UNIVERSIDADE DE LISBOA
FACULDADE DE CIÊNCIAS
DEPARTAMENTO DE FÍSICA

Decomposition of inflationary systems into scale-free networks

Ricardo José Velho Ramos

Mestrado em Engenharia Física

Dissertação orientada por:
Professor Doutor João Pires da Cruz
Professor Doutor Cristóvão Dias

2025

Acknowledgements

Firstly, I want to express my deepest gratitude towards Dr. João Pires da Cruz for the unceasing support, help and passion he demonstrated throughout my thesis. He was always present, helping me and guiding me throughout this journey and without his inputs, his drive and passion for the work being done I would have never been able to make this thesis.

A special thanks to Carlos Garrido and João Marques Dias from Closer, their help, brainstorming and constant input were the reason I was able to succeed in the early part of this work. Without their help this work would have become much more difficult.

I would also like to thank a person that helped me in this journey, Armando Teixeira. His inputs and critical look into my thesis helped me improve and structure my work better as well as tackle some problems.

Finally, to my friends and family, particularly my mom, my dad, my sister, my grandma and my uncle, I want to make a special regard for their support my whole life, particularly these past 5 years where I would have never been able to get this far without them. Without my dad, maybe because of his experience, but definitely because of his words, this journey would not have been concluded. He always gave me hope and discussed the issues I was facing whenever they arrived. My mom as always been my light and our walks, talks and moments have always lifted my spirit and helped me push forward, her and my dad were the reason I ever got this far. To my sister, Carolina, thank you for always being there in the toughest moments and for all the words and moments.

Abstract

Due to their overly complex nature, real-world networks cannot be understood through typical mathematic tools, such as diferential equations, for that reason we use networks and analyse them through algorithms to understand them. This work focuses on decomposing scale-free networks into Barabási-Albert networks to better understand them. We are able to make an analysis of these networks through their decomposition and to identify redundant data to reduce size and complexity. We use tools such as autoencoders to create the algorithms and to go from integer-dimensional spaces to networks and vice versa.

An integer dimension space, such as a Euclidian space is a projection of a network, for the case of BA (Barabási-Albert) networks, in the limit they tend to a one-dimension space, or a curved line. We can make use of this notion to try and recreate the original integer dimension space from the created BA networks.

We can tested these concepts using a network whose projection is a Euclidian space and a market network, whose projection is a fractal space, which are created with the help of autoencoders. Autoencoders are also used to try and understand their differences and to validate the results. An accuracy analysis will be made in order to identify redundant data in the networks and possibly simplify them.

This work can help to simplify and better comprehend real-world networks, such as the stock market and social networks.

Keywords: Barabási-Albert, autoencoders, integer-dimensional spaces, scale-free networks

Resumo

O mundo atual é constituído por um universo imenso de redes, mesmo as nossas vidas são regidas por redes. Desde as nossas redes de amigos, redes sociais, redes de trabalho, redes de mercado a redes de informação. A complexidade destes sistemas é demasiado grande para as nossas ferramentas convencionais, como a matemática e a física as compreenderem, uma vez que são sistemas com inúmeras dimensões. Desta forma são necessárias soluções inovadores de modo a conseguir percebê-las e interpretá-las.

Redes scale-free são estruturas complexas caracterizadas por uma distribuição de grau que segue uma lei de potência, $P(k) \sim k^{-\gamma}$, onde poucos nós (hubs) concentram a maioria das conexões, enquanto a maioria dos nós possui grau baixo. Esse comportamento emergente é típico de muitos sistemas naturais e tecnológicos, como redes sociais, redes metabólicas e a internet. A principal propriedade das redes scale-free é a sua robustez a falhas aleatórias e, ao mesmo tempo, a sua vulnerabilidade a ataques dirigidos aos hubs. O modelo de crescimento preferencial de Barabási–Albert é o paradigma mais comum para gerar esse tipo de rede, onde novos nós se conectam com maior probabilidade aos nós mais conectados. Essa topologia não homogênea tem implicações significativas na dinâmica de propagação, resiliência estrutural e controle de sistemas complexos.

A análise de redes scale-free enfrenta desafios substanciais devido à sua complexidade estrutural, heterogeneidade topológica e comportamento emergente. A presença de hubs altamente conectados, que rompem a suposição de homogeneidade típica em muitos modelos matemáticos clássicos, dificulta a aplicação de ferramentas tradicionais baseadas em estatísticas médias, como a teoria dos grafos regulares ou equações diferenciais convencionais. A distribuição de grau em lei de potência implica ausência de uma escala característica, o que inviabiliza métricas como grau médio ou variância como métodos descritivos eficazes. Além disso, muitos processos dinâmicos como difusão, sincronização ou propagação de falhas apresentam comportamentos não-lineares e não-locais nessas redes, exigindo abordagens alternativas, como métodos de teoria de sistemas complexos, análise fractal, teoria de percolação e ferramentas da física estatística fora do equilíbrio. Assim, a matemática convencional, desenvolvida para estruturas uniformes e sistemas lineares, mostra-se insuficiente para capturar a riqueza fenomenológica e os padrões auto-organizados que emergem nas redes scale-free.

Uma forma de analisar este tipo de redes parte da utilização de redes de Barabási-Albert com m=1, onde m representa as ligações por iteração. Estas redes são formadas através de um nodo inicial que se liga a um novo nodo na segunda iteração. Cada iteração cria um nodo e uma ligação onde novos nodos se conectam com maior probabilidade aos nós mais conectados numa estrutura de árvore. Desta forma surgem os mencionados hubs. Este tipo de redes é particularmente útil pois segue uma lei de potência e a sua simplicidade permite que seja feita uma análise rigorosa e mais prática. Utilizamos então uma nova abordagem através de redes para compreender este tipo de sistemas.

Esta metodologia está diretamente ligada com análise fractal. A análise fractal de redes scale-free revela que essas estruturas não são adequadamente descritas por espaços de dimensão inteira, como os tradicionalmente utilizados na geometria euclidiana. Em vez disso, apresentam propriedades típicas de objetos fractais, como auto-similaridade e escalabilidade, que requerem o uso de definições de dimensões não inteiras como a dimensão de Hausdorff ou de box-counting para uma descrição mais fiel. A geometria que projeta as redes sobre espaços euclidianos de dimensão inteira (como planos ou volumes), resulta numa simplificação excessiva da topologia real da rede, ignorando a distribuição desigual de conexões e a centralidade dos hubs. No caso específico das redes de Barabási–Albert, observa-se que, à medida que o número de nós tende ao infinito, a estrutura global da rede converge para um sistema com comportamento efetivamente unidimensional: os novos nós conectam-se preferencialmente a um subconjunto dominante, formando uma "espinha dorsal" hierárquica que se

assemelha a uma linha curva no espaço topológico. Essa linha não é reta nem regular, mas segue uma geometria emergente, onde a conectividade se distribui de forma altamente não convencional, refletindo a ausência de escala característica e reforçando a inadequação dos modelos geométricos convencionais.

Uma ferramenta útil para a análise e criação de redes são os autoencoders, que são redes neurais artificiais utilizadas para aprendizagem não supervisionada de representações compactas de dados. A arquitetura básica é composta por duas partes: um codificador (encoder), que mapeia a entrada de alta dimensionalidade para um espaço latente de menor dimensão, e um decodificador (decoder), que reconstrói a entrada original a partir dessa representação comprimida. O treino busca minimizar o erro de reconstrução entre a entrada e a saída, incentivando a extração de características relevantes. Variantes importantes incluem denoising autoencoders (robustos a ruído), sparse autoencoders (com representações esparsas) e variational autoencoders (que modelam distribuições probabilísticas). Os autoencoders são amplamente aplicados em redução de dimensionalidade, compressão de dados, deteção de anomalias e pré-treinamento de redes profundas.

Os autoencoders, mesmo nas suas versões lineares, desempenham um papel fundamental na análise de redes de Barabási–Albert ao permitirem a compressão e reconstrução de representações estruturais complexas, como matrizes de adjacência ou vetores de características nodais. Ao mapear uma rede scale-free para um espaço latente de baixa dimensão, os autoencoders identificam padrões de conectividade redundantes, regularidades topológicas e estruturas hierárquicas implícitas que seriam difíceis de capturar por métodos estatísticos convencionais. No caso das redes de Barabási, cuja topologia é altamente heterogênea devido ao mecanismo de crescimento preferencial, os autoencoders conseguem isolar os fatores latentes que explicam a centralidade dos hubs, a distribuição em lei de potência e a modularidade emergente.

Nesta tese foram utilizadas redes de Barabási-Albert m=1 e as suas características únicas para fazer a decomposição de redes scale-free teóricas e reais em conjuntos destas redes. Desta forma conseguimos obter um conjunto de redes de Barabási-Albert que quando recombinadas originam a rede original. É possível então analisar individualmente ou conjuntos destas redes de modo a retirar informação e melhor compreender a rede original. Com este algoritmo é possível fazer um conjunto de testes e análises à rede original que doutra forma seria muito complicado.

Foi também feito uso de autoencoders lineares para gerar redes scale-free teóricas e reais, partindo de sinais, à semelhança de transformadas de Fourier. Neste contexto, a rede é construída não a partir de regras explícitas de ligação (como no modelo Barabási–Albert), mas através de uma codificação vetorial (entrada) e da sua correspondente reconstrução (saída), que gera a matriz de adjacência ou outra representação estrutural da rede. A fórmula utilizada estabelece uma relação linear entre pesos do codificador e decodificador, funcionando como um mapeamento direto entre espaços latentes e a rede gerada. Esse formalismo permite o controlo explícito da densidade e da hierarquia de conexões via manipulação paramétrica e ao mesmo tempo preservando propriedades emergentes típicas das redes scale-free, como a concentração de conectividade em poucos nós. A linearidade do modelo facilita a análise matemática, e a estrutura da fórmula garante que os componentes possam ser diretamente manipulados. Assim, autoencoders lineares oferecem um enquadramento alternativo à geração estocástica, permitindo uma abordagem determinística e reprodutível na construção de redes complexas.

Combinando o algoritmo de decomposição com as redes geradas MNIST e utilizando um graph autoencoder é possível identificar redundâncias na rede original e simplificá-la. Desta análise conclui-se que, para o caso da rede teórica MNIST utilizada, cerca de 30% desta continha informação essencial e não sobre complexa que permitia ao autoencoder recriá-la facilmente. Desta forma, e tal como esperado, foi possível concluir que certas partes da rede constituem informação mais simples e que, possivelmente, utilizando apenas essa informação podemos recriar o espaço de dimensão inteira

original. Para a rede teórica o resultado foi completamente diferente, o resultado foi o oposto do esperado onde se viu que para uma rede relativamente pequena o autoencoder teve sempre dificuldade em recriá-la e que quanto mais a reduzíamos mais difícil se tornava a tarefa, o oposto do caso da rede teórica.

Desta forma com este trabalho foi possível concluir que as redes reais possuem um fator de complexidade desconhecido que não permite que a sua análise seja feita de forma igual às redes teóricas. É necessário primeiro identificar esse fator desconhecido e tê-lo em conta antes de ser feita uma análise e tentativa de simplificação da rede uma vez que esse fator é essencial para a sua recriação.

Palavras-chave: Barabási-Albert, autoencoders, espaços de dimensão inteira, redes scale-free

Acronyms

AI	Artificial Intelligence
BA	Barabási-Albert
ML	Machine Learning

Contents

List of Figures

List of Tables

1 Introduction

To tackle and study a complex inflationary system, one needs to first define what constitutes such systems. Given the broad applicability of the term, this task is rather complicated, the focus of discussions around inflationary systems is their economic manifestations. In economics, inflation is defined as “the tendency of prices to increase continuously” [1]. Ultimately, an inflationary system is characterized by permanent and continuous growth, regardless of its specific domain.

The study of inflationary systems is essential in many scientific fields, given they often manifest as networks [2], systems of interconnected entities that interact dynamically. “A network is a system of interacting agents” [2]. Such interactions are present in several contexts, from the web of connected connections between websites on the internet, the bonds between atoms in a molecule, to the complex social networks that shape human societies. For each case, the structure and dynamics of the network determine the system’s behaviour and evolution, making it essential to understand these underlying network properties. These systems can manifest as networks. These networks are called inflationary networks, due to being constantly expanding in size.

Conventional methods in statistical physics typically assume that a system is either in equilibrium or fluctuates near it. This premise, however, becomes invalid for systems driven by continuous inflationary dynamics, where no stable state or equilibrium exists. In the realm of networks, the situation is further complicated by the interdependence of nodes each one's behaviour and evolution are shaped by its links and interactions with others. This intrinsic interconnectedness introduces additional complexity, making straightforward statistical or mathematical modelling challenging, particularly when the network itself is undergoing expansion or inflation [3].

There is a well-defined, idealized model of networks, named: the Barabási-Albert (BA) model for complex scale-free networks. This model will serve as a foundational framework throughout this thesis due to its inherent properties of growth and preferential attachment, which produce a network with a power-law degree distribution.

1.1 This work and goals

These complexities highlight the central challenges addressed in this thesis: how do we study systems whose complexity requires novel approaches to solve? How do we explore a network space where the concept of dimension is not immediately clear or well-defined? Are complex inflationary networks decomposable into Barabási-Albert networks? Can we make use of Barabási-Albert networks to better understand a theoretical complex system? Is the same possible for a real-world complex system? Addressing these questions require novel approaches that move beyond conventional methods.

Decomposing complex systems into sets of Barabási-Albert networks can mathematically and computationally save us a lot of time and effort. Making use of this model we could better understand and analyse systems of integer dimensional spaces. These systems span from the market to social media to other networks and understanding them is the key to understanding complex systems of whose behaviours are still incomprehensible due to their extreme complexity. Using networks as a way to break down these systems seems to be the best way to comprehend them, since our current mathematical tools are not enough to understand them. Given that Barabási-Albert networks are simpler we can make deeper analysis of them. In theory the sum of these networks when decomposed from an original network are also a projection of the original integer dimensional space. We aim to go from a fractal dimension system and decompose this system into a set of Barabási-Albert networks with the use of

autoencoders, a tool that is essential to achieve our goal. However, to understand these systems more deeply, we must also find a way to project them onto spaces with well-defined mathematical properties. This is where the use of autoencoders becomes central to our methodology.

An autoencoder is a type of machine learning model that learns to compress and reconstruct input data, capturing the underlying structure of the input space. By applying autoencoders to our generic network, we seek to uncover the latent dimensions and intrinsic structure of the network space, revealing the patterns that govern its growth and connectivity. The autoencoder serves as a tool to map the network onto a lower-dimensional space, allowing us to study its properties in a mathematically rigorous way. We will make use of autoencoders in unconventional ways to create and understand redundancy in networks.

This thesis proposes and demonstrates a novel decomposition of complex scale-free networks into sets of Barabási–Albert (m=1) networks. By combining this decomposition with autoencoder-based network generation, we show that it is possible to isolate redundant information, identify minimal backbones, and simplify the analysis of both theoretical and real-world networks.

This work is organised into several chapters, starting on Chapter 2 with a theoretical and conceptual explanation of complex networks, metric spaces and how they are connected to networks and a study of dimensionality. Chapter 3 breaks down machine learning and explains the function of autoencoder. Chapter 4 is a breakdown of the methodology used in our work. Chapter 5 breaks down the decomposition algorithm and its results. Chapter 6 explains the process of network creation and accuracy analysis for a theoretical and a real-world system.

2 Theoretical concepts

2.1 Introduction to Complex Networks

"Complex networks describe a wide range of systems in nature and society, much quoted examples including the cell, a network of chemicals linked by chemical reactions, or the Internet, a network of routers and computers connected by physical links" [4]. These networks differ from simpler networks, such as lattices and random networks, in the sense that they possess nontrivial topologies.

When studying these types of networks, according to [5] there are three main proprieties that represent complex networks: small worlds, clustering and degree distribution. Small worlds is a key characteristic of many real-world networks, in which the average distance between two nodes is usually small when comparing to the size of the network. Clustering refers to the tendency of nodes to form communities, where nodes are more likely to be connected to each other within the community than to nodes outside of it [6].

Finally, the degree distribution. In a complex network the number of connections (also known as edges) is not the same for all nodes. The differences in edges can be expressed statistically by a distribution function, which defines the likelihood of finding a node with a certain number of edges (node degree). In a random graph (or network) this distribution is described as a Poisson distribution. The problem is that most real-life networks do not behave like that. In a particular case of these systems, the scale-free networks, this distribution function is instead proportional to a power-law [7]. Due to these characteristics, modelling these types of networks is of great importance in understanding real world systems.

2.1.1 Scale-Free Networks

We have come to an understanding that many large networks are scale-free [8]. As mentioned before, these networks are characterized by a degree distribution that follows a power law ($P(x) \alpha x^{-\gamma}$). For these kinds of networks, the power law's exponent γ is for most cases between two and three [5]. The authors of the book [5] identified that the formation of such networks follows two key mechanisms which are also present in many real-world systems. The first one is that these networks experience growth, meaning the number of nodes is not static, but instead increases over time. Another characteristic of the new nodes is that they showcase preferential attachment. This means that the new node does not create edges randomly (with equal probability to every node) but instead will preferably connect to nodes with a higher degree. [9] This phenomenon leads to the creation of hubs in the networks, which are high-degree nodes. This is also known as the rich get richer, which is also present in real world systems as pointed out by the authors. An example are papers citations, where a new paper is more likely to cite more cited articles than less known ones. To note that there may also be other mechanisms that generate scale-free networks, this one is simply the most used when it comes to replicating these networks [10].

Scale-Free Networks exhibit an inflationary behaviour and are a particular case of inflationary systems. This is of great importance for the present work. We shall now study the mechanism to create and simulate networks of this kind.

2.1.2 Barabási-Albert Algorithm

Generating these kinds of networks can be done by a simple algorithm comprised of two steps We start with a certain number of nodes, m_0, a new node is added at each timestep, creating a certain number of edges, m ($m<m_0$), with the network. Determining the probability (Π) of node i connecting with the new node j, we need to factor the node degree, as shown in the following equation [5]:

$$\Pi(k_i) = \frac{k_i}{\sum_j k_j} \tag{2.1}$$

This equation demonstrates the higher probability of new nodes of connecting to higher degree ones, where i is the new node and j is a node already present in the network.

We might be led to question whether a scale-free network can maintain its proprieties if we maintain only one of these two characteristics, as did the authors. This was answered by simulating two different models: one that starts with a small number of nodes and experiences growth, without preferential attachment (the probability of a new node to connect to an existing node is the same across all nodes); the other starts with a large number of nodes and no edges, where randomly chosen nodes create links based on the probability function described in equation (2.1), meaning by the end all of the network will be connected [9].

In both cases it was shown that none followed an expected distribution of a scale-free network. Thus, both mechanisms are needed to generate such networks. This example was demonstrated in section 5.6 of [5]

Throughout our work we used only the simplest form of scale-free networks, using the Barabási-Albert (BA) algorithm, creating networks where every new node creates only one edge (m=1). In these networks all the edges will be weightless and will have the same relevance. These networks will be addressed as Barabási-Albert networks.

2.1.3 Barabási-Albert networks: m=1 vs. m > 1

We will now explain the choice of this particular BA network. The main difference between a BA network with m = 1 and m > 1 lies in the complexity of their structure. The first case creates only one edge for each new node, which is a rather simple structure with tree-like structure with no loops. This means that the path between any two nodes is unique, and maintains the structure of the network straightforward, with no redundant connections. This network has clean and sparse connections with well-defined distances. When looking at the goal of this thesis, one can see why this network was chosen. For a decomposition of a complex system, our goal is for the decomposed systems (networks) to be as simple as possible for analysis purposes, for that reason the selection of such networks is crucial.

In contrast, when comparing to m > 1, each node connects to multiple nodes. This introduces loops and new paths to our network, which exponentially increase its complexity. Loops pose a big problem to the simplicity of a network, given that their presence no longer guarantees the shortest path between two nodes to be unique and many routes may exist to connect two different nodes. This added paths and loops complicate the network's structure, while making it more robust due to the added edges and removing the tree-like structure when compared to m=1.

When looking at the differences of the structures of these regimes, there are implications related to their behaviours. Networks with m = 1 are more prone to disconnections, as the removal of a single node can damage entire branches of the network. On the other hand, its simplicity allows for clearer

analysis as there are no different paths and a lack of redundancy. As for the case of $m > 1$, these networks are more resilient to node failures, due to loops and the redundancy that provide alternative paths for maintaining the network's integrity, although as mentioned before this creates a structural complexity that makes the network harder to analyse, compress and map to a simpler form without losing critical information.

2.2 Introduction to Metric Spaces

A metric space is a fundamental concept in mathematics and plays a vital role in various fields such as geometry, analysis, topology and computer science. Its importance comes from the structure it provides for discussing distances between elements and allows problems to be formulated and solved in a rigorous and consistent way. A metric space is defined as a set X with a metric d, which is a function with two elements $x, y \in X$ that returns a non-negative real number $d(x, y)$ [11]. This number is the distance between x and y. In order to be considered a metric space, the function must satisfy at least these properties:

1. **Non-negativity:** $d(x, y) \geq 0$ for all $x, y \in X$, and $d(x, y) = 0$ if and only if $x = y$.

2. **Symmetry:** $d(x, y) = d(y, x)$ for all $x, y \in X$.

3. **Triangle inequality:** For any $x, y, z \in X$, the distance between x and y must always satisfy $d(x, y) \leq d(x, z) + d(z, y)$.

These three fundamental properties of metric spaces ensure that the concept of "distance" is consistent and intuitive in different contexts. Non-negativity makes sure there is no negative distances, symmetry guarantees that the direction of measurement is irrelevant and the triangle inequality much like the Pythagorean theorem ensures that no "shortcut" through a third point results in a shorter path.

2.2.1 Barabási-Albert and metric spaces

The notion of a metric space can be extended to the domain of networks, where the elements of the set X are the nodes of the network and the metric d can be defined in terms of the shortest path between two nodes. Particularly in the context of a Barabási-Albert network, the distance $d(i, j)$ between two nodes i and j can be defined as the shortest path's length connecting them. This path is measured by the number of edges that separate i from j.

To assert that a BA network can be considered a metric space, we need to demonstrate that the distance function $d(i, j)$ defined as previously stated satisfies the three pillars of a metric [12]:

1. **Non-negativity:** The distance d(i, j) is clearly non-negative, as it is the number of edges in the shortest path, a positive integer. Also, d(i, j) = 0 if and only if i = j, meaning the node is connected to itself.

2. **Symmetry:** The symmetry condition is naturally satisfied in a BA network due to the network being undirected. This means that if there is a path from node i to node j, that path exists in the reverse direction. Therefore, d(i, j) = d(j, i),

3. **Triangle inequality:** The triangle inequality in the case of networks translates to the affirmation that the direct path between two nodes is at most as long as any indirect path through a third node. In a BA network, the shortest path between nodes i and k will never be longer than the sum of the shortest paths: from i to j plus from j to k. This means that d(i, k) $\leq$ d(i. j) + d(j, k) and the triangle inequality holds true.

Since the shortest-path distance in a BA network with m=1 satisfies all three conditions, we can conclude that these networks can be considered metric spaces. However, it is important to note that the nature of the metric in these networks reflects the network's topological structure rather than any background space. This difference is vital in understanding the importance of considering a network as a metric space.

Summing up, a BA network can be classified as a metric space, where the metric is defined as the shortest path between two nodes. This classification not only aligns with the theoretical framework of metric spaces but also boosts our comprehension of the structural proprieties of such networks.

2.2.2 Barabási-Albert Network Mapping

BA networks with m = 1 are the best examples of inflationary systemswith no redundancy, which give a clean representation of the networks structure. In this scenario, the network is tree-like in structure. Each new node connects to a single pre-existing node, avoiding loops and making sure the shortest path between any two nodes can be made through the direct edges between them. This simplicity makes these networks reliable for mathematical treatment as metric spaces, where the direct path between two nodes serves as the metric reflecting the topology of the system.

In the case of BA networks, the essential structure of the network can be captured by the edges alone, and in the case of m = 1, each edge plays a unique role in defining the system's topology. This allows for a clean mapping from the network to a low-dimensional space without the loss of a lot of information, if any, an essential characteristic that can be considered as the mathematical equivalent of a homeomorphism. In this scenario a homeomorphism allows for the comprehension of the network into an integer-dimensional representation that can be perfectly reconstructed without distortion or ambiguity. By having no loops in m = 1 it is possible for an autoencoder to perform this type of mapping, capturing the network's key features and reflecting them in the code layer with insignificant loss of structure.

As for the case of m > 1, the simplicity that allowed such a clean representation is no longer present. This addiction of multiple connections per new node necessarily introduces loops into the network. These loops break the tree-like structure seen in the m=1 case, where the graph is essentially a scale-free tree. Once loops are present, multiple shortest paths between nodes can exist. This destroys uniqueness, which has implications for algorithms that rely on shortest path uniqueness or tree structures. The addition of cycles and redundant connections introduces topological complexity, making low-dimensional embeddings (especially into integer-dimensional Euclidean space) more difficult. This challenges neural embedding models like autoencoders, which often assume some form of smooth, structured metric space. Although the network still satisfies the three formal requirements of a metric non-negativity, symmetry, and the triangle inequality, its structure no longer resembles that of a simple or low-dimensional Euclidean metric space.

With the presence of redundancy compression of the network becomes challenging. As the autoencoder attempts to map the network's structure to a low dimension space, it is faced with a problem of multiple paths possessing similar or the same information. In this scenario it becomes a rather complicated task if not impossible for the autoencoder to separate meaningful features from loops, which adds noise to the code layer. This means that the autoencoder can no longer perform a homeomorphism as we defined it. The complexity of the network becomes overwhelming for the autoencoder to create a fully invertible mapping between the input and the latent space.

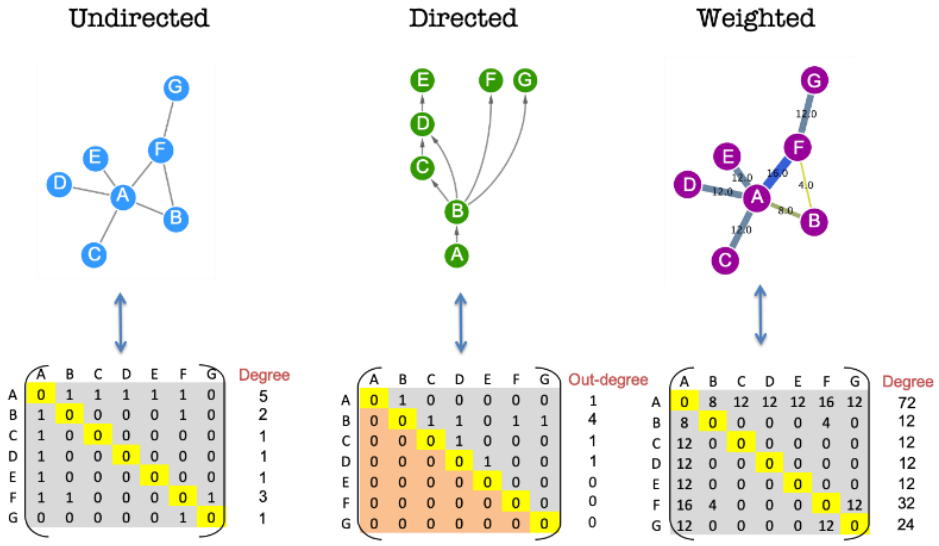
The autoencoder's ability to compress and reconstruct the network relies on the simplicity of the input structure and is here the big difference between m = 1 and m > 1. For the first case the network's structure can be perfectly captured by a homoeomorphic mapping. In this case, the code layer of the autoencoder carries just enough information to allow for an accurate reconstruction of the original network without loss of detail.

For these reasons we will be working with BA networks with m = 1. The deconstruction of scale free networks will be done into several BA networks with m = 1 which will be easier and more reliable to analyse as opposed to m > 1 networks. These networks are also easier to reconstruct and handle.

2.2.3 Adjacency Matrices

Our work with networks is highly dependent on matrix manipulations. A network can be represented as a square matrix where each entry represents a node connection. This matrix is characterized by only having 0s and 1s. If there is a 0 it means that those two nodes are not connected to each other and if there is a 1 it means that there is a link between those nodes. This allows for a very clean and transparent representation of a network where we can better understand the connections as well as the mapping of the network. It also is a reliable source in identifying certain characteristics of the network, given that the first column and first line both correspond to the connections of the same node, this means that the actual network is doubled in the matrix where we will have two triangles that are the same. This also means that the diagonal of the matrix represents self-loops, if there is a value 1 in the diagonal of the matrix, we have a node connecting to itself, this can be a quick way to see if we have this kind of behaviour in our network. Although this is true, this mechanism is not so reliable for large networks which will be the ones we work with, given the size of the matrices we cannot work with them directly and we will be using them instead to make our decomposition. This is due to the fact that it is much more computationally expensive to go through a whole matrix than an array of connection that can be obtained from the matrix itself. In figure 2.1 we can see how adjacency matrices are created, for our case we will be working with undirected matrices. As mentioned before the matrix does not have

the identity matrix present. Essentially, we need to work with the first triangle of the matrix, which is the same as the second one.



2. 1: Representation of adjacency matrices of networks [13]

2.3 Dimensionality

2.3.1 Fractal and Hausdorff dimension

Dimension is a concept that is fundamental to mathematics. It allows us to understand and categorize the complexity of various systems and structures. In classical geometry dimensions are rather straightforward: a line is one-dimensional; a plane is two-dimensional, and volume is three-dimensional. The complexity of this concept arises when we look at more complex structures where classical geometry is no longer valid [14].

For this case Hausdorff dimensions is one of the solutions to determining the geometry of such systems where Euclidian geometry cannot be directly applied. One such example are fractals; complex geometric shapes that possess self-similarity at different scales [15].

Let us consider a metric space. In this case, a Euclidian space suits us well. Consider the length of the system to be L. For one dimension this is a line of length L. In two dimensions we get a square, with side length L and area L^2. In the case of three dimensions the system becomes a cube with the same side length L and volume L^3.

Now we will scale the system. In this case this means scaling the system's basic length, L. When we scale the system by a factor of two, we get a line with length 2L in one-dimensional space. The length is increased be a factor of 2^1, the area $2^2 = 4$, and the volume by $2^3 = 8$. We can scale the system as we please.

This relation between scaling and measurement units is reflected in the following equation (equation 2.1):

$$N = r^d \Rightarrow \log(N) = d * \log(r) \Rightarrow d = \frac{\log(N)}{\log(r)} \quad (2.1)$$

In this equation, N is the measurement, r the system's basic length, and d the dimension of the system.

Let's use this notion of dimension in networks, particularly Barabási-Albert networks with m=1. In this case, each connection is bidirectional, there is no preferred direction for each connection. For each edge between two nodes, we have two objects. For example, for node n_1 and n_2 that are connected, we count two edge objects, (n_1, n_2) and (n_2, n_1).

For this case the number of nodes will be considered the system's basic length, and the number of edges the system's measurement. It may look that for every node there will be two edges, but that is not the case. The first node has 0 edges. Edges are only formed when adding the second node, where two edges (the average number of connections) between the first and second node will be added. This is proven by the following expression of the probability density function [16]:

$$p(x) = \frac{a x_0^{\alpha}}{x^{\alpha+1}} \ , \ 1 < \alpha < 2 \quad (2.2)$$

This is called the Pareto distribution, used for independent objects, which we will call particles, α = 1. But in economic systems, exchanges of equal value imply that both agents grow. So, for an idealized economy, α = 2 would correspond to symmetric exchanges. Hence why $1 < \alpha \leq 2$. Now from the p(x) we can obtain < x >, the average number of connections, from the following expression:

$$< x > = \int_1^{\infty} x\, p(x)\, dx = 2 \int_1^{\infty} x\, .\, x^{-3}\, dx = 2 \int_1^{\infty} x^{-2}\, dx = 2 \quad (2.3)$$

Where 2 is the average number of connections mentioned above. From that point on, adding a new node will add two edges to the network. This gives us a relation between number of nodes, M, and number of edges of 2(M – 1). When we input this relation into our equation, we get:

$$d = \frac{\log(N)}{\log(r)} = \frac{\log(2(M-1))}{\log(M)} = \frac{\log(2)}{\log(M)} + \frac{\log(M-1)}{\log(M)} \quad (2.4)$$

As M approaches infinity, meaning the network grows in nodes:

$$\lim_{M \to \infty} d = \lim_{M \to \infty} \frac{\log(2)}{\log(M)} + \frac{\log(M-1)}{\log(M)} = 0 + 1 = 1 \quad (2.5)$$

So, as the network grows (nodes and edges), the networks dimensionality tends to a one-dimensional system. Essentially, the space of all possible configurations for a network with N nodes can be thought of as lying in a high-dimensional space $\mathbb{R}^N$, where each node represents an independent axis or degree of freedom. In the absence of connections (edges), this space is fully unconstrained, and nodes are independent entities. However, each edge added to the system introduces a relational constraint, a dependency between two nodes. These constraints progressively reduce the system's freedom to vary independently across dimensions. Specifically, in a BA network with m=1, each new node attaches to exactly one existing node, thereby reducing the number of independent paths through the graph. This yields a spanning tree: a structure with exactly N−1 edges for N nodes, which is the minimally connected configuration required for full network connectivity. From a geometric or topological perspective, this

growing tree can be interpreted as embedding itself along a curved trajectory within the higher-dimensional configuration space. As the network expands, it does not fill the entire $\mathbb{R}^N$, but rather collapses onto a one-dimensional manifold, a structure that locally resembles a curve. The space is curved, rather than a “straight line” because the BA model imposes preferential attachment, which introduces heterogeneity, some nodes (early or well-connected) acquire many links, while most remain sparsely connected. This uneven distribution results in a hierarchical and self-similar structure, typical of fractal-like or curved spaces. The asymptotic behaviour (as $N \to \infty$) of such a network approaches a deterministic backbone that defines a clear geometric path, like a "curved line" in the high-dimensional space.

This notion is crucial for our work, by decomposing networks into a set of BA networks, we can obtain a new network whose dimension is close to one. This allows us to analyse it. Through the work we will be taking advantage of this geometry of networks. Essentially this means that a BA network will project into a one-dimensional space. We will make use of these networks to try and understand Euclidian geometry, such as stock price. With this in mind, we will try to understand integer dimensional spaces by transforming them into networks, which will be decomposed into sets of BA networks for analysis and later try and recreate the original integer dimensional space for result validation.

3 Machine Learning

Artificial Intelligence (AI) as the name implies is the simulation of human intelligence in machines using programs and / or algorithms. AI is divided into several branches, one of them is Machine Learning (ML), this subset is composed of algorithms that make use of mathematical techniques to learn and extrapolate patterns from data (similarly to humans). Machine learning can also be divided into three categories: supervised learning, unsupervised learning and reinforcement learning [17] [18].

3.1.1 Supervised Learning

In supervised learning the data used to train the AI algorithm is labelled, this means that for a certain input the output is known. This is particularly useful to identify patterns and relationships between input features and the output. Therefore, this type of algorithm focuses on matching these pairs of inputs and outputs. This method can be divided in two types, regression and classification, each type is chosen according to the data provided and the goal of the algorithm. Classification algorithms sort the data into categories by identifying specific characteristics and variables that compose each category. Regression algorithms try to predict numerical values based on the input data [19]. A couple of examples are email spam detection, here the model is trained on thousands of emails already marked as spam or not. Another case of supervised learning is loan approval prediction; in this case the algorithm learns from historical loan decisions.

3.1.2 Unsupervised Learning

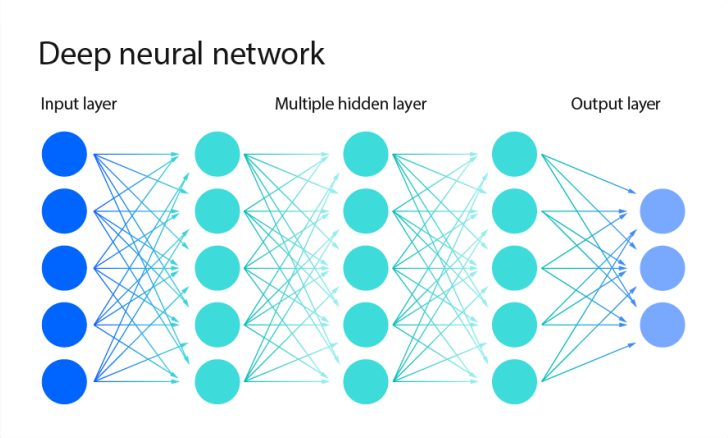
As one might guess, contrary to supervised learning, unsupervised learning does not have labelled data. Therefore, the model does not predict the output. Instead, these models work by trying to understand the relationships between data points and group similar data together. This can be done through clustering, dimensionality reduction or association [20]. An example is anomaly detection in network traffic, where anomalies are detected as deviations from normal behavior

3.2 Neural Networks

While Machine Learning is designed to try and mimic human processes of learning, then neural networks are a method that tries to replicate the human brain. This is achieved by creating layers of interconnected nodes (different than network nodes), called neurons. Neurons receive an input signal and process that signal producing an output that is passed forward to other neurons until reaching the output layer.

Each connection between two neurons has a certain weight, which determines the relevance of the connection. Typically, the neural network is initialized with random weights (these usually follow a distribution relevant to the problem). While training, similarly to the human brain, the network learns collectively by tweaking these weight values in order to minimize an error function. This is usually done following back propagation, where the difference between the predicted output and the actual one (for a certain input) is calculated. Using a gradient descent the weight contribution to the error is calculated, and their values are adjusted accordingly [21]. While there are other methods, this one is the most

relevant one to this thesis. Essentially the goal is to fit the neural network weights to the known results. We can see how neural networks are made of in figure 3.1.



3. 1: Neural Network scheme [22]

Neural networks can also vary in their architecture. Different models can have a different number of layers, where each can have a different activation function. The number of neurons in each layer is also important and we will address that later.

Hidden layers can be used to get key feature extraction and representation learning, being eager in navigating through the noise and unimportant data information in the input data. In the hidden layers, the dimension of the data can be reduced offering a “meaningful and compact representation” of the data’s space [23].

3.3 Autoencoders

An autoencoder is a type of artificial neural network designed to reconstruct its input as accurately as possible at the output. The structure of an autoencoder usually consists of an input layer, one or more hidden layers, also known as bottleneck and a symmetric output layer. These networks are structured into two parts: an encoder and decoder. The encoder transforms the input data into a reduced dimension latent representation, also known as the “code”, while the decoder takes this representation and tries to recreate the original input from it [24].

This task may seem rather trivial, but it is not. The first step requires the extraction of key features of the data to reduce the dimension, meaning the encoder and decoder are not mere identity functions. The goal of the autoencoder is to extract meaningful features of the data. The limitations (such as fewer nodes, lower dimensionality), in the structure of the autoencoder steer it, not forcibly but subtly to learn more meaningful or abstract features from the input data.

One popular constraint is to make the dimension of the hidden layer to a size smaller than the input space. This leads the network to focus on capturing the most important aspects of the data, while discarding redundancies. In essence the autoencoder compresses the input into a smaller representation and decompresses it when trying to restore the original data.

Autoencoders are powerful tools in tasks such as dimension reduction, denoising, anomaly detection and unsupervised learning where the goal is to extract meaning from the data without explicit labels or predefined categories.

3.3.1 Importance of Autoencoders for this thesis

In the context of this thesis, autoencoders provide two key advantages:

1. Dimensionality reduction without supervision enabling the extraction of latent structural patterns in networks that cannot be captured through simple graph-theoretical statistics.

2. Reconstruction-based validation by comparing reconstructed networks with originals, it becomes possible to quantify how much information is essential and how much is redundant.

These properties make autoencoders, and especially linear ones, central to the decomposition framework proposed here. While non-linear autoencoders are more powerful in theory, this work purposely employs linear autoencoders. The reason is interpretability, linear autoencoders allow the transformation from input to latent space (and back) to be expressed explicitly through weight matrices. This algebraic transparency makes it possible to extract adjacency matrices directly from the encoder-decoder weights, turning the autoencoder into a network generator rather than a black-box predictor.

Thus, linear autoencoders bridge the gap between statistical learning and graph-theoretical structure, providing a controllable and mathematically interpretable tool for this work.

The application of autoencoders in this thesis goes beyond conventional uses such as image denoising or dimensionality reduction. Instead, autoencoders are applied in a non-standard way to:

- Generate adjacency matrices: by treating the encoder–decoder weights as a mapping between nodes, autoencoders construct the connectivity structure of networks from underlying data (such as stock prices, MNIST digits).
- Enable network decomposition: the generated networks are then decomposed into BA(m=1) subgraphs, isolating simple tree-like structures that capture the backbone of connectivity.
- Identify redundancy: by comparing reconstruction accuracy across original networks and their decomposed BA subgraphs, the method reveals which parts of the network are essential and which are redundant.

This pipeline transforms autoencoders from passive data compressors into active tools for network analysis and simplification. Most applications of autoencoders in machine learning focus on feature extraction or dimensionality reduction. In contrast, the contribution of this thesis lies in repurposing autoencoders as structural operators on complex networks.

This re-framing of autoencoders positions them vas instruments of theoretical physics and network science, enabling a novel bridge between statistical learning and the geometry of complex systems [25].

4 Methodology

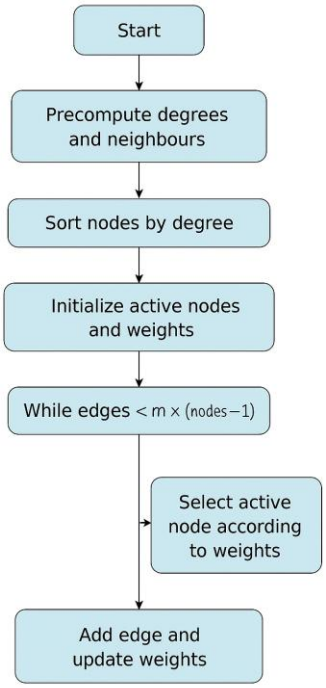
Our work consists of three major phases, the creation of an algorithm that can decompose a inflationary real-world network into a set of Barabási-Albert networks that when combined recreate the original network. We achieved this step by mapping the edges and nodes of the original network. We use this mapping to generate BA networks using only these nodes and edges. Once a network is generated, the used edges are removed, the list is updated, and the next network is created. This process goes on until all the edges of the original network are used to generated BA subgraphs.

Another step is generating network we will be working with, for that we will use a linear autoencoder with fully connected layers. In this case we will not be reducing the dimension of the hidden layer, but instead we will keep it the same as the input code. Essentially the dimension of the encoder, hidden layer and decoder is always the same. This will allow us to get the weight matrices that will generate an adjacency matrix for our network using only linear operations. It is important to normalise these weights since they will not be values of 1 and 0 exclusively, so we convert them. For that two criterions will be established which are discussed further in the results section. We will be working with the MNIST database and a signal stock prices extracted from Yahoo Finance to test the method. MNIST is a large database of handwritten numbers which are frequently used for training and testing machine learning algorithms, as is our case [26] [27]. The goal of this autoencoder is to extract latent representations, normalize them and generate an adjacency matrix of our network. This network will then go through our decomposition algorithm. We will be using the decomposed BA networks as well as the original network in a graph autoencoder with the goal of testing how accurately the autoencoder can recreate them. Essentially this last step consists of trying to understand how easily can the autoencoder extract the key features of the input network and later recreate it in the decoder. This indicator will let us understand if the network possesses a lot of redundant data and if we can simplify the networks to try and make the data less redundant. This last step is done by applying thresholds. The generated BA subgraphs in the decomposition algorithm have different sizes, we use a threshold that defines the recombination of these subgraphs. Essentially only networks whose sizes are equal or greater than the threshold is recombined to form a new network. We use this new network in our graph autoencoder to try and understand how accurately can the autoencoder recreate it. This accuracy value is expressed in probability of the autoencoder recreating the network successfully. For example, if we have a accuracy value of 0,5 this means the autoencoder can only recreate the network one every two tries, meaning there is a lot of complexity and redundant data that does not allow it to extract its key features. With this approach it will be possible to identify the most important information of the scale free networks and separate them from the redundant ones, a task that an autoencoder cannot do alone. With this method we can simplify networks and make their analysis much simpler and efficient.

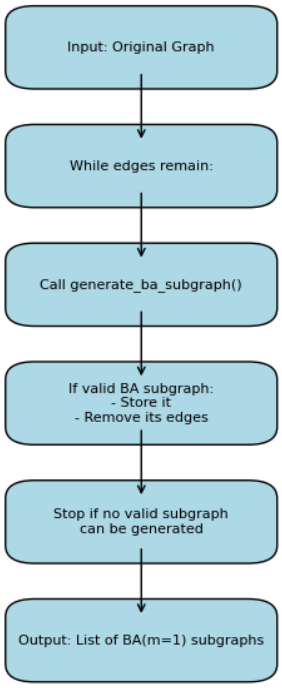
5 Decomposition Algorithm and Graph Creation

5.1 Decomposition Algorithm

A major part of this work consisted, as mentioned in the decomposition of a real-world inflationary system, into a set of Barabási-Albert networks with m=1. This problem comes with several constraints that a "pure" BA network does not have. For starters the idea of decomposing a graph, means that the created subgraphs must only contain edges and nodes that exist in the original graph. This means that we have limited connections that can be created. In a normal BA graph, the edges are added purely based on preferential attachment, meaning that the only relevant factor to determine the connection is the nodes weights, and a random selection based on that node the definition of preferential attachment. Our case is trickier given that the connections and nodes are limited. With that in mind we will break down the approach taken to tackle this problem. It is essential that the subgraphs do not reuse edges since that would make the decomposition chaotic and meaningless. For that reason, it is essential to track used edges and that each created subgraph does not have any edges in common with previous subgraphs. Unlike edges, our subgraphs must reuse nodes since the original graph's nodes are highly connected. From this constraint a natural sizing of networks arises, since less edges will be available to create subgraphs the subgraph's sizes should get increasingly smaller. To start this process, it is essential to keep track of nodes and edges in the original network, as well as used edges, for that three different lists were created to store those objects. We start by precomputing the nodes degrees and creating a neighbour map to better understand our connections. With this we can easily track all our node connections and make the process much simpler. We then sort the nodes by degree so we can index them and keep track of which nodes the subgraphs are using. This is essential since later we want to recreate the original network from the subgraphs to analyse our decomposition and for that we need to properly sum connections from different nodes, so it is imperial that nodes across different subgraphs have the same indexes. To start our creation process, we chose the highest degree node. Since it has more connections, it should be our starting node, we add that node to the subgraph and to the active nodes which is the list used to create connections. We then calculate the cumulative and total weight of the nodes to further use for preferential attachment. From there we enter a while cycle where the length of edges must be equal to the length of nodes minus one, which is the condition for BA networks with m=1. In this cycle we select a node based on the weights calculated beforehand and we then use the neighbour map created in the beginning to find valid connections to our new node. If this node has no valid neighbours, it is removed from the active nodes, and its weight is also removed from the total weights. If the node is valid from the possible connections it will attach to another node based on the weights of the valid active nodes, following preferential attachment. We then update the active nodes and the weights and remove this node from the neighbour map. This process is repeated until the subgraph is finished. After successfully generating the network, we remove the used edges from our original graph so they cannot be reused. This function is part of a cycle that will generate networks until we exhaust all edges, or it is no longer possible to generate a valid BA network from the original network. After the generation process is completed all the subgraphs are summed into one resulting subgraph which is later compared with the original. We saw for certain networks that a few edges were left out, this meant that they could not generate a BA subgraph. Now, we will show two figures that describe the two loops of the code.



5. 1: Workflow of the algorithm inner loop

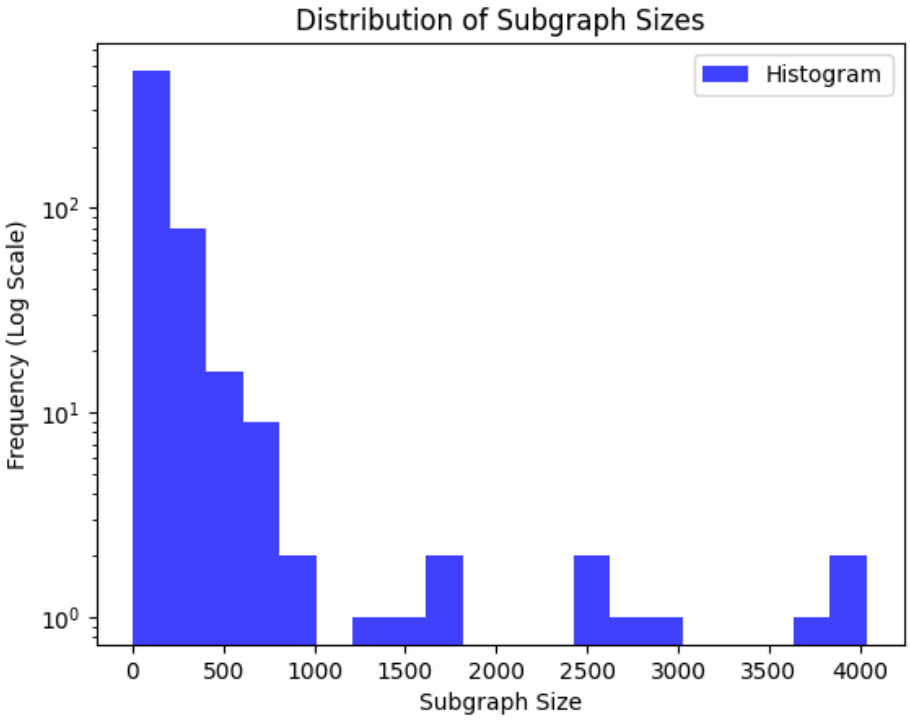


5. 2: Workflow of the algorithm outer loop

5.2 Testing the code

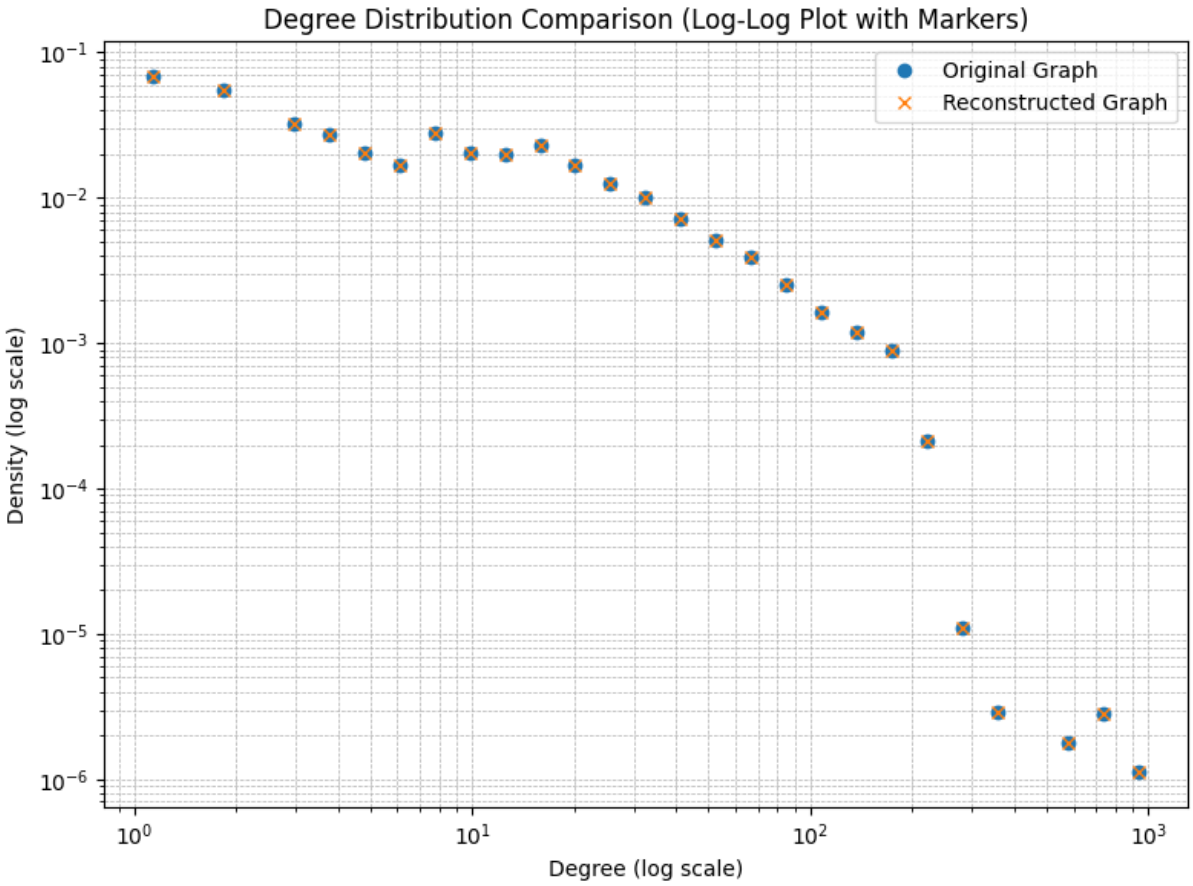
5.2.1 Facebook Network

This code was first tested out for a social circles network of Facebook [28]; this dataset is available in the Stanford University Large dataset collection. This network has 4039 nodes and 88234 edges, it is a rather small network but with a lot of connections, being the ideal case for testing and perfecting our code. For this network our code generated 580 networks, each increasingly small, the first network had 4039 nodes and 4038 edges, meaning it connected every node in the original network. All the created networks were trees which is a good indicator of a BA with m=1, given that only one connection is made by each new added node the BA network with m=1 must always be a tree. We will now show the results in graphs and explain them further.

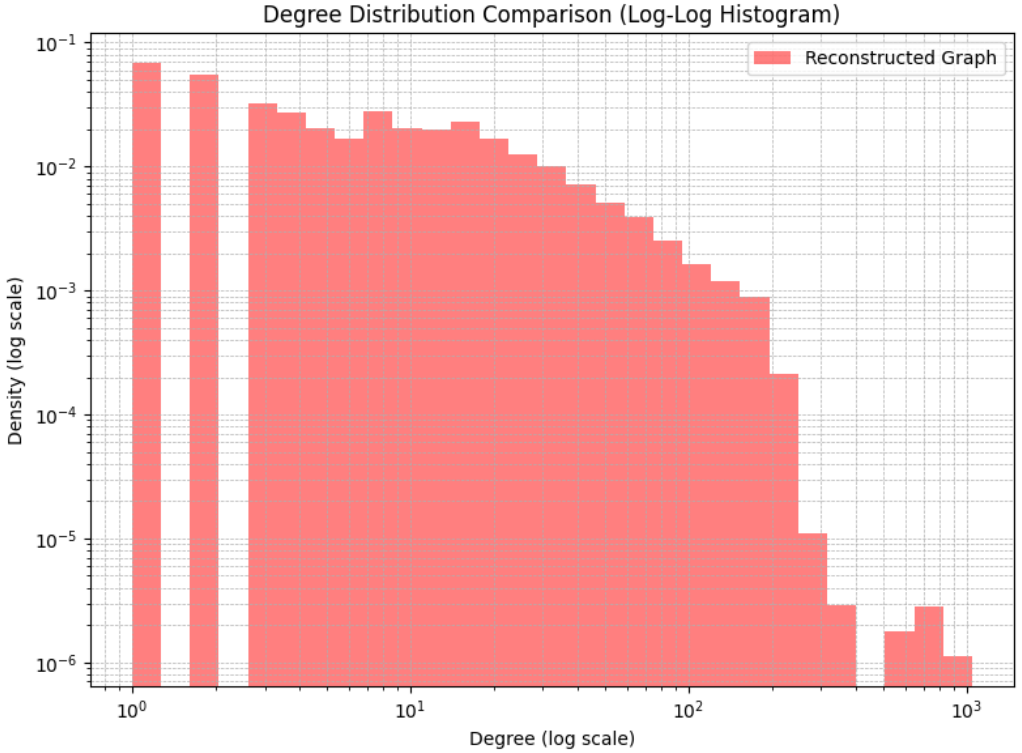


5. 3: Frequency of number of subgraphs created vs Subgraph size from the decomposition of the Facebook Network

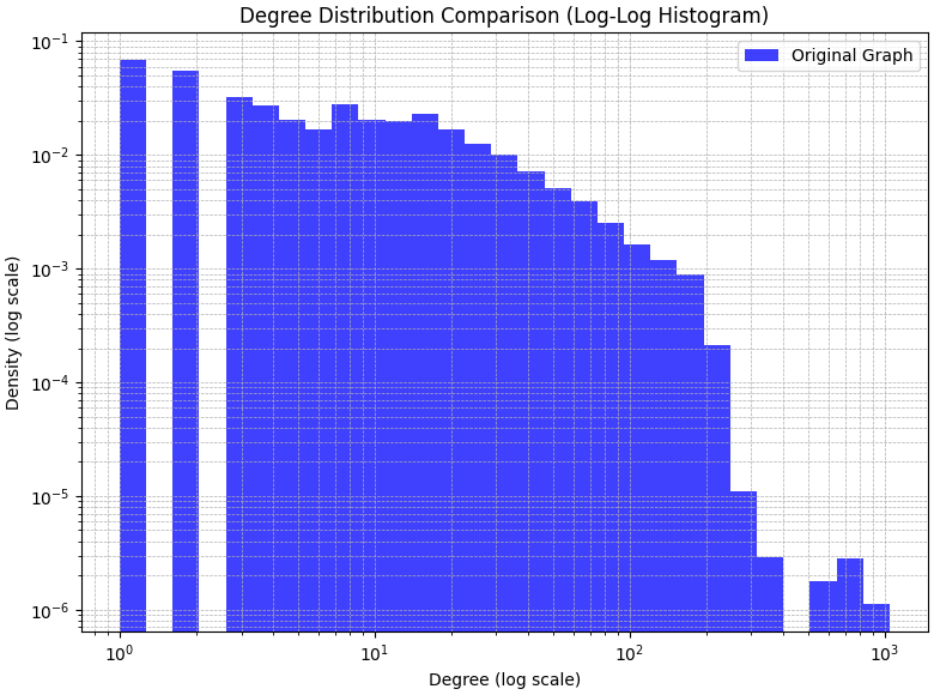
Figure 5.3 Represents the size distribution of the created subgraphs, this is particularly important to understand the sizes of our networks. If, for example no network created had size 4038 it would be an indicator that the subgraphs were not being properly created. Because the original graph is fully connected, there is, as the data shows, at least one possible subgraph whose size is equal to the original. As we can see from the image the sizes follow the expected behaviour, the largest graphs have a smaller frequency than the smaller graphs, and as we approach 0 the frequency increases drastically, this confirms our expectations and means that the less edges are available the smaller the subgraphs sizes will be. We have three main regions with sizes between 4000 and 1500, these are the largest graphs generated, and the code creates eight subgraphs with sizes over 1000 nodes where about 30% of the network's edges are contained. This again is an expected result as the first subgraphs generated should be the largest, therefore contain a large part of the edges of the original graph.



5. 4: Degree distribution comparison of summed graph and original graph overlapped for Facebook Network



5. 5: Degree distribution of the summed graph for Facebook Network



5. 6: Degree distribution of the original Facebook Network graph

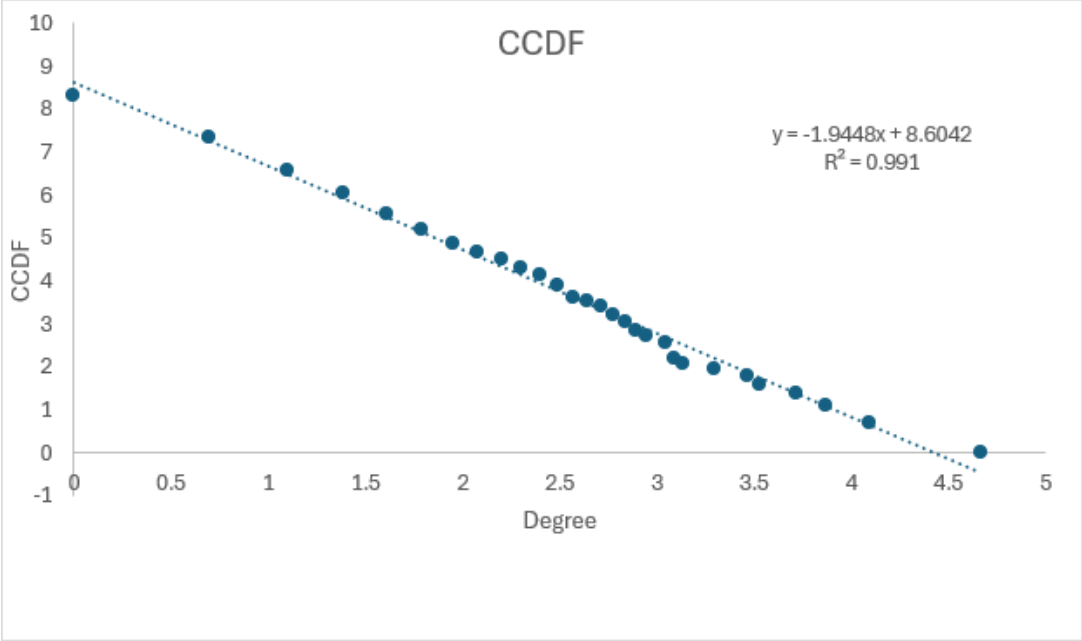
Figure 5.4 represents the degree distribution of our recombined graph overlapped by the original. A degree distribution is "the representation of the number of nodes in a network with each possible degree" [29]. The reconstructed graph corresponds to the sum of all the BA subgraphs created by the algorithm that are recombined into one network. This step essentially is made to verify whether our subgraphs can perfectly recreate the original. If they can't it means there were edges left out by the algorithm that were not used to make the subgraphs, which can mean it wasn't possible to create a BA subgraph with those edges, or the algorithm is not properly working. Now looking at figure 5.4, we can see that dots are on top of the circles, this means that the two graphs, original and reconstructed are overlapping. In other words, the summed graph is a perfect representation of the original. To make this more evident, Figure 5.5 and Figure 5.6 show the two-degree distributions side by side which are evidently identical. This also means that in this case the code was able to use all the available edges in the decomposition.

So far, the overall results have proven good and indicate that the decomposition of the original graph was successful, although it is still needed to validate whether the subgraphs created are Barabási-Albert with m=1. A good indicator as mentioned was that all of them were trees, but this alone is not enough, for that we used the first subgraph created, which by default is the largest one and make an analysis of its CCDF. CCDF or Complementary Cumulative Distribution Function is a statistical power measurement that provides a deep understanding of signal behaviour [30]. And is given by the following expression:

$$\overline{F_x}(x) = P\,(X > x) = 1 - F_X(x) \qquad (5.\ 1)$$

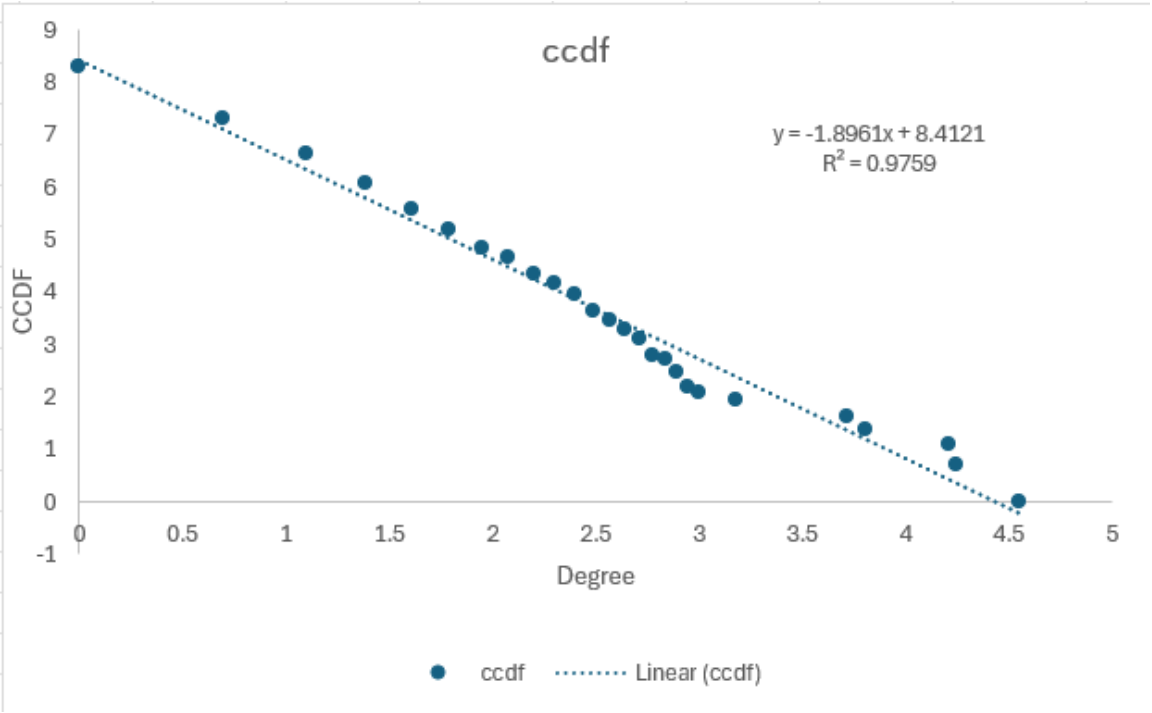
Where F_x is the cumulative distribution function. This means that the CCDF can be obtained from the cumulative distribution function or the probability distribution function [31]. Applying to networks, this

means that if the power law of the CCDF is between 1 and 2 (from the expression we subtract 1) the network is a guaranteed BA. Since no other network has a tree-like structure and follows a power law distribution, this is the strongest indicator we have that a network is or not BA. This transformation smooths the high-degree tail of the distribution and reduces noise compared to the raw histogram (Probability Distribution Function), making power-law trends easier to detect. The power law of a BA network has a slope between 2 and 3, so from the expression if $P(k) \sim k^{-\gamma}$ then $CCDF(k) \sim k^{-(\gamma-1)}$ which essentially means the slope must be between 1 and 2. Since the plot is log-log scaled the slope gives direct access to the exponent and we can verify whether or not the network is BA. To note that this is only true for BA networks with m = 1, if m is higher than 1 this indicator is not valid due to the simplicity of the m = 1 case compared to m > 1.



5. 7: CCDF for the largest subgraph generated from the decomposition of the Facebook Network

From figure 5.7, we can see that for the first graph generated by the algorithm we obtained a slope of approximately -1,95 which is indeed between -1 and -2. Given that as mentioned before the graph was tree-like we can confirm that the network generated is indeed Barabási-Albert. We will make the same test for the second generated graph to check if it is also a BA network. If so, we can safely conclude that we were able to successfully decompose a small real-world network into a set of BA networks and that our goal was achieved.



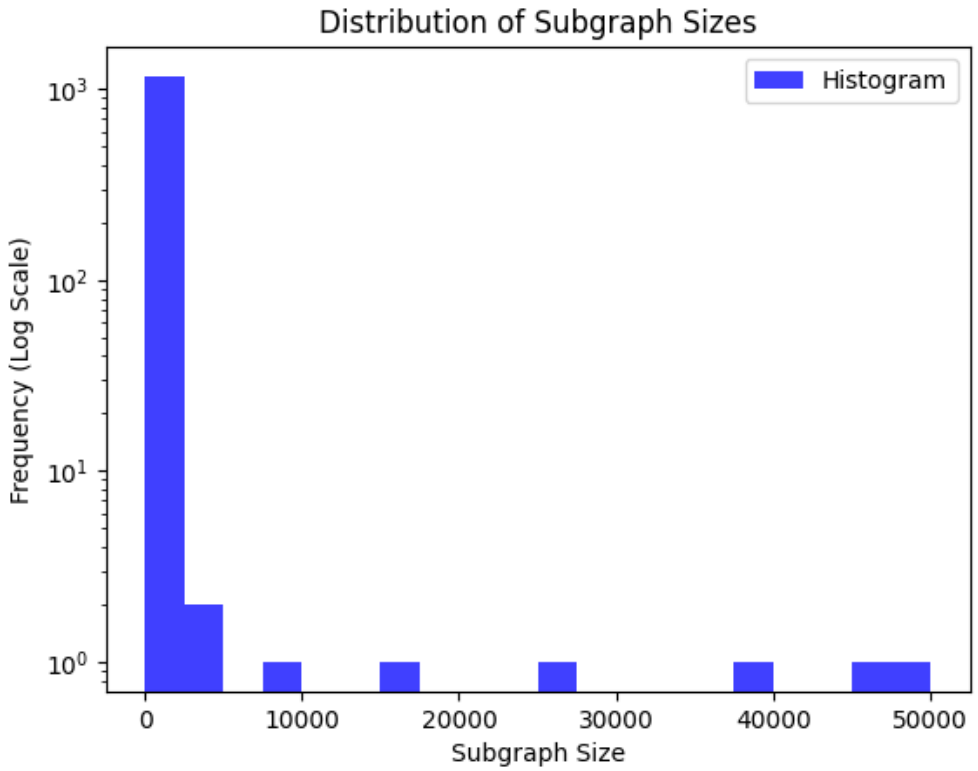
5. 8: CCDF for the second largest subgraph generated from the decomposition of the Facebook Network

Looking at figure 5.8 we can see that the slope is approximately -1,9 which falls into the acceptance interval and allows us to conclude the network is also BA. Another interesting behaviour that also is typical of a BA network is the split tail at the end, which was also present in the previous figure. Looking at the highest degrees, we can see that there is a higher distancing from the line and that the distancing shifts from the bottom of the line to the top of the line. While the bulk of the CCDF adheres closely to a straight line in log-log space indicating adherence to a power-law, the uppermost portion often deviates upward, curving above the line fitted to the central region. This phenomenon is most noticeable for the nodes with the highest degrees and becomes more pronounced in smaller or medium-sized networks, which is our case. In the BA model, new nodes attach preferentially to existing nodes with higher degree. As a result, nodes that are added early in the network's growth history accumulate disproportionately more links over time. These early nodes evolve into hubs with degrees significantly larger than what would be expected under a strict asymptotic power-law decay. Since there are only a few such nodes, they exert a strong influence on the upper tail of the degree distribution. The theoretical power-law for BA networks ($P(k) \sim k^{-3}$) assumes an infinite number of nodes. However, in finite networks, the maximum degree does not scale indefinitely but instead grows approximately as $k_{max} \sim N^{1/2}$ for m=1. Consequently, the few nodes that reach such high degrees can appear to deviate from the expected slope due to lack of statistical support in the tail, or in other words, there are simply not enough high-degree nodes to smooth out the fluctuations.

From all the results and tests made we can safely conclude that we were able to successfully decompose a real-world network into sets of BA networks. This is remarkable in the sense that the network we are working with was rather small and for real-world networks, the behaviours intrinsic to the original network are sometimes overly complex and could be a potential obstacle for recreation and decomposition.

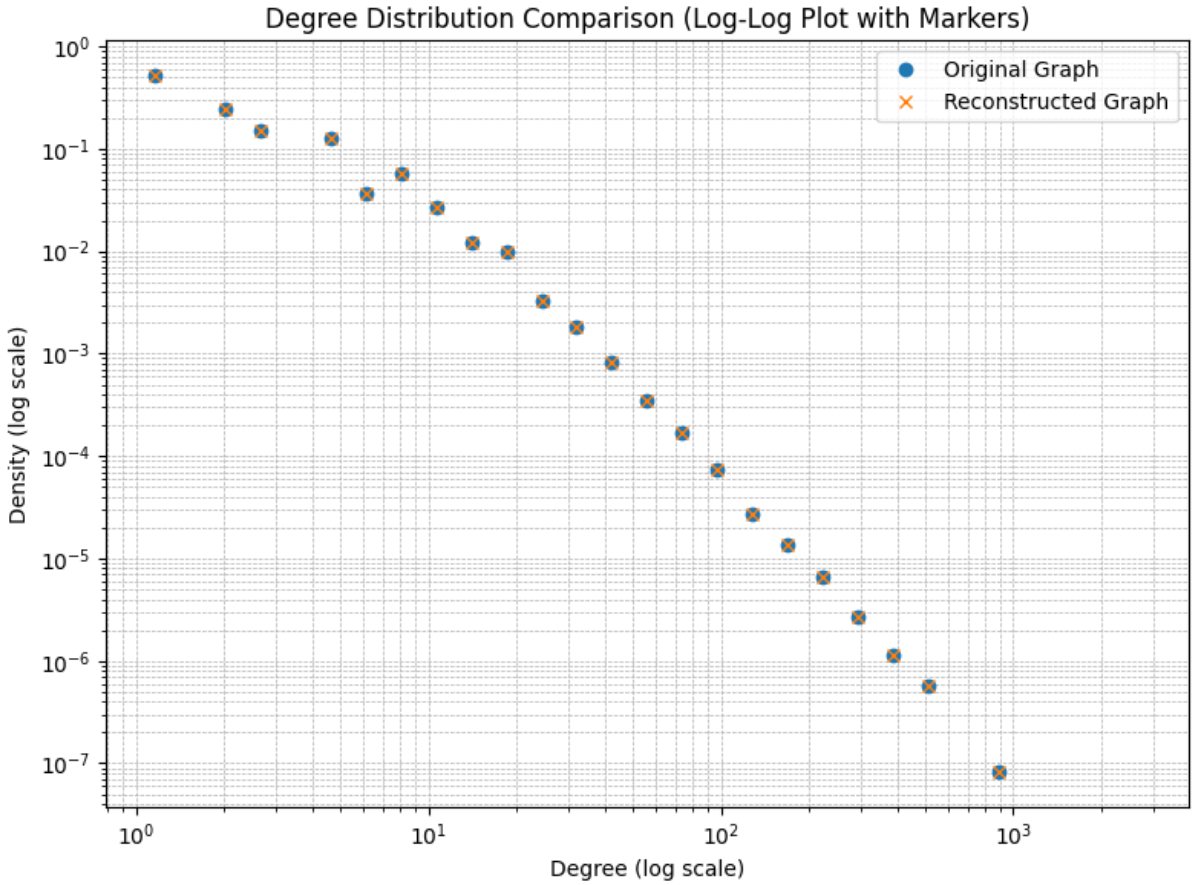
5.2.2 Artificial Network

The analysis for the Facebook network validated the algorithm for a small subgraph, but it is still needed to see how the code behaves for larger graphs, since real networks are mostly larger than the network used. Another factor to consider is the time increment, for the small network the code ran in less than a minute; it is important to see how much time a larger subgraph takes to decompose and if the results are also valid for sizeable networks. For this case we generated our network artificially, we used the library NetworkX [32] to generate a BA network with 50000 nodes and m=1, and we then added a set number of edges to recreate a real network. In our case we added 150000 edges through preferential attachment getting a total of 199999 edges and 50000 nodes. This network has approximately 10 times more nodes than the Facebook network and is will help us understand how the algorithm performs for considerably sized systems. After running the code, the time increase was significant, the code took around 50 to 60 minutes to run after multiple tries. This means essentially that our time complexity is of around $O(n^2)$, the representation of a quadratic time complexity. This means that when we increase the size of the network tenfold, the code takes around 100 times longer to run. These were the best results obtained after a lot of optimizations. This limitation means that for networks significantly larger than 50000 nodes and 200000 edges the code will take around n^2 longer time to run, being n the scalar factor in the networks size increase. To work around this problem, it is possible to use better hardware with more computing power to reduce this time complexity.



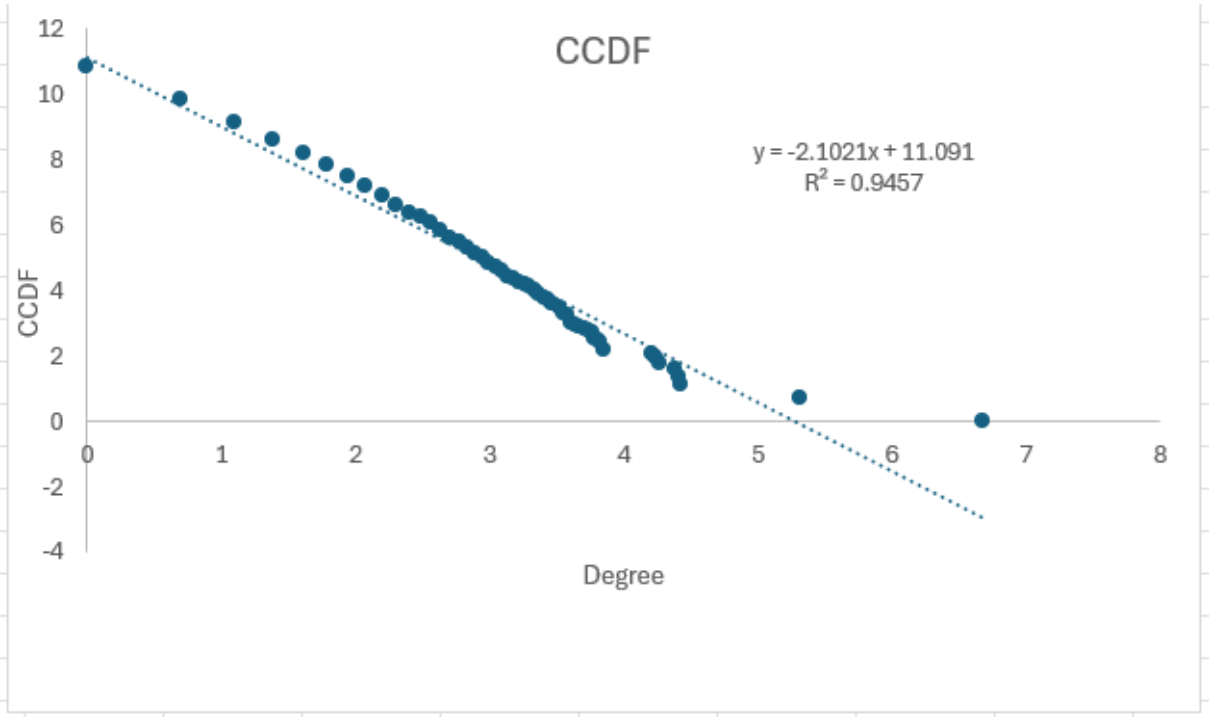
5. 9: Frequency of number of subgraphs created vs Subgraph size from the decomposition of the artificial network

Looking at figure 5.9 we can see that the sizes followed again the same patterns of graph creation as the previous network, which was expected. To note that once again the largest network generated has all the nodes available being used, given we started from a perfect BA network this is also a good indicator. In this case we can see that the size of the subgraphs decreases much faster than the previous network. This is since original network is initially a BA network. We then added additional edges following a preferential attachment mechanism, which makes the original network more connected around high degree nodes and less connected around low degree nodes, this makes the network more concentrated around higher degree nodes and as a consequence the original graph will have a set of smaller subgraphs and only few highly connected subgraphs, which is also explained in the dominance of small sized graphs that revolve around high degree nodes. This result shows that the graphs are being generated as expected, at least in terms of size.

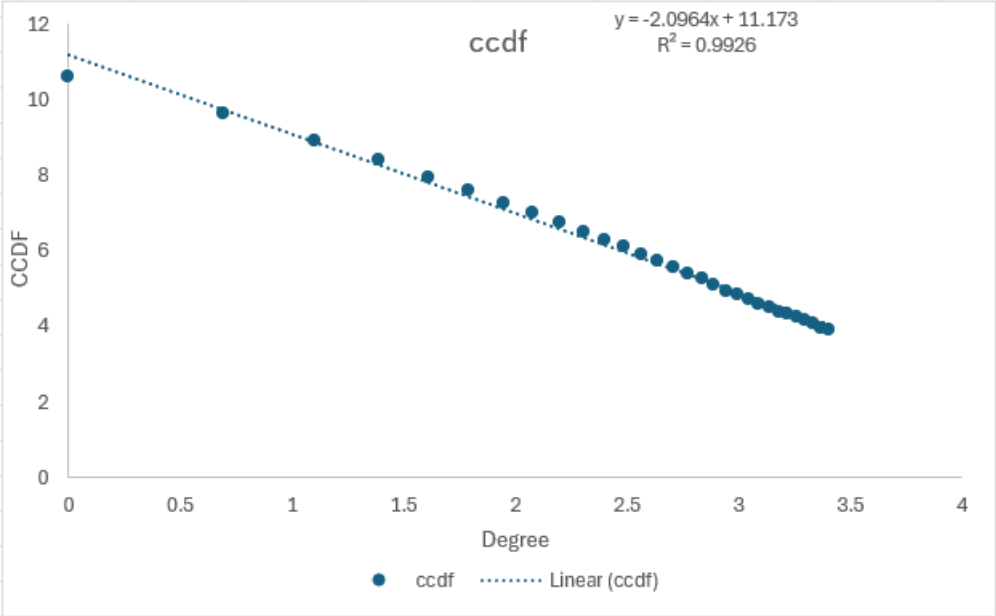


5. 10: Degree distribution of the artificial graph and reconstructed graph overlapped

As we can see from the degree distribution in figure 5.10, we were once again able to fully recreate the original network, in this case the networks had a significantly different distribution. As previously explained the network's decomposition revolves around high degree nodes. Comparing to the Facebook network we can see that there is a significantly higher density around low degree nodes. Although we are working with a different network, we were still able to make use of all edges and to fully decompose the network. What is left is to analyse the first subgraphs generated and check if they follow the BA model. The subgraphs passed the tree-like test, where all subgraphs gave a positive result. Now let's look at the first 2 subgraphs generated which by default are the largest and see if they are BA or BA like.



5. 11: First subgraph created from the decomposition of the artificial network



5. 12: Second subgraph created f from the decomposition of the artificial network

Looking at figure 5.11 of the first decomposed subgraph, which is the largest we can see that the slope is slightly above 2. As we explained previously a "true" BA needs to have a power law between 2 and 3 and for the case of the CCDF this value falls under 1 and 2. This case for both subgraphs the values slightly exceeded, although remaining close. There are several possible explanations, the most plausible one would be that the tailed distribution for this case is steeper than the first case. This will make the slope higher bringing it closer to one. This also comes from the fact that the artificial network that originated them is significantly larger than the first one and has a lot of hubs. This naturally makes the decomposed network tend to produce more hubs than a normal network would. Through these considerations and since the value is close to 2 we can consider these networks to be BA like. They are not pure perfect BA networks but considering how close they are to 2 and the original's characteristics and the process behind their creation we can assume them to be BA networks. This network proved helpful to understand better the algorithm, to see how it behaved for larger networks and how the structure of the original network impacts the decomposition. We were also quite pleased with the fact that the algorithm was able to fully decompose this unorthodox network.

6 Network Creation and Accuracy analysis

6.1 Autoencoder

In this part of our work, we created a code to generate networks through datasets of real-world and theoretical networks, The code consists of a complete training pipeline for a custom linear autoencoder, designed with TensorFlow/Keras, and it's intended for reconstructing and analysing datasets (such as price data or images like MNIST digits). It is essential we use a linear autoencoder, because autoencoders are neural network models designed to learn compact representations of data through an encoder–decoder architecture. In certain applications, such as analysing learned representations or simplifying the reconstruction process, it is desirable to extract the overall transformation matrices that represent the encoder and decoder. However, this is only feasible when the autoencoder is linear.

A linear autoencoder consists exclusively of linear transformations, where each layer performs an operation of the form:

$$y = Wx + b \qquad (6.\ 1)$$

Here, W is the weight matrix, and b is the bias vector. If no non-linear activation functions are used between layers, the overall effect of multiple linear layers can be algebraically combined into a single matrix multiplication followed by a bias addition. For example, the encoder becomes:

$$x \rightarrow W_{enc}x + b_{enc} \qquad (6.\ 2)$$

And the decoder becomes,

$$z \rightarrow W_{dec}z + b_{dec} \qquad (6.\ 3)$$

Because linear transformations can be composed, the entire encoder–decoder system can be described as a single function:

$$x \rightarrow W_{dec} * (W_{enc}x + b_{enc}) + b_{dec} \qquad (6.\ 4)$$

This is extremely useful since the full network's behaviour is captured by just two matrices and two bias vectors. These matrices can then be extracted, analysed, and even reused in other contexts (such as projections or reconstructions) outside the neural network.

We did not utilise non-linear activation functions (like ReLU, Sigmoid, or Tanh), because it makes such algebraic simplification impossible. A layer with a non-linear activation performs:

$$y = \phi(W_x + b) \qquad (6.\ 5)$$

where φ represents the non-linear function. Since the non-linearity depends on the specific input values, the transformation is no longer consistent across all inputs. Thus, it is not possible to represent the network using a fixed matrix. Each layer's behaviour becomes conditional on the input and therefore cannot be merged analytically with other layers.

In the purely linear case, autoencoders trained to minimize mean squared reconstruction error resemble Principal Component Analysis. The encoder projects the input into a lower-dimensional linear subspace, and the decoder reconstructs the input from that subspace. This connection motivates matrix extraction, as the matrices provide clear insight into the structure of the learned subspace.

Non-linear autoencoders, while more powerful in modelling complex patterns, do not yield such interpretable linear projections. Instead, they map data to non-linear manifolds, and the transformations they apply vary based on the input values.

6.1.1 MNIST

For the MNIST numbers we chose to work with numbers 0 and 1. In this case the network's dimension was 784 which corresponds to the input dimension, so our code layer's dimension was chosen to be the same.

After running the code for one of the numbers the adjacency matrix was normalized to contain only 0 and 1, there were two criterions chosen for our analyses, first we defined that every number above 0 was considered 1, essentially no matter how small the weight was of that node we consider that there is a connection between such nodes. This was done for both MNIST 0 and MNIST 1. The second criterion was to only consider a connection, or 1 in the adjacency matrix, values above 0,001. All values smaller than this threshold were considered 0, or in other words there was no connection there. This was done so we could also analyse how the threshold would affect the matrix, and if the results of the analysis would differ for both cases. Again, this criterion was applied to MNIST 0 and MNIST 1. Another step that needed to be done was the removal of the identity matrix from the adjacency matrix, given that in the network the nodes are connected to themselves, which makes sense when we consider the adjacency matrix to be given by the following expression:

$$M_t = (\mathrm{W}_e + \mathrm{b}_e)\,\mathrm{W}_d + \mathrm{b}_d \qquad (6.6)$$

Where M_t is our adjacency matrix, W_e is the encoder weight matrix, b_e is the encoder bias, W_d and b_d are the decoder weight matrix and bias respectively. In this case it makes sense that each node has a weight of 1 relatively to itself and for that reason the identity matrix is present in the adjacency matrix. As we know in the context of a network the identity matrix represents self-loops and for that reason, we removed it from the adjacency matrix before moving to the decomposition. For the MNIST 0 with the first criterion (with the threshold set to values above than 0) the obtained network had 68892 edges, and naturally 784 nodes, for the second criterion the network had 66685 edges, which means our threshold removed around 2000 links from the network. As for the MNIST 1 the network possessed 134446 edges, this value means that the number 1 is much more complex than 0 in terms of creation. The second criterion for this number reduced the number of edges to 131777, in this case around 3000 links were removed. Looking at the values from both cases we can conclude that the majority of the weights were superior to 0,001 but it is important to understand how this small difference in links will affect the decomposition and recreation of each network.

6.1.2 Stocks data

For the stocks data we loaded a file to the autoencoder with data from 16 different stocks. The procedure is quite like the MNIST one, the only major difference was the handling of the data format. Following the same steps as before we end up getting a 16x16 matrix with the stock data. The dimension of the matrix follows the number of stocks as expected, for 16 stocks we get a node corresponding to each one. In this case, given the data was already normalised there was no need for a criterion to be applied, therefore we ended up with a network of 16 nodes and 72 edges [33].

6.2 Accuracy analysis

6.2.1 Autoencoder

For this part of our work, we used a Graph Autoencoder (GAE) using PyTorch Geometric to reconstruct the adjacency matrix of a graph from node features and edge connectivity. The model is composed of two key parts: an encoder and a decoder. The encoder uses a Graph Convolutional Network (GCN) to transform input node features into a lower-dimensional latent space, capturing the structural and feature-based context of each node. The decoder then reconstructs the graph's adjacency matrix by estimating the likelihood of edges between node pairs based on their latent representations. This is achieved using a dot-product mechanism followed by a sigmoid activation.

The input graph data is formatted with node features and edge indices. Since the task focuses on structure reconstruction rather than feature prediction, the node features are initialized with uniform dummy values, for example all ones, ensuring that the model learns from connectivity patterns alone.

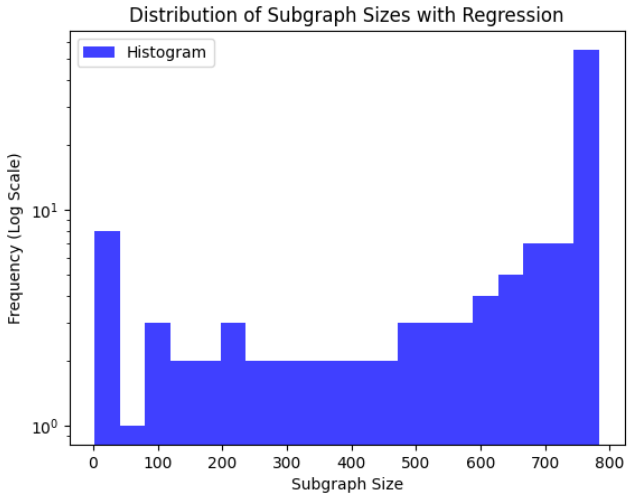
A binary cross-entropy loss is used to compare the reconstructed adjacency matrix to the true adjacency matrix. This loss function treats the edge prediction task as a binary classification problem for each possible edge. An optimizer (Adam) is employed to update the model weights via backpropagation, minimizing the reconstruction error over multiple epochs. During training, the model learns to encode node representations that preserve graph topology.

After training, the model’s ability to reconstruct the graph is evaluated using metrics such as AUC (Area Under the ROC Curve), accuracy, precision, recall, and F1-score. These metrics are computed by flattening the true and reconstructed adjacency matrices and comparing them elementwise. A threshold is applied to binarize the reconstructed matrix for classification-based metrics.

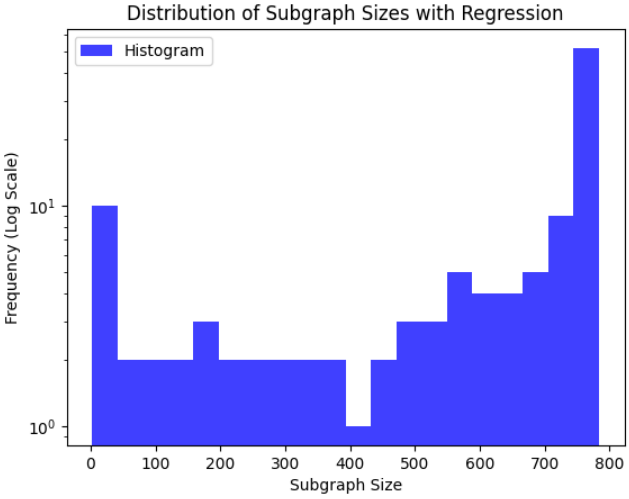
This implementation enables the extraction of meaningful node embeddings in an unsupervised setting. These embeddings can be used for tasks such as link prediction, node clustering, anomaly detection, or as input features for downstream supervised learning models. The framework demonstrates how structural patterns in graphs can be captured through learned representations without relying on labelled data.

6.2.2 MNIST data

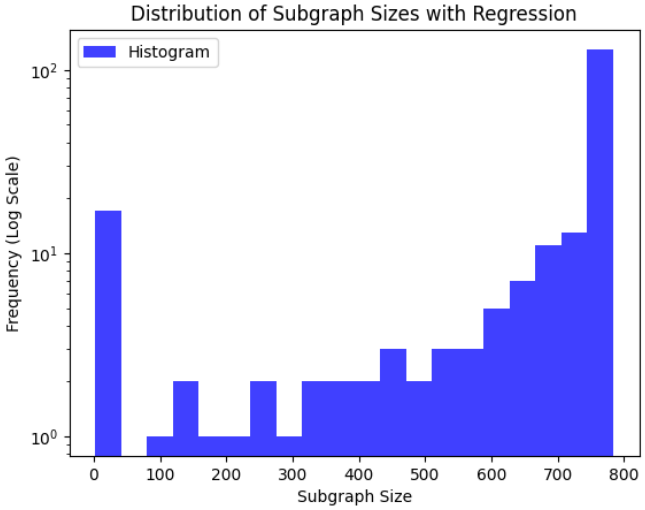
As mentioned, we decomposed our network into a set of BA networks, this means that we have a set of networks that contain the data of the original network and when recombined, recreate it. We can use these networks to make a deeper analysis of our network and with our accuracy analysis we can identify which ones are useful and which ones contain only redundant data. We can look at the distribution of our BA networks to understand the size frequency and the created networks.



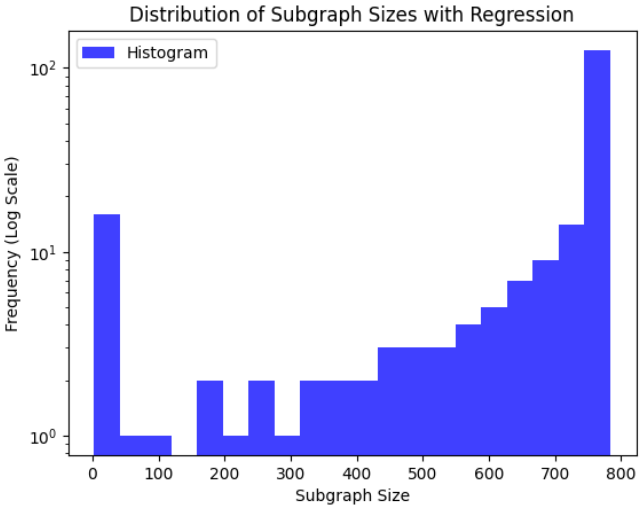
6. 1: Frequency of number of subgraphs created vs Subgraph size from the decomposition of the MNIST 0 with criterion 1



6. 2: Frequency of number of subgraphs created vs Subgraph size from the decomposition of the MNIST 0 with criterion 2

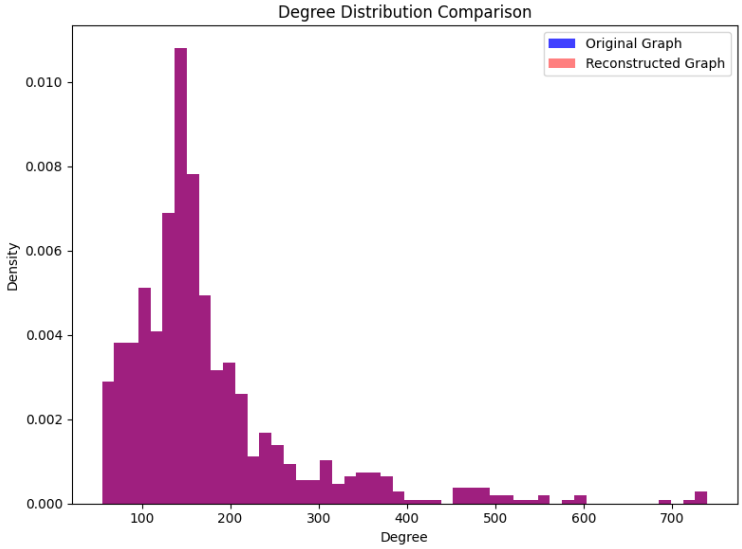


6. 3: Frequency of number of subgraphs created vs Subgraph size from the decomposition of the MNIST 1 with criterion 1

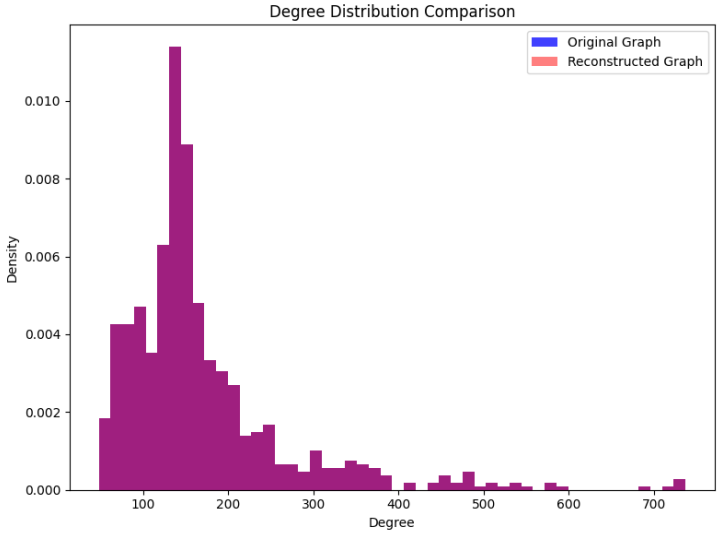


6. 4: Frequency of number of subgraphs created vs Subgraph size from the decomposition of the MNIST 1 with criterion 2

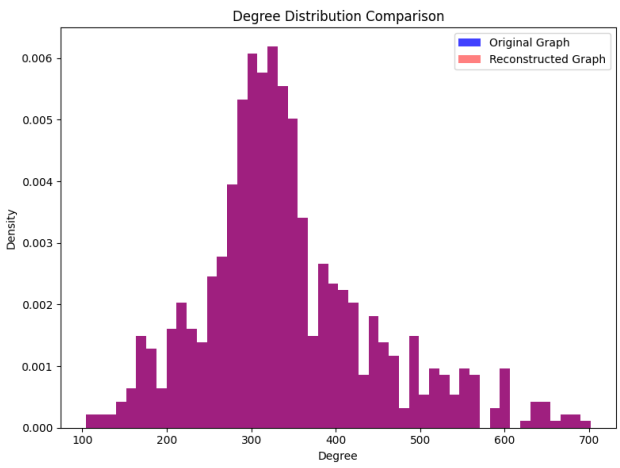
As we can see, for both criteria the frequency distribution is quite similar in both cases, as for MNIST 0 we see a distribution close to MNIST 1, but with higher frequencies in lower subgraph sizes compared to MNIST 1. This information and the fact that MNIST 1 has roughly double the number of edges compared to MNIST 0 means that MNIST 1 is more connected than MNIST 0, which makes sense given we have the same number of nodes in both networks. This also means that the algorithm should generate more subgraphs for MNIST 1 than MNIST 0, which again is the case. We have 120 and 200 graphs generated for both cases respectively. Now into a more detailed analysis we can see that in all cases we have a peak in the size of 784. This is particularly relevant because these subgraphs are our ideal case scenario. Given that they are fully connected and the largest subgraphs we assume they contain the biggest amount of information and the most relevant one as well. We need to also understand if the algorithm was able to fully decompose the original network and if the decomposition matches the original network. For that we will analyse the degree distribution, if the decomposition was successful, we can move to our accuracy analysis.



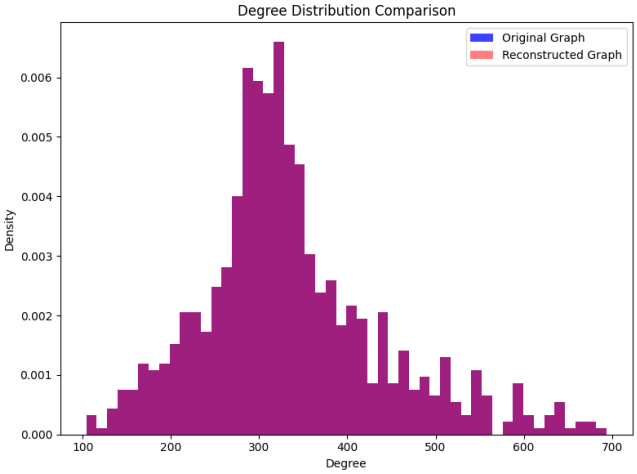
6. 5: Degree distribution of MNIST 0 with criterion 1



6. 6: Degree distribution of MNIST 0 with criterion 2



6. 7: Degree distribution of MNIST 1 with criterion 1



6. 8: Degree distribution of MNIST 1 with criterion 2

Looking at all four graphs we can note that they are fully purple, meaning the original and reconstructed graph coincide precisely. This means that we were able to completely decompose our networks into sets of BA networks. We can also note that for both cases the degree distributions vary slightly for each criterion, although the peaks remain the same in both cases, this means that the networks structure is still the same, which makes sense, and that the differences between both cases should not be as impactful. Yet, it is still interesting to see if there are any changes in the accuracy analysis. To note that given the networks sizes being much smaller than the ones tested for the algorithm the running time was quite fast for this process.

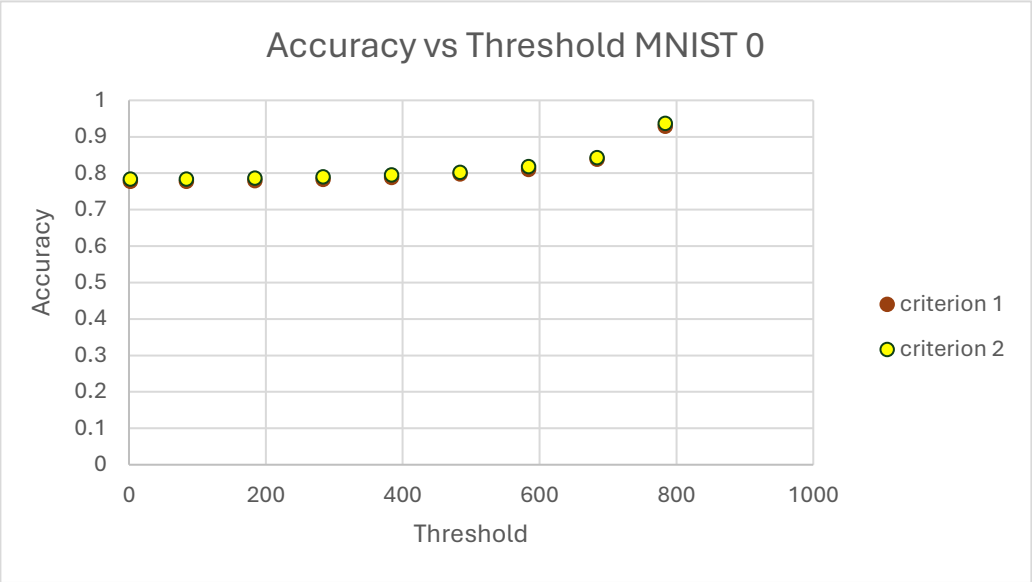
Now that we are sure the decomposition algorithm worked, we need to define our analysis. We will try to measure the accuracy of a certain sum of BA networks and compare it to the original. In other words, we will use a certain amount of decomposed BA networks and sum them together to create a new, smaller network and compare it to the original. In this case, accuracy represents how well an autoencoder can recreate a network. The approach taken will be based on decomposed network size, this means that we will be summing all networks whose size is equal or greater to a certain threshold and then make the analysis. We believe that the most important data of the original network for a high accuracy lies in the largest decomposed networks, given they are fully connected and the first ones to be decomposed from the original. We will define 9 different thresholds. The first will be 2 nodes, meaning we will consider the sum of all networks, which is the original one, our second threshold will be 84 nodes, which will represent the sum of all networks whose sizes is greater or equal to 84 nodes. From this point on we will increase the threshold a hundred-fold, from 184 to 784 nodes, which corresponds to the largest networks decomposed. The threshold selection, mathematically speaking corresponds to this sort of a “quasi-Fourier Transform”:

$$\Lambda = \sum \phi_i \Lambda_i \quad (6.7)$$

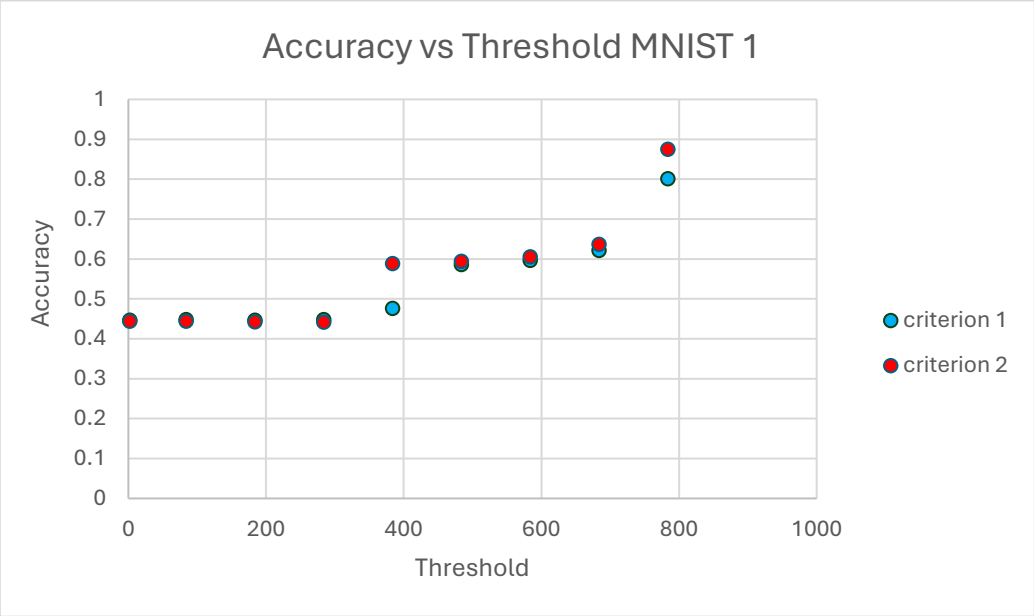
Where Λ is the obtained network, ϕ_i is the number of networks of the threshold and Λ_i is a template of a network, much like a sin wave is a template of a wave.

We intend to look at the correspondent thresholds set and understand how accurately we can recreate the original network using just a small part of that network. The threshold corresponds to that small part. For example, if the threshold is set to 700, only the BA subgraphs created whose sizes are greater or equal to 700 will be added to form a new network. This means that the thresholds and networks

are highly dependent on the decomposition algorithm and based on that the decomposition the values of the thresholds were chosen. For the case of this network, the original had 784 nodes, so to make the thresholds somewhat relevant we chose a 100 nodes interval between all of them, making thresholds from 784 to 84 nodes in 100 node subtractions. For the case of the 84-node threshold, we have a network formed of all the decomposed networks whose sizes are greater or equal to 84, which is going to englobe most edges of the original. To note that when we speak about a fraction of the original network, what we really have is a fraction of the edges of that network. As explained previously the algorithm generates BA subgraphs using the nodes and edges from the original network. Naturally we also added a 2 nodes threshold, which is the equivalent of the original network, since if there are at least 2 nodes, all the connections are englobed in the summed network. The smallest possible size for a network is exactly one connection, so if that is the threshold the sum of all networks whose sizes are greater or equal to 2 is the original. We hypothesize that the higher the threshold, the higher the accuracy should be, this comes from the fact that the less we use of sparse and low connected networks (small-sized networks) the less redundancy should the summed network have. There is also a need to understand if by setting the threshold too high and achieving high accuracy we lose key information of the network that is essential to project the original integer dimension space. As was explained previously, the integer dimensional space is a projection of the network and our goal here is to reduce the data as much as possible, making it more accurately recreated and create a network, that in theory can recreate the original space. The actual proof of this concept is out of the scope of this thesis, yet we want to understand how to increase the accuracy as much as possible. We can look at the accuracy of how an autoencoder recreates a network as a redundancy filter, if a network is overly complex, the autoencoder will struggle to recreate it and the accuracy will be low. For reference, an accuracy of 0,2 means that the autoencoder can only recreate the network once every five tries. In theory, we believe that high threshold networks should have the most important data needed to recreate the original. If that is the case, complex systems can be much more simplified and understanding them would be much more simple and easier. We will now look at the results for the MNIST, which corresponds to a theoretical system, given its generated and not a real-world network. This will be helpful to understand if our accuracy analysis follows the expected pattern and if the results are we hypothesized.
The results for both MNIST numbers are given in the following graphs:



6. 9: Accuracy analysis for MNIST 0 with both criterions



6. 10: Accuracy analysis for MNIST 1 with both criterions

Let's break down the first graph, it is evident that the accuracy values are all relatively high, starting around 0,8 accuracy. To note that the accuracy represents how "accurate" an autoencoder can replicate the matrix, meaning it is a sort of redundancy filter, as we mentioned before the autoencoder cannot perform well with redundancy. This means that the higher the accuracy the "cleaner" the data. As we discussed previously a BA network with m=1 has no redundant information, due to this fact the accuracy value obtained by only measuring one of the decomposed networks is close to 1. The reason why the MNIST 0 has such high accuracy values is likely due to its small size as a network and low connectivity when comparing to MNIST 1. We already mentioned that MNIST 0 has roughly 70000 edges, almost half of MNIST 1, which means the network is significantly less complex and is easier for

an autoencoder to replicate. To note that all networks have the same number of nodes, two times as many edges represents a high complexity increase of the network's data. In 6.9 we notice that the accuracy values did not change a lot until the 584-node threshold where the increase in accuracy was of around 0,03. The next two thresholds had a more significant increase in accuracy, particularly the last one, where the increase of that step was of around 0,09 a big increase to around 0,93 accuracy. This value means the autoencoder can almost perfectly recreate the last network.

The results obtained are according to our predictions, the 784-node threshold proved to give a very high accuracy. As we theorized this network is a sum of the best decomposed networks only, in the sense that they are all fully connected and ideal BA networks. This result means that the autoencoder can replicate with high accuracy 30% of the original network. This naturally means that the other 70% contain more redundant or overly complex data that are more difficult for the autoencoder to replicate. Another point is that the two criterions for this case are hard to differentiate. The second criterion proved to have better values than the first one, but these values are so close to the second criterion that we can consider them the same. In this case the choice of a higher threshold for edge consideration did not prove to have made a difference. This can be explained once again by the low complexity of the original network. Since the original network has already high values of accuracy, the criterions are not impactful in the results.

Looking at 6.10, we have a completely different scenario than the first one. As mentioned, the complexity of this graph is a lot higher than the first one and a quick glance at the graph shows that the node threshold made a great impact in the accuracy. There is very little variation in accuracy until the 384-node threshold where a "leap" happens, the accuracy increase grows with each step and we can notice once again that the last threshold has a large increase in accuracy compared to the rest, like MNIST 0. In this case the increase was even higher, for criterion 1 the last threshold had an increase in accuracy from 0,62 to 0,8 a 18% increase in accuracy. In this case it is evident that the networks smaller than 784 nodes add a significant level of redundancy to the network. Here, the sum of all networks with 784 nodes represents around 43% of all edges. The results are impactful in this case has we can jump from an accuracy of 0,45 to 0,8 by reducing the data to 43%. For MNIST 1, the criterions proved to have made a big difference, particularly in the "leap" thresholds, 384-node and 784-node. At first the values intertwined between criterions, but the first leap had a difference of over 10% in accuracy between the two criterions. The accuracy of criterion 1 for the 384-node was of about 0,48 and the second criterion was of 0,59. We can notice that in the 484-node threshold the criterions adjust once again and from that point on the difference between the two becomes increasingly higher until the 784-node threshold where the accuracy value for criterion 2 was of about 0,875 a 7,5% increase relative to criterion 1. We were able to achieve such a result by removing only 3000 edges out of the 134000 in the original network. This result proves that there was a significant amount of redundancy in around 3000 edges that changed drastically the network's data. From our analysis, we can speculate that the criterions prove useful in the case of large, more complex networks. This theory needs to be proven with further analysis.

We will now look at the number of networks each threshold adds to the summed network, this value represents the derivative of networks. Here, we consider all the networks obtained minus the networks present in the previous threshold. Essentially, we will only consider the networks that were added by the change of threshold. If, for example threshold 784 englobed 30 networks and in threshold 684 we have 43 networks, we will only consider the 13 new networks.

Table 6.1: Table of new added networks for each threshold

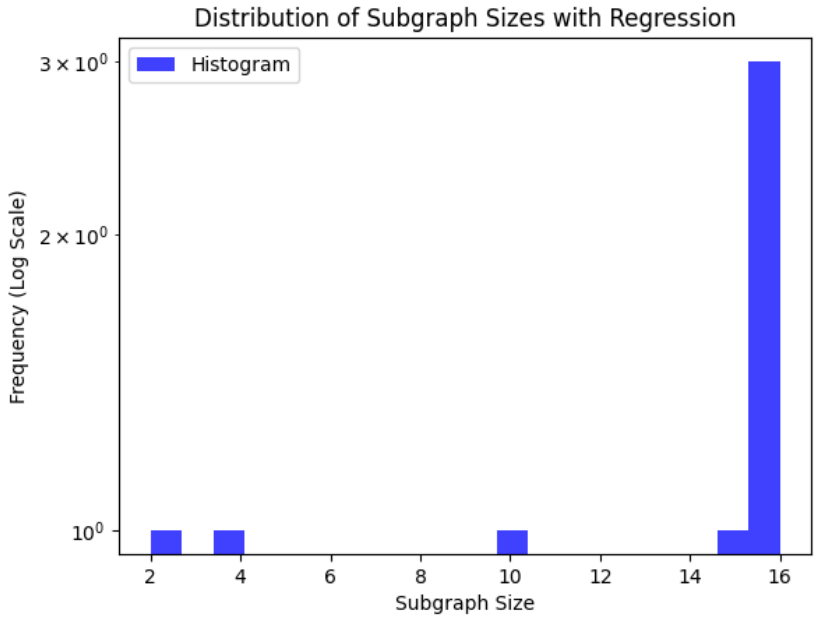
MNIST 0				**MNIST 1**			
Criterion 1		Criterion 2		Criterion 1		Criterion 2	
Threshold	New networks	Threshold	New networks	Threshold	New networks	Threshold	New networks
2	6	2	11	2	18	2	17
84	4	84	6	84	2	84	2
184	5	184	5	184	4	184	4
284	6	284	6	284	4	284	4
384	6	384	5	384	6	384	7
484	9	484	7	484	8	484	8
584	11	584	13	584	14	584	15
684	47	684	35	684	75	684	62
784	19	784	28	784	76	784	84

As we can see in Table 6.1, the largest number of networks are in the last 3 thresholds, which also as we already analysed correspond the highest accuracy thresholds. This is particularly relevant since we have most of the original network's data contain within these 3 thresholds. Looking at this fact and the accuracy values, we hypothesize that only using these three thresholds (the higher the better) we can recreate the original integer dimensional space from the summed network. Ideally, we want the original integer space to be recreated by 784-threshold, but it is highly likely that the 584-threshold can recreate the original space, since it contains most data from the original and for both cases have a high accuracy value. For the case of MNIST 1 it is even more likely that the 684-threshold will be enough, given that 80% of the data is contained in that threshold, so eliminating less than 20% of the data we should be able to recreate the original space using a network with low redundancy. To note these conclusions are only theoretical and based on the data we obtained from this analysis, it is still needed to corroborate this theory.

Summing up, for both cases we were able to significantly reduce the network's sizes and separate useful data from redundant data. This approach used in larger networks can prove effective in reducing the data size and removing unnecessary data simplifying the analysis process and the complexity of the network and possibly allow us to closely recreate the original network with this simplified data.

6.2.3 Stocks data

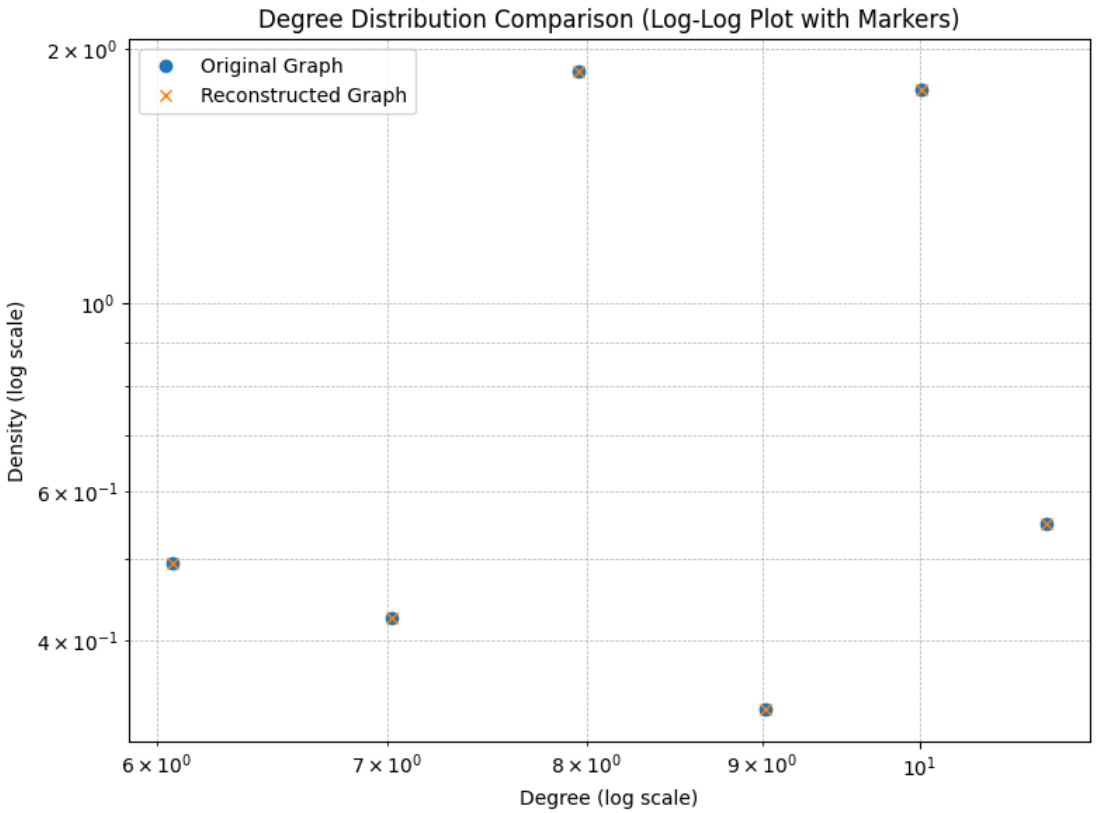
For the real network, we created a stock-market dataset from price fluctuations over a time period. We fetched data of stocks price fluctuations over that time period and created a signal with that data. We then passed the signal in our autoencoder to recreate the real-world network that represents that stock data. For the stocks data the network decomposition was rather straightforward, given that we were working with a small network. The following image shows us the subgraph size distribution.



6. 11: Frequency of number of subgraphs created vs Subgraph size from the decomposition of the Stocks network

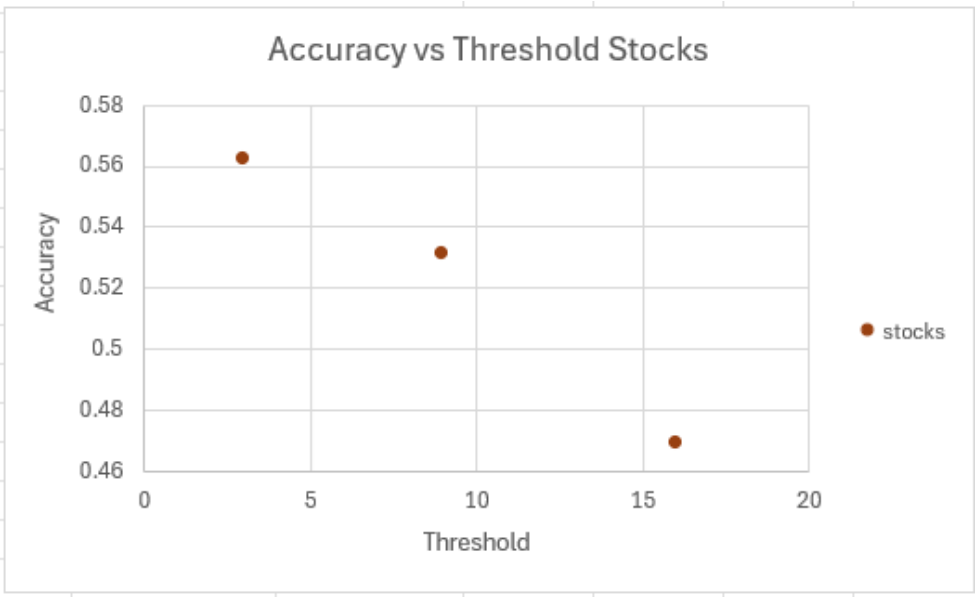
As we can see from figure 6.11, we only obtained seven different graphs, due to the original network only containing72 edges. As for the distribution we can see a peak of four networks with size 16, one peak with size 9 and two peaks with size 3. In the graph the perception is different, this is due how the code formatted the graphs axis. The thresholds represented in the graph are the ones described previously.

The degree distribution, figure 6.12, was also fully recreated by the sum of all decomposed graphs, meaning we successfully turned the original network into a set of BA networks. This is visible in the picture once again by the fact that we cannot distinguish the two graphs, meaning they perfectly overlap.



6. 12: Degree distribution of the Stocks network

Since we have a very small set of networks, we will set a threshold to the size of each network for accuracy analysis, or in other words the thresholds will be the sizes of the seven different networks. We then have three different thresholds, size 3, 9 and 16. The accuracy results are shown in the following graph:



6. 13: Accuracy values for the stocks network

As we can see from the graph we encountered a strange phenomenon, even for such a small graph the autoencoder could not recreate the graph with high accuracy. Even more curious is the fact that the accuracy values decrease when the threshold increases, something that goes against the expected behaviour of the results. This behaviour might be akin to the uncertainty principle. When we transform from a space to another with a mathematical object that is like a Fourier Transform, what happens is that if something is concentrated in one space, then it will be spread out on the dual space (and vice versa). We can see that the real-world network, although quite small has a rather complex data for the autoencoder to recreate accurately. As a comparison for this case the autoencoder has the same chance to recreate the network as we would have to get heads or tails in a coinflip. For the last case even less. This result is not unexpected, given that stocks prices are complex systems, and it would be unlikely to solve the market with this thesis, we can interpret from the results that there a few more steps needed in the case of the stock market to recreate it with accuracy. This case can be expanded to other real-world networks, where there is more to them then a theoretical network such as MNIST. A possible explanation, although speculative is that unlike the MNIST that can be considered a Euclidian space the stock market and other real inflationary systems are curved non-integer dimension spaces, not tending to a line like the case of MNIST. This complicates the mathematics of the projections and does not allow for linear transformations, complicating and adding uncertainty to the network. For that very reason the autoencoder will struggle to recreate accurately the networks and will lead to the obtained results. We could not test this algorithm for other real-world networks, which can be done to further cement these results.

Comparing artificial and real networks highlights important differences that promote a deeper discussion. Artificial networks, generated under controlled preferential-attachment rules, follow predictable scaling laws and are therefore easier to reconstruct and decompose with high fidelity. Real networks, however, often deviate from these idealised behaviours, they can include hidden correlations, hierarchical structures, noise, or domain-specific constraints that are not captured by the Barabási–Albert model. These hidden factors likely contribute to the reconstruction difficulties observed, particularly the lower accuracy of autoencoders when applied to real-world data. Overcoming these limitations would require not only more powerful computational resources, but also more refined models that explicitly account for heterogeneity, multi-layer interactions, or temporal evolution of the networks. A possible work for future doctoral research would be to extend the decomposition framework with richer generative models (such as graph variational autoencoders or multi-layer BA analogues), capable of capturing these hidden dynamics while retaining interpretability. This extension would allow the methodology to scale from theoretical validation towards robust applicability in the analysis of complex real-world systems.

7 Conclusions and Future Works

This thesis aimed to decompose real world inflationary networks into sets of Barabási-Albert networks, a type of scale-free networks known for their power-law degree distribution, which mimics many real-world systems. We also aimed at analysing those networks to try and get the original integer-dimensional space from them. We used autoencoders as a key tool to obtain networks and make the analysis.

From a theory point of view, we explored the concept of dimensionality of the Barabási-Albert networks using the Hausdorff dimension intuition. Using the network's edges as a growth measurement, we proposed the network's dimensionality approached one, or in other words a line. We successfully classified BA networks as metric spaces and explained how to go from integer dimensional spaces to networks using sort of "Fourier Transforms".

On the practical side, we successfully created an algorithm that can decompose inflationary networks into sets of BA networks. The decomposition worked from creating the largest BA network possible with the available edges and nodes and subtract those indexes, so the latter network does not reuse any edges. For that reason, the networks became increasingly smaller. We explained the constraints of the decomposed networks, given they are limited to the original network's edges and nodes the creation process is not completely randomized, meaning they are not "pure" BA networks, even though their structure is identical to one. We were also able to track nodes and edges to recombine the decomposed networks and we for all cases the summed network could recreate the original, as seen by the overlap of degree distributions. We tested the code for large networks, of up to 1 million edges and made a time complexity analysis, where we concluded the time complexity to be around $O(n^2)$, meaning the running time increases n^2 with the size increase of the network.

We used two datasets to generate networks with the goal of analysing a theoretical and real network. For the theoretical network we used MNIST which is a handwritten number database, and we picked numbers 0 and 1 to generate the networks. As for the real network we created a stock-market dataset from price fluctuations over a time period. We normalized the data and used a linear auto encoder to create the networks from the datasets. For this step it was imperative that we used a linear auto encoder as it allows us to obtain the network from a rather simple equation $M_t = (W_e + b_e) W_d + b_d$ (6. 6). With this we get a network whose projection is the database integer-dimensional space. We subtracted the identity matrix from the networks and were able to successfully generate our theoretical and real networks. For the MNIST network we used two criterions for the normalization of the matrix with the goal of understanding the impact of the normalization subjectivity in the results.

Using the decomposition algorithm we successfully decomposed our networks into sets of Barabási-Albert networks. Using a graph autoencoder we were able to make an accuracy analysis of networks. This means we were able to understand how easily an autoencoder can recreate a network and with that understand if it had overly complex or redundant data. The test was done for subsets of the original network. Using the decomposed networks, we summed them based on a node threshold and compared their accuracy values.

This test was done for both MNIST 0 and 1 following both criterions. For MNIST 0 we saw that the accuracy values were all relatively high which was directly related to the fact that the network had relatively low connectivity which allowed it to have less complex and redundant data. The results followed the expected behaviour that the higher the threshold, the higher the accuracy value was. In the last threshold the accuracy was close to one, meaning the autoencoder had almost a 100% chance of

successfully recreating the network. This makes sense given that we work with less, more connected networks the higher the threshold. As for the criterions we didn't notice any significant differences in accuracy values. For MNIST 1 we saw a significant drop in accuracy values, especially for the lower threshold and noticed significant increases in higher thresholds where we were able to achieve high accuracy. This drop in values is related to the increase in connectivity when comparing to MNIST 0, where in this case there were twice as many connections as in MNIST 0, making the network more complex and introducing more redundancy. In this case we saw a significant difference in both criterions for two thresholds. The most significant one was the last Threshold where the increase was of around 7,5%. This meant that for more connected networks, where redundancy and complexity are higher, there is a greater impact in the subjectivity of the normalization of the adjacency matrix.

For the stock dataset we had a relatively small network of a little over a dozen nodes and low connectivity. We made the same analysis for three different thresholds and obtained accuracy values of around 0,5. This result meant that the autoencoder had a 50% chance of successfully recreating the network. This result was quite surprising, given we were working with a small, low connected network. For this case, the higher the threshold, the lower the accuracy values were, a result that also came as quite a surprise. This could mean that we were losing important information in the threshold definition, which made it harder for the autoencoder to recreate the network. This result, although surprising was not unexpected. For this case we were working with a real-world network, composed of real stock market data and not a theoretical one like the MNIST, which means that there are other factors that we do not know of that influence the complexity of the network. In order to crack this problem, which would allow us to potentially solve the market we need to find and understand the extra step or steps, needed to be able to successfully decompose and increase the accuracy of real-world networks. One possible explanation for this phenomenon is that the projection of real-world networks is not an integer-dimension of even Euclidian space, but rather a curved space. Curved space analysis and mathematical operations are far more complex and cannot be solved or improved following this method. We can also say, that if a network projects onto an integer-dimension of even Euclidian space we can successfully decompose it and reduce its redundancy and size. This can be tested for other real-world networks, although, most likely they will fall into our stock market network category.

Summing up, we developed an algorithm capable of decomposing complex networks into sets of Barabási–Albert (m=1) trees. This was validated both in theoretical settings, such as artificial BA networks, and in real-world datasets including the Facebook network, MNIST data and stock correlations. Across these tests, the method consistently revealed that the essential structure of the network is concentrated in a minimal backbone, while smaller subgraphs mostly contain redundant or overlapping information. This demonstrates that the decomposition is not only mathematically consistent, but also practically useful in identifying the balance between essential information and redundancy in complex systems.

At the same time, the study also revealed important limitations. For smaller market networks, the autoencoder struggled to achieve accurate reconstructions, suggesting that the method is less effective when the amount of available structure is insufficient to support reliable compression or the system is overly-complex. The decomposition algorithm itself also presents computational challenges, with time complexity growing quadratically with the size of the network, which currently restricts its application to medium-sized datasets. Moreover, in some artificially constructed networks, the subgraphs obtained were "BA-like" rather than perfectly BA, showing that the method can deviate from the ideal case when the underlying structure departs from the pure preferential-attachment model.

For future work, several paths can be pursued. Transforming the theoretical networks with higher accuracy back to integer-dimension and Euclidian spaces and understand for the higher threshold networks if the obtained Euclidian space matches that of the original network. This would prove that the higher accuracy, less redundant networks can still project the same Euclidian space as the original

network. If such result holds true, we can simplify networks and potentially understand more about these spaces and how they project into networks while making them less computationally expensive. Another possible course of action would be to try to understand the step missing to obtain a set of decomposed networks from a real-world network and achieving high accuracy in the graph autoencoder analysis. This work would require a more refined approach and a deep understanding of curved, non-integer spaces, which by itself is a significant challenge. A natural next step is to extend this preliminary application to larger datasets and different domains. For instance, financial markets offer rich correlation structures that could benefit from redundancy detection, while social networks provide an ideal setting to test the ability of the method to identify meaningful communities versus redundant links. Future work could also explore hybrid approaches, combining decomposition with advanced machine learning models, to enhance interpretability while preserving predictive power. These directions would allow the methodology to transition from theoretical validation into practical tools for analysing large-scale complex systems. Finally, improving the existing algorithm of network decomposition for large networks, solving the time and computing power requirements constraint can help in simplifying scale-free networks into sets of BA that can be analysed individually and making the whole process simpler and more effective.

8 References

[1] R. Fahlevi, "A brief review on the theory of inflation", Journal of Critical Reviews, vol. 7, no. 8, pp. 2069–2076, 2020.

[2] K. Swanson, P. Roebber and A. Tsonis, "What do networks have to do with climate?", Bulletin of the American Meteorological Society, vol. 87, no. 5, pp. 585–596, 2006.

[3] Y. Wang, L. Yu, J. Li and X. Zhang, "Mathematical problems for complex networks", Mathematical Problems in Engineering, vol. 2011, pp. 1–17, 2011.

[4] A.-L. Barabási and R. Albert, "Statistical mechanics of complex networks", Reviews of Modern Physics, vol. 74, no. 1, pp. 47–97, 2002.

[5] A.-L. Barabási, Network Science", Cambridge, U.K.: Cambridge Univ. Press, 2016.

[6] A. Rényi and P. Erdős, "On random graphs,", Publicationes Mathematicae, vol. 6, pp. 290–297, 1959.

[7] M. E. J. Newman, "Networks: An Introduction", Oxford, U.K.: Oxford Univ. Press, 2010.

[8] A. B. a. E. Bonabeau, "Scale-free networks", Scientific American, vol. 288, no. 5, pp. 60–69, 2003.

[9] R. Albert e A.-L. Barabási, "Emergence of scaling in random networks", Science, vol. 286, no. 5439, pp. 509–512, 1999.

[10] D. J. W. a. S. H. Strogatz, "Collective dynamics of 'small-world' networks", Nature, vol. 393, no. 6684, pp. 440–442, 1998.

[11] H. D. Pfister, "Metric Spaces and Topology", Cambridge, MA: MIT Press, 2010.

[12] A. C. Cruz, "Projection of inflationary systems onto integer-dimensional spaces", unpublished manuscript.

[13] E. B. Institute, "Graph theory: Adjacency matrices", EMBL-EBI Online Training, 2021. [Online]. Available: https://www.ebi.ac.uk/training/online/courses/network-analysis-of-protein-interaction-data-an-introduction/introduction-to-graph-theory/graph-theory-adjacency-matrice.

[14] J. Sah, "Hausdorff dimension and its applications", Applied Mathematical Sciences, vol. 6, no. 108, pp. 5379–5406, 2012.

[15] B. Mandelbrot, "The Fractal Geometry of Nature", San Francisco, CA: W. H. Freeman, 1982.

[16] J. P. d. Cruz, "Introduction to Physics of Economy and Finance", Cham, Switzerland: Springer Nature, to be published..

[17] T. Hastie, R. Tibshirani e J. Friedman, "The Elements of Statistical Learning", Springer, 2009.

[18] Y. LeCun, Y. Bengio e G. Hinton, "Deep learning", Nature, vol. 521, pp. 436–444, 2015.

[19] I. Goodfellow, Y. Bengio e A. Courville, "Deep Learning", MIT Press, 2016.

[20] A. K. Jain, "Data clustering: 50 years beyond K-means", Pattern Recognition Letters, vol. 31, no. 8, pp. 651–666, 2010.

[21] C. Bishop, "Pattern Recognition and Machine Learning", Springer, 2006.

[22] IBM, "Deep Neural Network diagram", IBM, 2020. [Online]. Available: https://www.ibm.com/content/dam/connectedassets-adobe-cms/worldwide-content/cdp/cf/ul/g/3a/b8/ICLH_Diagram_Batch_01_03-DeepNeuralNetwork.png.

[23] S. Haykin, "Neural Networks and Learning Machines", Pearson, 2009.

[24] Y. Bengio, "Learning deep architectures for AI", Foundations and Trends in Machine Learning, vol. 2, no. 1, pp. 1–127, 2009.

[25] R. R. Salakhutdinov e G. E. Hinton, "Reducing the dimensionality of data with neural networks", Science, vol. 313, no. 5786, pp. 504–507, 2006.

[26] Kaggle, "MNIST dataset", Kaggle Datasets, 2019. [Online]. Available: https://www.kaggle.com/datasets/hojjatk/mnist-dataset.

[27] Y. L. e. al, "Gradient-based learning applied to document recognition", Proc. IEEE, 1998.

[28] J. McAuley e J. Leskovec, "Learning to discover social circles in ego networks", in Advances in Neural Information Processing Systems (NIPS), 2012. [Online]. Available: https://snap.stanford.edu/data/ego-Facebook.html.

[29] J. Golbeck, "Analyzing the Social Web. Burlington", MA: Morgan Kaufmann, 2013.

[30] D. Myers, "CS 547 Lecture 8: Continuous random variables", George Mason University, Lecture Notes, 2015.

[31] Y. Malevergne, V. Pisarenko e D. Sornette, "Power laws in financial time series: Theory and empirical evidence", Quantitative Finance, vol. 5, no. 5, pp. 379–401, 2005..

[32] NetworkX Developers, "NetworkX library documentation", NetworkX Project, 2021. [Online]. Available: https://networkx.org/documentation/stable/index.html.

[33] R. N. Mantegna, "Hierarchical structure in financial markets", The European Physical Journal B, vol. 11, no. 1, pp. 193–197, 1999.

A. Appendix

Decomposition Algorithm

In the appendix we will show the code created to decompose scale-free networks into BA networks and break it down. The first part of the code after the imports and generation of the network is the generate_ba_subgraph function and it is designed to construct a subgraph from an existing graph using principles inspired by the Barabási–Albert model of preferential attachment.

This function takes an input graph and incrementally builds a new subgraph, where new nodes are added based on the relative degree of existing nodes, thus simulating preferential attachment. The parameter m controls the density of the resulting subgraph.

```python
def generate_ba_subgraph(graph, m=1):
    ba_subgraph = nx.Graph()
    active_nodes = set()
    used_edges = set()

    # Precompute node degrees and neighbor map
    node_degrees = dict(graph.degree())  # Faster degree computation
    neighbor_map = {node: set(graph.neighbors(node)) for node in graph.nodes()}

    # Sort nodes by degree, descending
    nodes = sorted(node_degrees.keys(), key=lambda x: node_degrees[x], reverse=True)

    if not nodes:
        print("No nodes in the graph.")
        return None

    # Select the first node and add it to the subgraph
    initial_node = nodes[0]
    ba_subgraph.add_node(initial_node)
    active_nodes.add(initial_node)

    # Running total for active node weights
    cumulative_weight = {node: 1 for node in active_nodes}
    total_weight = sum(cumulative_weight.values())

    while len(ba_subgraph.edges()) <= m * (len(ba_subgraph.nodes()) - 1):
        if not active_nodes:
            print("No active nodes with available edges.")
            break

        # Preferential attachment: Select node using weights
        r = random.uniform(0, total_weight)
        running_total = 0
        for node, weight in cumulative_weight.items():
            running_total += weight
            if running_total >= r:
                selected_node = node
                break

        # Find valid neighbors for the selected node
        valid_neighbors = neighbor_map[selected_node] - set(ba_subgraph.nodes())
        if not valid_neighbors:
            # Remove node from active set if no valid neighbors
            total_weight -= cumulative_weight[selected_node]
            del cumulative_weight[selected_node]
            active_nodes.remove(selected_node)
            continue

        # Add a randomly chosen valid neighbor to the subgraph
        new_node = random.choice(list(valid_neighbors))
        ba_subgraph.add_edge(selected_node, new_node)
        used_edges.add((selected_node, new_node))

        # Update active nodes and weights
        ba_subgraph.add_node(new_node)
        active_nodes.add(new_node)
        neighbor_map[selected_node].remove(new_node)

        # Update cumulative weights
        cumulative_weight[new_node] = 1  # New node starts with weight 1
        total_weight += 1
        cumulative_weight[selected_node] += 1  # Add to selected node's weight
        total_weight += 1

    # Batch remove used edges from the original graph
    graph.remove_edges_from(used_edges)

    # Log the result
    if ba_subgraph.number_of_edges() > 0:
        print(f"Created subgraph with {ba_subgraph.number_of_nodes()} nodes and {ba_subgraph.number_of_edges()} edges.")
    else:
        print("Failed to create a valid subgraph.")

    return ba_subgraph if ba_subgraph.number_of_edges() > 0 else None
```

A. 1:BA subgraph generation function

Initialization:

The function is initialized by creating a new empty graph ba_subgraph to hold the resulting subgraph. Two sets, active_nodes and used_edges, are initialized to track nodes eligible for further expansion and to store edges added to the subgraph, respectively.

Preprocessing:

The function computes the degree of each node in the original graph and stores it in a dictionary called node_degrees. It also creates an adjacency map (neighbor_map) that maps each node to its set of neighboring nodes. This setup allows for efficient lookups during the graph construction process.

Seed node selection:

Nodes are sorted in descending order based on their degree, and the node with the highest degree is chosen as the initial seed node. This node is added to the ba_subgraph and marked as "active", meaning it is eligible to connect to new nodes.

Initialization of attachment weights:

Each active node is assigned a starting weight of 1, stored in the cumulative_weight dictionary. The total weight is used for weighted random selection during the preferential attachment step.

Subgraph Construction Loop:

The main loop continues growing the subgraph as long as the number of edges remains below the threshold defined by the formula m * (n - 1), where n is the number of nodes currently in the subgraph. This keeps the structure relatively sparse and tree-like for small m.

Inside the loop:

A node is randomly selected from the active_nodes set using weighted sampling based on the cumulative_weight values. Nodes with higher weights (i.e., higher degrees) have a higher probability of being selected, reflecting the preferential attachment principle. The function identifies valid neighbours of the selected node—those that are connected in the original graph but have not yet been included in the subgraph. If no valid neighbours are found, the selected node is removed from the active set, and its weight is subtracted from the total. Otherwise, one valid neighbour is randomly chosen, and an edge is added between the selected node and the new node. The new node is also marked as active, and the weights of both nodes are incremented to reflect their increased degrees. The newly added edge is stored in the used_edges set for later removal from the original graph.

Edge Removal from Original Graph:

After constructing the subgraph, all edges used in the subgraph are removed from the original graph. This ensures that subgraphs generated from the same original graph do not overlap in subsequent runs, if desired.

Return Value:

The function returns the constructed ba_subgraph if it contains at least one edge. Otherwise, it returns None and prints a failure message.

```python
def generate_ba_subgraphs(graph, m=1):
    ba_subgraphs = []
    print(f"Total available edges in graph: {graph.number_of_edges()}")

    while graph.number_of_edges() > 0:
        ba_subgraph = generate_ba_subgraph(graph, m)
        if ba_subgraph is None:
            break
        ba_subgraphs.append(ba_subgraph)
        print(f"Generated subgraph {len(ba_subgraphs)}. Remaining edges: {graph.number_of_edges()}")

    return ba_subgraphs
```

A. 2: Iterative BA graph generation

The generate_ba_subgraphs function is designed to iteratively generate multiple BA-style subgraphs from a given input graph. It serves as a wrapper around the previously described generate_ba_subgraph function and continues extracting non-overlapping subgraphs until there are no more usable edges in the original graph.

This approach is particularly useful in analysing large networks by partitioning them into smaller, preferentially grown components that mimic real-world scale-free structures. It allows researchers to extract latent modular substructures while preserving topological properties derived from preferential attachment.

Initialization:

The function initializes an empty list called ba_subgraphs, which will be used to store each generated subgraph. It also prints the total number of edges available in the original graph. This is useful for tracking the progress of the iterative subgraph extraction.

Iterative subgraph generation:

A while loop runs as long as the original graph still contains edges. The key idea is to repeatedly extract new BA-style subgraphs using the generate_ba_subgraph function until no more meaningful subgraphs can be constructed. Each call to generate_ba_subgraph may remove edges from the original graph (as implemented in that function). If a valid subgraph is returned, it is appended to the ba_subgraphs list. A message is printed to indicate the number of subgraphs generated so far and how many edges remain in the original graph.

Termination:

The loop terminates either when there are no remaining edges in the graph or when the internal subgraph generation fails (for example, when there is no valid starting node or isolated structure). Then, he function returns the list of generated subgraphs

```python
def sum_ba_subgraphs_to_recreate(ba_subgraphs):
    """Sum the BA subgraphs to reconstruct the graph."""
    reconstructed_graph = nx.Graph()  # New graph to store the reconstructed network

    # Add edges from each subgraph to the reconstructed graph
    for ba_subgraph in ba_subgraphs:
        if isinstance(ba_subgraph, nx.Graph):
            print(f"Adding {len(ba_subgraph.edges())} edges from subgraph.")  # Debug statement
            for edge in ba_subgraph.edges():
                reconstructed_graph.add_edge(*edge)  # Add each edge to the reconstructed graph
        else:
            raise TypeError("Each item in ba_subgraphs must be a NetworkX graph.")

    # After all subgraphs have been processed, print the reconstructed graph info
    print(f"Reconstructed graph has {len(reconstructed_graph.nodes())} nodes and {len(reconstructed_graph.edges())} edges.")

    return reconstructed_graph
```

A. 3: Subgraph sum to recreate original graph function

The sum_ba_subgraphs_to_recreate function is designed to reconstruct an approximate version of an original graph by combining a list of previously generated Barabási–Albert-style subgraphs, much like the name indicates. This is particularly useful for validating the decomposition approach employed by the generate_ba_subgraphs function, allowing for a reverse operation that merges the extracted subcomponents back into one graph.

The function operates by iterating through each subgraph in the list and re-adding all their edges to a new empty graph. The result is a composite structure that approximates the original graph from which the BA-style subgraphs were derived.

Initialization:

An empty networkx.Graph called reconstructed_graph is created to serve as the container for the rebuilt network.

Iterative edge recombination:

The function iterates over each subgraph in the ba_subgraphs list. For each subgraph:

- It first checks whether the current item is a valid networkx.Graph instance using Python's isinstance() function. If not, a TypeError is raised to alert the user to an invalid input.
- If valid, it prints the number of edges being added (for logging/debugging purposes) and then proceeds to iterate through the subgraph's edge list.
- Each edge is added to the reconstructed_graph using the unpacking operator *, which separates the edge tuple into two nodes.

Final reporting:

After all subgraphs have been processed, the function prints the total number of nodes and edges in the reconstructed graph. After, the reconstructed_graph is returned.

```python
def matrix_to_graph(matrix):
    return nx.from_numpy_array(matrix)
```

A. 4: Matrix to graph function

Finally, we convert a numerical adjacency matrix into a graph structure using the NetworkX library. This function makes use of the networkx.from_numpy_array() utility, which directly converts a NumPy array into a graph. This function offers a lightweight and efficient way to transition from adjacency matrix representations to NetworkX graph structures, which are more versatile and compatible with standard graph-theoretical algorithms.

B. Appendix

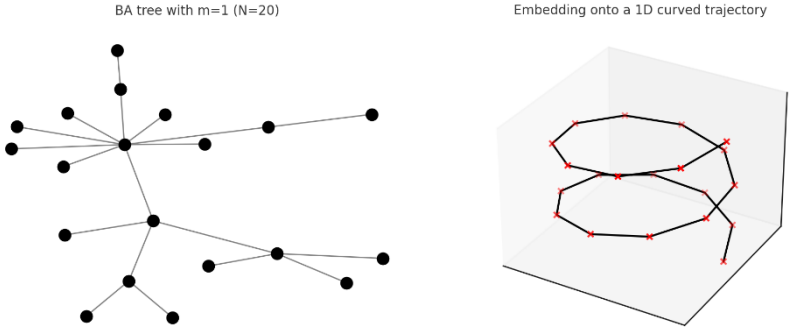
Heuristic Derivation of Dimensionality in BA Networks with m = 1

1. Model definition

In the Barabási–Albert model with m=1, each newly added node connects to exactly one existing node. This growth mechanism generates a tree like structure: for N nodes, the network contains exactly $N - 1$ edges.

2. Growth and effective dimensionality

Let N denote the characteristic system size (analogous to a length scale L). Since the number of edges grows linearly with the number of nodes, we obtain $E(N) \sim N$. Applying the standard definition of dimension in terms of scaling laws, $N \sim L^d$, we conclude that $d \approx 1$.



B. 1: Illustration of dimensionality in BA with m=1

On the left we have the representation of growth of a BA tree with m=1, each new node attaches to exactly one existing node, forming a tree without loops. On the right there is a schematic embedding of the tree into a high-dimensional space, where the structure aligns along a curved one-dimensional trajectory (the “backbone”).

3. Geometric interpretation

A tree is the minimally connected graph that ensures global connectivity without redundancy. Consequently, the topology of a BA network with m=1 collapses onto a one-dimensional manifold embedded in the high-dimensional configuration space $\mathbb{R}^N$.

Preferential attachment introduces heterogeneity in connectivity: a small subset of nodes accumulates many links, while most remain sparsely connected. This creates a hierarchical "backbone" that is not a straight line but can be interpreted as a curved trajectory in the embedding space.

4. Conclusion

In the asymptotic limit ($N \to \infty$), BA networks with m=1 behave as effectively one-dimensional objects. This heuristic derivation justifies the statement made in the main body of the thesis.