. Graphs are represented as specialised versions of RDD's and thus can be parallelised across a cluster. GraphX includes a selection of primitive graph algorithms including connected components and triangle count but additional algorithms must be implemented by the end user using one of the available GraphX graph traversal API's: Pregel and Aggregate Messages.

A.2 Parallel Feature Extraction

Both the GFP-X and GFP-C approaches are written in Scala for the Apache Spark GraphX package. Spark was chosen due to it's ability to scale across a distributed environment and it's use of in-memory computation. As the main goal of GFP-X was scaleability, using Spark allowed this to be achieved. GraphX offers a range of implicit functions for extracting common features from a graph, such as triangle counting, PageRank and connected components – where ever possible, these methods were utilised. Any features not provided by GraphX must be implemented via one of the available graph traversal algorithms. To implementing the non-implicit features for GFP-X, the Aggregate Messages API was utilised. Previous research has shown that key statistics about a vertex neighbourhood [25] can be very powerful in it's identification. The Aggregate Messages API passes information from a vertex to all it's neighbours and can be considered conceptually as Map / Reduce for graphs [235]. To use the Aggregate Messages API, a send message and merge message function must be created to perform the desired computation. The send message function, analogous to a Map, controls what message is sent by every vertex within a graph. The merge message, analogous to a Reduce, controls the aggregation of multiple messages arriving at the same vertex to create a single result. This process is performed in parallel across the Spark cluster.

The Aggregate Messages API is used in three of the vertex features for GFP-X; the mean PageRank score, number of two hop away neighbours and mean local clustering score for a vertex's neighbourhood. To capture the mean PageRank score for a vertex's neighbourhood, the PageRank score, computed for each vertex using the implicit GraphX function, is used as the attribute to be passed in the send message — as well as a counter variable. This results in each vertices PageRank score being sent to all it's neighbours. The merge message function then sums the incoming PageRank score messages at each vertex and divides by the total number of counters received, resulting in each vertex having the mean PageRank score for it's neighbourhood. The methodology is a generalised way of capturing the mean of any vertex feature across it's neighbourhood — using it also for the mean neighbourhood local clustering score and number of two hop away neighbours. This method is extremely efficient and is fully parallelised across a cluster. The method could be expanded to aggregate a feature from multiple hops away from a vertex, capturing information about it's extended neighbourhood, using multiple iterations of the send-merge process.

All the features for GFP-X and their extraction method are detailed in Table A.1. Each

vertex feature is returned as a VertexRDD, containing the vertex ID and the feature value. The global features are returned as a single DataFrame containing all global feature values. In order to scale to massive graphs, even when running on a single machine, memory management is a key concern. Spark allows data to be cached in memory to improve application performance, but programs can be unstable if the data requirements exceeds the amount of available memory. Due to this, we allowed the graph to cache to disk if memory space is limited. To improve the memory footprint of GFP-X, each feature is extracted and then immediately aggregated so that the original VertexRDD can be removed from memory.

A.3 Parallel Graph Comparison

The function for feature creation utilises the Spark DataFrame API, which allows for each set of vertex features to be aggregated efficiently and in parallel using the implicit statistics functionality. Once all the features have been aggregated, they are joined to create a vertex and global feature vector, both of which are stored as DataFrames. To compute the similarity between two graphs, these feature vectors are compared using the Canberra distance which has been implemented using the RDD API. The two vectors being compared are first joined together, then a single Map / Reduce iteration can be used to compute the distance. In the Map phase, the absolute difference between each vector elements is divided by their absolute sum. These results are then summed in the Reduce phase. Using Apache Spark for all components, not just the graph feature extraction, of GFP-X and GFP-C, ensures that they will still be scaleable as graph datasets continue to grow.

The GFP-X and GFP-C frameworks have been open sourced under a GPLv3 licence and are available on GitHub[1]. In addition, the code used to run each experiment, generate the synthetic datasets used and the implementation of NetSimile, written in the Graph-Tool package [63], are also available in the same repository.

[1] https://github.com/sbonner0/GFPX-GraphSimilarity

Table A.1: GFP-X Feature Extraction Method

Feature	Extraction Method
Eigenvector Centrality Value	Extracted using the Sparkling-Graph package [21].
PageRank Score	Extracted using the implicit GraphX method.
Average PageRank of Neighbourhood	Extracted using Aggregate Messages mean neighbourhood method described in section A.2.
Total Degree	Extracted by counting the number of vertices incident on each vertex.
Two-Hop Away Neighbours	Extracted using the Aggregate Messages methodology by each vertex sending the number of neighbours it has to its neighbourhood.
Local Clustering Score	Extracted via the Sparkling-Graph package.
Average Clustering of Neighbourhood	Extracted using the mean neighbourhood Aggregate Messages method.
Graph Order	Extracted by counting the number of vertices within the VertexRDD.
Graph Size	Extracted by counting the number of edges within the EdgesRDD.
Number of Triangles	Extracted using the implicit GraphX function and a custom Map / Reduce function.
Number of Components	Extracted via the implicit GraphX function.
Number of Vertices In Largest Component	Extracted via a custom Map / Reduce method.