NTNU
Norwegian University of
Science and Technology

Defining Roles in Transaction Networks Using Deep Learning

Eirik Vikanes

Master of Science in Informatics
Submission date: June 2018
Supervisor: Massimiliano Ruocco, IDI

Norwegian University of Science and Technology
Department of Computer Science

Abstract

Recent years have seen an immense increase in the amount of data our generated by humans. This data takes many different shapes and forms, and one of the types we find most often is in the form of network data. In order to be able to extract valuable insights from these data, we need robust tools to analyze these networks. An area of research that has received a lot of attention recently, has been finding methods that can be used to extract roles from graph data.

In this thesis, we examine an algorithm called struc2vec that embeds a graph into a vector space and how we can use these embeddings to extract roles from the graph. We consider embeddings of various dimensionalities and how the choice of dimensionality impacts how well the embeddings represent the roles in the graph. Where applicable, we use dimensionality reduction techniques to obtain lower-dimensionality embeddings from which we can extract the roles using clustering methods.

Our results show that struc2vec generates embeddings that separate nodes well with regard to their role, and manages to capture several aspects of the graph structure.

Sammendrag

De siste årene har vi sett en drastisk økning i mengden data som produseres av menneskeheten. Disse dataene kommer i mange former, hvorav en av de vanligste er data i form av et nettverk. For å kunne uthente nyttig informasjon fra disse dataene, trenger vi solide verktøy for å analysere disse nettverkene. Et forskningsområde som har fått mye oppmerksomhet, har vært å finne metoder som kan benyttes til å hente ut roller fra grafdata.

I denne oppgaven ser vi på en algoritme kalt struc2vec som omformer grafdata til et sett med vektorer, og vi ser på hvordan disse vektorene kan benyttes til å uthente roller fra den originale grafen. Vi betrakter vektorer av ulik dimensjonalitet, og ser på hvordan valg av dimensjonalitet påvirker hvor godt vektorene beskriver rollene i grafen. Der det er nødvendig, vil vi bruke dimensjonalitetsreduksjonsteknikker for anskaffe vektorer av lavere dimensjonalitet som vi kan hente ut roller fra.

Resultatene våre viser at struc2vec produserer sett med vektorer som klarer å separere noder godt med tanke på deres rolle, og klarer å fange ulike aspekter ved grafens struktur.

Preface

This thesis represents the culmination of a master's degree in Artificial Intelligence as part of the Informatics study program at the Department of Computer Science (IDI) at the Norwegian University of Science and Technology (NTNU). The following project was carried out over a period of 9 months, beginning in September 2017 and ending early June 2018. The idea for this topic came from Telenor ASA.

I would very much like to thank Geoffrey Canright, Arjun Chandra and Massimiliano Ruocco for invaluable help and helpful discussions throughout the project.

Table of Contents

List of Tables

List of Figures

Abbreviations

CC	=	Clustering coefficient
SNE	=	Stochastic Neighbor Embedding
t-SNE	=	t-Stochastic Neighbor Embedding
DBSCAN	=	Density-Based Spatial Clustering of Applications With Noise
D_{KL}	=	Kullback-Leibler Divergence
CBOW	=	Continuous Bag of Words
LLE	=	Locally Linear Embedding

Chapter 1

Introduction

Over the last decade, improvements in the speed and efficiency of computers have enabled us to build ever smaller devices. Today, portable devices like smartphones, smartwatches, and fitness armbands are ubiquitous, while sensors and smart devices, under the umbrella term ”Internet of Things” (IoT) are rapidly connecting our homes to the world. All of these devices constantly generate data that needs to be stored and perhaps analyzed.

In the same timeframe, we have been witness to a revolution within the field of machine learning. Much of the theoretical foundation for this revolution were already laid a decade ago, but advances within the field of parallel computing (much thanks to the gaming industry) along with new insights have enabled researchers to analyze data on a scale that few could have predicted ten years ago.

Companies like Google and Facebook have sunk billions of dollars into research into large-scale data storage and machine learning, and have consequently become tech titans. Many of these recent advances have been within a category of machine learning problems called *supervised learning*. When performing supervised learning, we utilize large amounts of clearly labeled data and find patterns that can be used to infer the label from the data, which in turn can be used to classify unseen samples. On the other hand, when doing *unsupervised learning*, we have no such labeled data and thus these kinds of problems are often considered more challenging to solve.

1.1 Background and Motivation

The enormous amounts of data generated and stored every second of every hour can take many forms. One of these forms is network data. Banks store transaction logs, Facebook possess a massive amount of data describing social interactions between people and even the internet itself constitutes a massive network.

Since these networks are so prevalent in our society, it is imperative that we have the right tools to analyze and understand their structure. The machine learning revolution currently unfolding has provided us with a lot of tools in this regard, but many of these algorithms assume that the input data takes the form of a vector space. Network data does not inherently form such a space, which is why recent research has been focused on finding ways to embed the network data into a vector space, making it easier to use with the aforementioned machine learning algorithms.

In addition to this, these types of data often lack clear labels describing either the type of an actor in the network or the interactions between them. Such unsupervised problems are challenging in several aspects. When using supervised methods, the quality of a solution can be determined by testing the solution on previously unseen data and measuring the difference between the expected outcome and the actual outcome. This is not possible for unsupervised methods, which means we'll have to find other ways to ensure the quality of our solution. Oftentimes, we'll have to rely on domain knowledge and manual verification to be in lieu of the testing that supervised methods utilize.

1.2 Goals and Research Questions

In order to limit the scope of the thesis and clarify its goals, we'll here present the goal for this thesis along with some research questions in order to determine how best to achieve these goals.

The overarching goal of this thesis is to be able to define roles for the nodes of a graph through the use of graph embeddings, which are vector representations of the graph which preserve some desired features of the graph. In order to achieve this, we need to look at state-of-the-art algorithms that handle the problem of embedding graphs into a vector space and then determine the best way for these to be visualized in a way that permits quick role extraction. This leads us to posing the following two research questions:

Research Question 1: What can be done to embed network structures into a lower-dimensional space?

As a graph with n nodes forms an n-dimensional space, performing machine learning directly on the graph can be hard. In order to reduce the cost and also be able to use the vast majority of machine learning tools that work on vectors and matrices rather than graph data, we want do examine what can be done to effectively represent the graph in d dimensions, where $d \ll n$

Research Question 2: Can these embeddings be clustered in a meaningful way?

In order to be able to extract meaningful information from the resulting embeddings, we need to be able to cluster the data in a way that allows us to effectively discover roles of the nodes in the graph. Preferably, this should be done in a way that allows for easy visualization as well.

1.3 Approach

In order to answer these research questions, we will discuss a range of methods that can be used to embed two different graph datasets into a space of lower dimensionality. Then, various clustering methods will be discussed. We will then select a combination of graph embedding algorithms and clustering methods that is likely to facilitate easy role discovery. To test and verify the correctness and robustness of the solutions, a set of metrics will be applied on the original graph data which hopefully will be preserved in the resulting clustering.

1.4 Outline

The structure of this thesis is as follows. Chapter 2 presents methods and concepts within a range of topics that are necessary in order to be able to fully understand the problem and the proposed solutions. Chapter 3 presents several algorithms and methods that constitute the state-of-the-art within the field of graph embedding. Chapter 4 goes into greater detail about which algorithms are suitable for solving our problem, and explains the datasets and metrics used to verify the quality of the results. Chapter 5 presents the various experiments to be conducted. Chapter 6 presents the results from the experiments along

with a discussion of these results. Finally, chapter 7 briefly summarizes the findings from the preceding chapter and describes some problems that are left as future work.

Chapter 2

Background Theory

In order to fully appreciate the problem at hand, some background theory about general concepts and methods are required. Initially, some basic graph theory will be presented to help the reader understand the problem domain. Then, we will point out some challenges that present themselves when dealing with high-dimensional data as well as a way to work around these challenges. Finally, we will clarify some concepts within deep learning and AI for the reader along with some NLP (Natural Language Processing) methods that utilize deep learning.

2.1 Introduction to Graph Theory

2.1.1 What Is a Graph?

A graph $G = (V, E)$ consists of a collection of nodes (or vertices) V and edges E. In layman's terms, the nodes V in a graph are objects that are related through the edges E. A social network is often used as an example of a graph; the people taking part in this network constitute the nodes, while the various types of relationships between them constitute the edges. Each such edge in the graph may be either *directed* or *undirected*. A directed edge forms a one-way relationship in the graph. Elaborating on the social network analogy, a person 'following' another constitutes a directed edge in the social network graph, since it is not given that the person is being 'followed' back. On the other

hand, the typical 'friendship' between two people in the social network graph constitutes an undirected edge since this relationship is a reciprocal one.

In addition to having a direction, each edge is often associated with a cost or a weight. The meaning of this cost is context-specific; it may denote the amount of money paid in a single transaction between two parties, the strength of a friendship in a social network, or the number of minutes it takes to drive from one city to another.

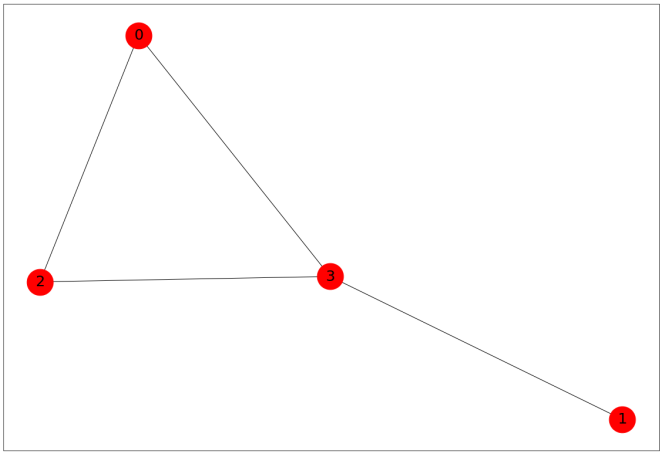

Figure 2.1: A visual representation of an unweighted, undirected graph with four nodes and four edges

2.1.2 Basic Terminology

We say that one node v_i is *adjacent* to another node v_j if there is an edge $\{i, j\}$ connecting them. The number of adjacent nodes of a node is termed the *node degree*. The *neighborhood* of v_i is the set of all nodes that are adjacent to v_i. We can extend this neighborhood outward to include nodes that are further away from the source node. A *2-hop* neighborhood of v_i would include all nodes that are connected to v_i by two edges or fewer.

2.1.3 Graph Representations

There are several ways to represent the nodes and edges in a textual format, each with their own advantages and disadvantages. We will present the most used ones here.

Adjacency Matrices

Adjacency matrices is arguably the most used format to represent graph data. The adjacency matrix for a graph with $N = |V|$ nodes is a matrix M of size NXN. For undirected graphs, M is a symmetric matrix, where each entry $m_{ij} = m_{ji}$ denotes the presence or absence of an edge between node i and j in the graph. A non-zero number for this entry denotes the weight or cost of this edge, while a value of 0 means that there is no edge between node i and j. Unweighted graphs can be viewed as a special case of the weighted graph, where every weight is equal to 1.

$$\begin{bmatrix} 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 1 \\ 1 & 1 & 1 & 0 \end{bmatrix}$$

Figure 2.2: The adjacency matrix for the graph in figure 2.1

Representing a graph as an adjacency matrix has several advantages. For an undirected graph, we can simply sum either the rows or columns to obtain the degree of the node. For directed graphs, the sum of the row gives the out-degree, while the sum of the columns gives the in-degree. Determining the existence of an edge from i to j can be done in constant time, but the memory complexity of storing the adjacency matrix is $O(n^2)$. This means that if the graph is sufficiently large and sparse (i.e. the graph has few edges relative to the number of nodes), another representation should be used.

Adjacency Lists

The adjacency list is another widely used format for representing graphs. For this type of format, each row is on the format $n \rightarrow i_0, i_1, \ldots, i_k$ where n is a node in the graph and $i_0 \ldots i_k$ are all nodes that are adjacent to n. Representing the graph in this way has some advantages over using an adjacency matrix. First, since only the adjacent nodes are stored in the list, the adjacency list is better suited for sparse graphs. However, determining whether there is an edge from node i to node j has a time complexity of $O(|V|)$, but can be improved by keeping each list of adjacent nodes sorted and using binary search.

$$\begin{aligned} 0 &\rightarrow 2, 3 \\ 1 &\rightarrow 3 \\ 2 &\rightarrow 0, 3 \\ 3 &\rightarrow 0, 1, 2 \end{aligned}$$

Figure 2.3: The adjacency list for the graph in 2.1

Laplacian Matrices

A Laplacian matrix is a third format that is used for representing graphs. It is equal to $L = D - A$, where D is the *degree matrix* of the graph and A is the adjacency matrix. The degree matrix of a graph is a diagonal matrix where each entry on the diagonal indicates the degree of that node. For directed graphs, this may be either the in-degree or the out-degree.

$$L = \begin{bmatrix} 2 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 2 & 0 \\ 0 & 0 & 0 & 3 \end{bmatrix} - \begin{bmatrix} 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 1 \\ 1 & 1 & 1 & 0 \end{bmatrix} = \begin{bmatrix} 2 & 0 & -1 & -1 \\ 0 & 1 & 0 & -1 \\ -1 & 0 & 2 & -1 \\ -1 & -1 & -1 & 3 \end{bmatrix}$$

Figure 2.4: The Laplacian matrix for the graph in 2.1

The Laplacian matrix has several interesting attributes. Finding the eigenvectors and eigenvalues of this Laplacian matrix can in fact tell us something about the underlying connectivity of the graph.

Edge Lists

Another commonly used format is an edge list. As its name implies, this is a list of all the edges (v_i, v_j) in the graph. In the case of an undirected graph, each edge is bidirectional whereas one would add an additional edge (v_j, v_i) to represent a bidirectional edge in the directed case.

$$\begin{matrix} 0 & 2 \\ 0 & 3 \\ 1 & 3 \\ 2 & 3 \end{matrix}$$

Figure 2.5: An edge list for the graph in 2.1

2.1.4 Ego Graphs

A special type of node neighborhoods we'll refer to extensively in this thesis, are so-called *ego graphs*. [8] defines the ego graph of a node as:

> The subgraph induced over vertices directly connected to a specific vertex, called an ego, but excluding the ego itself

In other words, the ego graph of a node includes that node's 1-hop neighborhood, in addition to any edges between its neighbors.

Ego graphs are of interest to us because they are of a manageable size, while containing the most immediate local structure of a node. Ego graphs, and subgraphs in general, can be given attributes that summarize their structure on a high level. In this thesis, three such attributes will be used:

Clustering Coefficient (CC)

The clustering coefficient of a graph is a measure describing the degree to which the neighbors of a given node are connected. This measure is applied locally on individual nodes, which can then be aggregated to obtain a value for the graph as a whole. Watts et. al. [4] give the following definition:

> Suppose that a vertex v has k_v neighbours; then at most $k_v(k_v - 1)/2$ edges can exist between them (this occurs when every neighbour of v is connected to every other neighbour of v). Let C_v denote the fraction of these allowable edges that actually exist. Define C as the average of C_v over all v.

Gamma (Γ)

We introduce a related metric [2], which we have termed Γ. We'll label any edge that connects two neighbors in the ego-graph a *closure link*. Given a number of closure links t, Γ gives us a measure of the tendency of the ego's neighbors to be clustered. In the following explanation, k_t denotes the number of neighbors of the ego involved in at least one of these closure links.

Worst case: $k_t = 2t \equiv M \rightarrow \Gamma = 0$

If each closure link connects two nodes that are not involved in any other closure links, we would experience the worst possible clustering of closure links and by definition this would correspond to $\Gamma = 0$.

Best case: $k_t = m \rightarrow \Gamma = 1$

Noting from the definition of the clustering coefficient that the maximum possible number of closure links is equal to $k_t(k_t - 1)/2$, we can solve for k_t to obtain

$$k_t = \frac{1}{2}(\sqrt{8t + 1} + 1) \equiv m$$

In this situation, the smallest possible number of nodes are involved in the closure links. This would be lead to the best possible clustering for that number of closure links and by definition this would correspond to $\Gamma = 1$

For every case in between these two extremes, we use the following function that is linear in k_t.

$$\Gamma = \frac{M - k_t}{M - m}$$

We note that there's a case where this function cannot be used. In particular, $t = 1$ yields $M = m = k_t = 2$, which leads to $\Gamma = \frac{0}{0}$. Since this is simultaneously the "best" and the "worst" clustering possible, we find it fitting to set $\Gamma = \frac{1}{2}$ in this case.

For an illustration of the difference between the clustering coefficient and Γ, take a look at figure 2.6. Here we see two ego-graphs centered on 0.

Both have a clustering coefficient of 0.2, but we can clearly see that the closure links in the two graphs are distributed very differently between the nodes. In the leftmost graph, we get the best possible clustering of the closure links; we have maximum overlap and the three nodes that are involved in the closure links form a clique with the ego, resulting

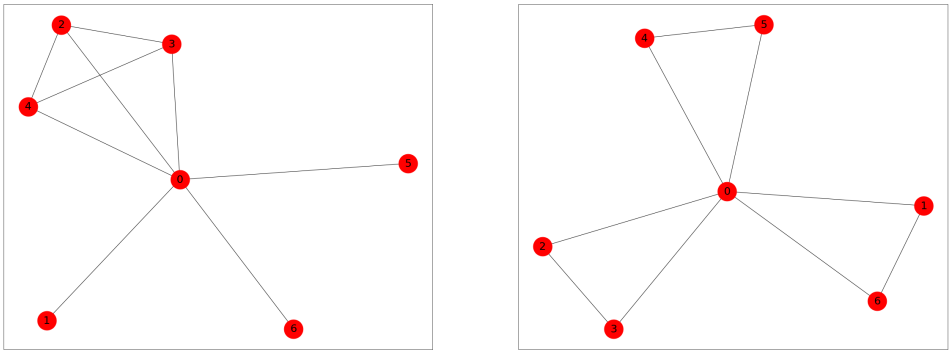

Figure 2.6: Left: $cc = 0.2, \Gamma = 1$ Right: $cc = 0.2, \Gamma = 0$

in $\Gamma = 1$. For the graph to the right, each node is involved in *exactly* one closure link, yielding $\Gamma = 0$.

2.1.5 Community Detection

One of the most studied problems within graph theory is the problem of *community detection*. While there has been a lot of debate about what exactly a community is [5], a classic view is that a community of a graph G is a subgraph S of G, whose nodes have a higher degree of interconnection than that of the remaining graph. Figure 2.7 illustrates this on a very simple graph, known as a *barbell graph*. One can obtain such a graph by creating two identical fully connected graphs, each with n nodes and then connecting these two graphs with an additional edge.

For such a simple example, it is trivial to identify the communities. There are two separate communities $C_1 = \{0, 1, 2, 3\}$ and $C_2 = \{4, 5, 6, 7\}$, connected by the link between 0 and 4. It is useful for illustrating concepts such as community detection because its mirrored nature makes the communities easily identifiable by visual inspection. Due to this, the barbell graph and similar graphs are often used as benchmarks for testing new algorithms and methods. On real-life data, however, this approach to community detection is rarely feasible due to the size and structural complexity of the graphs. Figure 2.8 shows another famous graph, nicknamed the Zachary Karate Club graph. This is a graph depicting the social interactions between 34 members of a university karate club over a three-year period and was compiled by Wayne Zachary for use in [22].

Compared to the barbell graph in figure 2.7, the Zachary Karate Club graph immediately

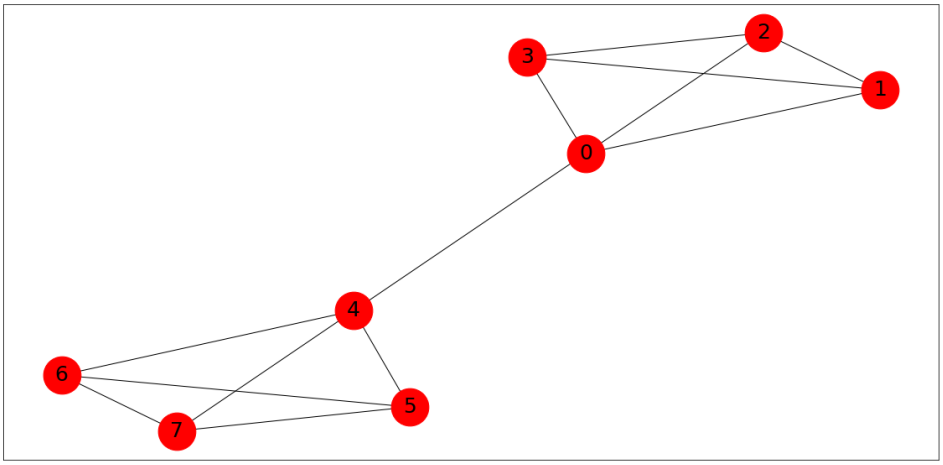

Figure 2.7: A barbell graph ($n = 4$)

strikes us as more complex. It is not apparent what communities exist in this graph, nor how many there are. Interestingly, at some point during the three years of data this graph represents, a conflict between the karate club president (node 34 in the graph) of and the instructor (node 1) resulted in the club being split into two. This led to researchers attempting to identify these two communities from the graph itself, resulting in the Zachary Karate Club graph becoming a common benchmark graph for community detection algorithms.

In the case of the Zachary Karate Club graph, we have domain knowledge explicitly telling us at least one likely split of the nodes into two separate communities. In particular, we have the the ”correct” number of communities we want to find, which can potentially reduce the search space by a lot. We may allow overlapping communities, i.e. a node can be part of several communities, further complicating matters. In addition, if the graph becomes sufficiently dense we may have trouble detecting communities due to a decreasingly small difference between the internal and external interconnectedness.

2.1.6 Structural Equivalence and Role Discovery

Another problem closely related to, but slightly different from the community detection problem is the problem of determining what role each node has in the graph. Lorrain and

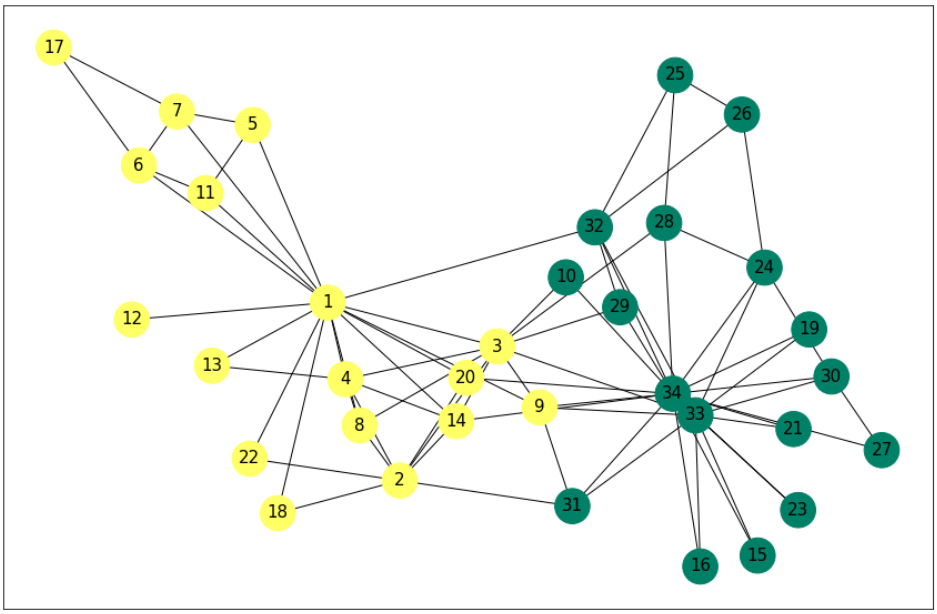

Figure 2.8: The Zachary Karate Club graph, colored by club membership after split

White [13] proposed the term *structural equivalence* between actors in a social graph. To paraphrase their definition: node a is structurally equivalent to node b if a relates to every node x in the graph in exactly the same ways as b does. For all intents and purposes, structurally equivalent nodes are indistinguishable from each other and can be substituted without consequence. This means that we can compress the graph by representing this group of structurally equivalent nodes as one single node along with a count. As an example, consider the Zachary Karate Club graph again. Nodes 22 and 18 are structurally equivalent because they both relate in exactly the same way to all other nodes in the graph; they are both connected to 2 and 1. The same goes for many of the nodes in the bottom right part of the graph: 16, 15 and 23 are all structurally equivalent according to the previous definition.

As pointed out by Sailer [20], structural equivalence as previously defined has several flaws. Primarily, since the requirements for structural equivalence are so strict, we often won't find it in complex real-world graphs. In addition, the definition relies exclusively on nodes being in close proximity in the graph. If we want to compare the local structure of nodes in different parts of the graph, structural equivalence is of little use.

A recent field of research, termed role discovery, proposes methods of grouping nodes in

the graph that have a *similar*, not equivalent, local structure regardless of their placement in the graph. This means that we'll have to rely on some other metric than proximity between nodes in the graph. Additionally, it implies a relaxation of the strict requirements that structural equivalence poses. We'll once again use the Zachary Karate Club graph for illustration, as it is sufficiently complex for our use. Recall that nodes 34 and 1 represented the club president and the instructor respectively. Even though the local structure of these two nodes is clearly distinct and share few neighbors, it should be clear to the reader that they are much more similar than if we were to compare for instance nodes 34 and 16. This illustrates the need for a continuous similarity measure between the local structures of two nodes.

One of the approaches suggested by Rossi and Ahmed [18] has been to extract a number of features from either the local neighborhood or the k-hop neighborhood for each node. We can then assign roles based on this feature vector, rather than based on the original graph.

2.1.7 Graph embeddings

Traditional methods for solving graph-based problems have typically been working directly on the graph representation in order to obtain a solution. These graph representations are often very high-dimensional, making it hard to create scalable algorithms. Additionally, many rely on inflexible matrix manipulation, resulting in brittle models.

To mitigate these issues, *graph embeddings* have been proposed as a possible solution. Graph embedding algorithms attempt to create vector representations for each node that preserve some predefined property of the original graph. According to Goyal and Ferrara [6], there are three main considerations to keep in mind when designing a graph embedding algorithm.

1. **Choice of property** A graph has many different properties that we can attempt to preserve. Common choices for this property include the first-order and second-order proximities, mapping nodes that are in close proximity in the graph closer in the embedding. Alternatively, we can opt to preserve the structural identity of each node. Since we effectively reduce the dimensionality when embedding the graph in a vector space, we cannot hope to preserve all of these properties at once. Thus, a choice has to be made as to which properties we want to preserve.

2. **Scalability** Real-life networks can be huge, so the algorithm used to perform the graph embeddings has to be able to scale to handle these massive data sets. Due to the relational nature of graphs, this is often difficult to achieve.

3. **Dimensionality** There is no silver bullet regarding the number of dimensions that best preserves the desired properties. The number of dimensions necessary to effectively capture the desired properties may depend on the size of our dataset, and there may be application-specific considerations to be made.

2.2 Clustering

Clustering is the act of taking each data point and assigning it to one of a number of sets such that data points within a set are more similar to the other points in that set than to those in other sets. These sets are what we call clusters. Clustering is a well-researched problem and there are several effective and performant algorithms available for solving this problem, given certain conditions that we'll discuss later.

2.2.1 Similarity Measures

Central to the entire process of clustering is how we define similarity between two data points. A variety of such measures (known as *similarity measures* or *distance measures*) are available for different data types. When the data objects that we want to cluster are real-valued vectors, a common choice of similarity measure is the *Euclidean distance*. It is defined as:

$$d(\boldsymbol{p}, \boldsymbol{q}) = \sqrt{(p_1 - q_1)^2 + (p_2 - q_2)^2 \dots (p_n - q_n)^2} \tag{2.1}$$

In visual terms, it can be described as the length of the straight line drawn from $\boldsymbol{p}$ to $\boldsymbol{q}$. This metric works well when the number of dimensions in the data set is relatively low.

2.2.2 Evaluation of Clustering Quality

Many evaluation methods exist to measure the quality of a clustering. Many of these are supervised methods, relying on some ground truth. Other methods use only the clustering

itself, and typically utilize intra-cluster or inter-cluster distances to provide a view of how good the quality is without using any auxiliary data.

The *Rand index* (RI) is an example of an unsupervised clustering quality measure. In brief terms, it measures the fraction of pairs of data points that are in agreement on cluster membership:

$$RI = \frac{a+b}{\binom{n}{2}} \tag{2.2}$$

where a and b are pairs of data points belonging to the same cluster in both clusterings and pairs belonging to different clusters in both clusterings, respectively, and n is the total number of data points.

The Rand index can be adjusted for chance, yielding the *adjusted rand index* (ARI) [10]. This leads to a measure that is close to 0 for random clusterings, and 1 when the clusterings are identical.

2.2.3 Low-dimensional Clustering

There are several different paradigms of clustering algorithms available that efficiently find clusters in low-dimensional data. Unfortunately, no algorithm exists that is guaranteed to do so in every case. We give a brief description of the different paradigms below.

Centroid-Based Clustering Algorithms

Centroid-based clustering algorithms start with k (often randomly placed) points in the space, known as *centroids*. Each of these centroids correspond to one cluster, meaning we can predefine the number of cluster we want the algorithm to find and it will attempt to find the best fit. As the algorithm progresses, these points will shift around in the space trying to separate the data as cleanly as possible. K-Means is an example of such an algorithm. Such algorithms excel at finding approximately circular clusters of the same size, but struggles with data where the clusters are oddly shaped or of varying sizes.

Density-Based Clustering Algorithms

Density-based clustering algorithms attempt to find areas in the space with high densities of data objects and deems this area a cluster if the density is above a predefined threshold. Algorithms that fall into this category include DBSCAN and OPTICS. These algorithms do not perform very well when confronted with data where the density varies a lot, as the parameters that guide the search have to be tailored to a certain density. When tuned appropriately, however, they find clusters of different shapes and sizes.

2.2.4 High-Dimensional Clustering and the Curse of Dimensionality

As the number of dimensions of the data we want to cluster increases, the algorithms mentioned above start to break down. The reason for this is a phenomenon that has been termed *The Curse of Dimensionality*. Assume that we have 1000 data points randomly distributed in n dimensions. As n increases, so does the volume of the space spanned by the n dimensions and our 1000 data points are likely to be spread too sparsely in this space for any meaningful clusters to form.

Beyer et. al. [1] argued that as n approaches infinity, the ratio of the distances between the two nearest and the two furthest points in the an n-dimensional space as measured by various similarity measures approaches 1. This means that these similarity measures become effectively useless, and so do the algorithms that depend on them. Table 2.1 shows statistics for the Euclidean distance applied on pairs of 1000 randomly sampled points for various dimensions.

Table 2.1: Distances between 1000 randomly distributed points for different dimensions (n)

n	Min	Max	Ratio (max/min)
1	$4 * 10^{-8}$	0.9993	$2.5 * 10^{7}$
2	0.0005	1.374	2748
3	0.003	1.556	518.6
10	0.261	2.218	8.5
50	1.663	4.212	2.5
100	2.99	5.152	1.72
1000	12.11	14.40	1.19

2.3 Dimensionality Reduction

In order to cope with the curse of dimensionality, we can apply a technique called *dimensionality reduction*. As the name implies, this technique attempts to reduce the number of dimensions in which the data is represented while attempting to retain as much useful and interesting information from the original space as possible.

2.3.1 PCA

One of the first and most famous techniques for dimensionality reduction is *Principal Components Analysis* or *PCA* for short. The intuition behind PCA is that we can summarize high-dimensional data using a lower number of dimensions by identifying new coordinate systems where the variance of the data is maximized and then projecting the data onto these new coordinate systems. We can then choose to use the first n dimensions in this new space, knowing that these dimensions are the n dimensions where the data varies the most.

2.3.2 Stochastic Neighbor Embedding

The Stochastic Neighbor Embedding (SNE) algorithm is a modern take on dimensionality reduction. Proposed by Hinton and Roweis [9], SNE starts out by calculating probability distributions p_{ij} that each point i would pick point j as its neighbor in the original, high-dimensional space:

$$p_{ij} = \frac{\exp(-d_{ij}^2)}{\sum_{k \neq i} \exp(-d_{ik}^2)} \tag{2.3}$$

where d_{ij}^2 is the dissimilarities of two points according to some established metric. In the original paper, a measure called the scaled *squared Euclidean distance* is used here. Each high-dimensional points x_i and x_j have low-dimensional counterparts y_i and y_j. Given these low-dimensional counterparts, we can generate a probability distribution q_{ij} that point i picks j as its neighbor:

$$q_{ij} = \frac{\exp(-||\boldsymbol{y}_i - \boldsymbol{y}_j||^2)}{\sum_{k \neq i} \exp(-||\boldsymbol{y}_i - \boldsymbol{y}_k||^2)} \tag{2.4}$$

The goal of SNE is to make these two probability distribution match as closely as possible, or in other words minimize the difference between them. To achieve this goal, a useful metric called the Kullback-Leibler Divergence (D_{KL}) [11] is used. This metric measures the difference between two probability distributions. SNE calculates the D_{KL} between the two probability distributions and generates a cost function which is then minimized through the use of gradient descent.

$$C = \sum_i \sum_j p_{ij} \log \frac{p_{ij}}{q_{ij}} \tag{2.5}$$

2.3.3 t-Stochastic Neighbor Embedding (t-SNE)

t-SNE [14] is an evolution of the SNE algorithm described previously. It introduces a minor change to the cost function in 2.5 that makes it easier to optimize during gradient descent without loss of accuracy. In addition, it changes the probability distribution that is used to generate probabilities from the Gaussian distribution to the Student's t-distribution (with 1 degree of freedom).

The reasoning for changing the probability distribution is to cope with the so-called *crowding problem* that appears when embedding high-dimensional data in two or three dimensions. When we are trying to map points from a high-dimensional space to a lower-dimensional space, we can't accurately retain similarities in all dimensions. In order to be able to perform well anyway, we have to make some compromises. Using the Student's t-distribution instead of the Gaussian results in moderately-spaced points in the high-dimensional space being mapped much farther apart in the low-dimensional space. This is because of the heavier tails of the latter probability distribution. In turn, this makes the clusters more compact and easy to visualize.

2.4 Deep Learning

Deep learning is a specific subfield within the broader field of machine learning. In particular, it describes a specific approach to machine learning where we utilize a model architecture called *neural networks* or a variation thereof. Many specialized types of such networks exist for use with for instance image data or data where time is a significant

factor. We will not discuss these further, but rather give a high-level description of what basic artificial neural networks are, and how they work.

2.4.1 Artificial Neural Networks

An artificial neural network is a machine learning architecture that uses several layers of small units called *artificial neurons*. When combined, these layers can learn highly non-linear patterns and representations.

The artificial neuron takes its inspiration from the similarly named units that exist in the human brain. In short, a neuron receives a number of weighted inputs which are summed, transformed using a non-linear *activation function* and then output to the next layer.

An artificial neural network contains several layers of such neurons, each of which can contain from a few to several thousand neurons. The first layer in a neural network, predictably named the *input layer*, receives the input data. The final layer of the neural network is called the output layer. Depending on the application and whether we are performing regression (predicting a continuous value based on the input) or classification (predicting the category of the input), this final layer may implement some additional processing in order to transform activations from the preceding layer further. Softmax, which we will describe shortly, does exactly this.

2.4.2 Backpropagation

The layers between the input layer and the output layer are called *hidden layers*. These layers are responsible for learning representations of the data. The algorithm used by the hidden layers to learn the patterns of the input data is called *backpropagation*. In essence, each time we run a sample through the network the output will differ from the so-called *ground truth*. The difference between the actual output and the ground truth is called the *error* of the network and is the value that we want to minimize across all samples. The backpropagation algorithm propagates this error back trough the network, making adjustments to the weights of the neurons in a way that is likely to reduce the error.

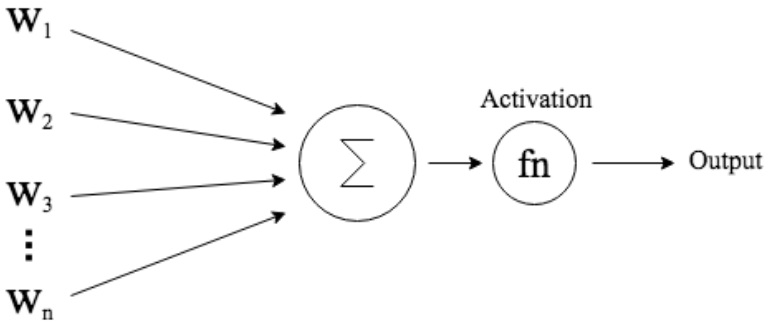

Figure 2.9: An artificial neuron

2.4.3 Softmax

When using neural networks to perform classification, we often want the output of the neural network to act as a probability distribution over the number of possible classes that the input could potentially belong to. The softmax function solves this and is consequently often used as the final layer in a neural network: given a K-dimensional real-valued vector, softmax outputs a K-dimensional vector where each element is in the range $[0, 1]$ and all the elements sum to 1. More formally, it is given by:

$$\text{Softmax}(\boldsymbol{v})_i = \frac{\exp(v_i)}{\sum_{k=1}^{K} \exp(v_k)} \tag{2.6}$$

2.4.4 Word2Vec

The Word2Vec model, brought to life by Mikolov et. al. [15], was initially proposed to generate word embeddings from a corpus of sentences. In fact, Word2Vec can be used on any sequences of objects, and recently it has found use with algorithms solving graph embedding. This model uses a sliding window W centered on a word w with a specified size s across each sentence in the corpus to generate training samples for a neural network in order to predict the words surrounding w within the sliding window W.

Word2Vec comes in two different variants that can be used to solve different problems. The first variant, named *CBOW (Continuous Bag of Words)*, takes a number of words as input and outputs a single word that is likely to co-occur with these words. This can be useful for text generation applications. The other variant is named *skip-gram* and performs

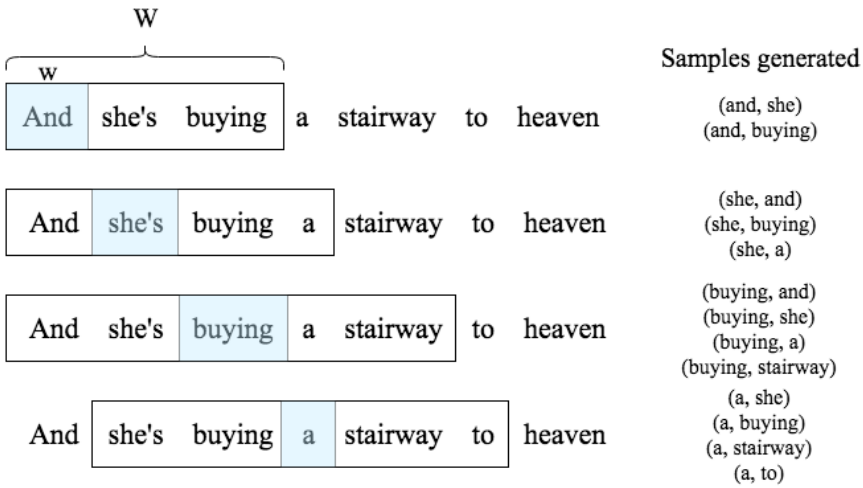

Figure 2.10: An illustration showing the sliding window mechanism of Word2Vec. This particular window has a size of 5

the inverse operation: given an input word, it outputs the most likely words to co-occur with this input word.

The neural network used in the training process contains a single hidden layer. The weights of this hidden layer are continually adjusted throughout the training process and upon the completion of the training phase they constitute the word embeddings that we sought to learn.

Chapter 3

Related Work

In this chapter, some state-of-the-art methods within graph embedding methods are described.

3.1 Graph Embedding Methods

Over the last few years, a variety of different paradigms have emerged with regard to how graph embeddings are performed.

3.1.1 Factorization-Based Methods

The first graph embedding methods that were proposed relied on performing matrix factorization operations on some kind of matrix representation of the graph. either directly on the graph's adjacency matrix, its Laplacian matrix or some other kind of matrix representation.

Locally Linear Embedding (LLE)

Among the first graph embedding methods, was a method called *Locally Linear Embedding* [19]. This algorithm first finds the nearest neighbors of each data point. Then

it attempts to find a set of weights that best reconstruct each data point from its neighbors. Finally, the algorithm uses these weight to find vectors in the low-dimensional space that best minimize some defined cost function.

As the algorithm only considers each point's neighbors, the goal of Locally Linear Embedding is to try to capture the proximity of each point in the graph.

GraRep

GraRep [3] uses the graph's adjacency matrix to obtain a *transition probability* for each node in the graph to every other. In contrast to many other algorithms based on matrix factorization that capture first-order and maybe second-order similarities, GraRep attempts to learn embeddings that utilize global information and subsequently manages to captures kth-order proximities.

3.1.2 Methods Based on Random Walks

Another family of graph embedding methods relies on sampling the local neighborhoods of each node by performing random walks from each node. The intuition here is that a sufficient number of such random walks is enough to completely characterize the local neighborhood, and nodes that frequently share subwalks within these random walks are likely to be located in close proximity to each other in the graph.

Deep Walk

The DeepWalk algorithm, pioneered by Perozzi et. al. [16] uses random walks in conjunction with concepts from NLP (Natural Language Processing) in order to obtain a low-dimensional representation of the input graph. In short, the algorithm works like this: For a given number of iterations, generate random walks for each node in the graph. Recognizing that these random walks constitute sequences of data where we are very interested in the co-occurring nodes, we maintain a skip-gram model, training it on these random walks.

Upon termination, the skip-gram model can take a node as input and output the nodes with the highest probability of having co-occurred with it in the random walks. The graph

embedding we sought to find is then simply the neuron weights of the hidden layer in the same skip-gram model.

Node2Vec

Node2Vec, proposed by Grover and Leskovec [7] extends the DeepWalk model in several aspects. The inventors argue that a purely random walk procedure is not sufficient in order to capture both communities and structural equivalence. Instead they propose a *biased* random walk procedure that, according to two parameters p and q, can be tweaked to favor traversing nodes close to the source node or to favor exploration outward.

Struc2Vec

A third algorithm that uses random walks, although in a different manner than the two previously mentioned algorithms, is struc2vec. This algorithm was proposed by Ribeiro et. al. [17] and differs from the others mentioned here in that it is especially suited for role discovery. Initially, the algorithm computes pairwise similarities for each node for increasing neighborhood sizes k. This is done by taking all nodes exactly k hops from the source node, obtaining an ordered degree sequence from them and comparing these two sequences using an algorithm called Dynamic Time Warping (DTW). DTW outputs a number denoting the similarity between these two ordered degree sequences. This number is summed with similarities for lower values of k, such that the similarity is non-decreasing as k increases.

Having done this, the algorithm uses these pairwise similarities to construct a multilayered weighted graph, where each node is present in every layer and connected to every other node in that layer as well as its corresponding node in the layers above and below. The struc2vec algorithm differs from the previously mentioned algorithms in that the random walks are not performed on the original graph, but rather on this latent multilayered graph. The weights between two nodes in each layer k of this multilayered graph is inversely proportional to the similarity between these two nodes at neighborhood size k. The weight of an edge between a node and its counterpart in the layer above is given by the degree to which that node has many similar nodes in the current layer k. Moving up to layer $k + 1$ will introduce a stricter measure of similarity between nodes, since the similarities between two nodes will be non-decreasing as k increases.

Having generated this multilayered weighted graph, struc2vec proceeds to performing random walks on this graph. Since the edge weights in this latent graph are determined not by proximity in the original graph, but rather structural similarity through comparing degrees, these random walks will generate completely different walks than those generated by for instance node2vec and DeepWalk. These random walks are then fed to skip-gram, which in turn generates the final embeddings.

3.1.3 Methods Based on Deep Learning

The most recent approach to generating graph embeddings relies on using deep neural networks of different varieties in order to obtain the embeddings.

Structural Deep Network Embedding (SDNE)

A recent algorithm using deep neural networks to generate graph embeddings is the SDNE algorithm, proposed by Wang et. al. [21]. This algorithm utilizes what's known as a deep autoencoder to encode each node's neighborhood to a latent representation and then using the decoder to attempt to reconstruct the neighborhood. The result of this is that nodes with similar neighborhoods have similar latent representations. This deep autoencoder uses an objective function that attempts to minimize both the first-order proximity and the second-order proximity between nodes. In practice, this means that nodes that are neighbors and that have the same neighbors will get similar embeddings. As such, the algorithm does not capture the structural similarity, but rather communities within the graph.

Chapter 4

Methods

In this chapter, I'll go into greater details about what algorithms were deemed suitable for solving the problem. In addition, the unsupervised nature of the problem requires a robust procedure for validating and comparing the results, which will be described as well.

4.1 Choice of Algorithms

4.1.1 Embedding Algorithm

While several algorithms exist for embedding graph structures, as seen in the previous chapter, not all of these are ideal fits with regard to solving the problem of role discovery. DeepWalk, node2vec, SDNE, LLE and GraRep all attempt to preserve the local neighborhood of each node in the embedding. This is contrary to what we want to achieve, namely preserving the role that the node plays in the graph.

Struc2vec, on the other hand, preserves the structural identity of each node which is analogous to preserving the role of the node. Because of this feature, we will use struc2vec as the embedding algorithm. A range of dimensionalities will be attempted, in order to see how the dimensionality might influence the quality of the embedding.

4.1.2 Dimensionality Reduction

In order to be able to effectively cluster the embeddings as they reach higher dimensions, a method of dimensionality reduction has to be applied before clustering is applied. We'll use the t-SNE algorithm mentioned in chapter 2 to achieve this.

4.1.3 Clustering Algorithm

Having obtained a lower-dimensionality embedding through the application of t-SNE, we can apply a clustering algorithm on this in order to extract the roles of the nodes. We expect to find many different roles in the graph, with some roles having many nodes and others having fewer. Consequently, it is important that our clustering algorithm can find clusters where the size varies a lot. This excludes centroid-based clustering algorithms like K-means, but makes a good case for using a density-based algorithm. Based on this, we have chosen to use DBSCAN as the algorithm we use to find clusters in the low-dimensional embeddings.

4.2 Validation and Baselines

As mentioned earlier, the exploratory nature of this thesis means that we have no labels or supplemental information for any of our nodes. Instead, we'll have to rely on metrics supplied by graph theory and visual inspection to determine the quality of our clusterings.

4.2.1 Metrics and Baseline

In order to be able to verify that our solution captures interesting information and compare different solutions, we want to establish a baseline embedding to which we can compare our models. These baseline embeddings should be easily attainable, and capture some useful information about the structure of the graph.

As previously explained, the clustering coefficient is a metric for describing how interconnected the neighbors of a node is. In order to be able to claim that struc2vec captures the structural identity of the graph, we would expect that nodes with similar

clustering coefficient belong to the same clusters. Recall that Γ measures the degree to which interconnected neighbors are clustered together or separated in the graph. As Γ is closely related to the clustering coefficient, we would hope to see that struc2vec is able to cluster nodes with similar values for both the clustering coefficient and Γ-values closely together in the graph.

We note that the three metrics (CC, Γ and node degree) together capture much of the information about the local graph structure of each node, and used together they form a three-dimensional space which we can use as a starting point for our baseline. The CC and Γ features both fall in the range $[0, 1]$ and can be readily used. Since the node degrees cover a large range and are not evenly distributed over this range, log-scaling along with normalization to the same interval as the CC and Γ will be applied. This is done according to the equation 4.1.

$$p_{scaled} = \frac{\log(p)}{\log(p_{max})} \tag{4.1}$$

Chapter 5

Experiments

In this chapter, we will describe the various experiments performed during the lifetime of this project. First, we will describe the datasets used in this thesis, before presenting the baselines used. Finally, we'll give a brief overview of the experiments themselves.

5.1 Datasets

5.1.1 Bitcoin OTC Web of Trust Weighted Signed Network

The first dataset is a trust network in the Bitcoin network first used by Kumar et. al. [12]. It is directed, and for the purposes of this thesis, the directionality has been removed. The nodes in the dataset represent persons trading a cryptocurrency known as Bitcoin on an over-the-counter (OTC) marketplace. The dataset comes in the form of an edge list, containing $5\,881$ nodes and $35\,592$ edges. Since the trading is done between anonymous parties, the marketplace has established a trust rating system for keeping track of fraudulent users. Each edge in this edge list represents a rating in this system and is associated with a weight denoting the level of trust put in one party by another. A timestamp is also given for each edge, giving the exact time of the rating. Since we are only interested in the graph structure, these edge attributes have been discarded. The full dataset is available here

Table 5.1: Table showing various metrics for Bitcoin OTC

Metric	Mean	Median
Cluster coefficient	0.177	0
Γ	0.31	0
Degree	7.30	2

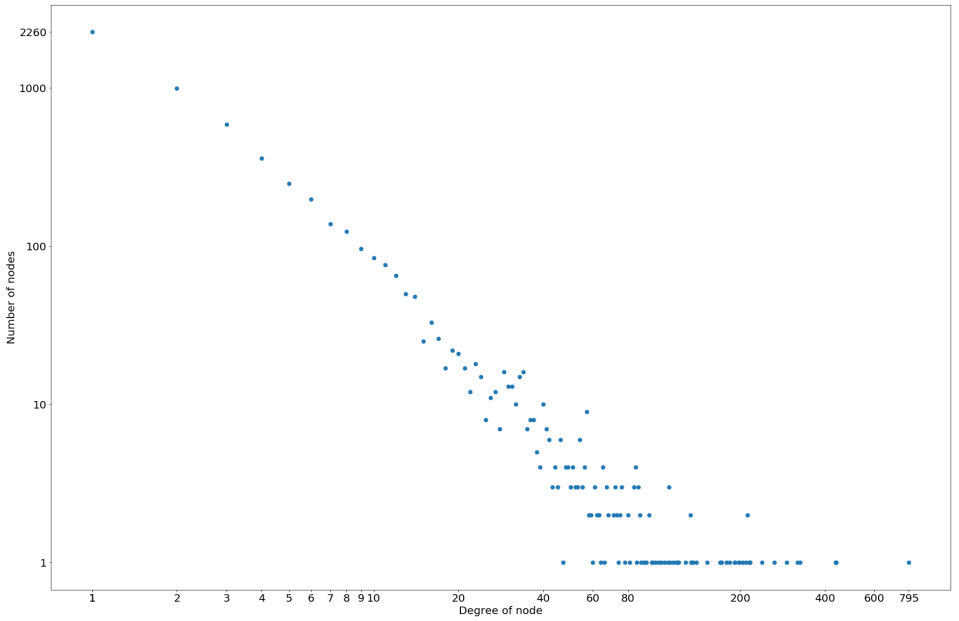

Figure 5.1: The number of nodes of various degrees in the Bitcoin OTC dataset. Note the log-log scale.

Statistics

In order to have a clearer view of the dataset, some general statistics about the Bitcoin OTC dataset is provided. Table 5.1 shows the mean and median for the cluster coefficient, Γ and degree. Figure 5.1 plots the degree against the number of nodes of that degree. From these figures, it is clear that the majority of the nodes have very few neighbors. In fact, 86.6% of the nodes have 10 neighbors or fewer. 12.6% of the nodes have between 11 and 100 neighbors, while the remaining 0.08% (a grand total of 45 nodes) have more than 100 neighbors.

5.1.2 Anonymized Telenor Dataset

The second dataset was provided by Telenor for the purpose of this thesis. This dataset represents users of a mobile payment service sending money during May 2016. The original data is in the form of a directed edgelist, with $494\,018$ nodes and $703\,528$ edges, making the full-size version of this dataset considerably larger than the Bitcoin dataset. Associated with each edge in the graph is the amount sent, as well as the date of the transaction.

Preprocessing

Because the edges in the graph were originally directed, we have removed the directionality of the edges. This reduces the number of edges to $540\,196$. Due to issues with the scalability of the implementation of the algorithms used, we will not use the entire dataset but rather a subset of it. We chose five days of data (from the 12th to the 17th), which got us down to a total of $101\,694$ edges and $140\,120$ nodes. Since the nodes here outnumber the edges, we know that this graph is disconnected. In fact, it contains close to $40\,000$ connected components. Considering that the nodes in this dataset are people sending money, and the edges are transactions that took place over five days, we would not expect every user to be connected in this graph. Although the algorithms we will use can handle disconnected graphs, we will omit the smaller components and instead make use of the largest one. This component contains $9\,810$ edges and $9\,514$ nodes, which is what we'll use as the final dataset.

Statistics

Here we will briefly discuss some statistics related to the Telenor dataset that will provide a clearer picture of the data we are dealing with. Figure 5.2 shows the distribution of the degree of nodes. $98,8\%$ of the nodes in the dataset have a degree of 10 or less, a significantly higher fraction than the Bitcoin OTC dataset. As this is transactional data from a mobile payment service, it is unsurprising that the vast majority of the users have performed less than 10 transaction over this five-day period.

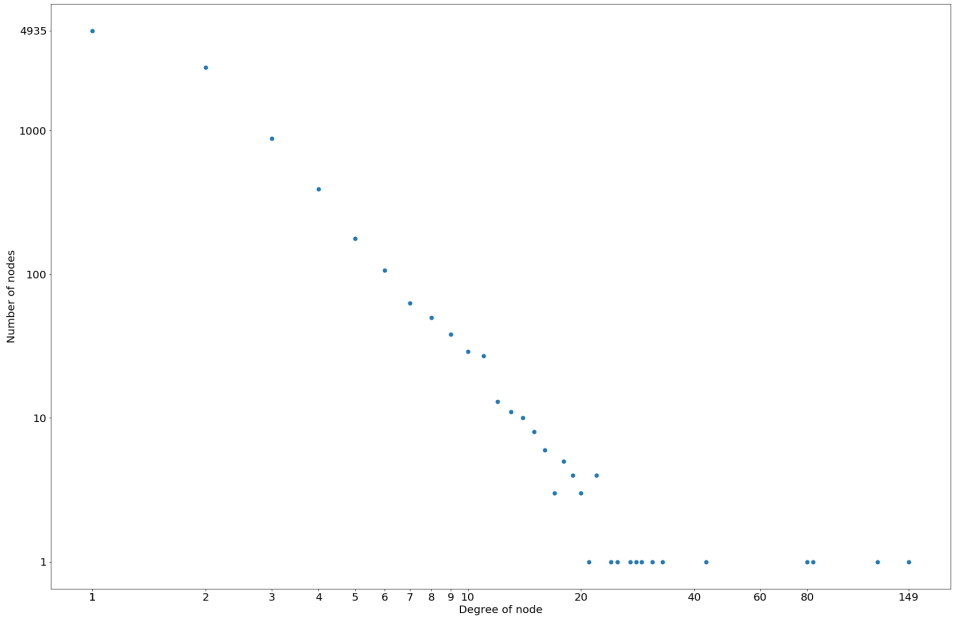

Figure 5.2: The number of nodes of various degrees in the Telenor dataset. Note the log-log scale

5.2 Baselines

5.2.1 Bitcoin OTC Baseline

The full three-dimensional baseline embedding that resulted from using the three metrics mentioned in chapter 4 for the Bitcoin OTC dataset can be viewed here, but for now we will use a 2D embedding obtained by applying t-SNE to this 3D embedding. Figure 5.3 shows this 2D embedding along with the clusters obtained through DBSCAN, while figures 5.4 through 5.6 shows the same embedding with the nodes colored according to the metrics specified earlier.

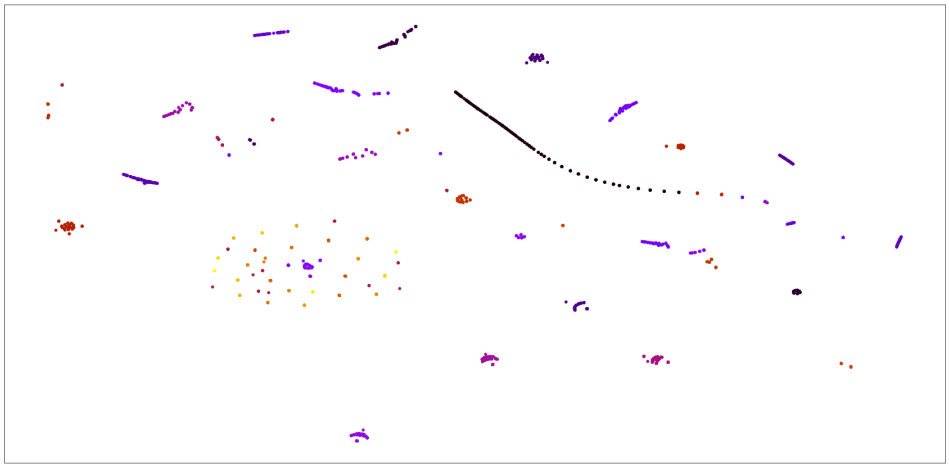

Figure 5.3: 2D baseline embedding for Bitcoin OTC dataset, color denotes cluster membership

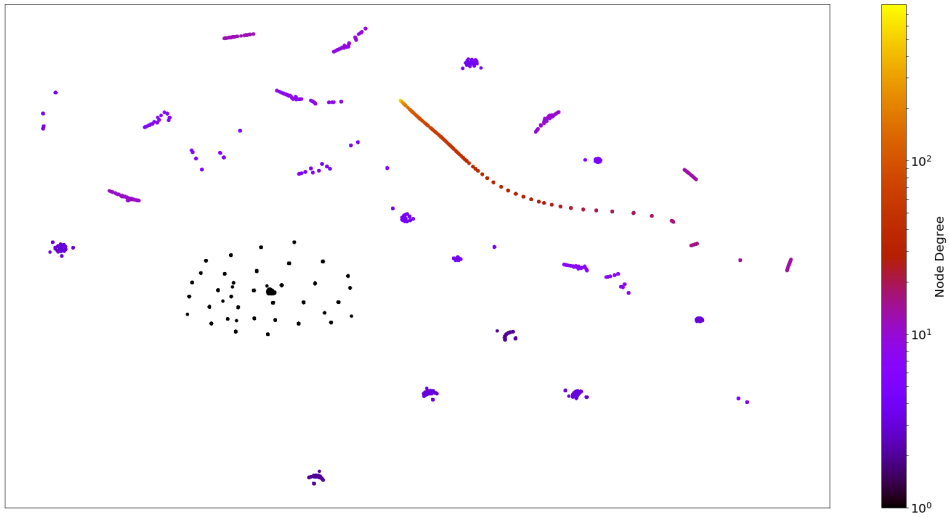

Figure 5.4: 2D baseline embedding for Bitcoin OTC, color denotes logarithm of node degree

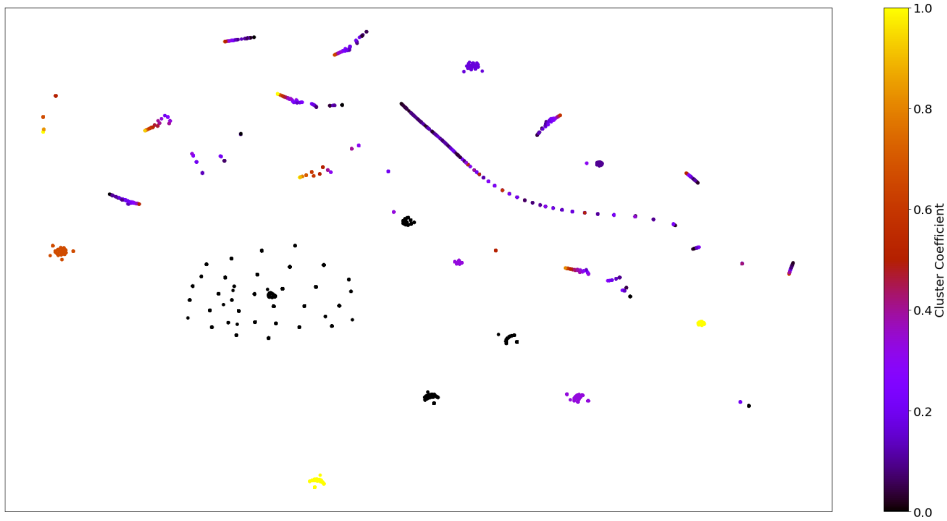

Figure 5.5: 2D baseline embedding for Bitcoin OTC, color denotes cc of node

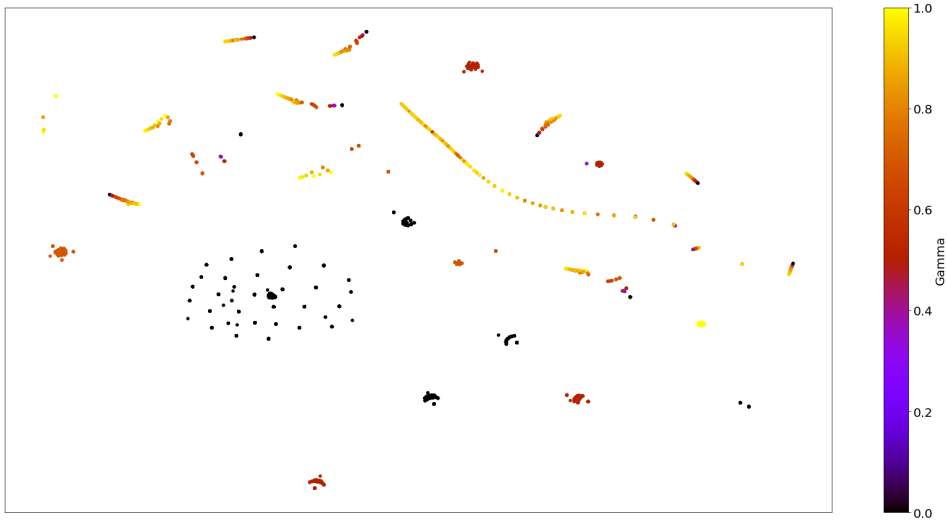

Figure 5.6: 2D baseline embeddings for Bitcoin OTC, color denotes Γ of node

5.2.2 Telenor Baseline

Due to privacy reasons, a 3D baseline like the one for the Bitcoin OTC dataset will not be made available publicly. As with the Bitcoin OTC dataset, we'll use a 2D version of this

baseline where we have applied t-SNE on the 3D embedding. Figure 5.7 shows this 2D embedding along with the clusters obtained through DBSCAN, while figures 5.8 through 5.10 shows the same embedding with the nodes colored according to the metrics specified earlier.

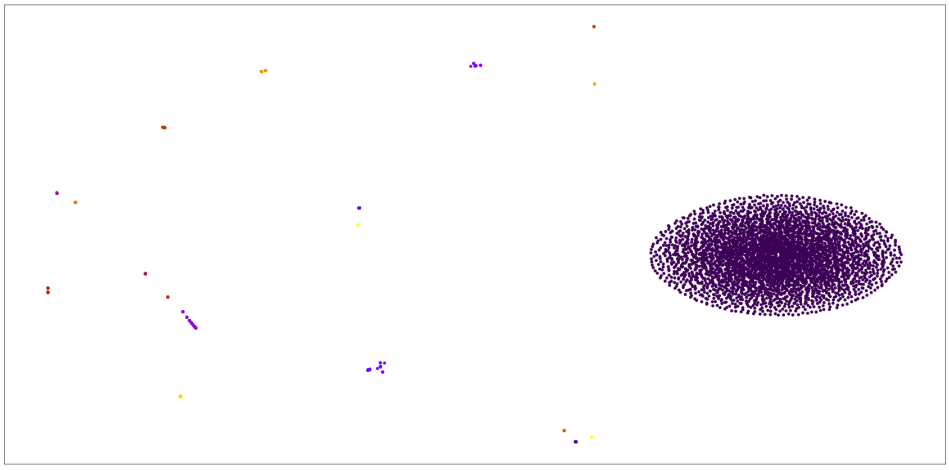

Figure 5.7: 2D baseline embedding for Telenor, color denotes cluster membership

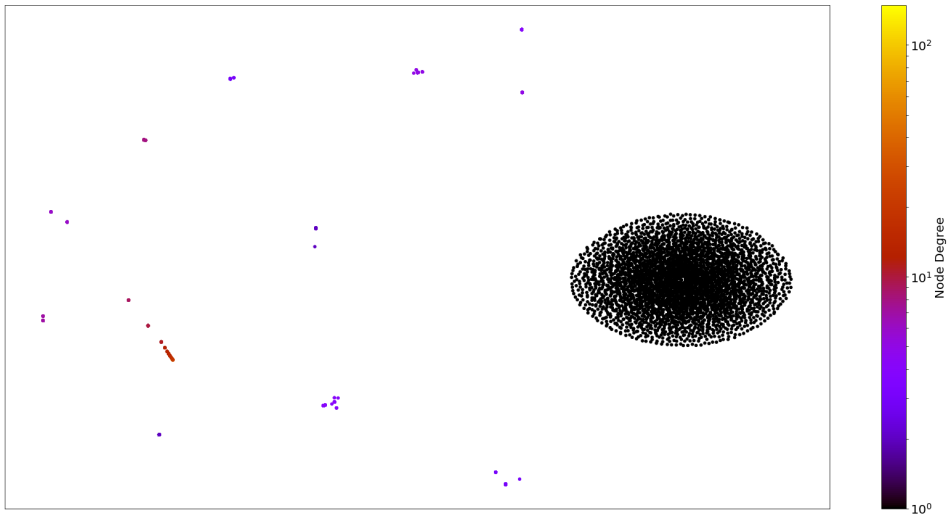

Figure 5.8: 2D embeddings for Telenor, color denotes degree of node

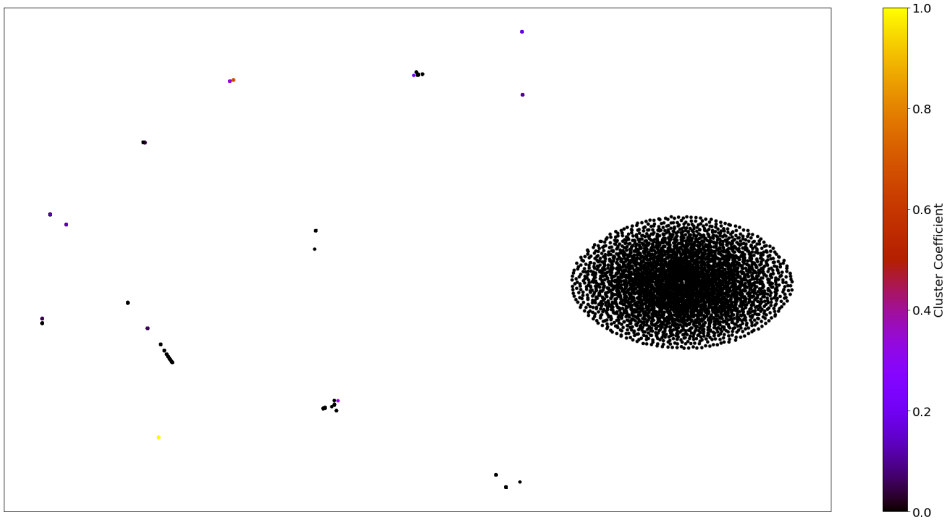

Figure 5.9: 2D embeddings for Telenor, color denotes cc of node

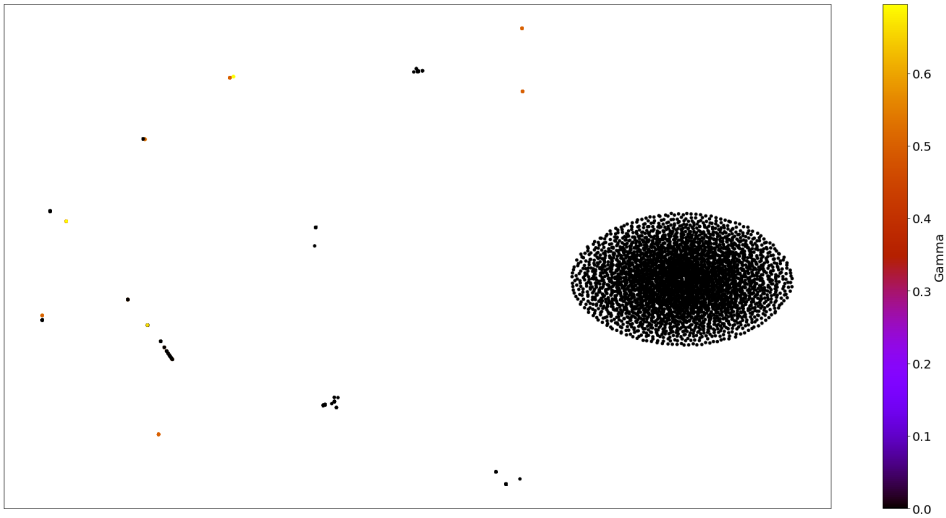

Figure 5.10: 2D embeddings for Telenor, color denotes Γ of node

5.3 Experiments

For all struc2vec experiments conducted, the reference implementation of struc2vec was used. All the struc2vec optimizations mentioned in [17] were used throughout. Struc2vec was run with default settings for the number and lengths of the random walks. The t-SNE and DBSCAN implementations were provided by scikit-learn. t-SNE was run with the default settings as well. The settings used for the DBSCAN clustering were $\epsilon = 0.2$ and $\mathrm{minSamples} = 10$ for each experiment, unless specified otherwise. All the code produced during this thesis is available here.

In order to determine the quality of the embeddings by struc2vec, we will see how well the embeddings capture the metrics used to establish the baseline: the clustering coefficient, Γ, and the node degree. In order to easily visualize these metrics, each point in the embedding space has been colored according to the value of the relevant metric. In order to provide a measure of the difference between the baseline embedding and the generated embeddings, the adjusted rand score was used.

5.3.1 Generating 2D Embeddings Directly

The first experiment we attempted was to obtain the 2D embeddings directly, using struc2vec without any form of post-processing. This meant that we did not need any dimensionality reduction, and consequently we could perform DBSCAN directly on the graph embeddings in order to extract the roles.

5.3.2 Generating nD Embeddings

Regardless of the results for the low-dimensional embeddings, we wanted to generate embeddings of higher dimensionalities as well in order to see whether increasing the dimensionality of the embeddings yielded proportionally better results. In order to be able to perform clustering on these nD embeddings, we used t-SNE to obtain a 2D embedding on which we could perform clustering. Various dimensionalities (50, 100, 300) were attempted.

Chapter 6

Results and Discussion

6.1 2-Dimensional embeddings

6.1.1 Bitcoin OTC

For the initial experiment, two-dimensional embeddings were generated directly by struc2vec. This meant that no dimensionality reduction was needed. Doing this direct-to-2D embedding, we could determine whether the final phase of struc2vec, using skip-gram, can capture the local structure of each node utilizing just two neurons in the hidden layer.

As we can see in figure 6.1, where the nodes have been colored according to which cluster they belong to, several distinct clusters have been identified by DBSCAN. Figure 6.2 shows the struc2vec embeddings colored according to the degree of the nodes, rather than cluster membership. It shows that the 2D struc2vec embeddings have managed to separate the nodes into clusters that clearly represent sets of nodes with different degrees. We cannot say the same for the cc and the Γ. There seem to be no consistent clusters formed with regard to these two values, with the exception of the single cluster containing all nodes of degree 1. This is of course due to the fact that the ego graph of 1-degree nodes always will have $cc = 0$ and $\Gamma = 0$. For the remaining clusters, no one value of either cc or Γ stands out.

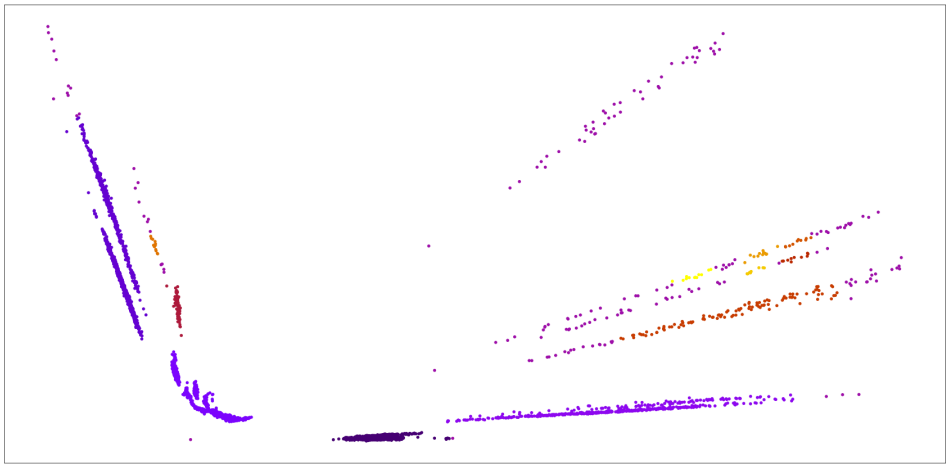

Figure 6.1: 2D embeddings for Bitcoin OTC, color denotes cluster membership

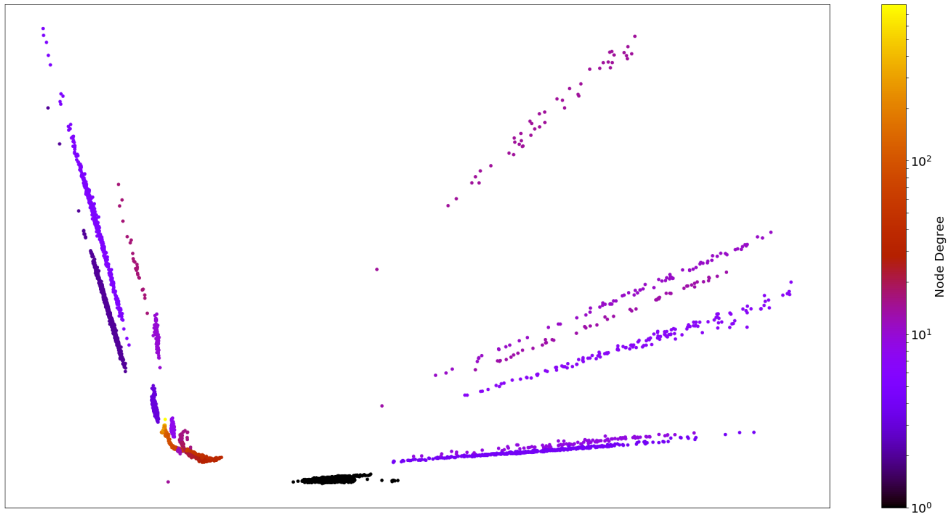

Figure 6.2: 2D embeddings for Bitcoin OTC, color denotes logarithm of node degree

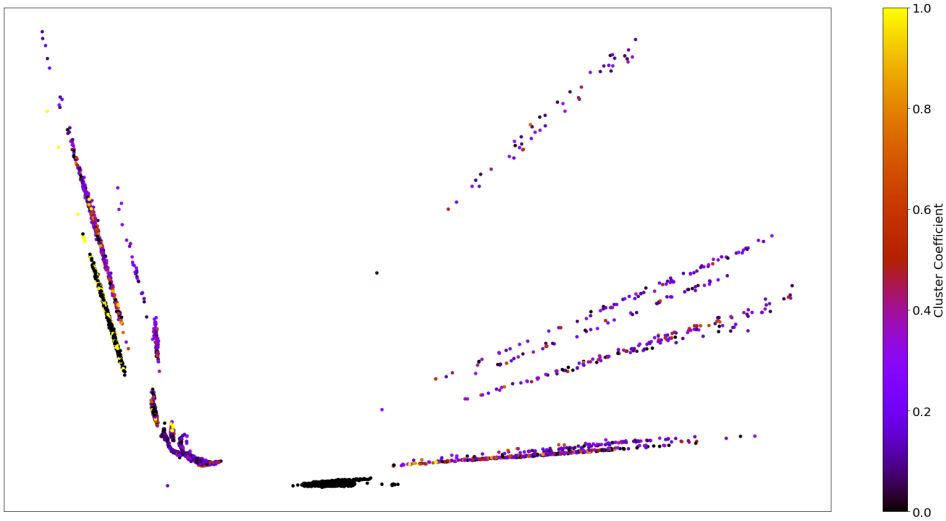

Figure 6.3: 2D embeddings for Bitcoin OTC, color denotes cc of node

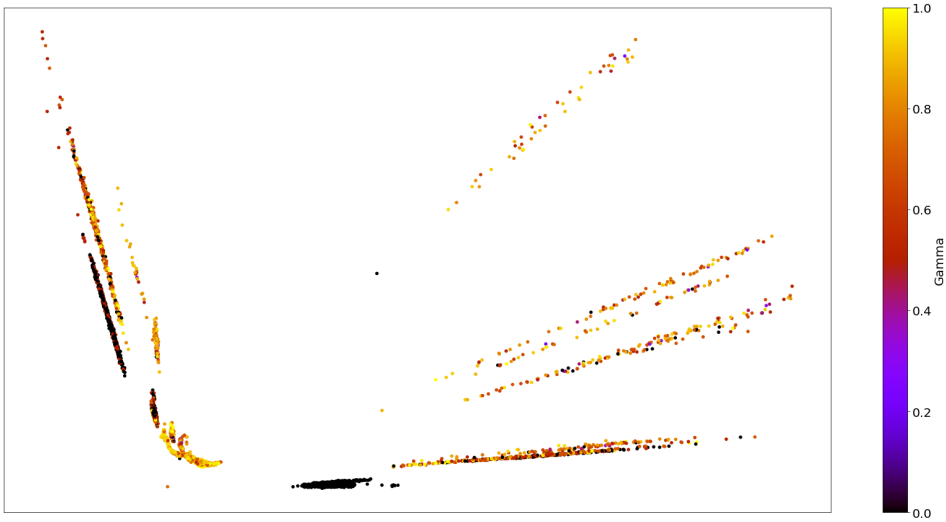

Figure 6.4: 2D embeddings for Bitcoin OTC, color denotes Γ of node

6.1.2 Telenor Dataset

As with the Bitcoin dataset, we started by generating 2D embeddings directly. Figures 6.5, 6.6, 6.7, and 6.8 show the embeddings generated by node2vec colored according to cluster membership, node degree, cc and Γ respectively. Upon visual inspection of figure 6.5, we see that most of the nodes have been clustered in a small area of the 2D space while the remaining nodes are scattered all around. The resulting disparity in density makes it difficult for DBSCAN to effectively cluster the nodes. Considering figure 6.6, which shows the degree of each node, we see that while there are certain areas within the supercluster that better represent nodes of various degrees, the difference is too small for DBSCAN to be able to distinguish between them. Viewing figures 6.7 and 6.8 we see that the 2D embeddings do a poor job of capturing the cc and Γ.

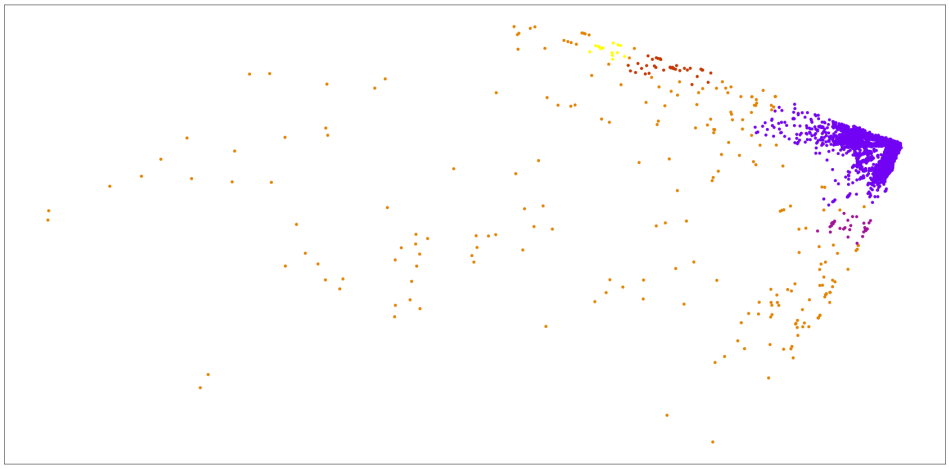

Figure 6.5: 2D embeddings for Telenor, color denotes cluster membership of node

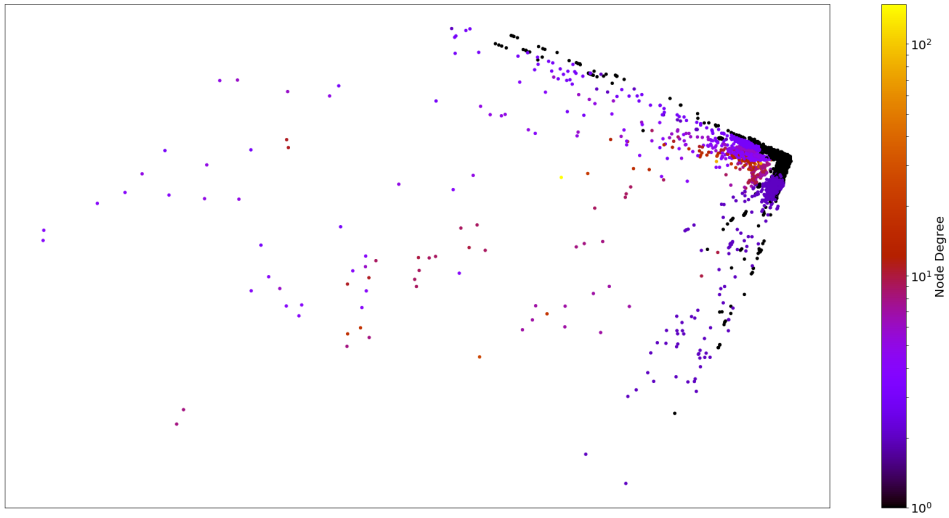

Figure 6.6: 2D embeddings for Telenor, color denotes the logarithm of node degree.

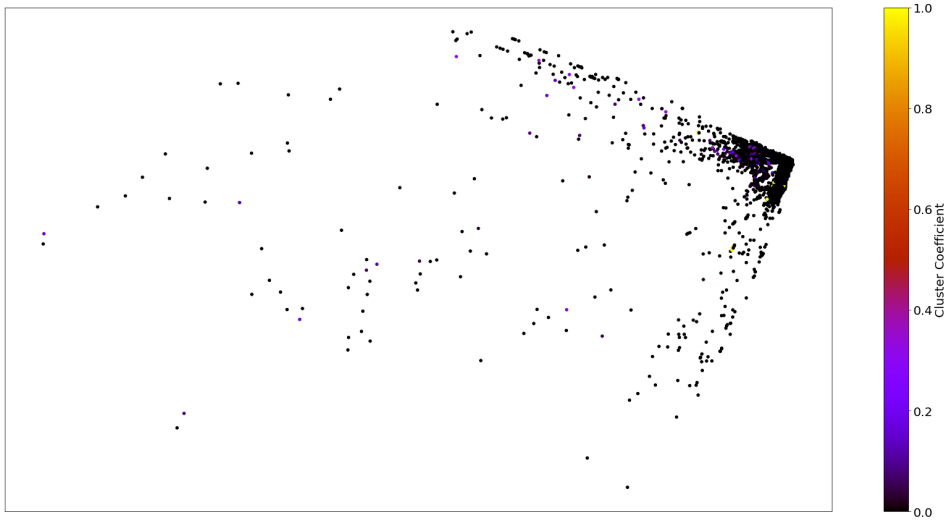

Figure 6.7: 2D embeddings for Telenor, color denotes cc of node

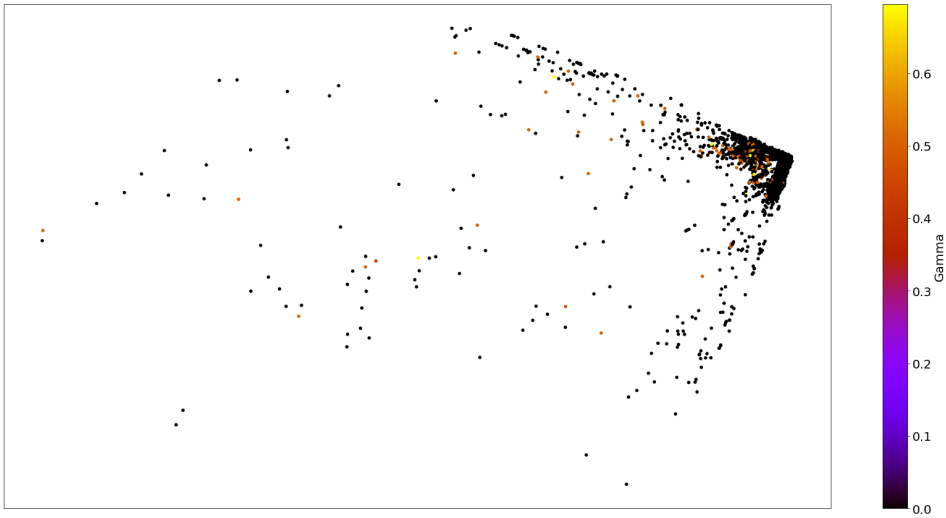

Figure 6.8: 2D embeddings for Telenor, color denotes Γ of node

6.1.3 Discussion

As we saw in the previous chapter, the 2-dimensional embeddings generated by struc2vec for the Bitcoin OTC dataset captured the node degree to a certain degree, but that was just about all it captured. Considering the cc and the Γ, there are no clear clusters formed (except in obvious cases where the node degree is 1 or 2).

For the Telenor dataset, there was really no clear separation with regard to either node degree, cc or Γ. The vast majority of the nodes in the graph were mapped to the same area in the space.

The explanation for this is likely that embedding the graph directly into a 2D space discards too much of the information we want. Recall that the final phase of struc2vec consists of learning the weights of a neural network via skip-gram. When struc2vec attempts to embed a graph with $|V|$ nodes directly into n dimensions, the skip-gram uses n hidden layer neurons to achieve this. Since each neuron has one input weight for each neuron in the preceding layer, we have a total of $n \cdot |V|$ weights. Because each node has n weights associated with it, it is likely that using a small value for n (e.g. 2 or 3) as $|V|$ increases leads to skip-gram being unable to generate embeddings that effectively capture the space of similarities between the nodes.

6.2 50-Dimensional embeddings

6.2.1 Bitcoin OTC

As we mentioned in the previous chapter, we will also generate embeddings of a higher dimensionality (50, 100, and 300 were attempted) and then utilize t-SNE for visualizing these high-dimensional embeddings. Figures 6.9 through 6.12 show the results obtained.

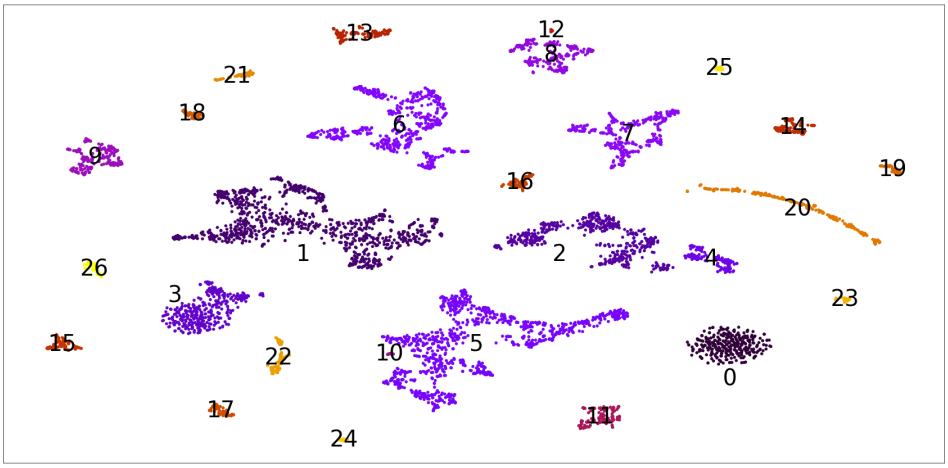

Figure 6.9: 50D embeddings for Bitcoin OTC, color denotes cluster membership of node

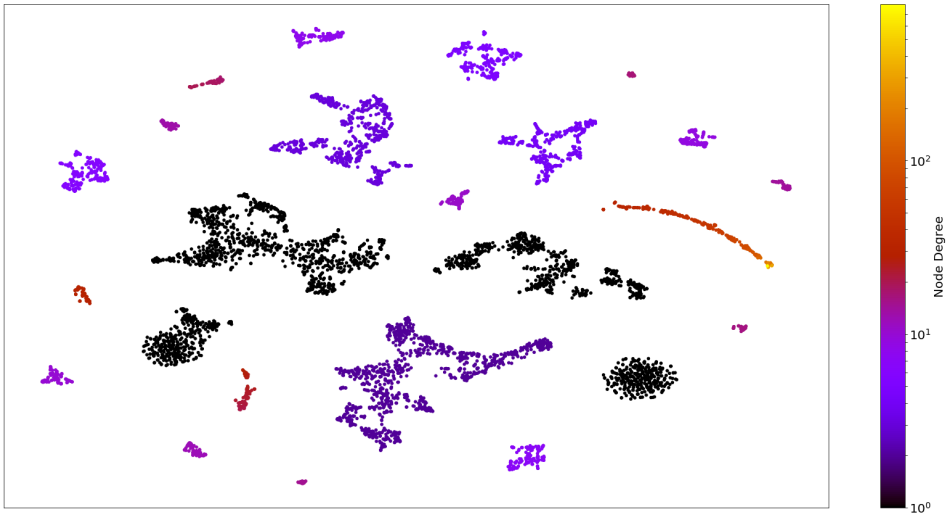

Figure 6.10: 50D embeddings for Bitcoin OTC, color denotes logarithm of node degree

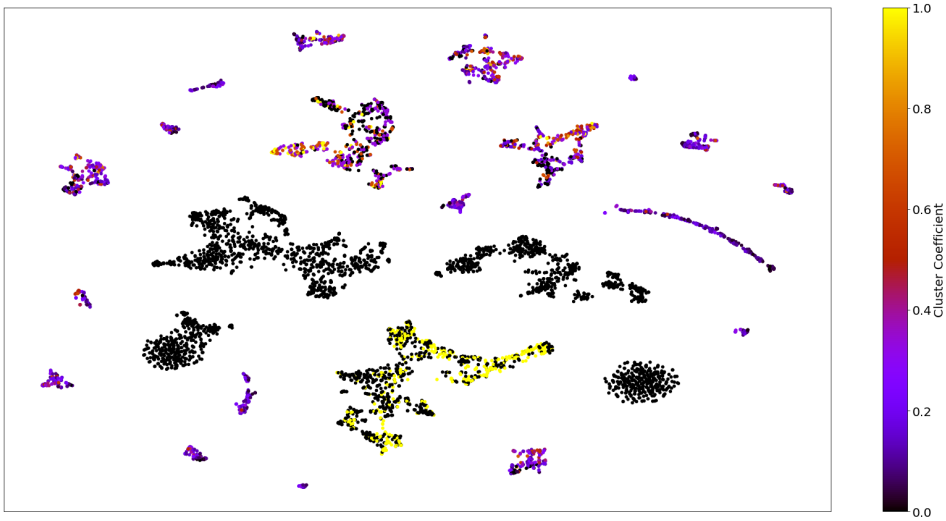

Figure 6.11: 50D embeddings for Bitcoin OTC, color denotes cc of node

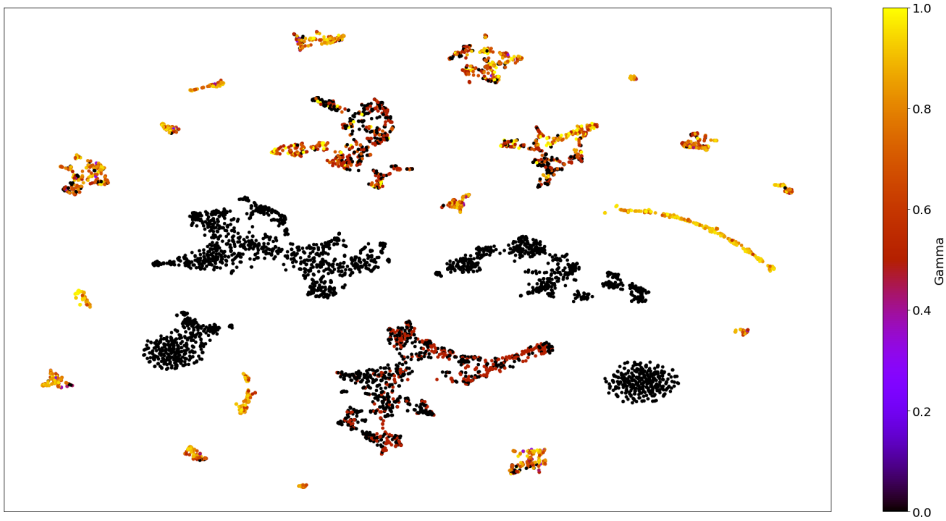

Figure 6.12: 50D embeddings for Bitcoin OTC, color denotes Γ of node

6.2.2 Telenor Dataset

When increasing the dimensionality from 2 to 50 dimensions and applying t-SNE on the resulting embeddings, we get a much more cleanly separated result as shown in 6.13. Considering the degrees of the nodes in 6.14, we see that the clusters corresponds well to the degrees of each node. Moving on to the cc and Γ in figures 6.15 and 6.16 respectively, it's hard to see any patterns in particular.

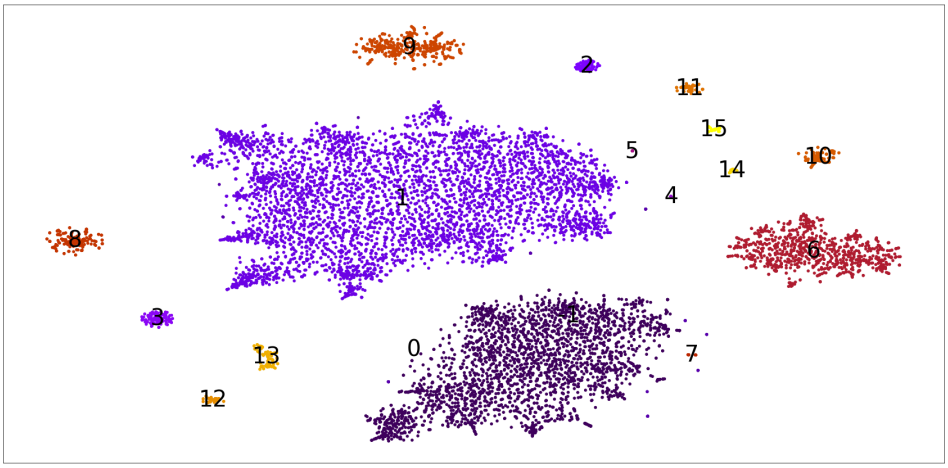

Figure 6.13: 50D embeddings for Telenor, color denotes cluster membership of node

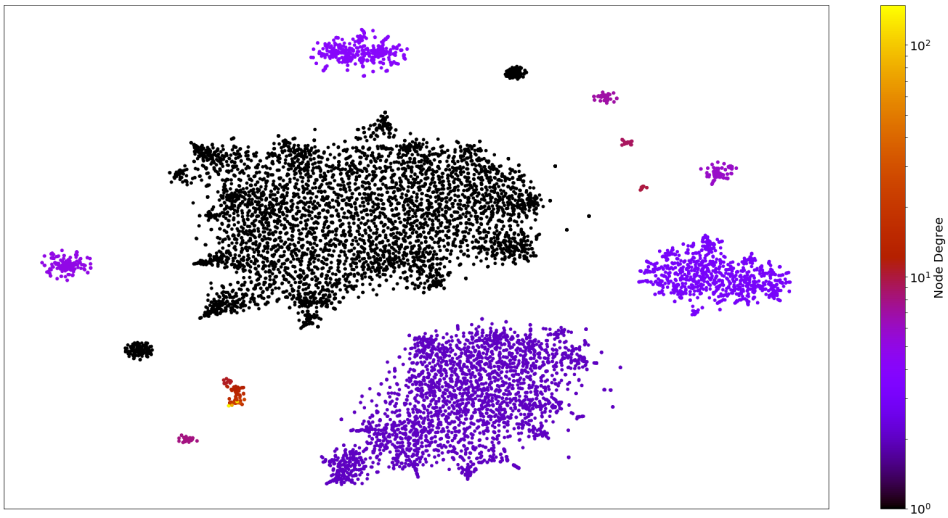

Figure 6.14: 50D embeddings for Telenor, color denotes logarithm of node degree

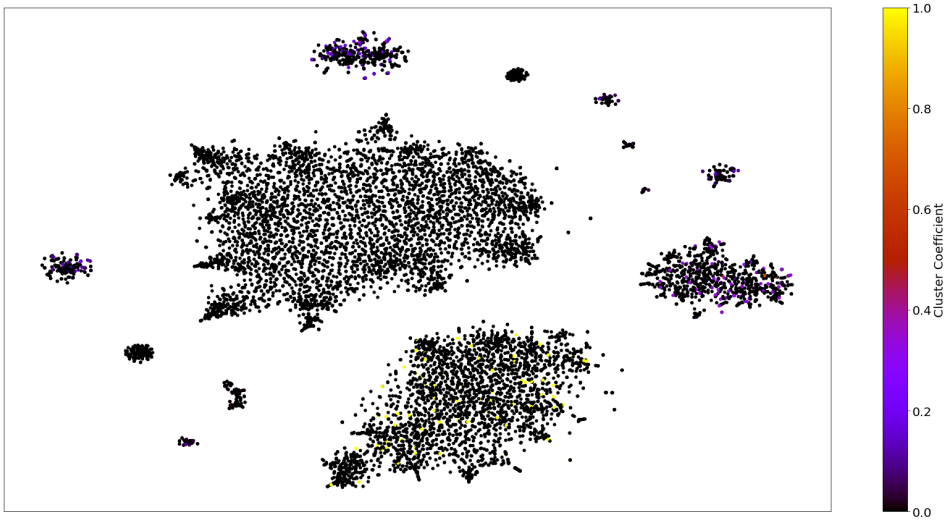

Figure 6.15: 50D embeddings for Telenor, color denotes cc of node

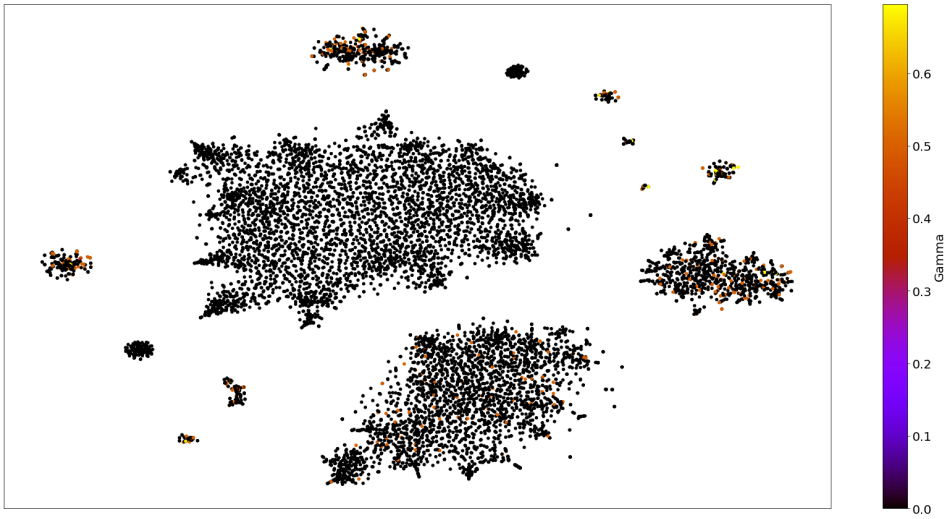

Figure 6.16: 50D embeddings for Telenor, color denotes Γ of node

6.2.3 Discussion

As was mentioned previously, low neuron count in the hidden layer of skip-gram is a likely explanation for why we could not obtain 2D embeddings that captured the properties of

the nodes' neighborhood. By increasing the dimensionality to 50 dimensions, we are increasing the ability of skip-gram to retain higher-order similarities.

As figure 6.9 (Bitcoin OTC) and 6.13 (Telenor) clearly show, using t-SNE to visualize the 50D clusterings produced by struc2vec leads to clusters that are separated much more cleanly than the 2D embeddings previously discussed. Recall that t-SNE simulates strong attraction between points that are close in the high-dimensional space, and a small repelling force between points that are far away in the high-dimensional space. When combined, these two "forces" result in clusters in the low-dimensional space that are more cleanly separated and easier to visualize. In addition, the resulting clusters are of comparable density, making density-based clustering feasible.

Figures 6.10 (Bitcoin OTC) and 6.14 (Telenor) show the 50D embeddings, colored by the number of neighbors for each node. From these figures, it is clear that the embeddings capture the number of adjacent nodes well. For both datasets, we can see that clusters are clearly separated and all contain nodes of approximately the same degree.

For the Bitcoin OTC dataset, we see that the 1-degree nodes have split into four separate clusters. For the Telenor dataset, we see the 1-degree nodes split into three clusters, echoing the same tendencies as for the Bitcoin OTC. Considering that the nodes contained in all of these clusters have the same degree, a naïve expectation would be that these should all have ended up in the same cluster. Knowing that struc2vec works by comparing ordered degree sequences, we would assume that this separation of nodes of the same degree into different clusters is due to a dissimilarity of the local graph structure beyond the immediate neighborhood the nodes.

Figures 6.17 (Bitcoin OTC) and 6.18 (Telenor) show the same embeddings with the nodes colored according to the mean of the degrees of their neighbors. Here we can clearly see that the split of 1-degree nodes into separate clusters is in fact due to differences in the degrees of their neighbors.

Looking at these two plots, we can make out clusters of nodes whose neighbors have the exact same degree. For the Bitcoin OTC datset, the cluster in the lower right corner of figure 6.17 consists of 298 nodes whos fe single neighbor is a node of degree 795 (the node with the highest degree in the Bitcoin OTC dataset). From the definition of structural equivalence in chapter 2, it follows that all the nodes within this graph are structurally equivalent and consequently the random walks emanating from these nodes will be virtually identical. In turn, this leads to them being mapped very closely in the

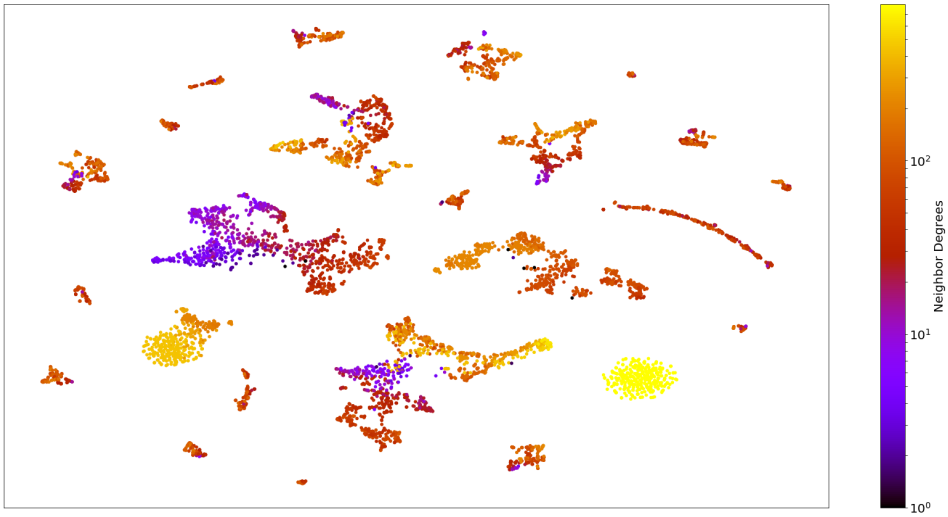

Figure 6.17: 50D embeddings for Bitcoin OTC, color denotes logarithm of mean of neighbor degrees

resulting embedding.

We can see a similar tendency in the Telenor dataset, shown in figure 6.18; one small cluster in the bottom left and one in the top middle of the graph. These clusters contain 113 and 83 nodes respectively, and all nodes belonging to each of these clusters are structurally equivalent. In the context of the Telenor dataset being a transactional network, these clusters contain nodes presumably representing people interacting with nodes representing some form of agent or hub.

Figure 6.19 shows 2-hop ego graphs for representative nodes from the four different clusters of 1-degree nodes in the 50D embeddings for the Bitcoin OTC dataset, again clearly showing the differences between nodes belonging to the different clusters.

Figures 6.11 (Bitcoin OTC) and 6.15 (Telenor) show the 50D embeddings colored according to the clustering coefficient of each node.

Considering clusters 0, 1, 2 and 3 for the Bitcoin OTC dataset, it's unsurprising that the nodes contained in these clusters have the same clustering coefficient: all nodes with a single neighbor necessarily have zero closure links and thus a clustering coefficient of 0. A more interesting case is that of cluster 5; all nodes contained in this cluster have two neighbors, and consequently their egograph can take one of two forms: there can be a closure link between the neighbors or not, which in turn leads to clustering coefficients of

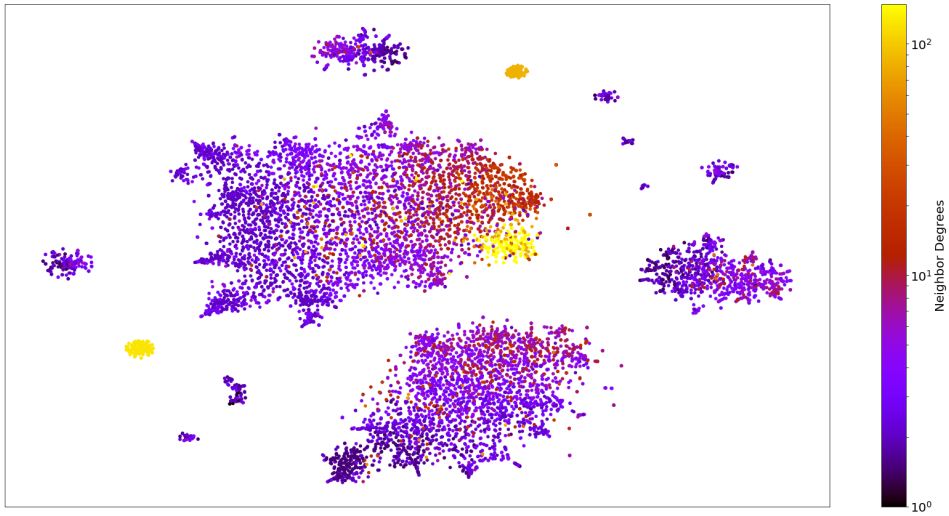

Figure 6.18: 50D embeddings for Telenor, color denotes logarithm of mean of neighbor degrees

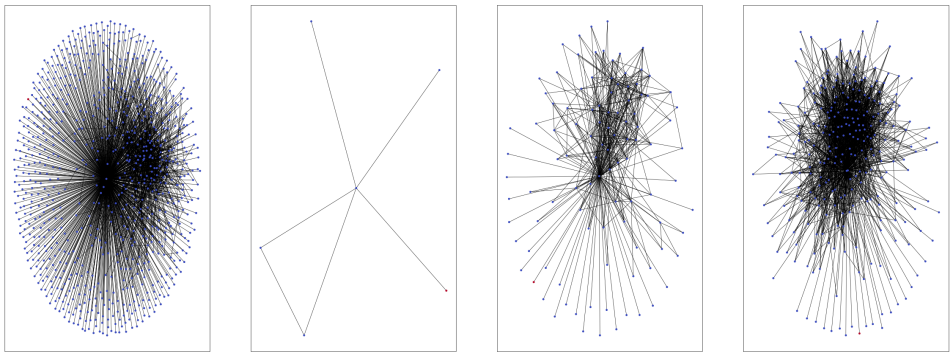

Figure 6.19: Representative nodes for cluster 0, 1, 2 and 3 in Bitcoin OTC

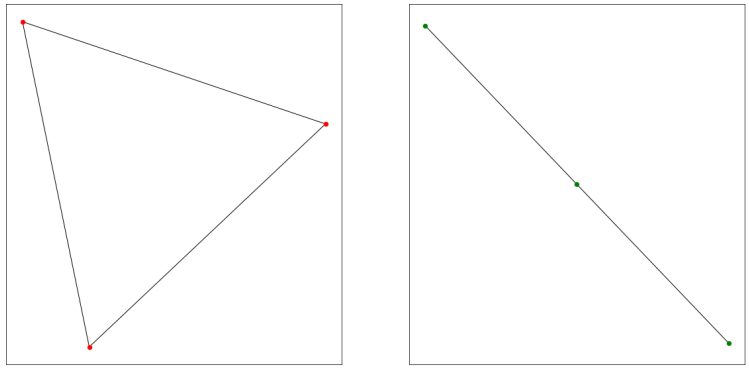

Figure 6.20: Left: $cc = 1, \Gamma = 0.5$ Right: $cc = 0, \Gamma = 0$

1 and 0 respectively. This explains why cluster 5 looks the way it does. The structure of the egonet for each of these cases can be viewed in figure. 6.20.

The Γ-value of each node is presented in figures 6.12 (Bitcoin OTC) and 6.16 (Telenor). As with the clustering coefficient, nodes within each cluster tend to vary a lot and no single value of Γ seems to dominate. Again, for the Bitcoin OTC, cluster 5 is a special case. In this case, the special case in the definition of Γ for nodes with two adjacent nodes and one closure link results in some of the nodes having a Γ-value of 0.5. For the Telenor dataset, we see the same trend.

A brief recap is in order: we have seen that 50-dimensional embeddings from struc2vec cleanly clusters the nodes according to the number of adjacent nodes they have. This is not at all surprising, considering that the degree of a node is the clearest and most indicative attribute a node has of its structural identity. At the same time, we have seen that the clustering coefficient and the Γ are captured poorly. We'll get back to why this is the case.

6.3 Higher dimensions (100D and 300D)

We have seen that increasing the dimensionality of the embeddings from 2D to 50D leads to more meaningful clusterings. This section will explore whether increasing the

dimensionality even further leads to clusterings that capture even more of the local structure of a node. In their seminal paper on word embeddings, Mikolov et. al. [15] generated word embeddings of up to several hundred dimensions, suggesting that we may see improvements by increasing the dimensionality even further.

6.3.1 100D Embeddings - Bitcoin OTC

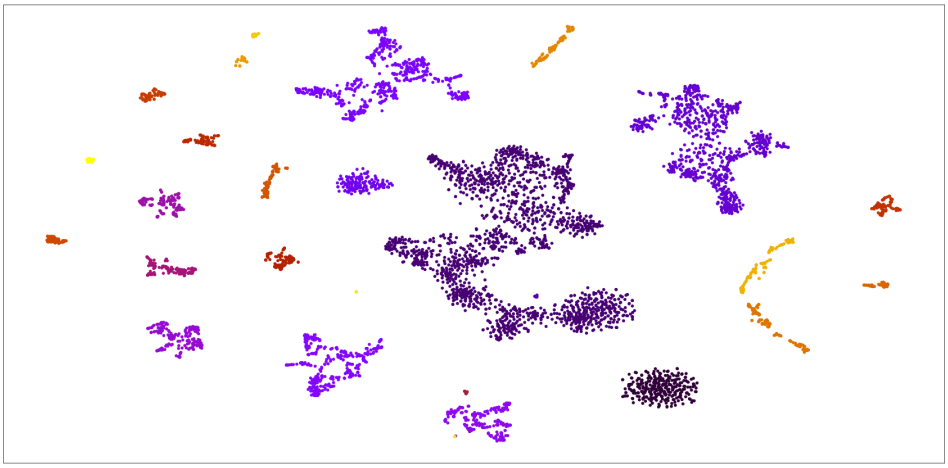

Figure 6.21: 100D embeddings for Bitcoin OTC, color denotes cluster membership

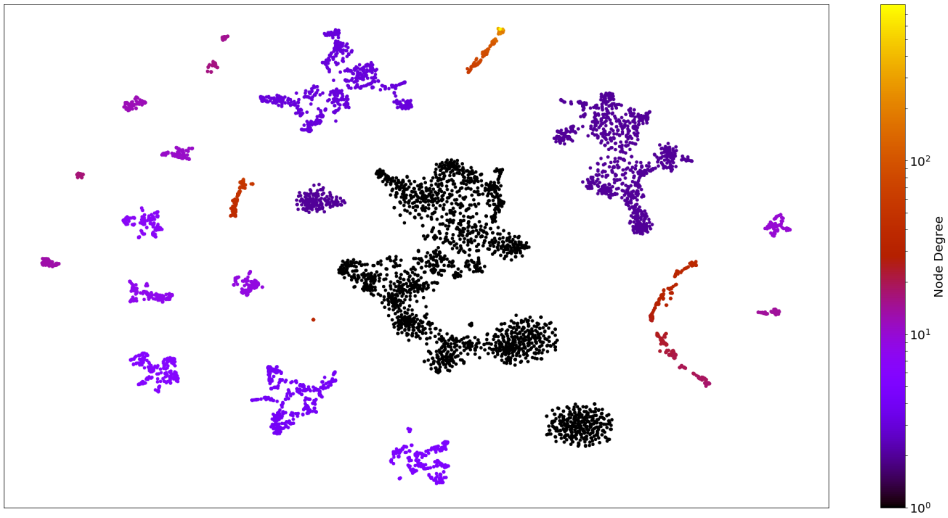

Figure 6.22: 100D embeddings for Bitcoin OTC, color denotes logarithm of node degree

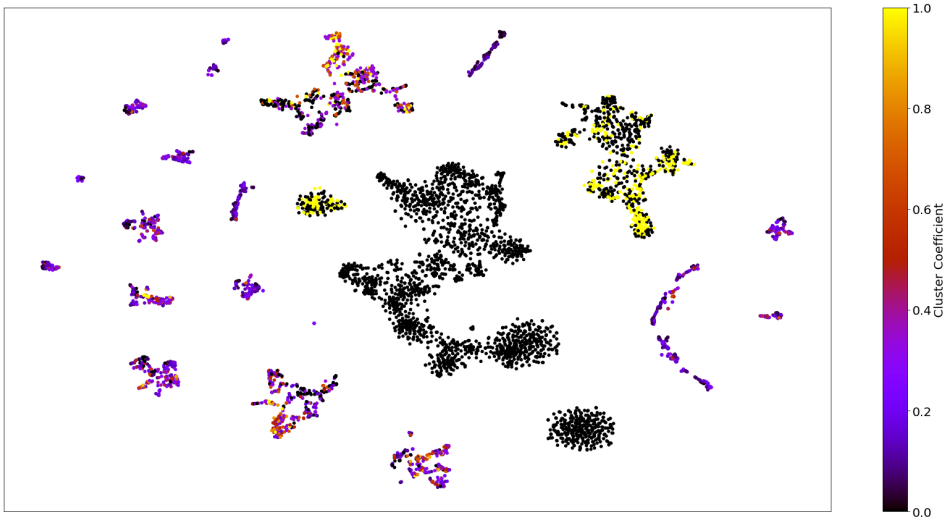

Figure 6.23: 100D embeddings for Bitcoin OTC, color denotes cc of node

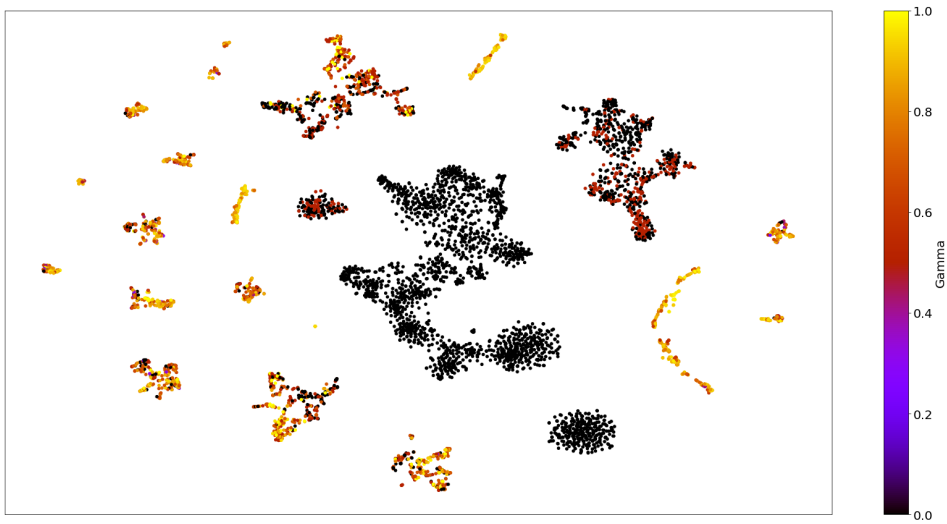

Figure 6.24: 100D embeddings for Bitcoin OTC, color denotes Γ of node

6.3.2 100D Embeddings - Telenor

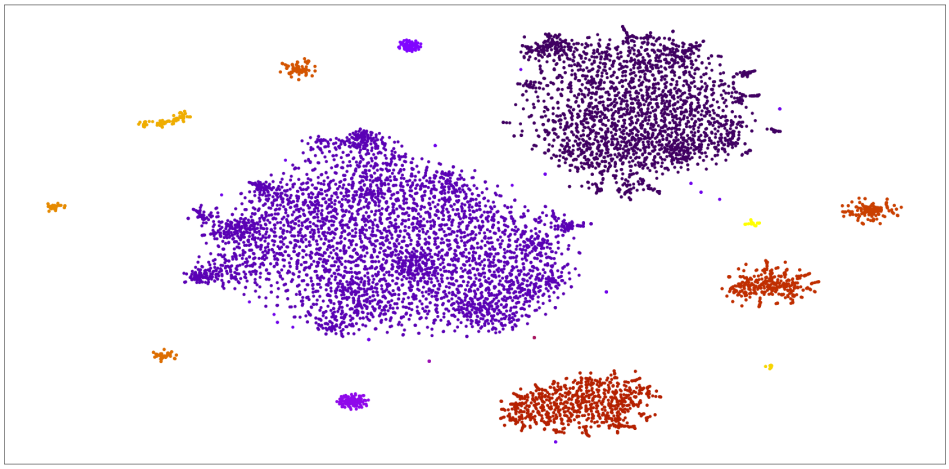

Figure 6.25: 100D embeddings for Telenor, color denotes cluster membership of node

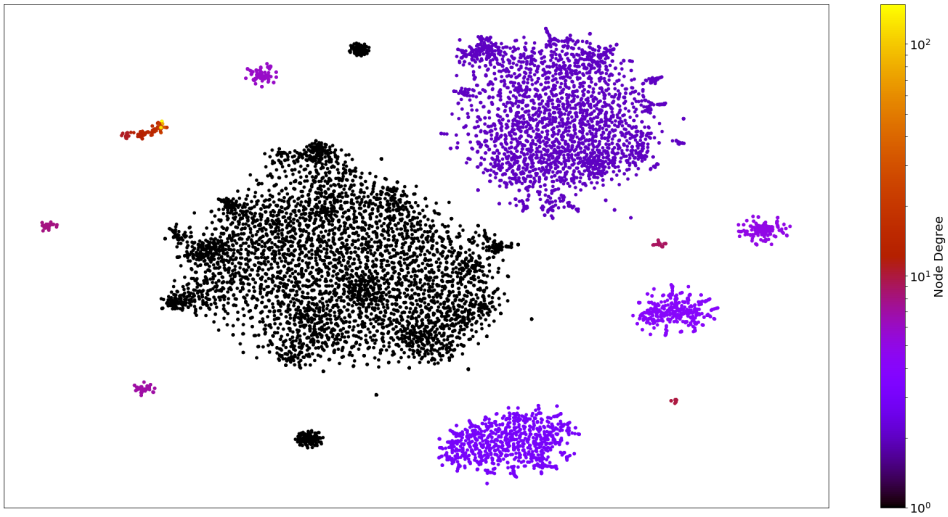

Figure 6.26: 100D embeddings for Telenor, color denotes the logarithm of node degree

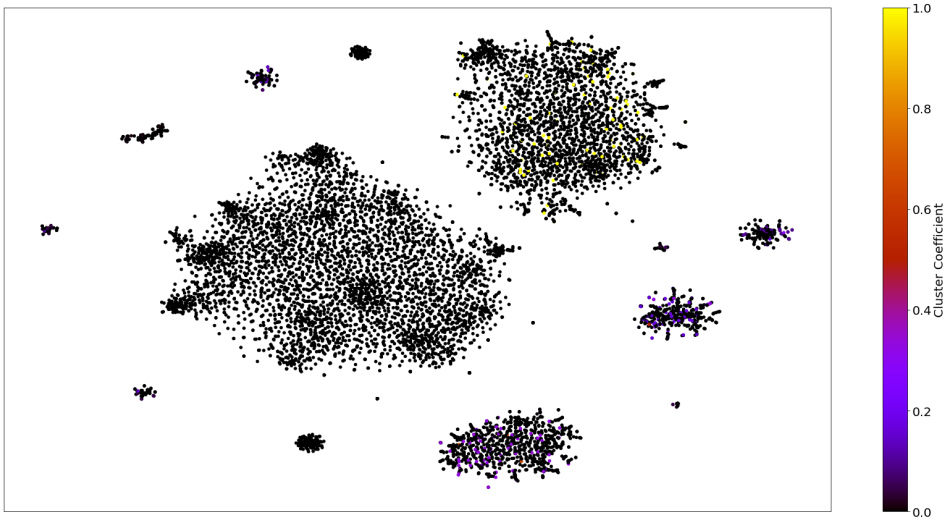

Figure 6.27: 100D embeddings for Telenor, color denotes value of cc

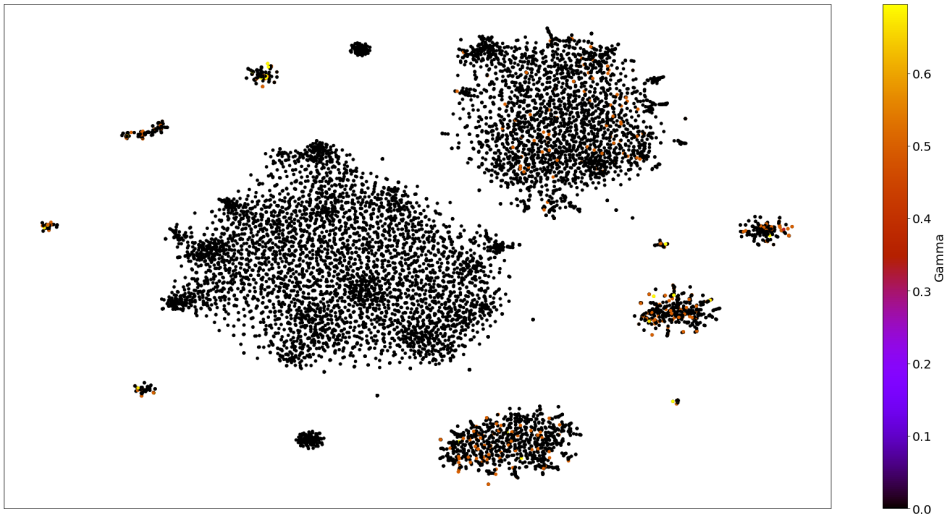

Figure 6.28: 100D embeddings for Telenor, color denotes value of Γ

6.3.3 300D Embeddings - Bitcoin OTC

The clusterings in figure 6.29 are overall very similar the previously discussed 100D embeddings. As we've seen before, the clusters have shifted around but this is to be expected due to the probabilistic nature of t-SNE.

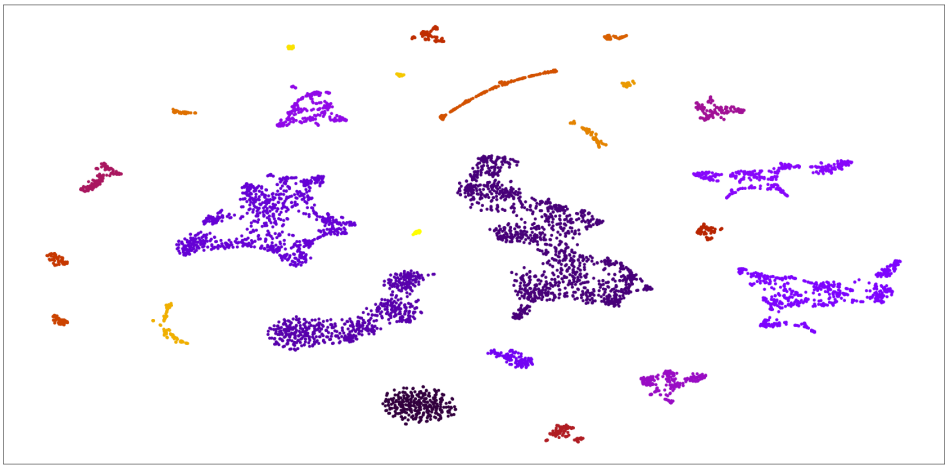

Figure 6.29: 300D embeddings for Bitcoin OTC, color denotes cluster membership

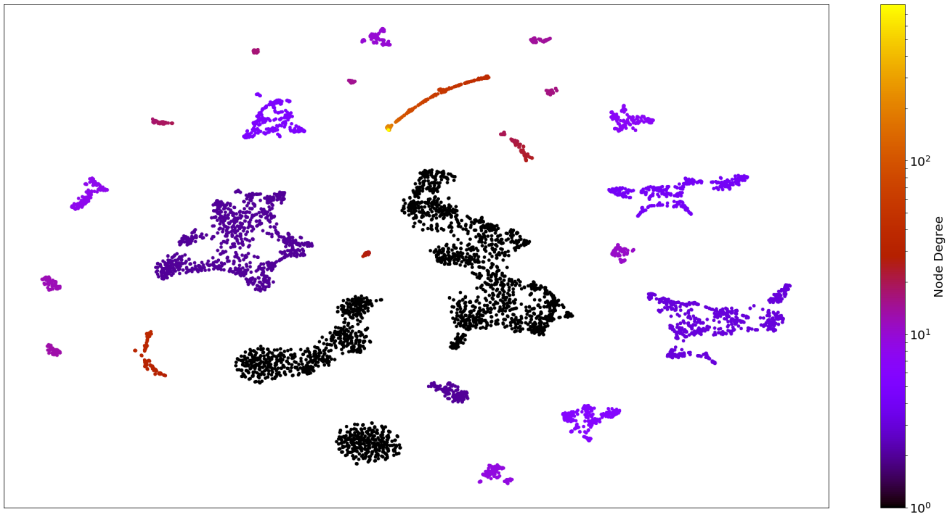

Figure 6.30: 300D embeddings for Bitcoin OTC, color denotes logarithm of node degree

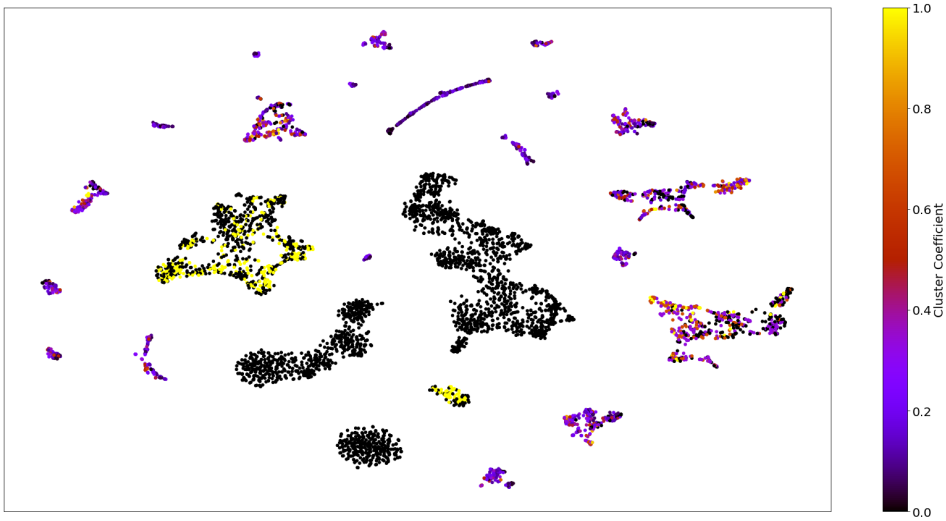

Figure 6.31: 300D embeddings for Bitcoin OTC, color denotes cc of node

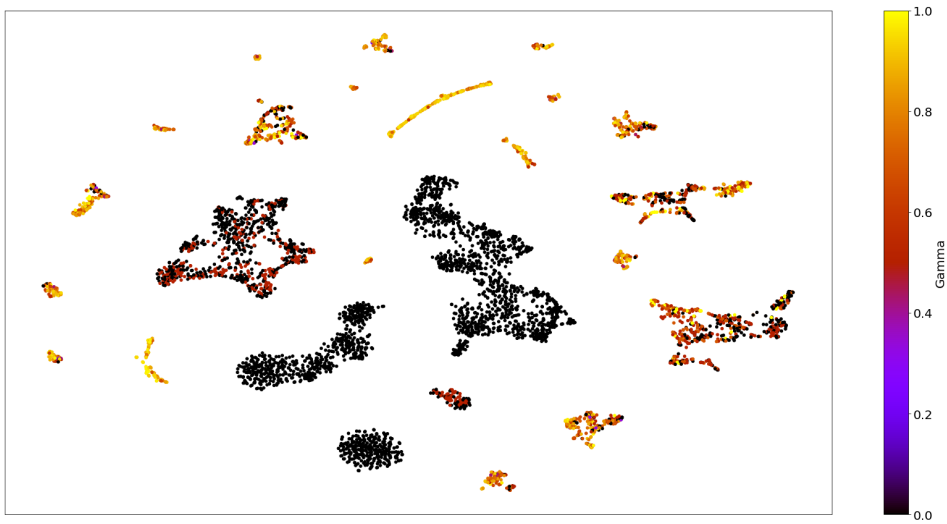

Figure 6.32: 300D embeddings for Bitcoin OTC, color denotes Γ of node

6.3.4 300D Embeddings - Telenor

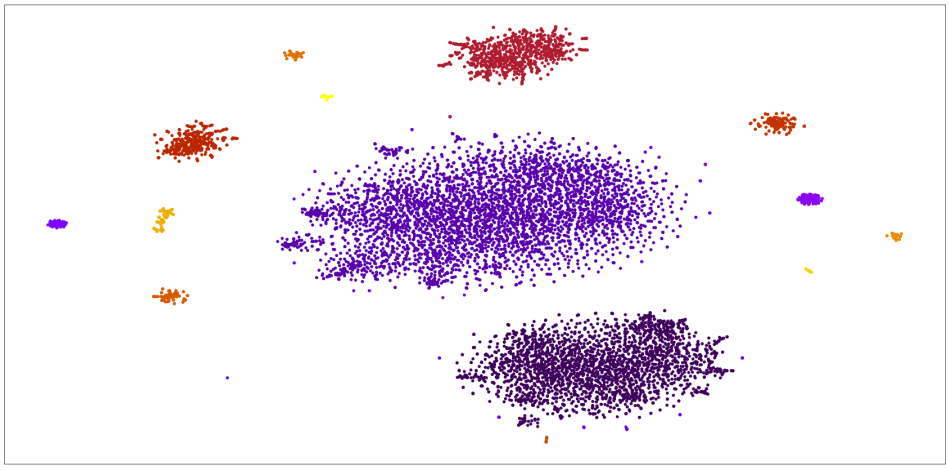

Figure 6.33: 300D embeddings for Telenor, color denotes cluster membership of node

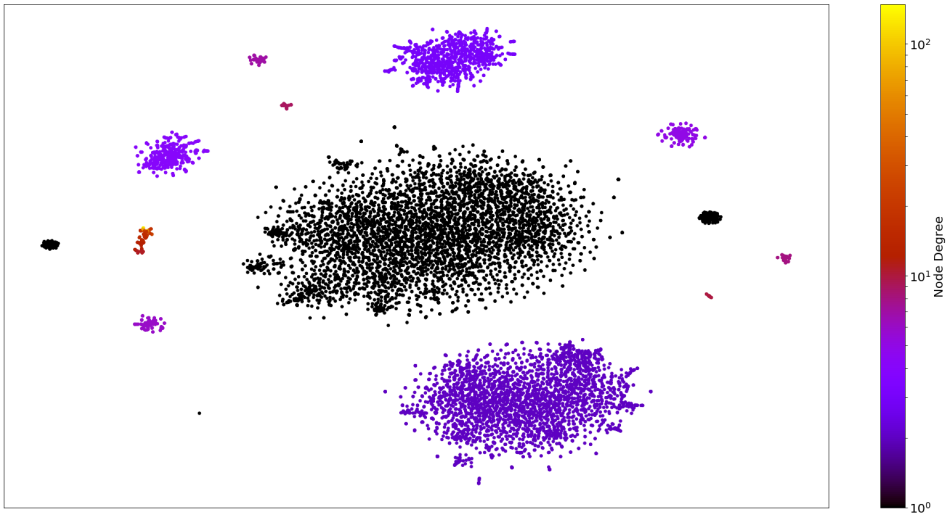

Figure 6.34: 300D embeddings for Telenor, color denotes the logarithm of node degree

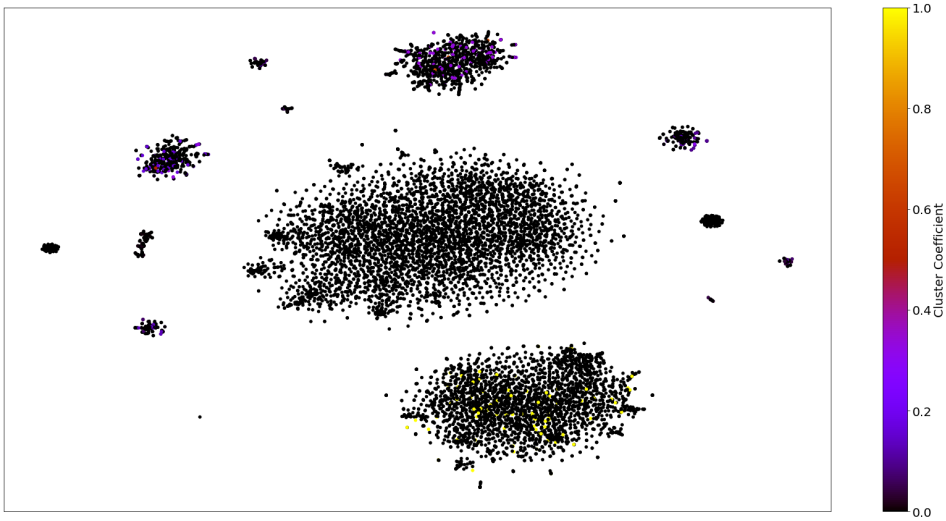

Figure 6.35: 300D embeddings for Telenor, color denotes value of cc

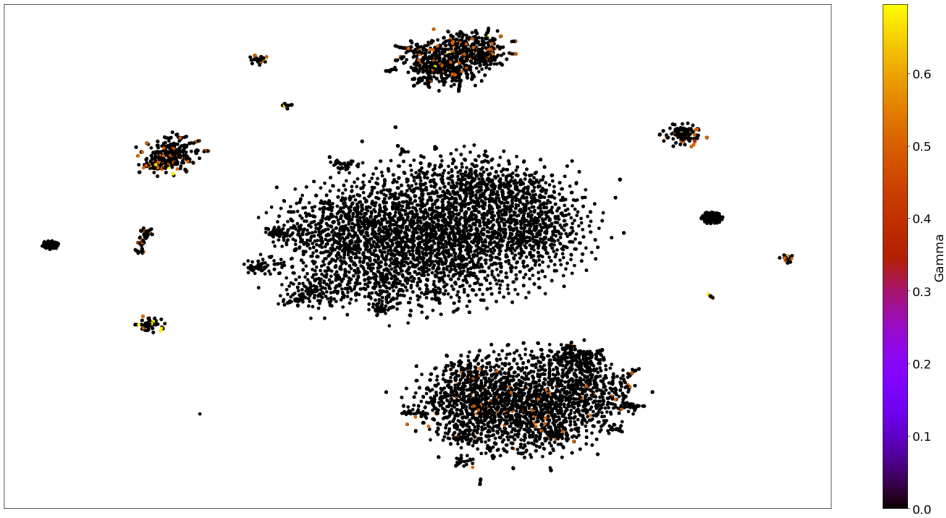

Figure 6.36: 300D embeddings for Telenor, color denotes value of Γ

6.3.5 Discussion

Comparing figures 6.10 and 6.22 we can quickly tell that the 100D embeddings for the Bitcoin OTC dataset capture the node degrees in a manner that is comparable to the 50D embeddings. Some minor differences are present, however. Considering the clusters containing nodes that have a single neighbor, we can see that some of the clusters present in the 50D embeddings have merged, forming one larger cluster. The cluster containing neighbors of the single 795-degree node still persists. We also note that the cluster containing two-degree nodes have split into two separate clusters, while for the 50D embeddings they were all contained in a single cluster.

Table 6.1 shows some metrics for the neighbors of nodes belonging to these two clusters. We can tell that the neighbors of the nodes in cluster 3 have on average fewer neighbors than their counterparts in cluster 4.

Table 6.1: Table showing statistics for neighbors of nodes in cluster 3 and cluster 4

	Cluster 3		Cluster 4	
Metric	Mean	Median	Mean	Median
Degrees	173.26	71.0	330.925	239
Cluster coefficient	0.075	0.058	0.052	0.028
Γ	0.797	0.856	0.853	0.923

For the 300D embeddings, we notice that the 1-degree nodes have formed an additional cluster, making these nodes look more similar to the 50D embeddings than the 100D ones. We once again recognize the familiar circular cluster containing neighbors of the 795-degree node, and we'll omit further discussion of this cluster.

In order to be able to say something about the difference between the distribution of nodes into clusters, we have calculated the adjusted rand index between the 50D clustering and each of the 100D and 300D clusterings.

Adjusted rand index for Bitcoin OTC

- ARI(50D, 100D): 0.628
- ARI(50D, 300D): 0.793

Although the adjusted rand index here seems to indicate that the clusterings have a certain degree of dissimilarity, the result here is somewhat misleading. If we exclude all nodes having a degree of 2 or less from the calculation of the adjusted rand index, we get the following results:

Adjusted rand index for Bitcoin OTC (nodes with degree $>$ 2)

- ARI(50D, 100D): 0.946
- ARI(50D, 300D): 0.966

This means that it is predominantly the very low-degree nodes that change cluster membership between clusterings and that the vast majority of high-degree nodes belong to the same cluster regardless of the dimensionality. Because the low-degree nodes are so many and belong to a few large clusters, any merge or split of these large clusters will lead to a major perturbation in the ARI measure. As we have already seen some of these merges and splits of the clusters containing low-degree nodes across different dimensions (that are likely due to the probabilistic nature of t-SNE), we can conclude that ARI here

gives a skewed view of the differences between the clusterings of the 50D, 100D and 300D embeddings, and that these clusterings for the the Bitcoin OTC dataset are qualitatively very similar.

Considering the Telenor dataset, the 100D and 300D embeddings look virtually identical to the 50D ones. This is reflected in the ARI calculated between the clustering of the 50D embedding and the two high-dimensional embeddings generated. As we can see, the ARI in both cases are close to 1, confirming that the clusterings are very close to identical.

Adjusted rand index for Telenor

- ARI(50D, 100D): 0.99
- ARI(50D, 300D): 0.984

6.4 Do Smaller Clusters Find More Descriptive Roles?

Viewing the clusters in 6.9, we see that some of them have irregular shapes. As we've mentioned previously, the nature of t-SNE is to place similar nodes in the high-dimensional space close in the low-dimensional space, while nodes that are dissimilar in the high-dimensional space will be mapped farther away. Consequently, clusters of irregular and weird shapes suggest that there are some dissimilarities between the nodes within each of the clusters in figure 6.9.

We know that the larger clusters found previously capture meaningful patterns in the data. In order to determine whether the shapes of the clusters suggest dissimilarities or whether they can be attributed to the probabilistic nature of t-SNE, we re-ran the DBSCAN algorithm on the Bitcoin OTC data with a focus on finding smaller clusters (lowering the ϵ parameter to 0.012) and obtained a clustering where the nodes are much smaller. These clusters are shown in figure 6.37.

We are going to use what was cluster 2 in figure 6.9 (now split into clusters 2, 36, 37 and 41) as an example to visualize the differences. We know from before that all the nodes in these four clusters have one single neighbor. We also know that struc2vec performs random walks based on similarities obtained by comparing ordered degree sequences and

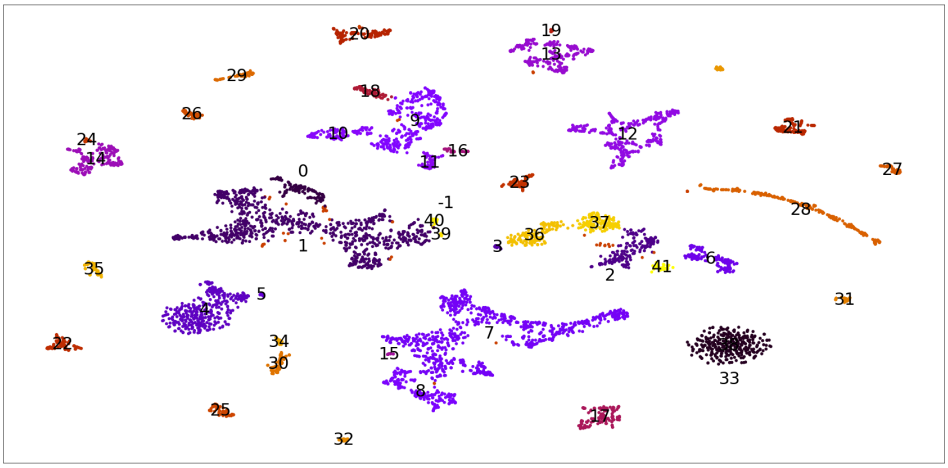

Figure 6.37: 50D embeddings generated by struc2vec, reduced to 2D with t-SNE, color denotes cluster membership

that these are the bases for the embeddings. Considering that all the nodes contained in these four clusters have one single neighbor, the degree of this single neighbor is vital to the calculation of the similarities and consequently to how the random walk proceeds further.

Figure 6.38 visualizes these four clusters up close, colored according to the degree of their single neighbor. We see a clear tendency here; the degree of the single neighbors becomes gradually lower as we move to from left to right in the figure. We'd like to examine nodes of all degrees, however, not only those with a single neighbor. Figure 6.39 shows cluster 28 in 6.9 which contains predominantly high-degree nodes. The nodes in this figure are colored according to the *mean* of the neighbors' degrees. In this case the tendency is not as apparent, but again there seems to be a slight tendency for the mean degree of the neighbors to decrease slightly as we move from left to right.

Finally, figure 6.17 shows an overview of all the nodes of the 50D embeddings colored according to the mean of the degree of their neighbors. Viewing all the nodes like this, we see a clear trend showing that within each cluster there is clear separation that can be attributed at least in part to the degrees of their neighbors.

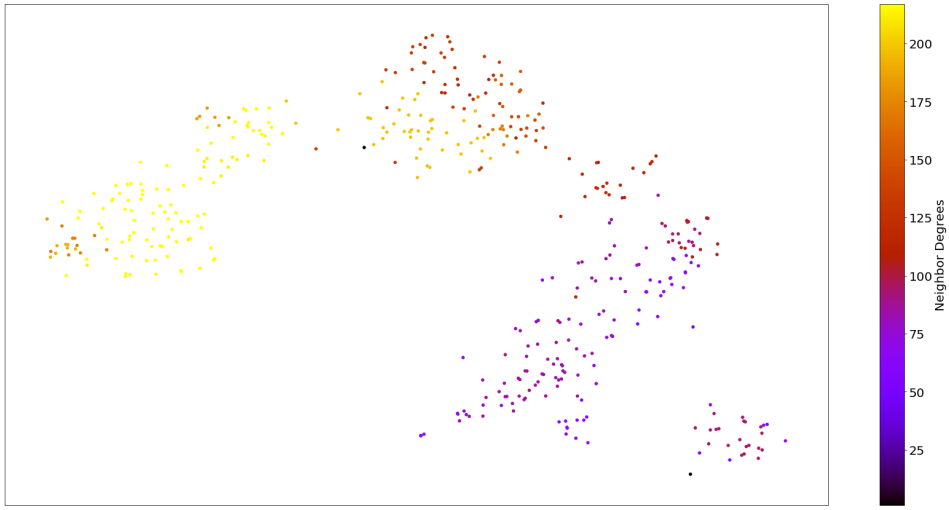

Figure 6.38: 50D embeddings for Bitcoin OTC, zoomed to clusters $2, 36, 47$ and 41, color denotes mean of neighbor degrees

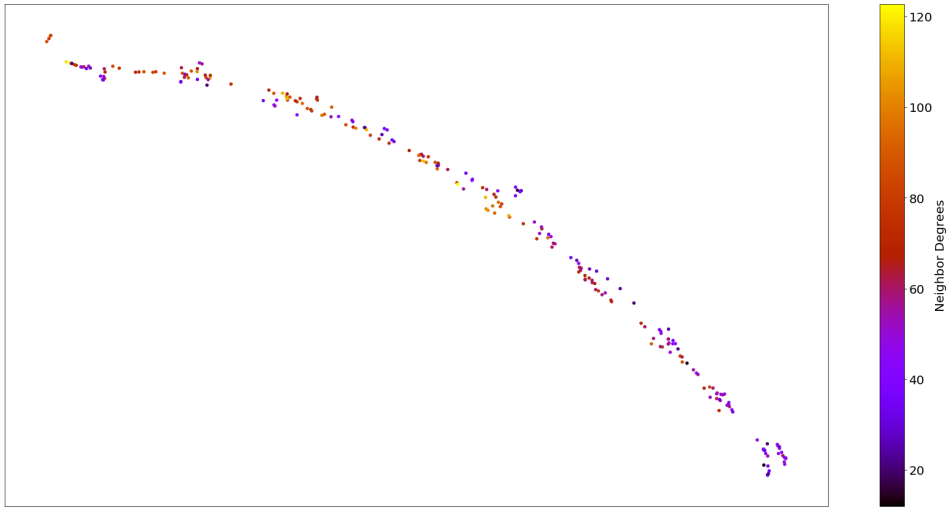

Figure 6.39: 50D embeddings for Bitcoin OTC, zoomed to cluster 28, color denotes mean of neighbor degrees

6.5 Why Do the Embeddings Seemingly Fail to Capture the Clustering Coefficient and Γ?

Having examined embeddings of different dimensionalities, it would at first glance seem like the clustering coefficient and the Γ are not very well captured by the embeddings. Looking more closely, we find that they are captured well, although in a subtler way than can be detected by looking just at each node's 1-hop neighborhood.

For all the 1-degree nodes, we know that the clustering coefficient and the Γ are both 0. This means that we don't really expect to see any large differences between these nodes. Instead, struc2vec has captured the Γ-values of the single neighbor of these 1-degree nodes. Examining figure 6.40, where the nodes are colored according to the mean of the Γ-values of its neighbors, we can see that within each cluster of 1-degree nodes there are clear tendencies for nodes with neighbors of similar Γ-values to be co-located within the cluster.

For the remaining k-degree nodes, we can see that when colored according to the Γ of the nodes, there is seemingly no pattern. When viewing the mean of the Γ-values of each node's neighbors, we can see a much clearer tendency of intra-cluster variations.

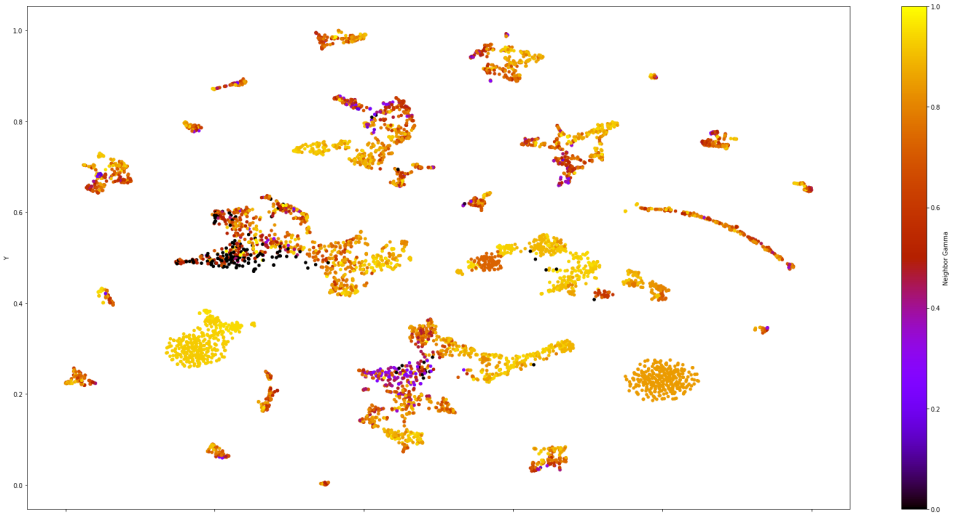

Figure 6.40: 50D embeddings generated by struc2vec, color denotes mean of Γs of neighbors

6.6 Lower-Dimensional Embeddings

In order to see few dimensions we can use and still obtain clusterings of high quality, we have generated embeddings for dimensionalities in between the relatively low-dimensional 3D embeddings and the 50D embeddings we just considered. In particular, we chose 10D and 25D as these sizes. Figures 6.41 and 6.42 show the their respective embeddings, both for the Bitcoin OTC dataset, colored by degree.

As we can see from these two figures, there is really no qualitative difference between these lower-dimensional clusterings and the 50D clusterings for the Bitcoin OTC dataset, suggesting that we can use a lot fewer dimensions for our embeddings and still obtain good results.

The ARI index confirms this: **Adjusted rand index for Bitcoin**

- ARI(50D, 10D): 0.81
- ARI(50D, 25D): 0.79

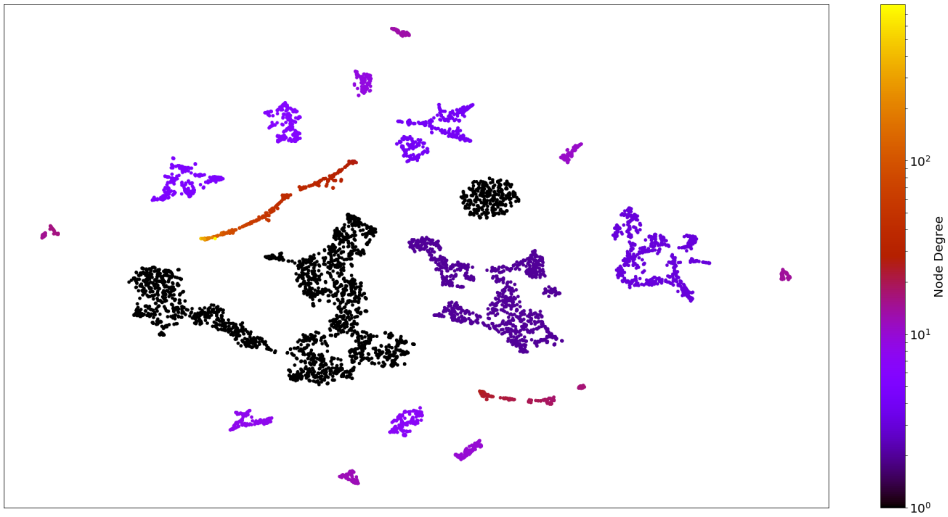

Figure 6.41: 10D embeddings generated by struc2vec, reduced to 2D with t-SNE, coloring denotes log of the number of adjacent nodes

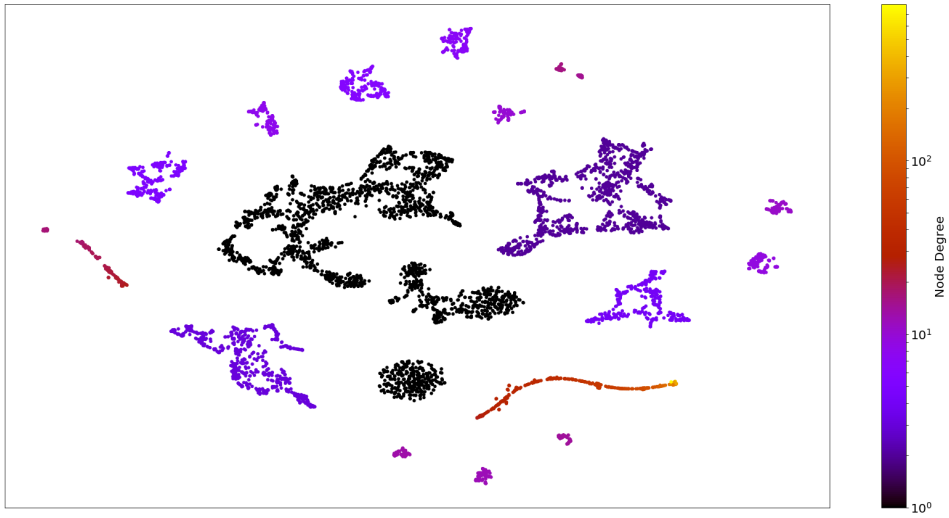

Figure 6.42: 25D embeddings generated by struc2vec, reduced to 2D with t-SNE, coloring denotes log of the number of adjacent nodes

Chapter 7

Conclusion

In this chapter we will summarize the results as well as answer the research questions that were posed in the introductory chapter. Finally, we will provide some direction as to how future work may improve our solution.

7.1 Summary and Conclusions

This thesis has shown how struc2vec can be a valuable and useful tool for generating node embeddings that preserve the graph's local structure. More importantly, we have generated embeddings that not only capture trivial information like the node degree, but features of the extended neighborhood of the node. We determined that struc2vec struggles to embed large networks directly to two dimensions, likely due to too few neurons in the hidden layer used by word2vec. As we increased the dimensionality of the space into which the nodes are embedded, the quality of the clusterings increased. Increasing the dimensionality beyond 50 dimensions yielded little to no gains, while we were able to get meaningful clusterings using as few as 10 dimensions.

7.2 Answers to Research Questions

Research Question 1: What can be done to embed network structures into a lower-dimensional space?

We have shown that struc2vec successfully generates embeddings that preserve the network structure for graphs of non-trivial sizes. In addition, these embeddings can have as few as 10 dimensions that still capture the relevant features.

Research Question 2: Can these embeddings be clustered in a meaningful way?

We have shown that when using the 2D embeddings directly, it is hard to effectively cluster these embeddings. When dealing with embeddings of much higher dimensionalities, we have used t-SNE to cope with this. The clusterings obtained from t-SNE clearly capture several aspects of the network structure.

7.3 Future Work

7.3.1 Use Node And Edge Attributes

Although nodes and edges in both the Bitcoin OTC graph and the Telenor graph contain additional features such as weights and timestamps, they were discarded for the purpose of this thesis. Using these kinds of auxiliary features is likely to improve the quality of the role extraction, and should be explored further.

Bibliography

[1] Kevin Beyer, Jonathan Goldstein, Raghu Ramakrishnan, and Uri Shaft. When is "nearest neighbor" meaningful? In *International conference on database theory*, pages 217–235. Springer, 1999.

[2] Geoffrey Canright. About gamma, unpublished writeup. 2018.

[3] Shaosheng Cao, Wei Lu, and Qiongkai Xu. Grarep: Learning graph representations with global structural information. In *Proceedings of the 24th ACM International on Conference on Information and Knowledge Management*, pages 891–900. ACM, 2015.

[4] Steven H. Strogatz Duncan J. Watts. Collective dynamics of 'small-world' networks. *Nature*, 393:440–442, 1998.

[5] Santo Fortunato and Darko Hric. Community detection in networks: A user guide. *Physics Reports*, 659:1–44, 2016.

[6] Palash Goyal and Emilio Ferrara. Graph embedding techniques, applications, and performance: A survey. In *ArXiv e-prints*, volume 1705, 2017.

[7] Aditya Grover and Jure Leskovec. node2vec: Scalable feature learning for networks. In *ArXiv e-prints*, volume 1607, 2016.

[8] Sidharth Gupta, Xiaoran Yan, and Kristina Lerman. Structural properties of ego networks. Social Computing, Behavioral-Cultural Modeling, and Prediction, pages 55–64, Cham, 2015. Springer International Publishing.

[9] Geoffrey E Hinton and Sam T Roweis. Stochastic neighbor embedding. In *Advances in neural information processing systems*, pages 857–864, 2003.

[10] Lawrence Hubert and Phipps Arabie. Comparing partitions. *Journal of classification*, 2(1):193–218, 1985.

[11] Solomon Kullback and Richard A Leibler. On information and sufficiency. *The annals of mathematical statistics*, 22(1):79–86, 1951.

[12] Srijan Kumar, Francesca Spezzano, VS Subrahmanian, and Christos Faloutsos. Edge weight prediction in weighted signed networks. In *Data Mining (ICDM), 2016 IEEE 16th International Conference on*, pages 221–230. IEEE, 2016.

[13] Francois Lorrain and Harrison C White. Structural equivalence of individuals in social networks. In *Social Networks*, pages 67–98. Elsevier, 1977.

[14] Laurens van der Maaten and Geoffrey Hinton. Visualizing data using t-sne. *Journal of machine learning research*, 9(Nov):2579–2605, 2008.

[15] Tomas Mikolov, Kai Chen, Greg Corrado, and Jeffrey Dean. Efficient estimation of word representations in vector space. In *ArXiv e-prints*, volume 1301, 2013.

[16] Bryan Perozzi, Rami Al-Rfou, and Steven Skiena. Deepwalk: Online learning of social representations. In *ArXiv e-prints*, volume 1403, 2014.

[17] Leonardo F. R. Ribeiro, Pedro H. P. Saverese, and Daniel R. Figueiredo. struc2vec: Learning node representations from structural identity. In *ArXiv e-prints*, volume 1704, 2017.

[18] Ryan A Rossi and Nesreen K Ahmed. Role discovery in networks. *IEEE Transactions on Knowledge and Data Engineering*, 27(4):1112–1131, 2015.

[19] Sam T Roweis and Lawrence K Saul. Nonlinear dimensionality reduction by locally linear embedding. *science*, 290(5500):2323–2326, 2000.

[20] Lee Douglas Sailer. Structural equivalence: Meaning and definition, computation and application. *Social Networks*, 1(1):73–90, 1978.

[21] 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. ACM, 2016.

[22] Wayne Zachary. *An Information Flow Model for Conflict and Fission in Small Groups1*, volume 33. 1976.

Appendix

Write your appendix here...