Tampere University

Veronika Blazhko

# LINK PREDICTION FOR X2 HANDOVER INTERFACE IN LTE NETWORKS

## Predicting the network topology of X2 LTE interface architecture based on real-time performance metrics data

Master of Science Thesis
Faculty of Information Technology and Communication Sciences
Thesis Examiner: Prof. Ari Visa
April 2023

# ABSTRACT

Veronika Blazhko: Link prediction for X2 handover interface in LTE networks
Master of Science Thesis
Tampere University
Data Engineering and Machine Learning
April 2023

---

Predicting the links in a network, based on the node features, has been a prominent problem of the network science of the past few years. The fast changing industry implements more and more complex systems, which creates a demand for network analysis solutions. One of the biggest industries built upon the complex networks is the telecommunications industry, which is working towards increasing the connection speed and decreasing the latency. With the LTE standard introduced, a novel technique of distributing the users has been implemented, that is called 'handing over' or simply 'handover'. Handover technology heavily relies on the topology of the nodes communicating with each other.

In this thesis we propose an effective method of estimating links between the nodes of the network, based on their real-time performance management metrics. Faulty handover interface topology in an LTE network can lead to decreased network performance, inefficient resource allocation and loss of data, ultimately causing a negative impact on user experience. The research is important for such fields as Configuration Management and Performance Management in the telecommunications domain.

The methodology includes Graph Neural Networks (GNNs), which have proven to be useful for graph analytical tasks, such as link prediction and node classification. In this work, we propose a node feature generation method, based on the seasonality analysis and Discrete Fourier Transform (DFT) of the time series signal of each node, combining analytical and machine learning methods for predicting the links in the network. Additionally, in comparison to the latest methods, classic methods will be used as a baseline algorithm.

As a result, we expect the handover interface topology to be recovered and enhanced, user equipment handover accelerated, signal latency is decreased and various business solutions enabled.

Keywords: complex networks, link prediction, topology estimation, graph neural networks, GNN, adjacency recovery, LTE, 4G, 5G, X2

# TIIVISTELMÄ

Veronika Blazhko: Linkkien ennustaminen X2-kanavan rajapinnassa LTE-verkoissa
Pro gradu -tutkielma
Tampereen yliopisto
Data Engineering and Machine Learning
Huhtikuu 2023

---

Verkon linkkien ennustaminen solmuominaisuuksien perusteella on ollut viime vuosien verkkotieteen näkyvä ongelma. Nopeasti muuttuva teollisuus ottaa käyttöön yhä monimutkaisempia järjestelmiä, mikä luo kysyntää verkkoanalyysiratkaisuille. Yksi suurimmista monimutkaisiin verkkoihin rakentuvista toimialoista on tietoliikenneteollisuus, joka pyrkii lisäämään yhteysnopeutta ja vähentämään latenssia. LTE-standardin käyttöönoton myötä on otettu käyttöön uusi tekniikka käyttäjien jakamiseksi, jota kutsutaan "handoveriksi". Handover-tekniikka on vahvasti riippuvainen toistensa kanssa kommunikoivien solmujen topologiasta.

Tässä opinnäytetyössä ehdotamme tehokasta menetelmää verkon solmujen välisten linkkien arvioimiseksi niiden reaaliaikaisten suorituskyvyn hallintamittareiden perusteella. Viallinen kanavanvaihtorajapintatopologia LTE-verkossa voi johtaa verkon suorituskyvyn heikkenemiseen, tehottomaan resurssien allokointiin ja tietojen katoamiseen, mikä lopulta vaikuttaa negatiivisesti käyttökokemukseen. Tutkimus on tärkeä tietoliikennealan konfiguraatioiden hallinnan ja suorituskyvyn hallinnan aloille.

Menetelmä sisältää Graph Neural Networks (GNN:t), jotka ovat osoittautuneet hyödyllisiksi graafien analysointitehtäviin, kuten linkkien ennustamiseen ja solmujen luokitteluun. Tässä työssä ehdotamme solmuominaisuuksien generointimenetelmää, joka perustuu kunkin solmun aikasarjasignaalin kausivaihteluanalyysiin ja Discrete Fourier Transform (DFT) -muunnokseen, jossa yhdistyvät analyyttiset ja koneoppimismenetelmät verkon linkkien ennustamiseen. Lisäksi perusalgoritmina käytetään uusimpiin menetelmiin verrattuna klassisia menetelmiä.

Tämän seurauksena odotamme kanavanvaihtorajapinnan topologian palautuvan ja parannettavan, käyttäjälaitteiden vaihdon nopeutuvan, signaalin latenssin pienenevän ja erilaisten liiketoimintaratkaisujen mahdollistuvan.

Avainsanat: verkkoteoria, yhteyksien ennustaminen, verkonrakenteen arvioiminen, graaffi neuroverkot, GNN, LTE, 4G, 5G, X2

# PREFACE

This research has been conducted under the project Network Operations Manager at Nokia Mobile Networks, Tampere.

First and foremost, I would like to express my deepest gratitude to my thesis supervisor, Ari Visa, for their unwavering support, invaluable advice, and continuous motivation throughout my thesis work and Master's degree journey. Their expertise and insightful comments have played a pivotal role in shaping my research and my progress in my Master's studies. I feel truly fortunate to have had the opportunity to work under their guidance, and I cannot thank them enough for their dedication and patience.

I would like to extend my heartfelt appreciation to amazing people around me - Aali, Vita, Lona, Ganna, Mila, Evan and Leevi, who have been an endless source of moral support during the highs and lows of this journey. Their encouragement and understanding have made the challenges more bearable and the victories more rewarding. In addition, I must acknowledge my dear cat Sima, who has been by my side throughout the entire process.

I would also like to thank my colleagues at Nokia and my manager, Pasi Antila, for their encouragement and support during my studies while working at Nokia. Their mentorship have greatly contributed to my academic success, and I am honored to be part of such a remarkable team.

Lastly, I want to express my gratitude to everyone who has directly or indirectly contributed to my research and my academic journey in Tampere University. I am eternally grateful for the opportunities and experiences that have helped me grow both personally and professionally.

Thank you all for being an essential part of this adventure.

Tampere, 26th April 2023

Veronika Blazhko

# CONTENTS

# LIST OF FIGURES

# LIST OF TABLES

# LIST OF SYMBOLS AND ABBREVIATIONS

| | |
|---|---|
| 2G | Second-generation cellular network standard |
| 3G | Third-generation cellular network standard |
| 4G | Fourth-generation cellular network standard |
| 5G | Fifth-generation cellular network standard |
| ANOVA | Analysis of Variance |
| AP | Average Precision summary statistic of the Precision-Recall curve |
| AUC | Area Under the Curve performance metric |
| BTS | Base Transceiver Station |
| CNN | Convolutional Neural Network |
| DBA | Dynamic Time Warping Barycenter Averaging |
| DFT | Discrete Fourier Transform |
| DTW | Dynamic Time Warping |
| E-UTRAN | Evolved UMTS Terrestrial Radio Access Network |
| eNodeB | Evolved Node B |
| EPC | Evolved Packet Core |
| FRESH | Feature Extraction based on Scalable Hypothesis tests |
| GAN | Graph Attention Network |
| GARNN | Graph Attention Recurrent Neural Network |
| GAT | Graph ATtention Network |
| GCN | Graph Convolutional Network |
| GNN | Graph Neural Network |
| GSM | Global System for Mobile Communications |
| HSS | Home Subscriver Server |
| KPI | Key Performance Indicator |
| LNCEL | Long-term Evolution (LTE) Cell |
| LNREL | Long-term Evolution (LTE) Relation in the X2 interface architecture |
| LSTM | Long Short-Term Memory type of recurrent neural network (RNN) |

| | |
|---|---|
| LTE | Long-Term Evolution standard (see 4G) |
| MLP | Multilayer Perceptron |
| MME | Mobility Management Entity |
| MRBTS | Multi Radio Base Transceiver Station |
| NodeB | A base station in 3G UMTS networks |
| PCA | Principal Component Analysis |
| PCRF | Policy and Charging Rules Function component of the EPC |
| PDN | Public Data Network |
| PGW | PDN Gateway |
| QoS | Quality of Service |
| RAT | Radio Access Technology |
| ReLU | Rectified Linear Unit activation function |
| RGB | Red, Green, Blue color model |
| RNC | Radio Network Controller |
| RNN | Recurrent Attention Network |
| S1 | Protocol used for handover between two eNodeBs, belonging to different MMEs or when X2 is not available |
| SGW | Serving Gateway |
| STL | Seasonal and Trend decomposition using Loess |
| UE | User Equipment |
| UMTS | Universal Mobile Telecommunications System |
| VAR | Vector Autoregression |
| WWW | World Wide Web |
| X2 | Protocol used for handover between two eNodeBs, belonging to the same MME |

# 1. INTRODUCTION

Mobile network industry has been developing rapidly since the very moment it emerged. Since the first mobile network has been launched in 1991, the user data rates have grown from 270 Kb/s in GSM to 326 Mb/s for LTE [1] and even up to 20Gb/s for 5G[2]. Mobile cellular networks have increased their coverage, dramatically decreased the latency and failure rate, which has resulted in the 5.2 billion individual mobile phone subscribers and 1.06 trillion US$ of revenue in 2019 [3]. One of the critical aspects of improving the user experience is a seamless coverage, provided by the novel LTE-based eNodeB cells and smallcell decentralized network, capable of handing over the users to each other without a need in communicating with the Radio Network Controller (RNC), in contrast to Base Stations in the previous cellular network generations as GSM, 2G and 3G [3]. The Figure 1.1 presents a modern LTE network infrastructure.

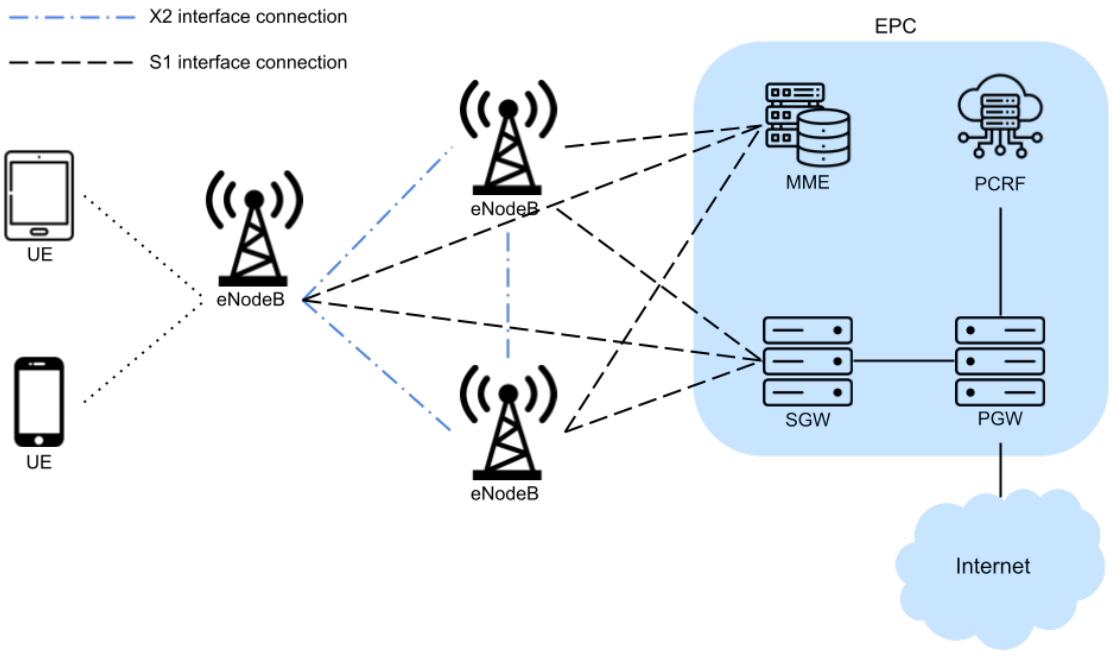



***Figure 1.1.*** *LTE network architecture*

In contrast to the older network standards, LTE networks have a more complex architecture. On the figure above, UE (User Equipment) is an end-user device, such as smartphone, that connects to the LTE network for voice and data services, eNodeB (Evolved

Node B) is a base station that manages wireless communication between UE and the core network, EPC (Evolved Packet Core) is the core network architecture in LTE, responsible for handling user data, mobility, and connectivity management between eNodeB and external networks, SGW (Serving Gateway) is the routing point between eNodeB and PGW, managing user mobility and data forwarding, PGW (PDN Gateway) connects the LTE network to external IP networks (PDN: Packet Data Network), handling IP address allocation and policy enforcement, MME (Mobility Management Entity) controls plane entity that manages UE signaling and mobility, such as authentication and handovers, PCRF (Policy and Charging Rules Function) governs policy control and charging for data services, defining and enforcing rules for QoS and billing.

There are many interfaces and connection types, used in the LTE networks: in this works, we will be focusing on the most important two: X2 and S1. X2 (X2 Interface) connects eNodeBs for direct communication, coordinating handovers and load balancing, while S1 (S1 Interface) connects eNodeB to the core network, divided into S1-MME for control plane traffic and S1-U for user plane traffic.

X2 interface is a novel interface, introduced in the LTE standard, supported by eNodeBs (in contrast to the NodeBs in, for example, 3G standard). As mentioned above, one of the primary uses of the X2 interface is performing handover. In a LTE network, handover refers to the process of transferring the connection (an active call or data session) from one base station to another without interruption. Handover is necessary when a user equipment (UE) moves out of the coverage area of one cell or base station and into the coverage area of another cell or base station.

There are several types of handovers that can occur in an LTE network [4], including:

- Intra-eNodeB handover happens when a UE moves from one cell (coverage area) to another cell within the same base station (eNodeB).
- Inter-eNodeB handover happens when a UE moves from the coverage area of one base station to the coverage area of another base station.
- Inter-frequency handover happens when a UE moves from one frequency band to another within the same base station.
- Inter-system handover happens when a UE moves from one type of wireless network (e.g. LTE) to another (e.g. 3G).

The eNodeBs are aware of each other's presence in the network through the X2 interface architecture: this way, they can request the X2 handover of the UE based on the connection metrics. However, the X2 interface architecture might be incomplete due to various reasons - for example, the connections might be lost during the reconfiguration.

This thesis will be focusing on improving the network performance through recovering and enhancing the X2 interface topology. At the same time, X2 interface is only available

for handovers, happening withing the same frequency and the same system. The inter-frequency handover and inter-system handover will remain out of the scope of the work.

## 1.1 Research question

This master's thesis aims to investigate the potential of applying link prediction techniques to the X2 interface architecture in an LTE network.

The research question can be formulated as following: is it possible to accurately predict the formation of X2 interface links between cells in an LTE network by analyzing time series Key Performance Indicator (KPI) metrics data collected from the cells?

This problem will be approached from a network and graph science perspective, where cells are considered as nodes, and X2 interface links represent the edges in the graph.

To explain our problem, let us include the problem formulation of estimating the X2 interface adjacency here for a graph $\mathrm{G} = (\mathrm{N}, \mathrm{L})$ [3], where $\mathrm{N}$ is the set of nodes and $\mathrm{L}$ is the set of edges.

Hence, with known $\mathrm{N}$ and available metrics (KPIs) for each node, presented as node attributes in a graph, we need to identify the adjacency of the network $A$.

## 1.2 Hypothesis

The hypothesis of this research is that by analyzing the time series KPI metrics data collected from cells in an LTE network, it is possible to establish a relationship between the cell performance indicators and the formation of X2 interface links. We expect that specific KPI patterns will emerge, providing valuable insights into the network's behavior and the likelihood of X2 interface link changes.

This understanding can then be used to develop models for X2 interface link prediction, leading to more efficient network management, optimization and fault recovery.

## 1.3 Structure of the Thesis

The structure of the thesis is as follows: Chapter 2 explains the necessary theoretical basics which the research is built on, Chapter 3 thoroughly defines the problem, explains how it is approached in the literature, lists the previous methods and explains the existing algorithms on estimating the unknown graph adjacency. Chapter 4 explains our data analysis and adjacency estimation algorithm. Chapter 5 illustrates the experimental setting and explains the combination of the pre-processing methods used, Chapter 6 presents the quantitative results alongside with a few examples to visually examine the performance of our method, and Chapter 7 provides the conclusive remarks, discussion and possible future work.

# 2. THEORETICAL BASICS

In this chapter, we introduce the theoretical basics that the thesis is built on. A brief introduction of the concepts X2 interface parameters, complex networks theory, graph parameters analysis and adjacency estimation will be provided in this chapter.

## 2.1 Telecommunicational networks

Telecommunication field has a great potential for being a subject of the complex network study. As the first mobile phone call has been made when GSM was introduced in 1991, mobile connection has become available in nearly every geographical location of the world[5].

The central feature of the communicational protocols, coming after the 3G generation is increased bandwidth and decreased latency. That is being achieved through three improvements: first one is the change in the frequency of the carrying signal, second is the introduction of indoor and outdoor “micro-cells” and thirdly, the change in the architecture of the network.

In 3G, every base transmission station (BTS) is connected to a central unit – Radio Network Controller, RNC (also knows as Base Station Controller in 2G). If a user, connected to this BTS and carrying the user equipment – UE is moving towards another base station, the central unit has to make decision and reassign the UE to a different BTS. This process is called handover. The handover process scheme utilizing BTS as a handover decision making unit is presented on Figure 2.1. It can be seen, that the UE, that is showing declining connection metrics with the NodeB 1 due to moving closer to the NodeB 2, has to be handed over to the NodeB 2. The decision is made by the RNC, that analyzes the metrics and reassigns the UE to the better suitable NodeB 2.

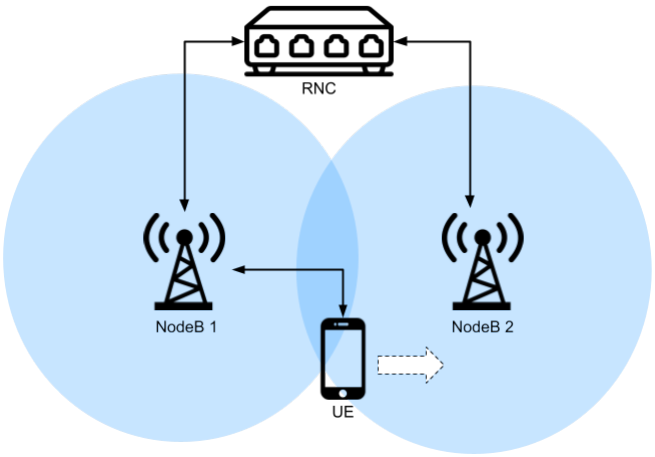



***Figure 2.1.*** *UE handover in 3G*

Another use case of the handing over is the bandwidth consumption control. A common pattern of the data consumption these days is a burst of requests, followed by the idle state – for example, when a user is watching a video for some time, and then puts the phone away. To optimize the traffic, the UE has to be reassigned to a less wide channel, to allow another UE to consume the bandwidth.

In contrast to 3G, the following generations of networks – 4G and 5G do not rely on a central unit for managing the handover process. eNodeB (evolved Node B) in the LTE standard is the analog of a 2G BTS, or a NodeB in 3G. eNodeB combines the capabilities of RNC and NodeB. As opposed to BTS and RNC, eNodeB is capable of making a decision of handing over the UE to another eNodeB. That results in the latency being is decreased by up to 17% in 4G [6] and 83% in 5G [7] compared to the 3G generation networks.

The handover process in LTE is quite a complex process. The handover type may vary on whether the EPC (Evolved Packet Core, the backbone unit of the LTE network) is changed in the process of the handover.

- Intra-LTE Handover: This occurs within the same LTE network. It happens when a user moves from one LTE cell to another LTE cell under the coverage of the same LTE network, managed by a single operator. The handover is managed by the same core network, ensuring a smooth transition for the user. Handover is managed within the same LTE EPC.
- Inter-LTE Handover: This type of handover also occurs between two LTE cells, but the cells belong to different LTE networks managed by different operators. In this case, a user moves from an LTE cell of one network to an LTE cell of another network. This handover requires coordination between the two different LTE networks to ensure a seamless transition. Handover is happening withing two different LTE EPCs.

- Inter-RAT (Radio Access Technology) Handover: This handover takes place between cells belonging to different radio access technologies, such as when a user moves from an LTE cell to a 3G cell or a 2G cell. This requires coordination between the different RATs to maintain connectivity as the user transitions between the networks. In this scenario, handover is happening between different network standards.

In the scope of this work, we will be focusing on the modern standard Intra-LTE Handover. This handover type itself can be further categorized, based on whether EPC is involved in the handover process decision making. The process of the decision is displayed on Figure 2.3.

- X2 Handover: This type of handover takes place when a user moves between two cells that are served by different eNodeBs, which have a direct X2 interface connection between them. The X2 interface allows eNodeBs to communicate with each other directly, without involving the EPC [8]. X2 Handover is managed by the source and target eNodeBs, and it is generally faster and more efficient, as it bypasses the EPC, reducing latency and resource usage. X2 interface handover in detail is presented in Figure 2.2.

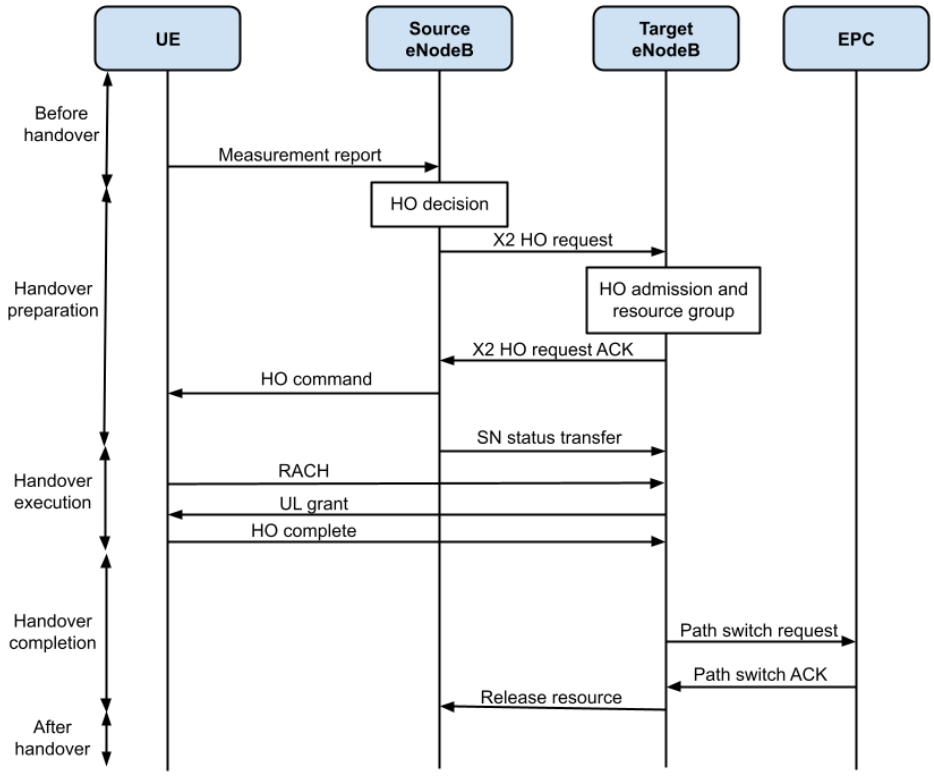



***Figure 2.2.*** *Handover through the X2 interface*

- S1 Handover: In contrast, S1 Handover occurs when a user moves between two

cells served by different eNodeBs that do not have a direct X2 interface connection. In this case, the handover process involves the EPC, particularly the Mobility Management Entity (MME) and the Serving Gateway (S-GW). The source eNodeB communicates with the MME, which then coordinates with the target eNodeB to manage the handover. Since the core network is involved, S1 Handover can be slower and less efficient than X2 Handover due to increased latency and resource usage. However, S1 Handover ensures connectivity when the X2 interface is not available or when handover between different networks is required.

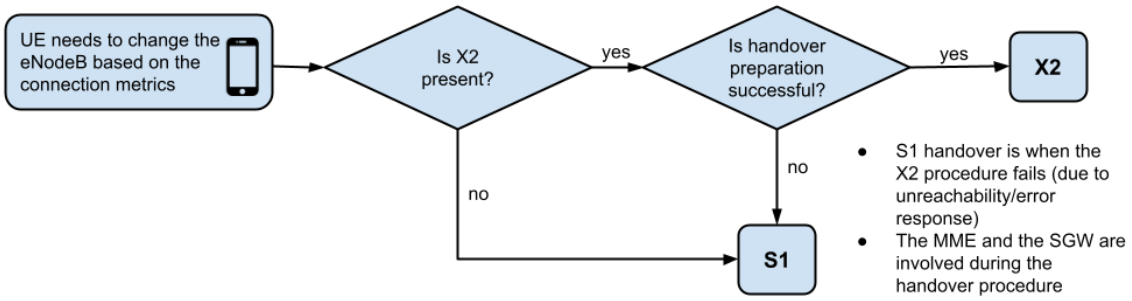



***Figure 2.3.*** *Basic concept of X2 and S1 handover selection*

To further clarify the LTE architecture, we will be also defining the concept of a cell and it's relation to the eNodeB. A cell in an LTE network represents the geographic area covered by a specific radio frequency [9]. A cell is the smallest unit of coverage in a cellular network, allowing the mobile devices to communicate with the eNodeB, whereas eNodeB is a base station that serves as the main component of an LTE network's radio access network (RAN). eNodeBs are responsible for managing and coordinating the radio resources for one or more cells. The relation between the cells and the eNodeBs is "many-to-one", which means that there are typically multiple cells, belonging to the same eNodeB and providing the area coverage. The user could be moving within the cells, belonging to the same eNodeB, that will provide seamless handover and fault-free connection.

An example, demonstrating the different number of cells and their impact on the coverage is presented on Figure 2.4.

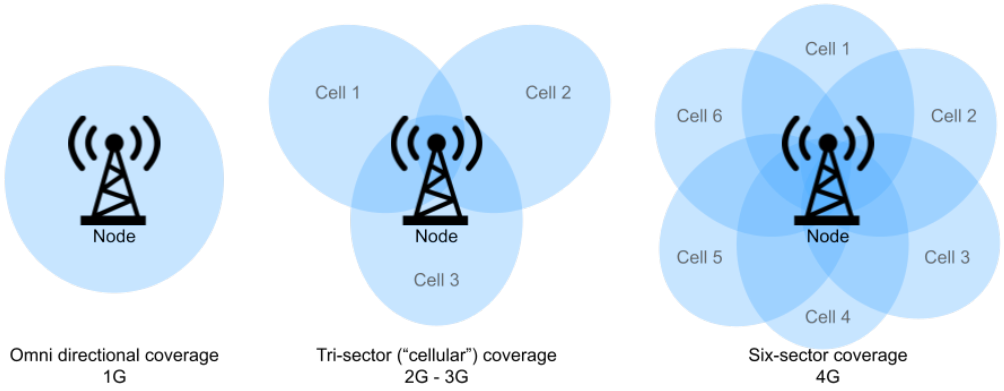



***Figure 2.4.*** *Cells coverage example*

This way, X2 handover is happening in a peer-to-peer complex network of eNodeBs and the cells that they consist of. It can be seen, that X2 interface is always preferable to the S1 interface. In cases, when it is not available, due to an architecture issue or a failure otherwise, a missing link has to be created or recovered, to ensure the quick and seamless handover of the UE. That allows us to focus on the X2 handover as a link analysis and prediction problem in a complex network.

## 2.2 Complex networks

Network science is a truly interdisciplinary field. In the modern world anything can be described as a graph [10]. Network science helps to take a fresh look at issues in various fields, like neural biology, social networks, transportation optimization and telecommunications analysis. Examples of these structures can be found everywhere in the surrounding world, starting from the very core of our own existence, rooted in genome and the way humans interact and cooperate with each other. More example include:

- Power grids, that define the network of transmission lines
- Social networks, that define how individuals interact with each other and other groups of individuals
- Supply networks, that define how groups of people exchange goods and services
- Communication networks, that define how devices interact with each other and the gateways

These structures can be described as complex networks. Each complex network, either natural or man-made can be described with a mathematical model [11]. The growing population and rapidly increasing scale of telecommunication networks, as well as the official start of the Internet in 1971 has boosted the interest in the graph theory. Collecting this data has helped to identify the core laws of the complex networks. It has been shown that despite the size, scale or origin of the networks they can be described using the set

of common principles. Detection of these principles, that will be described further, have allowed to plan and create the networks that will satisfy the requirements. It has been also demonstrated that realistic scale-free networks can be modeled with a limited set of parameters, derived or adjusted from the real-life networks [12][13].

### 2.2.1 Graph

The main subjest of the network science is a graph. We define a graph by a set of nodes or vertices and edges or links that connect them 3.1.

$$G = \{V, E\} \tag{3.1}$$

where V is a set of vertices, E is the set of edges. Typically $N$ denotes the number of nodes in the system and $L$ defines the number of links in the system. For example, Figure 2.5 shows a graph with $N$=10 and $L$=25.

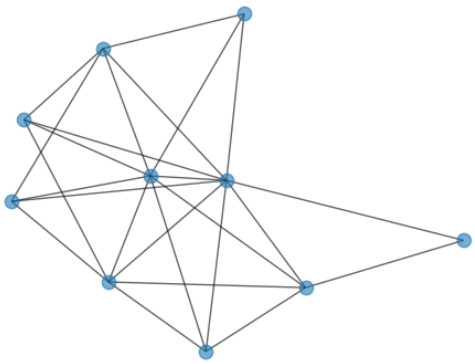


***Figure 2.5.*** *Single undirected graph*

The figure above demonstrates an undirected graph, which has edges that equally connect two nodes and provide bidirectional information flow. If a graph has edges liming the communication to one direction, it is called a directed graph. For example, the Internet is an undirected network of routers connected to each other, communicating on network interface level. World Wide Web (WWW) is a directed network that consists of web pages linked to each other.

It can be noticed that the same model can describe a variety of different graphs. An exact graph blueprint can be derived from a graph adjacency matrix $A$.

Adjacency matrix of a graph is denoted as following (5.1.2):

$$A_{ij} = \begin{cases} 1, if\{i,j\} \in E \\ 0, if\{i,j\} \notin E \end{cases} \tag{2.1}$$

For a weighed graph, $a_{ij}$ can take any value $\in [0,1]$. To better analyze the network, basic parameters can be derived from $A_{ij}$ [10].

Degree $k$ is one of the key parameters of the network. $k_i$ represents the number of links of a node $i$ (4.1).

$$k_i = \sum_{i=1}^{N} A_{ij} = \sum_{j=1}^{N} A_{ij} \tag{2.2}$$

Average degree is denoted as $\langle k \rangle$ (2.3).

$$\langle k \rangle = \frac{1}{N} \sum_{i=1}^{N} k_i = \frac{2L}{N} \tag{2.3}$$

### 2.2.2 Degree properties

Degree distribution $p_k$ is a probability of a node in a network having a specific degree $k$ with a network with number of nodes $N$ 2.4.

$$p_k = \frac{N_k}{N} \tag{2.4}$$

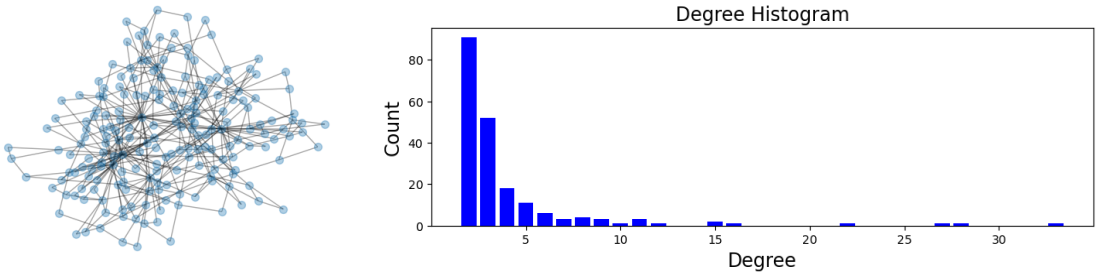
That way, $\langle k \rangle = \sum_{k=0}^{\infty} k p_k$.

Degree distribution plays a significant role in analyzing the real-life networks [14]. Precise functional form of $p_k$ determines many network phenomena, from network robustness to the spread of viruses, and crucial for analyzing and enhancing a telecommunication network performance.

One more parameter that can be used for network analysis in our work is the average clustering coefficient, that defines the connectedness degree of a network (2.5).

$$C = \frac{1}{N} \sum_{v \in G} c_v \tag{2.5}$$

An example of the degree distribution of a random graph, generated with Holme and Kim algorithm for growing graphs with power law degree distribution and approximate average clustering (with $N$ = 200, $L$ = 396, $\langle k \rangle$ = 3.96, $C$ = 0.44) is presented on Figure 2.6.




(a) (b)

***Figure 2.6.*** *A sample graph (a) and a corresponding degree distribution plot (b)*

The last property that we will consider for the further graph analysis is the scale-free property of the graph. In network science, the scale-free property refers to a network where the degree distribution of the nodes follows a power-law distribution. In other words, a few nodes have a significantly high number of connections, while the majority of nodes have only a few connections. This type of network is commonly observed in real-world systems such as social networks, the internet, and biological networks.

The degree distribution in a scale-free network can be described by the following power-law equation 2.6:

$$P(k) \sim k^{-\gamma} \tag{2.6}$$

where $P(k)$ represents the probability of a node having degree k, $\gamma$ is a positive constant exponent, and $k$ is the degree of a node.

The scale-free property implies that the network remains structurally similar as it grows, maintaining the same degree distribution characteristics across different scales. That way, we are able to analyze how a generated network, or a network with predicted or recovered edges (links) would behave in a situation of high demand or node failure.

# 3. BACKGROUND

In this chapter we briefly cover the literature of the complex network analysis and network link prediction and explain the recent and state-of-the-art methods.

## 3.1 Analytical approach in link prediction

Graph topology analysis, including link prediction, has been an object of the network science interest throughout the whole history of the field. Alongside with that, it has been identified that improving the mobile network or other technological system (like a power grid) with the help of network science and graph-oriented approach can show impressive progress and increase in network performance.

Classic link prediction methods are based on predicting a possible link in a network, with already known topology, for example, suggesting a new friend in a social network, based on already existing contacts. The methods include: Jaccard coefficient [15], that measures similarity between two nodes in a network by calculating the ratio of their common neighbors to the union of their neighbors; Adamic-Adar Index [15], similar to Jaccard Coefficient, but weights common neighbors by the inverse of their degree (logarithmically); Common Neighbor Centrality [16], that measures the importance of a node in a network by counting the number of common neighbors it shares with other nodes, indicating that a node is more central and well-connected within the network, playing a vital role in maintaining the network's connectivity; Preferential Attachment Score [15] predicting the likelihood of a link forming based on the product of the degrees of the two nodes — nodes with higher degrees are more likely to form connections, representing the "rich get richer" phenomenon.

Network topology planning and optimization solutions are in high demand on the market, hence it has resulted in a variety of research published during the past decade. For example, optimal site locations are planned by formulating optimization problems, e.g. integer linear programming problem for disaster recovery [17], or an extended from the clique-based approach that improves the throughput by 155 percent [18].

Another example of the graph signal processing in the presence of unknown topology [19] is using a Bayesian approach to estimate the presence of certain edges based on

the statistics of the data. In this work, it is pointed out how the uncertainty on the edges translates onto the eigencomposition of the graph Laplacian. Finally, there is a joint diagonalization of correlation matrices approach [20]. It demonstrates that the topology can be identified by jointly diagonalizing the slabs of a three-way tensor. This tensor is constructed using second-order statistics of the node measurements, which makes the method effective in terms of resisting to the noise or partially missing measurements. Most importantly, this method can used for both directed and undirected networks.

## 3.2 Machine learning approach in link prediction

On the other hand, machine learning approach has become a popular method of identifying patterns in data.

Convolutional Neural Networks (CNNs) have been traditionally applied for pattern recognition problems, e.g. objects identification, anomaly detection or natural language translation. However, these examples usually assume grid-like data, for example an RGB image. With this approach, it is possible to reuse the local filters and learnable parameters by reapplying them on all input positions. In contrast to the grid-like data, graph-represented data belongs to an irregular domain.

There is a number or examples of applying the neural network approach for the graph-like data. Originally, recursive neural networks have been used for analyzing acyclic graphs[21]. Graph Neural Networks (GNNs) were introduced already in 2005 [22] and improved in 2009 [23] as a generalization of recursive neural networks that can directly deal with a more general class of graphs, e.g., cyclic, directed, and undirected graphs. GNNs consist of an iterative process, which propagates the node states until equilibrium, followed by a neural network, which produces an output for each node, based on its state.

A recent modern approach for graph data processing and node classification is proposed in the Graph Attention Network (GAN or GAT) [24], that enables attending the nodes neighbors and specifying different weights to different nodes in the neighborhood, without knowing the graph topology upfront. The attention-based approach has shown prominent results in the node classification in the presence of the topology uncertainty. Combined approaches as well have proven to be effective for solving the problem of unknown graph topology [25].

One of the latest works featuring learning over graph has been published in March 2021 [26], and introduced a novel GARNN framework, that utilizes the recurrent neural networks to take into account temporal dependency while paying attention to the adaptive weight matrices learned from the graph, built based on the multi-head attention mechanism[24] to consider the correlations among time series.

Lastly, GraphSAGE [27] that stands for "Graph Sample and Aggregate" and is a novel

inductive learning framework for graph-structured data, introducing a fresh approach to the GNNs. GraphSAGE utilizes a neighborhood sampling strategy to reduce the computational complexity and memory requirements of the learning process. It aggregates the sampled neighborhood information using various aggregator functions, such as mean, LSTM, or max-pooling. The aggregator function is learned through backpropagation, allowing the model to adapt and optimize the embeddings based on the task at hand.

It is worth noting, that the majority of the works, focusing on the link prediction problem on the graph-structured data, are utilizing datasets with node feature vectors that describe the node itself. There are a few very commonly used datasets, that are described in the Table 3.1.

| **Dataset** | **Description** | **Node** | **Node features** | **Edges** |
|---|---|---|---|---|
| Cora [28] | Citation network dataset commonly used for node classification tasks. | Scientific publications (2708) | Bag of words (1433) | Citation links between the publications |
| Citeseer [28] | Citation network dataset similar to Cora. | Scientific publications (3327) | Bag of words (3703) | Citation links between the publications |
| Pubmed [28] | Citation network dataset, consisting of publications from the field of diabetes. | Scientific publications (19717) | Term Frequency-Inverse Document Frequency (500) | Citation links between the publications |
| Protein-Protein Interactions (PPI) [29] | Represents protein-protein interaction networks from different organisms | Human protein (21557) | Protein sequences, functional annotations, or topological properties | Physical interaction between proteins in a human cell |

***Table 3.1.*** *Most commonly used graph-structured datasets in the link prediction research*

There are only few works, that focus on the link prediction problem with node features presented as timeseries structured data. In this [30] study, the authors tackle the issue of predicting connections in changing networks, that evolve over time and change such

parameters as node degree, assuming they follow a vector autoregressive (VAR) model. A similar work [31], that also observes networks evolution over time, proposes building time series for each pair of non-connected nodes by computing their similarity scores at different past times.

This way, we can see that we will need to find a novel feature extraction method, suitable for the graph data structure, timeseries-based node features and constant degree parameters of the nodes, used in this study. GNN models, demonstrating most prominent link prediction results, such as GraphSAGE, are utilizing bag-of-words node feature vectors or other descriptive feature types. Hence, these GNN models can be used after the data pre-processing and finding an effective feature extraction method.

# 4. METHODOLOGY

In this section we identify the possible ways of resolving the research question and compare the methods to choose the most suitable approach. The methods discussed include methods for clustering the timeseries data, identifying the correlation between the timeseries data, timeseries features extraction and network link prediction using higher order features datasets.

## 4.1 Correlation identification

### 4.1.1 Pearson correlation coefficient

To start with, one of the straightforward methods of identifying the connection between two vectors is Pearson correlation coefficient [32].

$$r = \frac{\sum (x_i - \bar{x})(y_i - \bar{y})}{\sqrt{\sum (x_i - \bar{x})^2 \sum (y_i - \bar{y})^2}} \tag{4.1}$$

That method can be used as a baseline formula, since it is successful at identifying correlation uniformly distributed data and data without failures or anomalies. It is more rarely applied to the raw time series datasets because coefficient $r$ is measuring the global synchrony, rather than the local synchrony.

### 4.1.2 Dynamic Time Warping

Dynamic Time Warping (DTW) is a popular technique for measuring the similarity between two time series with varying lengths or time scales. DTW allows non-linear alignment of the time series, enabling the comparison of sequences with different rates of progression [33]. This is particularly useful in applications such as speech recognition, gesture recognition, and time series clustering.

The main idea behind DTW is to find the optimal alignment between two time series that minimizes the cumulative distance between their corresponding data points. This way, a new timeseries can be obtained, that summarizes the features of the original input timeseries. The following steps outline the DTW algorithm:

1. Compute the pairwise distance matrix $D$ between the data points of the two time series, $X$ and $Y$, with lengths $M$ and $N$ respectively:

$$D_{i,j} = d(x_i, y_j)$$

   where $d(\cdot, \cdot)$ is a distance function, such as the Euclidean distance.

2. Define a warping path $P = \{p_1, p_2, \ldots, p_L\}$, where $p_l = (i_l, j_l)$, with $1 \leqslant i_l \leqslant M$ and $1 \leqslant j_l \leqslant N$. The path starts at $(1, 1)$ and ends at $(M, N)$.

3. Calculate the accumulated cost matrix $C$ using dynamic programming:

$$C_{i,j} = D_{i,j} + \min\{C_{i-1,j}, C_{i,j-1}, C_{i-1,j-1}\}$$

   with initial conditions $C_{1,1} = D_{1,1}$, $C_{i,1} = D_{i,1} + C_{i-1,1}$ for $i = 2, \ldots, M$, and $C_{1,j} = D_{1,j} + C_{1,j-1}$ for $j = 2, \ldots, N$.

4. Find the optimal warping path $P^*$ by backtracking from $(M, N)$ to $(1, 1)$ through the accumulated cost matrix $C$.

5. The DTW distance between the two time series is the sum of the pairwise distances along the optimal warping path:

$$\text{DTW}(X, Y) = \sum_{l=1}^{L} D_{i_l, j_l}$$

### 4.1.3 Dynamic Time Warping Barycenter Averaging

DTW algorithms has some limitations when it comes to clustering large datasets of time series data. One of these limitations is that DTW is computationally expensive, especially when dealing with long time series or large datasets.

To address this issue, a technique called Dynamic Time Warping Barycenter Averaging (DBA) was developed [34]. DBA is a method for summarizing a group of time series data points into a single representative time series, which can then be used in place of the original data points for clustering purposes. The idea behind DBA is to compute an average time series, or barycenter, that is representative of the entire group.

The DBA algorithm works as follows:

1. Choose an initial barycenter, which can be any time series in the group.
2. For each time series in the group, compute the DTW distance between the time series and the current barycenter.
3. Align each time series to the barycenter using DTW.
4. Compute a new barycenter by averaging the aligned time series at each time step.

5. Repeat steps 2-4 until the barycenter converges to a stable value.

The resulting barycenter is a time series that represents the group of time series data points in a way that minimizes the DTW distance between the barycenter and the original time series data points.

DBA can be used as a pre-processing step for clustering algorithms, where the representative barycenter is used as a surrogate for the original time series data points. This can reduce the computational cost of clustering, as the number of time series to be compared is reduced to just one representative time series per group.

## 4.2 Dimensionality reduction

### 4.2.1 Principal Component Analysis

Principal Component Analysis (PCA) is a popular technique for feature extraction and dimensionality reduction. It can be used to analyze timeseries data features and identify the most important patterns and trends in the data. As we should not perform PCA directly on the timeseries dataset, we would need to extract the timeseries features of the timeseries data, and further select the most relevant feature vectors. Then, PCA is performed using the algorithm, described below. First, the dataset is standardized by centering the data by subtracting the mean of each feature from the feature values. Optionally, we can scale the data by dividing each feature by its standard deviation to have unit variance.

$$X_{ij} = \frac{X_{ij} - \mu_j}{\sigma_j} \tag{4.2}$$

where $X_{ij}$ represents the value of the $j$-th feature for the $i$-th observation (cell or eNodeB), $\mu_j$ is the mean of the $j$-th feature, and $\sigma_j$ is the standard deviation of the $j$-th feature.

Then we compute the covariance matrix of the standardized dataset:

$$S = \frac{1}{N-1} X^T X \tag{4.3}$$

where $S$ is the covariance matrix, $X$ is the standardized dataset, and $N$ is the number of observations (cells or eNodeBs).

After obtaining the covariance matrix of the features dataset, eigenvalue decomposition is performed.

$$S v_k = \lambda_k v_k \tag{4.4}$$

where $\lambda_k$ and $v_k$ are the eigenvalues and eigenvectors of $S$, respectively.

Then we sort the eigenvalues in descending order and select the corresponding eigenvectors. Finally, we create the projection matrix $W$ by concatenating the top $M$ eigen-

vectors, where $\mathrm{M}$ is the desired number of dimensions after reduction and transform the standardized dataset into the lower-dimensional space:

$$\mathrm{T} = XW \tag{4.5}$$

where $\mathrm{T}$ is the transformed dataset. The resulting transformed dataset can then be used for further analysis, such as clustering or classification.

### 4.2.2 Hypothesis testing

Hypothesis testing is a statistical method employed to determine the significance of features in a dataset. It can be used to identify the most relevant features in a feature selection process by comparing the null hypothesis ($\mathrm{H}_0$) against the alternative hypothesis ($\mathrm{H}_1$). The null hypothesis assumes that the feature under consideration has no significant impact on the target variable, while the alternative hypothesis assumes the opposite. If the test statistic falls into the critical region, the null hypothesis is rejected, indicating that the feature is relevant.

One such hypothesis testing-based feature selection algorithm is the FRESH (Feature Relevance Estimation using Statistical Hypotheses) algorithm [35]. FRESH assesses the relevance of each feature by estimating the p-value for the null hypothesis. Features with p-values below a pre-defined threshold are considered relevant and retained for further analysis.

The FRESH algorithm can be briefly described as follows:

1. Define the null hypothesis $\mathrm{H}_0$: The feature has no significant impact on the target variable.
2. Define the alternative hypothesis $\mathrm{H}_1$: The feature has a significant impact on the target variable.
3. For each feature, perform a hypothesis test (e.g., t-test, ANOVA, etc.) to calculate the test statistic and corresponding p-value.
4. Compare the obtained p-value with a pre-defined significance level ($\alpha$). If the p-value is less than $\alpha$, reject the null hypothesis and consider the feature relevant.
5. Retain the relevant features for further analysis.

## 4.3 Timeseries signal decomposition

Timeseries signal decomposition, or Seasonal-Trend Decomposition (STL) is a widely used technique for decomposing a time series into its seasonal, trend, and residual components. The Discrete Fourier Transform (DFT) is a mathematical tool that can be em-

ployed for this purpose. By transforming the time series data into the frequency domain, the DFT allows for the separation of different frequency components, enabling the extraction of seasonal and trend patterns.

The following steps outline the seasonal-trend decomposition of time series data:

1. Calculate the Discrete Fourier Transform of the time series data:

$$X_k = \sum_{n=0}^{N-1} x_n \cdot e^{-j\frac{2\pi}{N}kn}$$

   where $X_k$ is the $k$-th frequency component, $x_n$ represents the $n$-th data point in the time series, $N$ is the total number of data points, and $j$ is the imaginary unit.

2. Identify the seasonal and trend frequency components by analyzing the amplitude of the frequency spectrum.
3. Remove the seasonal frequency components by setting their amplitude to zero in the frequency domain.
4. Inverse Discrete Fourier Transform the modified frequency domain data to obtain the detrended time series data:

$$x_n = \frac{1}{N} \sum_{k=0}^{N-1} X_k \cdot e^{j\frac{2\pi}{N}kn}$$

5. Perform a moving average or other smoothing technique on the detrended data to estimate the trend component.
6. Subtract the trend component from the detrended data to obtain the residual component.

These extracted components can be further used for clusterization and link prediction between the components, instead of the raw timeseries data or timeseries features, extracted from the dataset.

## 4.4 Adjacency recovery

### 4.4.1 Node classification with Graph Neural Networks

Node classification is a common task in network analysis where the goal is to predict the class labels of the nodes in a network based on their structural properties and features. Recently, graph neural networks (GNNs) have emerged as a powerful tool for node classification, allowing for the incorporation of both local and global network information into the learning process.

One of the most popular GNN models for node classification is the Graph Convolutional Network (GCN) [36]. The GCN uses a convolutional-like operation to aggregate information from neighboring nodes in a graph, allowing for the propagation of information across the entire network. Those node representations node are calculated by combining the connectivity and features of a local neighborhood. The formula for computing the representation of a node in a GCN is presented below:

$$h_v^{(l+1)} = \sigma\left(\sum_{u \in \mathcal{N}(v)} \frac{1}{\sqrt{|\mathcal{N}(v)||\mathcal{N}(u)|}} h_u^{(l)} W^{(l)}\right) \tag{4.6}$$

where $h_v^{(l)}$ is the representation of node $v$ at layer $l$, $\mathcal{N}(v)$ is the set of neighbors of node $v$, $W^{(l)}$ is the weight matrix for layer $l$, and $\sigma(\cdot)$ is a non-linear activation function, such as the ReLU function.

$$f(x) = \max(0, x) \tag{4.7}$$

This approach can be used for predicting the intra-node adjacency, by labeling a cell as belonging to a eNodeB as belonging to a class. However, the biggest restriction of this method in application to this study is the fact that the number of classes, or clusters, grows with the number of cells. The potential effect of this restriction will be tested further in the Experiments chapter.

### 4.4.2 Link prediction with Graph Neural Networks

Graph neural networks have been proven to be an effective method of graph-based learning. The algorithm, selected for this work and adapted for the timeseries data is GraphSAGE [27]. GraphSAGE is a machine learning algorithm designed for learning representations of nodes in large, complex graphs. The GraphSAGE algorithm works by sampling and aggregating information from a node's local neighborhood, or subgraph. The algorithm first randomly samples a fixed number of nodes from a node's local neighborhood, including the node itself. It then aggregates the feature information from the sampled nodes and feeds it through a neural network to generate a new embedding for the node. The main idea behind GraphSAGE is to learn a function that generates node embeddings by aggregating information from a node's local neighborhood. The model consists of multiple layers, where each layer performs neighborhood aggregation and transformation operations.

The GraphSAGE algorithm can be described as follows:

1. For each node $v$, sample a fixed-size set of neighbors $N(v)$.
2. For each layer $k$ in the GNN model, update the embeddings of node $v$ and its

neighbors using the following aggregation function:

$$h_v^{(k)} = \sigma\left(W^{(k)} \cdot \text{AGGREGATE}\left(\{h_u^{(k-1)} : u \in N(v)\}\right)\right)$$

where $h_v^{(k)}$ is the k-th layer embedding of node $v$, $W^{(k)}$ is a learnable weight matrix for layer $k$, $\sigma(\cdot)$ is an activation function (e.g., ReLU), and AGGREGATE is an aggregation function (e.g., mean, max, or sum).

3. After $K$ layers of aggregation and transformation, the final node embeddings are obtained, which can be used for various downstream tasks such as node classification, link prediction, or clustering.

The model then predicts the probability of existence of an edge by computing a score between the representations of both incident nodes with a function (e.g. an MLP or a dot product) 4.8.

$$\hat{y}_{u\sim v} = f(h_u, h_v) \tag{4.8}$$

# 5. EXPERIMENTS

After the methods are introduced, we implement the algorithms described in the methodology section for data pre-processing, feature extraction and further link prediction. The input datasets are first presented in Section 5.1. and then prepared for link prediction through a pre-processing method described in Section 5.2. Then, the different combinations of dimensionality reduction, clustering and features extraction algorithms are detailed in Section 5.3. Next, Section 5.4 introduces the link prediction methods using cell clustering techniques and Graph Neural Networks (GNNs). Further, the evaluation process and metrics are explained in Section 5.5. Finally, the software tools used for the implementation are listed in Section 5.6

## 5.1 Datasets

The are two datasets, that have been provided for the study. Sections 5.1.1 and 5.1.2 describe and analyze the timeseries dataset and the adjacency dataset respectively.

### 5.1.1 Timeseries dataset

The first dataset is composed of the performance management data (KPIs) collected on separate cells, belonging to the corresponding eNodeBs. There are three major parameters that we observe for each of the cells. The first one is $MAX\_AVG\_ACTIVE\_USER\_CELL$, that represents the number of User Equipment (UEs) connected to the cells at the time. The other two parameters are $CELL\_THROUGHPUT\_DL$ and $CELL\_THROUGHPUT\_UL$, that represent cell throughput uplink and downlink respectively.

The timeseries dataset includes 7659072 discrete timeseries cell performance records, with distinct timestamps for 17 consecutive days. KPIs are sampled once per 15 minutes per distinct cell. Figure 5.1 represents plotted performance management data for a randomly chosen set of cells, belonging to different eNodeBs.

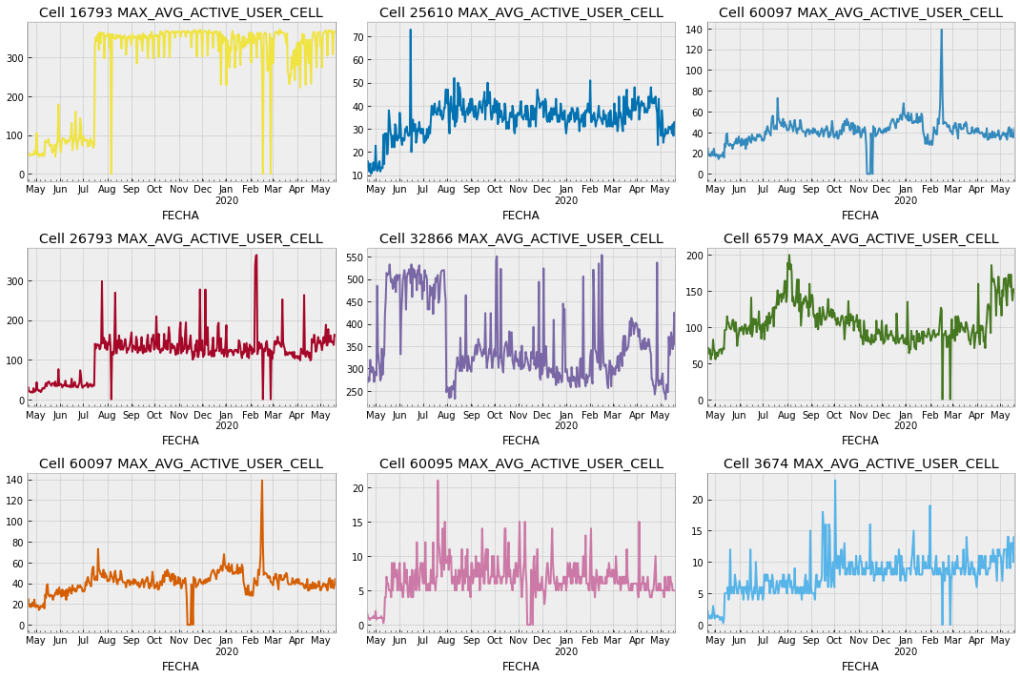



***Figure 5.1.*** *UEs connected to the sample set of cells*

It can be seen, that the KPIs are varying for both randomly chosen cells, belonging to different eNodeBs, as well as cells inside the same eNodeB. Figure 5.2 presents how cells of the same node have varying scale of KPIs, while presenting the same function outline.

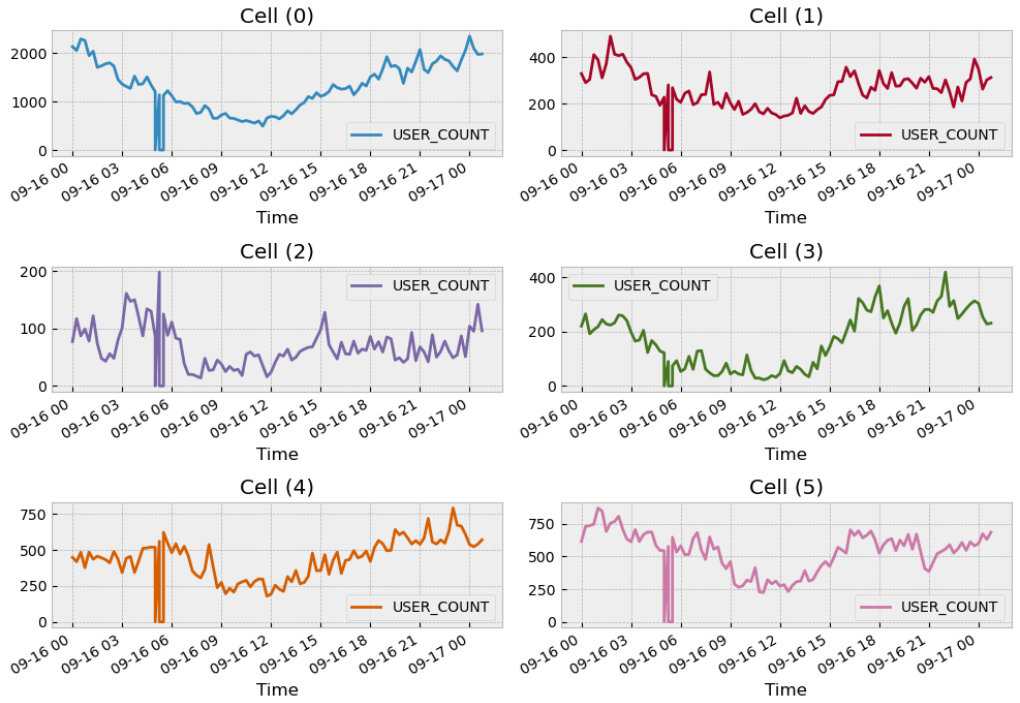



***Figure 5.2.*** *UEs connected to the sample set of cells*

To better understand the dataset, we analyze typical parameters of the timeseries data: autocorrelation, seasonality and stationarity. Figure 5.3 shows the standard deviation distribution among the sample of the 10% cells, selected randomly.

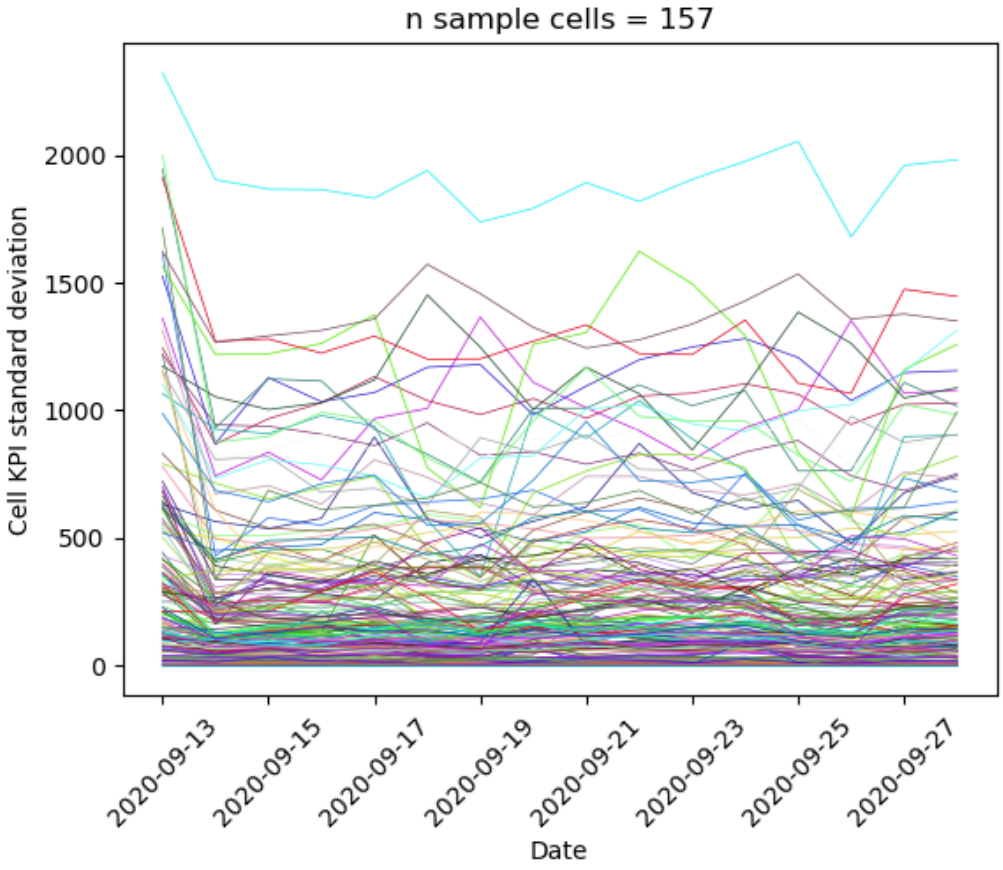



***Figure 5.3.*** *Sample cells standard deviation distribution by day*

It can be observed, that while the majority of the selected cells display steady standard deviation with a 1 day lag, there are a few exceptions. To determine, whether the time-series is stationary or not, we run the Augmented Dickey-Fuller unit root test for each of the cells, in order to test the hypothesis that a unit root is present in each series and, hence, prove that the data is non-stationary 5.1. The null hypothesis is that the unit root is present in the series ($\alpha = 1$), and confirmed if $\mathrm{p} > 0.05$.

$$\Delta \mathrm{y}_\mathrm{t} = \alpha + \beta \mathrm{t} + \gamma \mathrm{y}_{\mathrm{t}-1} + \delta_1 \Delta \mathrm{y}_{\mathrm{t}-1} + \cdots + \delta_{\mathrm{p}-1} \Delta \mathrm{y}_{\mathrm{t}-\mathrm{p}+1} + \varepsilon_\mathrm{t} \tag{5.1}$$

After running the test for each cell, we determine that 99.94% of the cells' time series are identified as stationary, with p-value being $\mathrm{p} < 0.05$. That result is satisfactory for the assumption that the dataset is stationary.

We also run the seasonality analysis, presenter on Figure 5.4, as well as identify. It can be seen, that the cells display explicit seasonality with 1 day lag.

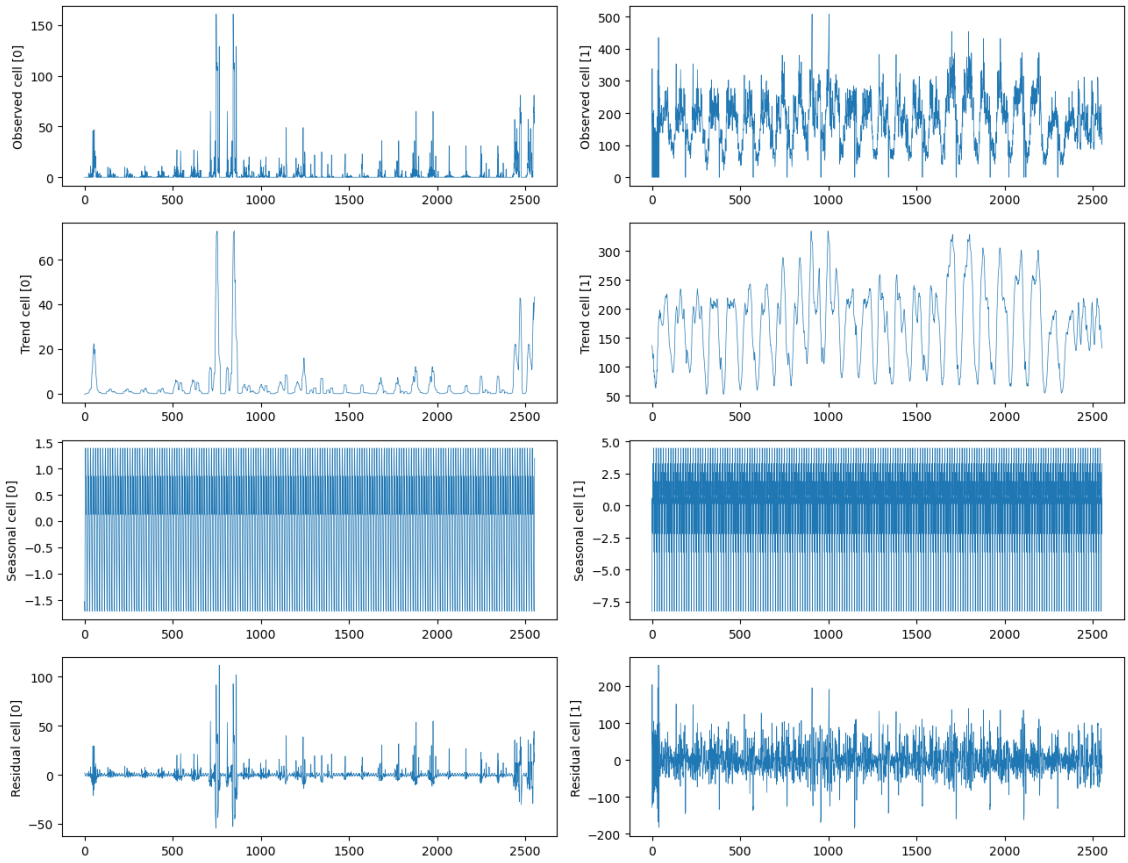



***Figure 5.4.*** *Selected cells seasonality analysis*

### 5.1.2 Adjacency dataset

Each cell and eNodeB element has a unique ID in the form of LNBTS-xxxx and MRBTS-xxxx respectively, where xxxx is the unique identifier. That identifier is used to describe the networks topology in the second dataset of the X2 interface relational recordings [37] in the following form 5.1.2:

$$\mathrm{Source} = \mathrm{MRBTS} - \mathrm{xxxx}/\mathrm{LNCEL} - \mathrm{xxxx}/\mathrm{LNREL} - \mathrm{xxxx}$$
$$\mathrm{Destination} = \mathrm{MRBTS} - \mathrm{xxxx}/\mathrm{LNCEL} - \mathrm{xxxx}$$

where MRBTS is the eNodeB identifier, LNCEL the cell identifier and LNREL is the relation identifier.

The cell and eNodeB adjacency dataset contains record of 2610 distinct cells and 953904 connections. A common notation in the network science for these entities are the node and the edge [11].

Additionally, dataset inconsistency is a commonly observed issue with the non-synthetic

data, collected from equipment in the field. We have analyzed the timeseries dataset and compared it to the entries in the adjacency dataset, identifying 614 unique cell entries, that are missing the corresponding adjacency information. We will remove these cells from the dataset, forming an additional dataset that will be further used for testing and validation purposes. We split the timeseries dataset into the parts, containing know and unknown adjacency, as well as filtering out adjacency entries, that are missing the KPIs, associated with them. It can be seen, that non-synthetic data is missing KPIs for around 39% of the cells, present in the original adjacency, as shown in the Table 5.1. Then, we build an undirected graph with Kamada-Kawai algorithm [38]. The graph is presented on Figure 5.5

| **Component** | **Known KPIs + known adjacency** | **Total** |
|---|---|---|
| Cells | 1577 | 2610 |
| eNodeBs | 328 | 811 |

***Table 5.1.*** *Counters of the cells and eNodeBs, present in the network*

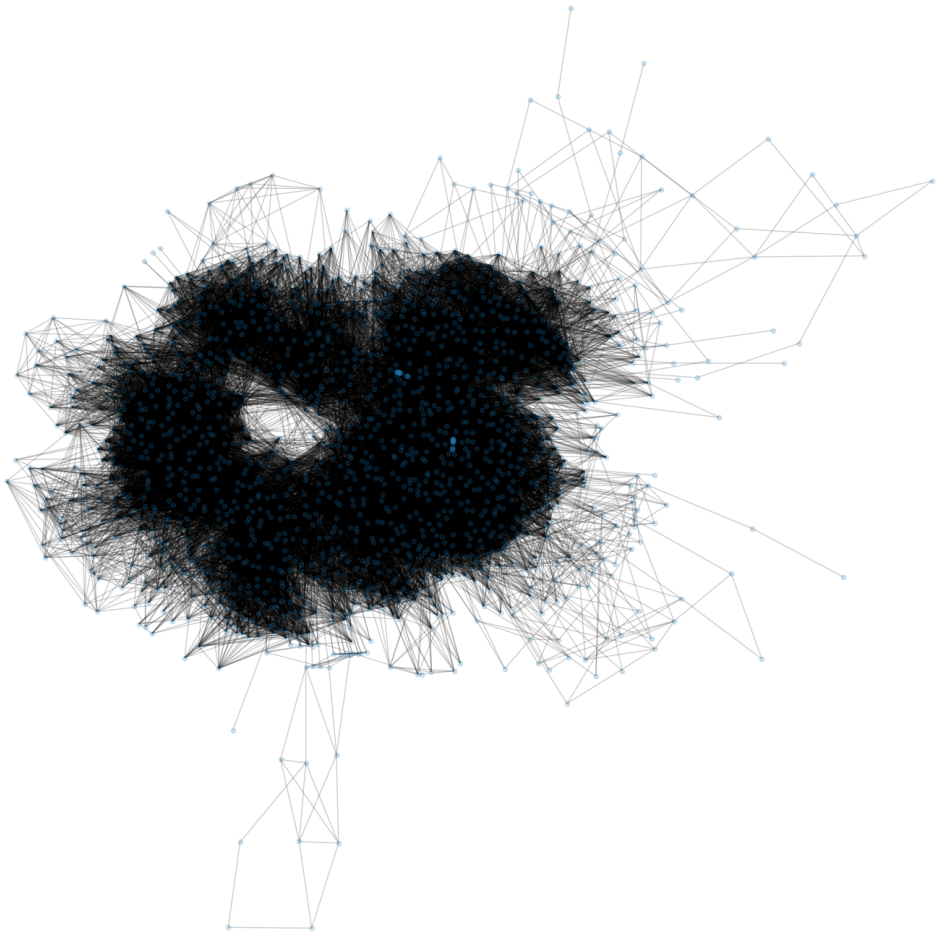


***Figure 5.5.*** *Full graph of the cells with known KPIs*

It can be noticed in the adjacency dataset that the relation between cells and eNodeBs is "many-to-one", "many-to-many" between the cells and "many-to-many" between eNodeBs. Taking this into account, we can build the inter-eNodeB graph, in order create a graph with a reduced number of both nodes and edges. The graph plot is presented on Figure 5.6. It can be seen, that the eNodeB graph is more sparse, compared to the cells graph.

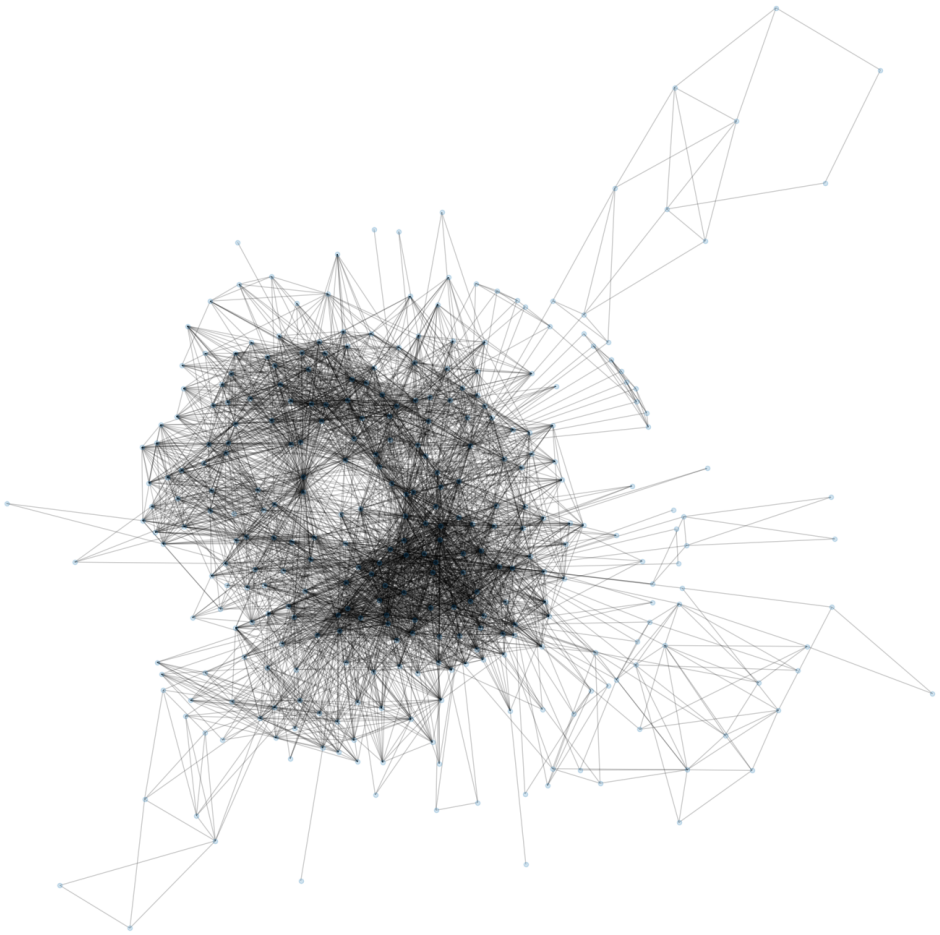


***Figure 5.6.*** *Full graph of the eNodeBs with known KPIs*

To further analyze the graphs, presented above, we calculate the common properties of the graphs. Average degree of a graph is measuring the ratio between the sum of all nodes degrees and the number of edges 5.6.

$$\langle k \rangle = \frac{1}{N} \sum_{i=1}^{N} k_i = \frac{2L}{N} \tag{5.2}$$

The resulting parameters of the graphs are presented in the Table 5.2.

It can be noted that the average degrees $\langle k \rangle$ are 53.12 and 26.45, which corresponds to supercritical regime, yet not reaching the connected regime:

$$ln(N) > \langle k \rangle > 1 \tag{5.3}$$

| Graph parameters | Cell adjacency graph | eNodeB adjacency graph |
|---|---|---|
| Number of nodes | 1577 | 322 |
| Number of edges | 41910 | 4259 |
| $\langle k \rangle$ | 53.12 | 26.45 |
| $\sigma$ of the degree distribution | 39.59 | 21.18 |

***Table 5.2.*** *Graphs degree parameters*

Degree distribution histogram of the cells graph and the eNodeBs graph are presented on Figure 5.7 and Figure 5.8 respectively. It can be seen from the figures, that the both cells and eNodeBs graph preserve the same degree distribution, which means that the eNodeB graph can be used for data point reduction of the original cells network.

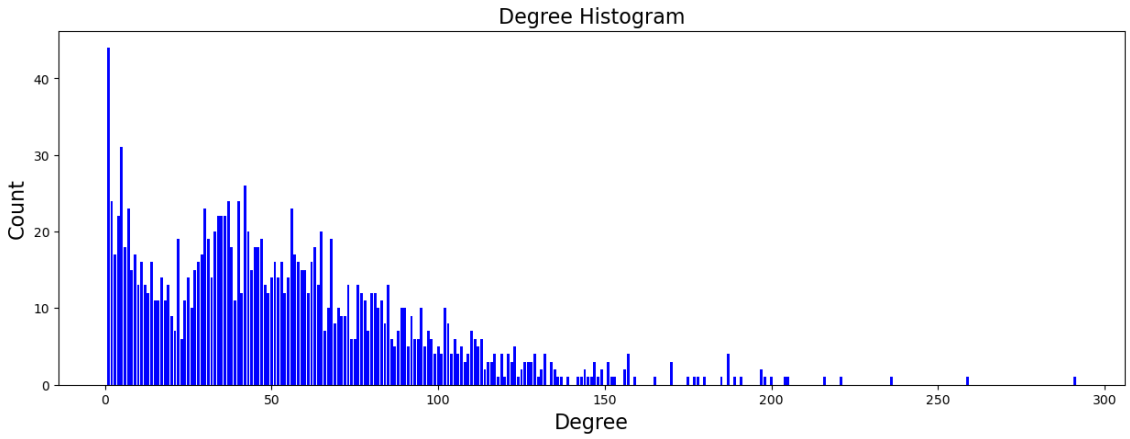



***Figure 5.7.*** *Degree distribution of the cells graph*

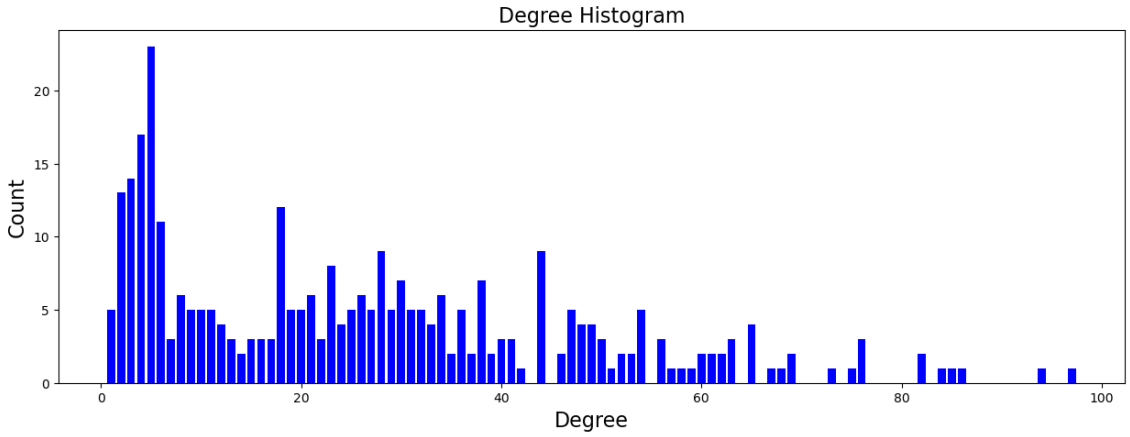



***Figure 5.8.*** *Degree distribution of the eNodeBs graph*

It can be seen, that there are only few celebrity nodes, while the majority of the nodes falls in the 1 $\sigma$ distance from $\langle k \rangle$. That way, we confirm that the data is describing a real world

scale-free network. Additionally, extra artifacts have been removed from the datasets as a part of the pre-processing, including NaN values and duplicates entries removal.

## 5.2 Data normalization and aggregation

At it has been demonstrated above (Figure 5.2), the KPIs of the cells, belonging to the same eNodeB exhibit substantial divergence, notwithstanding their adherence to a shared pattern. Hence, given the presence of features in the cells and eNodeBs time-series data that exhibit differing orders of magnitude, the application of mean variance scaling 5.4 is necessary.

$$x' = \frac{x - \bar{x}}{\max(x) - \min(x)} \tag{5.4}$$

Such scaling technique is implemented to ensure that the data features are placed on a consistent scale, thus mitigating potential biases that may arise due to the relative magnitudes of the features. Consequently, this will facilitate the identification of relevant patterns and the generation of accurate predictions in the subsequent analysis, clusterization and link prediction. A comparison between scaled and non-scaled data is displayed on Figure 5.9

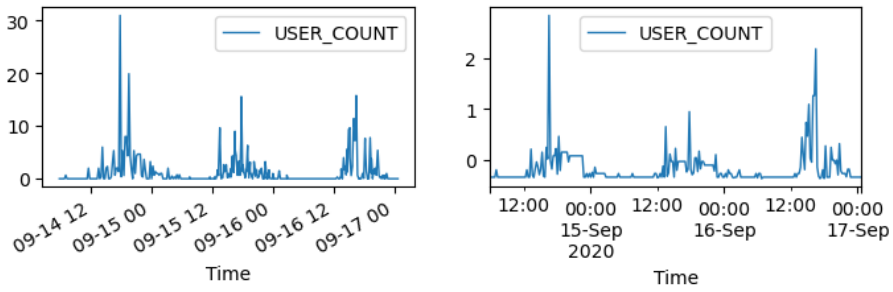



***Figure 5.9.*** *Comparison between the non-scaled (left) and mean variance scaled (right) KPIs*

## 5.3 DBA aggregation

After normalizing the data, it is possible to apply the Dynamic Time Warping Barycenter Averaging (DBA) algorithm [39] to aggregate the cells' time series and reduce the total number of the datapoints. DBA is generally considered to be a better approach than signal mean averaging for several reasons: DBA takes into account the alignment of time series data, it is more robust to noise and it can capture more complex temporal patterns, as signal mean averaging only considers the values of the signals at each time point, whereas DBA considers the temporal structure of the signals. After aggregating the scaled KPIs of the cells per eNodeB, we build a novel dataset with reduced number of datapoints, containing timeseries data for eNodeBs, as presented on Figure 5.10.

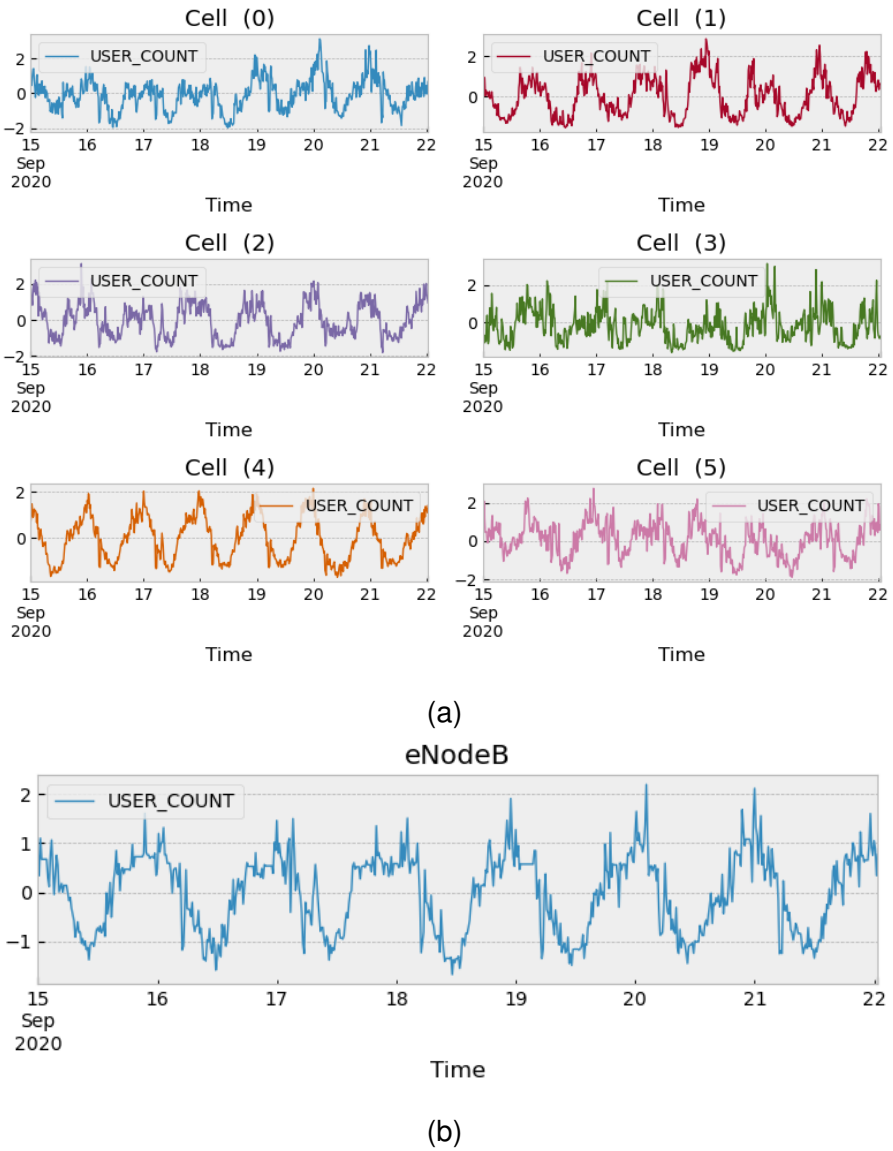



***Figure 5.10.*** *(a) Mean variance scaled KPIs for the cells, belonging to the same eNodeB (b) DBA-aggregated KPIs, resulting in a novel KPI timeseries for the eNodeB specifically*

## 5.4 Timeseries features extraction

The analysis of time series data involves the extraction of various features that provide insight into the underlying patterns and characteristics of the data. These extracted features are highly useful in reducing the dimensionality of the data and will be used in the subsequent classification task. The most widely used type of features is known as time domain features, which are derived from the temporal aspect of the data. In contrast, frequency domain features focus on the amplitude of different frequencies present in the signal. Accordingly, section 5.4.1 is dedicated to the extraction of time domain features and section 5.4.2 is focused on the extraction of frequency domain features.

### 5.4.1 Time domain features extraction

There are typical sets of features that can be extracted from a time series, using the in-build python libraries (for example, commonly used $pandas$ library) functionality, e.g. $.mean()$, $.median()$, $.max()$, $min()$, $var()$, $mode()$, $std()$. However, those features are insufficient, as they do not provide the insight into more complex features of the data, for example, autocorrelation, autocovariance and classic features calculated for different lags and splits. In order to calculate a wider set of features, we are using an automated tool, available in a form of a python library $tsfresh$ [35], that extracts up to 800 timeseries features for each time series. We extract the features for both cells time series and eNodeBs time series, obtained as described in the section 5.2. Figures 5.11 presents the novel features dataset for the cells. After dropping the faulty and NaN features, we have 732 and timeseries features for both cells and eNodeBs datasets.

| | USER_COUNT__sum_values | USER_COUNT__abs_energy | USER_COUNT__mean_abs_change | USER_COUNT__mean_change |
|---|---|---|---|---|
| **0** | 568897.0 | 1.675743e+08 | 40.458483 | 0.0 |
| **1** | 905733.0 | 4.455052e+08 | 47.795506 | 0.0 |
| **2** | 1074718.0 | 5.162960e+08 | 59.383263 | 0.0 |
| **3** | 1592069.0 | 1.310216e+09 | 69.843048 | 0.0 |
| **4** | 1420014.0 | 9.860446e+08 | 69.844350 | 0.0 |
| **...** | ... | ... | ... | ... |
| **1572** | 24718.0 | 1.861486e+06 | 4.679909 | 0.0 |
| **1573** | 30440.0 | 3.530160e+06 | 3.626180 | 0.0 |
| **1574** | 39142.0 | 1.757878e+06 | 6.086617 | 0.0 |
| **1575** | 10660.0 | 5.819740e+05 | 2.239661 | 0.0 |
| **1576** | 14205.0 | 2.289750e+05 | 3.465972 | 0.0 |

***Figure 5.11.*** *Extracted features for the cells time series*

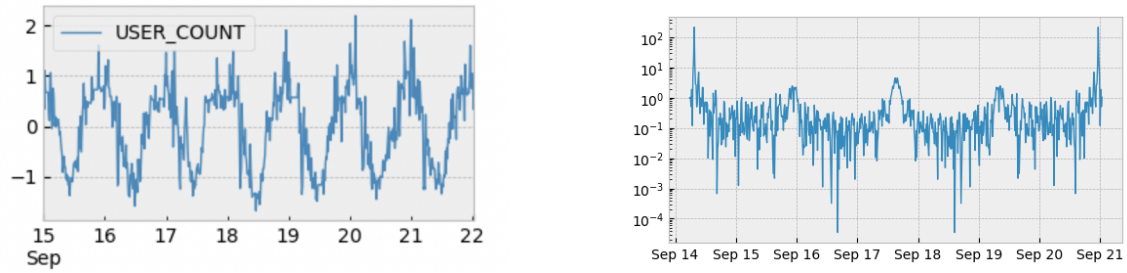
It is possible to further reduce the number of the features of the cells. The $tsfresh$ library, used for the feature extraction, also offers hypothesis testing based algorithm FRESH [40] for selecting the most relevant features for the dimentionality reduction purpose. This method is competing with PCA, that is also used for the dimentionality reduction. PCA can be useful for visualizing high-dimensional data, identifying clusters or patterns in the data, and reducing the computational complexity of subsequent analyses. In this case, FRESH algorithms is more beneficial, as it considers not only the input data vectors, but also the target data vector, that represents the target eNodeB, that each cell belongs to, which increases the accuracy and relevancy of the feature selection algorithm. After running the feature selection algorithm, 375 featurs were identified as relevant. These features were saved as cells features in the graph, and further used for the adjacency recovery.

### 5.4.2 Fourier transform features extraction

As it has was determined that the data is stationary, it is useful to calculate the Discrete Fourier Transform (DFT) for the KPIs timeseries. DFT analyzes discrete timeseries signals by transforming them from the time domain to the frequency domain. DFT is commonly used in the analysis of time series data to identify dominant frequencies and filter out noise from the signal. By identifying the frequencies associated with noise and dominant patterns in the signal, a set of features is created, that is further used in a classification model. DFT algorithm takes a sequence of numerical values that represent an original variable, with one value for each time step. The algorithm then calculates the amplitude, or signal strength, for each frequency in the sequence using Fourier coefficients. The output of the algorithm is a set of values that represent the strength of each frequency component in the original variable 5.5.

$$X_k = \sum_{n=0}^{N-1} x_n \cdot e^{-\frac{i2\pi}{N} kn} \tag{5.5}$$

Alongside with the DFT features, we also extract the seasonality features of the signal, creating 3 sets of features in total: DFT of the observed signal, signal trend and residual signal. An example of such features for a selected eNodeB is displayed on Figure 5.12 and Figure 5.13.




(a) (b)

***Figure 5.12.*** *Comparison between the eNodeB original KPIs (a) and eNodeB KPIs DFT transformation (b)*

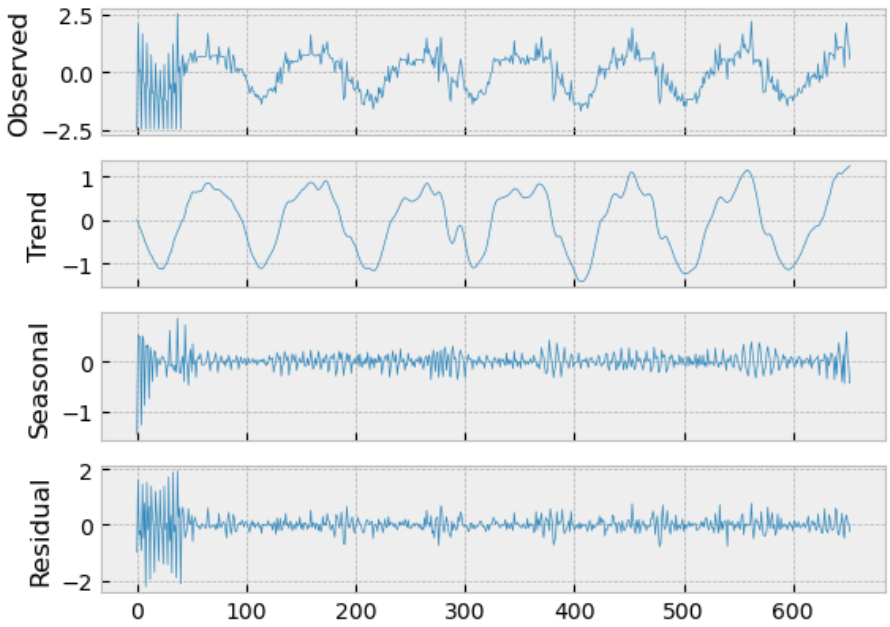



***Figure 5.13.*** *Extraction of the seasonality features of the eNodeB KPIs*

## 5.5 Link prediction

### 5.5.1 Intra-node link prediction

Intra-node adjacency, also known as the link between a cell and a eNodeB, can be derived from the timeseries features using multiple methods, depending on the dataset used. The partial link prediction problem of this study can be also formulated as node classification or node clustering, as it is typical for cells, belonging to the same eNodeB, to have the full adjacency inside the eNodeB, forming a fully connected graph. Figure 5.14 displays the ratio of the missing intra-node adjacency nodes to the full intra-node adjacency nodes.

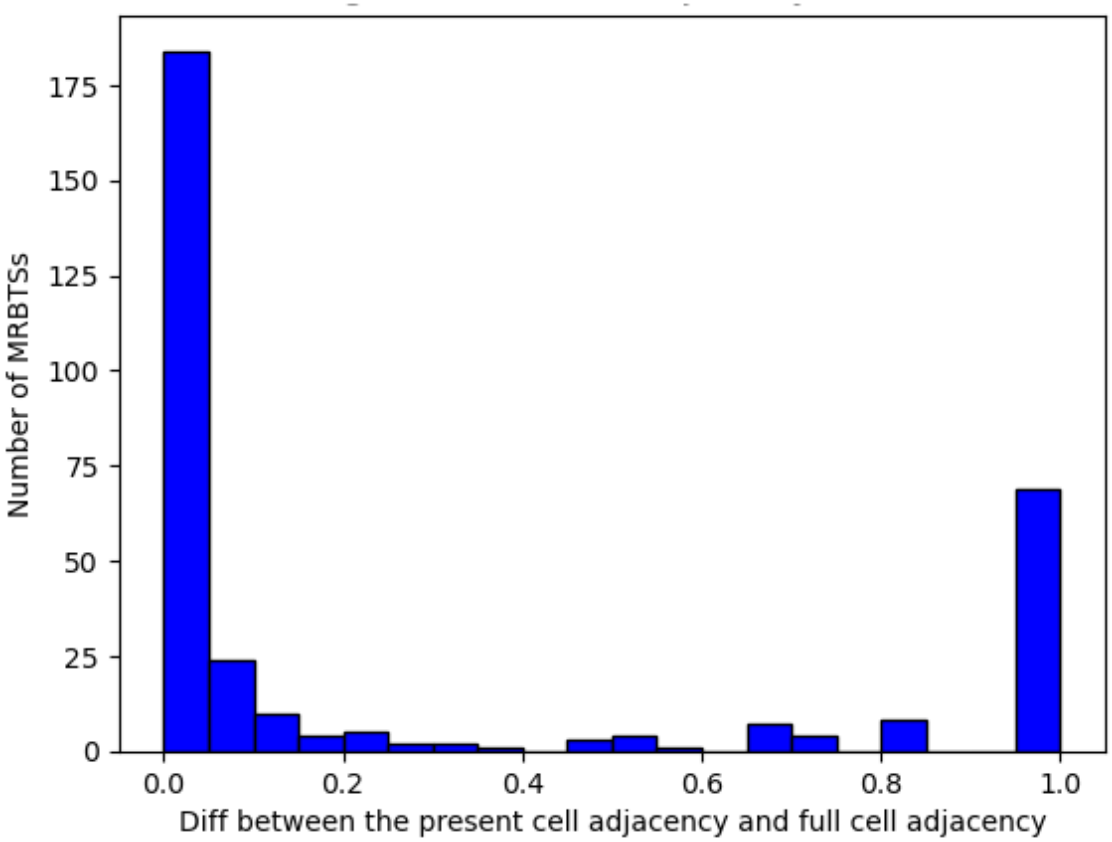



***Figure 5.14.*** *Intra-node adjacency consistency*

It can be seen, that the the data either has full connectivity, or no connectivity at all. That is explained by the fact, that some cells are lacking the adjacency data, but where it is present - it displays full intra-node connectivity, as it is supposed.

There are two main limitations to this approach. The first one is the impossibility of prediction of the exact number of clusters on the data, that is lacking adjacency. The best estimate can be achieved determining the average number of cells in a eNodeB:

$$\text{Number of clusters} = \frac{\text{Number of cells}}{\text{Number of eNodeBs}} \tag{5.6}$$

The seconds limitation is directly derived from the first limitation: the number of classes (clusters) is increasing linearly with the number of data samples (cells and eNodeBs), present in the networks. The classic approaches include Pearson correlation and K-means clustering with Euclidean distance metrics. Pearson correlation is a commonly used similarity metric for clustering time series data. However, when the number of time series becomes large, computing pairwise correlations can be computationally expensive and time-consuming. Moreover, Pearson correlation assumes that the time series are linear and stationary, which may not always hold true in practice. Therefore, it may not be the most efficient or effective similarity metric for clustering large collections of time series

data.

K-means with DBA metrics is a promising alternative for clustering time series data. DBA (Dynamic Time Warping Barycenter Averaging) is a technique for aligning and averaging time series that can handle non-linear and non-stationary data. By using DBA metrics as a similarity measure, k-means can effectively cluster large collections of time series data without the computational overhead of pairwise correlations. This approach has been demonstrated to be effective in clustering time series data in several studies [41].

Hence, with the limitations described we proceed with the K-Means clustering with DBA metrics. First, a selected sample of data, including 20 cells and 4 clusters has been tested with the different metrics of the K-means algorithm. Example is presented on the Figure 5.15

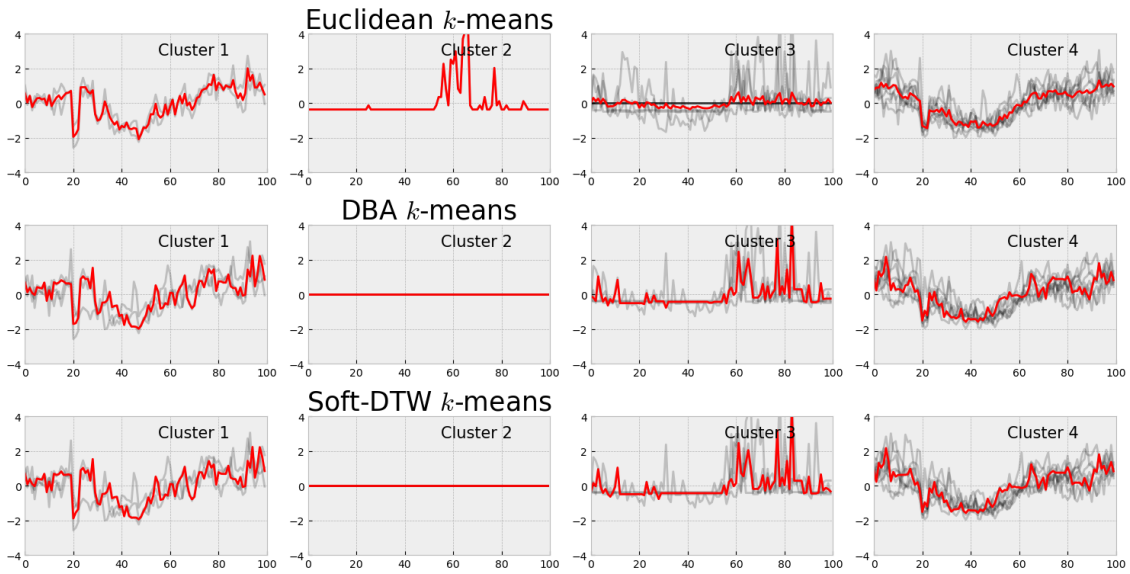



***Figure 5.15.*** *Comparison of the cells signals clustering methods*

However, clustering the full dataset with 1577 cells takes significant time. The clusterization results are also affected by the increasing number of classes. The results are analyzed further in the Chapter 6.

GNNs are another promising alternative for clustering time series data. By leveraging the graph structure of the time series data, GNNs can learn meaningful representations of the time series that capture their underlying patterns and relationships [42] [43]. There are two distinct approaches of utulizing GNNs in this work: intra-node adjacency recovery via node classification and inter-node adjacency recovery via link prediction. The latter approach will be discussed in the section below. The task of node classification can be viewed as a semi-supervised problem, whereby a graph neural network can make accurate predictions about the category of nodes with only a limited number of labeled nodes [36].

The proposed approach involves the use of a two-layer Graph Convolutional Network (GCN) which is capable of computing new node representations by aggregating information from neighboring nodes [44].

To enable the training and testing of the model, the network is partitioned into training, validation, and test splits. This is done by allotting each node a mask or label such as $trainmask$, $valmask$, or $testmask$, which serve as Boolean tensors to indicate the inclusion of nodes in their respective sets. Additionally, each node is assigned a ground truth category label indicating the eNodeB, it is connected to, and node features are extracted as feature tensor.

This approach serves as an alternative pre-processing step that can improve the further Graph Neural Network driven link prediction.

### 5.5.2 Inter-cell link prediction

#### 5.5.2.1 Problem formulation

The main objective of this study is to predict the presence of X2 interface relationships between two cells in an LTE network. To achieve this, the problem is formulated as a binary classification task. Specifically, the edges in the network graph are treated as positive examples, while a set of non-existent edges (i.e., node pairs with no edges connecting them) are sampled to serve as negative examples. The positive and negative examples are then divided into two distinct sets, one for training the model and the other for testing the model.

For this task the GraphSAGE model has been selected, and the original adjacency dataset has been converted into a graph, with time or frequency domain feature vectors used as original nodes attributes. This way, the link prediction problem is formulated as a binary classification task, which simplifies the approach to the originally multidimentional scope of the task.

#### 5.5.2.2 Preparing training and testing sets

We start by creating a graph, using the original adjacency for the cells that have known KPIs. Each node is assigned a feature vector as a node attribute. In the current experiment the following features that were extracted before were selected and compared: observed signal Discrete Fourier Transform, observed signal residual, timeseries data features vector.

The output below presents the created graph's parameters.

```
Graph(num_nodes=1533, num_edges=83506,
```

```
ndata_schemes={'feats': Scheme(shape=(1210,),
dtype=torch.float32)}
edata_schemes={})
```

It can be noticed, that the number of edges is twice as large as stated in the original graph. That happens due to the conversion between the undirected and the directed graph, where both incoming and outgoing connections are created. Then, in order to create datasets for training and testing, we first treat existing edges as positive samples and select 10% (8350 edges) of them for the testing dataset, leaving the remaining 90% (75156 edges) for the training dataset. Secondly, we create a set of the negative samples by generating non-existing in the original graph edges. We will create the same number of negative samples, as the number of the positive samples, by creating non-existent edges, randomly connecting two cells, that are not connected in the original adjacency. In the same way, we will split the negative samples into the training and testing dataset in the 90% and 10% ratio respectively. Each node preserves it's feature vector of shape (1210,).

#### 5.5.2.3 Defining the GNN model

As described in the Methodology chapter, GraphSAGE model is calculating the representations of chosen pairs of nodes, applying these representations as novel edge features.

The link prediction task involves a positive graph, comprising all positive examples as edges, and a negative graph, encompassing all negative examples. Both the positive and negative graphs share the same set of nodes as the original graph, thereby facilitating the transfer of node features across multiple graphs for computational purposes. As demonstrated in subsequent sections, the node representations computed for the entire graph can be directly applied to the positive and negative graphs, enabling the calculation of pairwise scores.

In this study, we are creating our custom model, using the pre-defined GraphSAGE layers.

The model class has the following structure:

- $\_\_init\_\_(self, in\_feats, h\_feats)$: Initializes the GraphSAGE model with input feature size in_feats and hidden feature size $h\_feats$. It defines two layers of $SAGEConv$ with mean aggregation. The first layer transforms input features to hidden features, while the second layer transforms hidden features to output features, also with hidden size $h\_feats$.
- $forward(self, g, in\_feat)$: This method is responsible for the forward pass of the model. Given a graph $g$ and its input features $in\_feat$, the method performs the following steps:
    1. Apply the first GraphSAGE layer ($self.conv1$) to the graph $g$ with input fea-

tures $in\_feat$.

2. Apply the ReLU activation function to the output of the first GraphSAGE layer.
3. Apply the second GraphSAGE layer ($self.conv2$) to the graph $g$ with the output of the ReLU activation function as input.
4. The forward method returns the final node embeddings after processing through the two-layer GraphSAGE model.

The code below describes the model parameters.

```
    ==========================================================================
Layer (type:depth-idx)                   Param #
==========================================================================
GraphSAGE                                  --
SAGEConv: 1-1                             --
    Dropout: 2-1                         --
    Linear: 2-2                          19,360
    Linear: 2-3                          19,376
SAGEConv: 1-2                             --
    Dropout: 2-4                         --
    Linear: 2-5                          256
    Linear: 2-6                          272
==========================================================================
Total params: 39,264
Trainable params: 39,264
Non-trainable params: 0
==========================================================================
```

#### 5.5.2.4 Calculating the edge score

The model estimates the likelihood of an edge's existence by calculating a score between the representations of the two incident nodes using the Multi-Layer Perceptron (MLP) function. We define a custom $\mathrm{MLPPredictor}$ class, which is a PyTorch implementation of a two-layer MLP predictor with the following structure:

1. An input layer that takes the concatenated features of the source and destination nodes (size $h\_feats$ * 2) and applies a linear transformation.
2. A ReLU activation function applied to the output of the input layer.
3. A final linear layer that maps the output of the ReLU activation to a scalar score.

This approach provides a quantitative measure for determining potential connections within the graph structure.

#### 5.5.2.5 Loss function

The loss function typically suggested for binary classification problems is binary cross entropy loss 5.7.

$$\mathcal{L} = -\sum_{u\sim v\in\mathcal{D}} (y_{u\sim v}\log(\hat{y}_{u\sim v}) + (1 - y_{u\sim v})\log(1 - \hat{y}_{u\sim v}))) \tag{5.7}$$

We compute the loss by predicting the scores separately for the negative and positive samples, generating labels for them (0 or 1), concatenating positive and negative scores, generated with MLP Predictor, as well as positive and negative labels and passing both scores and labels vectors to the binary cross entropy loss function.

In order to update the model in response to the output of the loss function, we use the Adaptive Moment Estimation optimizer (Adam) with 0.01 learning rate.

#### 5.5.2.6 Training loop

As GraphSAGE layer only selects a limited number of adjacent nodes into the batch, when calculating the novel node representations, it is possible to increase the number of training epochs, without facing the overfitting issue.

The training process consists of the following steps, executed for 3000 epochs:

1. Forward pass: Compute the node embeddings for the training graph ($\mathrm{train\_g}$) using the GNN model. The input node features are extracted from the training graph's node data ($\mathrm{train\_g.ndata['feats']}$). Then, the positive and negative scores are calculated using a prediction function (MLP) on the positive ($\mathrm{train\_pos\_g}$) and negative ($\mathrm{train\_neg\_g}$) graphs, respectively, with the obtained node embeddings ($\mathrm{h}$) as input.
2. Loss computation: Calculate the loss for the positive and negative scores using the $\mathrm{compute\_loss}$ function. This function measures the discrepancy between the predicted scores and the actual target values, guiding the model to improve its predictions.
3. Backward pass: Perform the backward propagation process to update the model's parameters. First, reset the gradients by calling $\mathrm{optimizer.zero\_grad()}$. Then, compute the gradients for the loss by invoking $\mathrm{loss.backward()}$. Finally, update the model's parameters with the calculated gradients using the $\mathrm{optimizer.step()}$ method.

By iterating through these steps for 3000 epochs, the GNN model is trained to make accurate link predictions for the given graph.

The results of the link prediction are discussed further in the Discussion chapter.

### 5.5.3 Inter-node link prediction

Similarly to the link prediction between the cells, the link prediction between eNodeBs takes a node feature vector as an input and applies it to the graph model.

This method of recovering the adjacency combines the two methods, described above. First, the inter-node adjacency is recovered, based on the cell classification and/or clusterization. Then, using the DBA algorithm a new time series is generated for each node, from which the time and frequency domain features are extracted and can be further used in the same algorithm, applied for the inter-cell adjacency recovery problem. It can be beneficial to recover inter-node adjacency, as in X2 interface cells typically have full adjacency with other cells in the node, producing a fully connected graph inside the eNodeB.

This way, we significantly reduce the number of the data points and remarkably decrease the algorithm execution time.

## 5.6 Software

In order to load, store and process the data, implement the different algorithms and compare their efficiency, a set of software tools were used. The project is implemented in the Python programming language using, among the others, the following libraries:

- NumPy is the fundamental package for scientific computing, which allows for the manipulation of data matrices and mathematical operations such as sorting, reshaping, averaging, and calculating percentiles.
- SciPy provides advanced statistical and mathematical functions and can be used for computing the Pearson correlation coefficient, among other things.
- Pandas offers data structures and analysis tools, and its use in the project involves extracting data and removing inadequate items.
- Scikit-learn encompasses a range of machine learning features.
- DGL is a Python package for deep learning on graphs, that provides layers for the model.
- Tsfresh provides time series feature extraction on basis of scalable hypothesis tests.
- Tslearn is a machine learning toolkit for time series data.
- NetworkX is a multiapplicational tool for creating, storing and displaying the graphs.
- PyTorch is a versatile library for building custom deep learning models with GPU-accelerated tensor computations.

The exact versions of the packages, listed above, as well the supporting packages listing can be found in the Appendix B, that displays the full Conda environment. This package listing can be further imported in the Conda python package manager, which allows replicating the original development environment of this study.

# 6. DISCUSSION

In this section we discuss the results of the experiments, described in the Experiments chapter 5. In order to correctly evaluate the performance of the models, we compare the combinations of the different feature extraction and clustering methods, their influence on the further link prediction algorithm efficiency on synthetic, training and testing datasets.

## 6.1 Performance metrics

There are multiple algorithms, used in the pre-processing steps accuracy evaluation, however, as this study is focusing on the link prediction problem, the baseline metrics for evaluating has been selected to be Area Under the Curve (AUC), that is the most suitable for the binary classification tasks 6.1. It is a convenient metrics for evaluating the performance of a model, classifying the edges in a networks as positive (present) or negative (missing) samples.

$$AUC(f) = \frac{\sum_{t_0 \in \mathcal{D}^0} \sum_{t_1 \in \mathcal{D}^1} \mathbf{1}[f(t_0) < f(t_1)]}{|\mathcal{D}^0| \cdot |\mathcal{D}^1|} \tag{6.1}$$

where, $[f(t_0) < f(t_1)]$ denotes an "indicator function" which returns 1 if $f(t_0) < f(t_1)$ otherwise returns 0; $\mathcal{D}^0$ is the set of negative examples, and $\mathcal{D}^1$ is the set of positive examples.

We will be also paying attention to the average precision, as the negative samples are synthetic and there is a chance that randomly chosen negative samples are much easier to correctly identify, than the positive samples. The average precision in a binary classification problem is calculated as follows:

$$AP = \frac{1}{N_{pos}} \sum_{k=1}^{n} P(k) \cdot rel(k) \tag{6.2}$$

where $N_{pos}$ is the number of positive samples, $n$ is the total number of samples, $P(k)$ is the precision at cut-off $k$, and $rel(k)$ is an indicator function that returns 1 if the sample at position $k$ is positive, and 0 otherwise.

As we have created the exact number of negative edge samples, as the number of positive

edge samples in the graph, the baseline AUC, as well as baseline AP is 0.5.

## 6.2 Pre-processing methods selection

In the Experiments section, we have set up the following pre-processing methods: data normalization with mean-variance scaling, DBA-based aggregation for the training dataset, time domain features extraction, frequency domain (DFT) features extraction, seasonal features extraction, combined with DFT, clusterization of the cells using DTW distance K-means clustering, Pearson coefficient and clusterization of the cells using GNN model based classifier.

Firstly, we have selected the data normalization with mean-variance scaling to be applied to all the timeseries data, as it was shown that cells, belonging to the same eNodeB display identical signal shapes, while having significantly varying absolute signal strength, compared to one another.

Secondly, we have tested the clustering methods on the cells, in order to study, whether it is possible to identify the intra-node adjacency. Pearson coefficient based method has been discarded, as the test algorithm run has shown poor accuracy, alongside with very high computational resources demand for calculating the pairwise coefficient for 1577 cells. Both more advanced methods, DTW distance K-means clustering and GNN model based classifier (with DFT and seasonality vectors as node features) have shown poor results: 0.01 and 0.34 validation accuracy respectively for 1577 cells, associated with 322 eNodeBs (clusters or classes in the target label). That is easily explained by the exceptionally high number of clusters, low average number of cells in a cluster and increasing number of new clusters with introducing new cells in the validation dataset. Hence, these methods were discarded and not used for the further link prediction.

Thirdly, we have aggregated the normalized cells signals in the training dataset using DBA algorithm, producing an additional training and validation dataset with lower chance of having identical signal in the training and validation steps.

Finally, we have selected the following methods for the signal features extraction: timeseries features extraction and hypothesis testing based feature selection, as this method provides descriptive features for each node in the time domain and seasonal-trend decomposition trend, residual and denoised observed signals, with DFT applied. The GNN model selected for the final solution is build on top of GraphSAGE layers, described in the Experiments section.

## 6.3 Model performance

We evaluate the performance of our model using various feature extraction methods for node representation, used as a pre-processing step, and compare the model performance on the cell dataset and pre-clustered eNodeB dataset. The Table 6.1 below provides a comprehensive comparison of these methods in terms of node feature vector shape, inter-cell adjacency recovery, and inter-eNodeB adjacency recovery. The performance metrics used for this comparison are Area Under the Receiver Operating Characteristic Curve (AUC) and Average Precision (AP).

Comparing inter-cell adjacency recovery to the inter-eNodeB adjacency recovery allows assessing the model's performance more accurately, as cells can display very similar performance metrics, while belonging to the same eNodeB, which can artificially increase the model's link prediction accuracy. On the other hand, eNodeB network displays unique eNodeBs metrics, composed of the cells metrics using DBA, hence eliminating the chance of similar features appearing both in the training and the testing datasets.

| | **Feature extraction method** | **Node feature vector shape** | **Inter-cell link prediction** | | **Inter-node link prediction** | |
|---|---|---|---|---|---|---|
| | | | **AUC** | **AP** | **AUC** | **AP** |
| 1 | Timeseries features extraction and hypothesis testing based feature selection | (732, ) | 0.46 | 0.50 | 0.54 | 0.51 |
| 2 | Seasonal-trend decomposition trend signal DFT | (1210, ) | 0.96 | 0.86 | 0.95 | 0.84 |
| 3 | Seasonal-trend decomposition residual signal DFT | (1210, ) | 0.50 | 0.50 | 0.50 | 0.50 |
| 4 | Seasonal-trend decomposition observed signal DFT | (1210, ) | 0.97 | 0.90 | 0.96 | 0.88 |

***Table 6.1.*** *Comparison of inter-cell and inter-eNodeB adjacency recovery performance for different feature extraction methods*

## 6.4 Model comparison

Table 6.1 presents the results of using four distinct feature extraction methods, combined with a Graph Neural Network model, assessed using the Area Under the Curve (AUC)

and Average Precision (AP) metrics. Notably, the baseline accuracy for both adjacency recovery types is 0.5, as the number of positive and negative samples is equal.

Comparing inter-cell adjacency recovery to inter-eNodeB adjacency recovery enables a more accurate assessment of the model's performance. As cells may display similar performance metrics while belonging to the same eNodeB, this can artificially inflate the model's link prediction accuracy. However, eNodeB networks exhibit unique metrics, generated from the cells' metrics using the DBA method, thus reducing the likelihood of similar features appearing in both the training and testing datasets.

The first method, which applies timeseries features extraction and hypothesis testing-based feature selection, produces a feature vector shape of (732,). The results show relatively poor performance in inter-cell adjacency recovery, with an AUC of 0.46 and an AP of 0.50, falling below the baseline accuracy. The method demonstrates slightly improved performance in inter-eNodeB adjacency recovery, attaining an AUC of 0.54 and an AP of 0.51, just above the baseline.

The second method employs seasonal-trend decomposition derived trend signal DFT, providing a feature vector shape of (1210,). This method exhibits superior performance in both inter-cell and inter-eNodeB adjacency recovery, achieving AUC values of 0.96 and 0.95, respectively, and AP values of 0.86 and 0.84, respectively. These results significantly surpass the baseline accuracy.

The third method, which utilizes seasonal-trend decomposition and residual signal DFT, also generates a feature vector shape of (1210,). However, the results reveal that this method merely attains the baseline accuracy of 0.5 for both AUC and AP in inter-cell and inter-eNodeB adjacency recovery.

Lastly, the fourth method, involving seasonal-trend decomposition derived observed signal DFT, again generates a feature vector shape of (1210,). This method demonstrates exceptional performance in both inter-cell and inter-eNodeB adjacency recovery, with AUC values of 0.97 and 0.96, respectively, and AP values of 0.90 and 0.88, respectively, markedly outperforming the baseline accuracy.

When comparing the trend signal DFT and the observed signal DFT methods, it is essential to focus on the AP metric, as it is a more reliable indicator of performance in predicting positive samples, while the AUC metric measures the overall performance of the classifier across all classification thresholds, quantifying the model's ability to differentiate between positive and negative samples. The AP value for the trend signal DFT method is 0.84, while the observed signal DFT method yields AP value of 0.88 for inter-eNodeB link prediction. The higher AP values for the observed signal DFT method indicate its superiority in predicting positive samples, making it the preferable choice for both inter-cell and inter-eNodeB adjacency recovery.

In light of the emphasis on predicting positive samples, the observed signal DFT method, which employs seasonal-trend decomposition, is the superior choice for inter-cell and inter-eNodeB adjacency recovery, as it demonstrates higher AP values compared to the trend signal DFT method. Consequently, future research and application development should prioritize the utilization of the observed signal DFT method for wireless communication network link prediction.

## 6.5 Performance on the synthetic dataset

To additionally assess the model's effectiveness, we created a synthetic network featuring KPIs that mimic the patterns observed in the original network. The Figure 6.1 below depicts this example, and an interactive demonstration can be accessed at blazhko.tech/#/link-prediction.

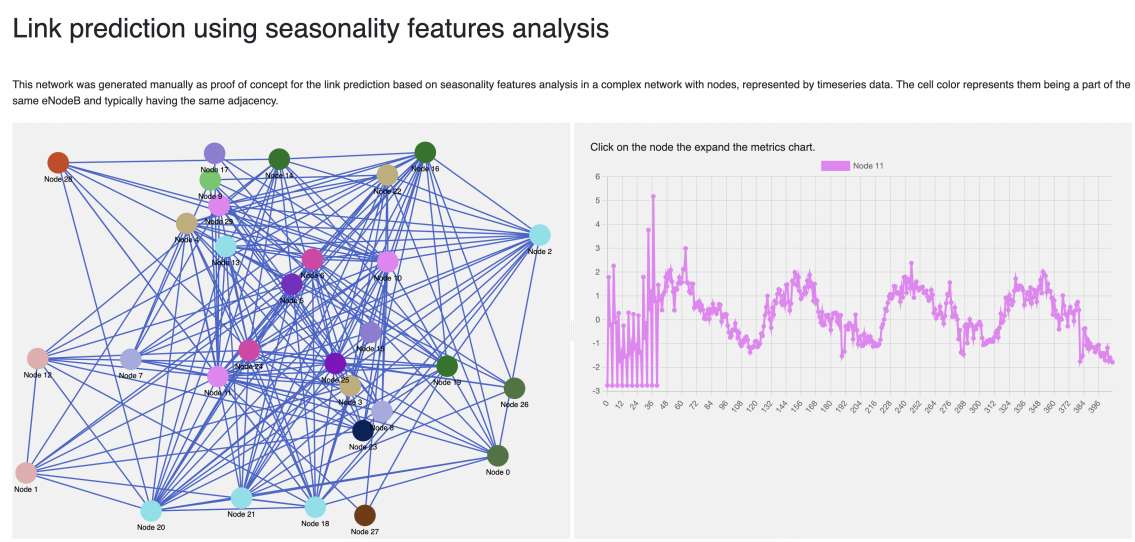



***Figure 6.1.*** *Example of link prediction on synthetic data*

## 6.6 Selected model performance

The training loss plot, presented on the Figure 6.2 showcases the model's improvement in reducing loss over a series of 2500 epochs, trained with the observed signal DFT features. Starting with a relatively high loss value, the model quickly learns and demonstrates a rapid decrease in loss during the initial epochs, but the end of the training, the loss settles at a significantly lower value, reflecting the model's effective learning and convergence to an optimal solution. It should be also noticed, that the high number of learning epochs is justified by the nature of the GraphSAGE model: as it is based on selecting only a small batch of node's neighbours for creating the node embeddings on each iteration, and further selectively walking through the network, the model overfitting is not happening: the learning process might be slower, but significantly less demanding of the computational

resources.

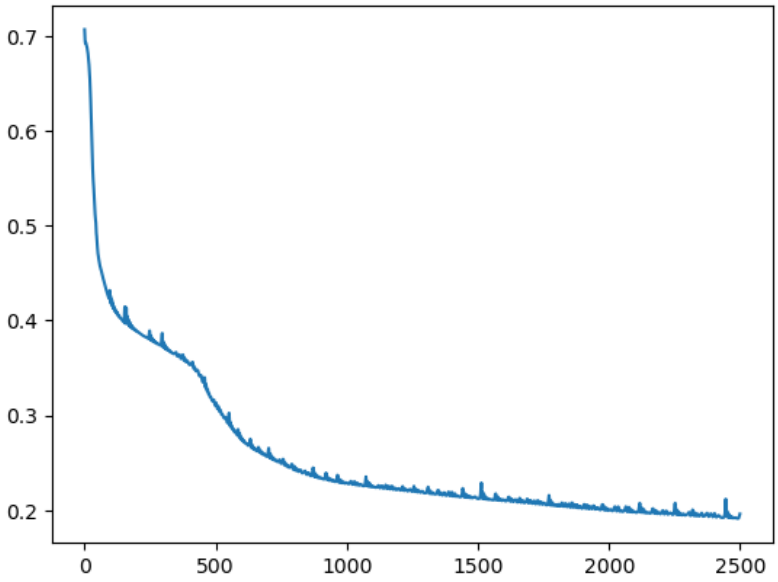



***Figure 6.2.*** *Loss curve for the model training with the observed signal DFT features*

To give more insight into the model's performance, Table 6.2 presents the confusion matrix of the model's prediction for 1692 test edges, out of which 846 edges are positive and 846 edges are negative.

| Total number of validation edges = 1692 | **Actual Positive** | **Actual Negative** |
|---|---|---|
| **Predicted Positive** | 762 | 85 |
| **Predicted Negative** | 84 | 761 |

***Table 6.2.*** *Confusion matrix of the model's predictions on the test dataset*

It can be seen, that as the false edges are synthetically generated, it the model's performance in detecting the negative samples is slightly better, than it's performance at detecting the positive samples correctly.

### 6.6.1 Computational efficiency

Pre-processing and link prediction tasks are both highly computationally intensive. However, data scaling as a pre-processing step has significantly accelerated such steps, as timeseries feature extraction, feature selection and seasonality features calculation. The efficiency of machine learning models is also dependent upon the input data, and these models are generally characterized by considerable computational demands. Another computationally heavy step of the study is timeseries data clustering. In the following

list, we present the tested clustering algorithms in ascending order of computational resources.

1. K-Means with Euclidean Distance
2. K-Means with Pearson Distance
3. K-Means with DTW Distance
4. K-Means with DBA Distance

Overall, the timeseries processing and analyzing algorithms consume significant amount of computational resources, but can be optimized with suitable data scaling and preprocessing.

# 7. CONCLUSION AND FURTHER WORK

In this master's thesis, we have analyzed the existing approaches to link prediction problem in complex networks, applied to the X2 handover interface of the LTE networks. Our research contributes to the understanding and development of innovative techniques in the domain of link prediction using Graph Neural Networks (GNNs) by combining time series data analysis and network science.

Throughout the study, we explored different methods for creating node features, including time series feature extraction and selection using hypothesis testing, Discrete Fourier Transform (DFT) of the node metrics signal, and signal seasonality features. We conducted extensive comparisons of the algorithm performance based on clustered and aggregated data (for eNodeBs) as well as raw data (for cells). Our approach achieved an average precision accuracy of up to 0.88 and 0.90 for eNodeBs and cells respectively, demonstrating the effectiveness of the proposed method in the link prediction problem.

As with any research, there are opportunities for further exploration and improvement. Some potential directions for future research include:

- Expanding the dataset: By increasing the number of cells in the dataset, we can evaluate the robustness and scalability of our proposed method on more complex networks. This would enable us to better understand the limits and potential of the approach in real-world scenarios.
- Extending the time span of data collection: In our study, we only utilized roughly two weeks of data. By increasing the time span of data collection, we can capture more dynamic changes in the network and potentially improve the accuracy of our link prediction algorithm.
- Deploying the proposed links in real-world test networks: A key step towards validating our approach is to deploy the algorithm-generated links in a real-world test network and measure its performance. This would allow us to evaluate the practicality and effectiveness of our method in an operational environment.

The original hypothesis of this thesis suggested that analyzing time series KPI metrics data from LTE network cells could reveal a connection between cell performance indicators and the formation of X2 interface links. The results of this study demonstrated that by utilizing the Discrete Fourier Transform (DFT) of the observed metrics signal of the

node, a final precision accuracy of 0.88 and 0.90 for eNodeBs and cells respectively was achieved. This confirms both the research question and hypothesis, indicating that accurate link prediction in X2 interface architecture is possible through the analysis of time series KPI metrics data.

In conclusion, this master's thesis provides a contribution to the field of link prediction in complex networks by introducing a novel approach that combines GNNs, time series data analysis, seasonality features extraction in the graph structured data and network science. Our findings offer a strong foundation for further research and potential real-world applications in the ever-evolving domain of LTE networks and beyond.

# REFERENCES


[1] Holma, H. and Toskala, A. *LTE for UMTS : evolution to LTE-Advanced*. 2nd ed. John Wiley Sons, 2011. ISBN: 978-0-470-66000-3.

[2] Daengsi, T., Ungkap, P. and Wuttidittachotti, P. A Study of 5G Network Performance: A Pilot Field Trial at the Main Skytrain Stations in Bangkok. *ICAICST 2021 - 2021 International Conference on Artificial Intelligence and Computer Science Technology* (June 2021), pp. 191–195. DOI: 10.1109/ICAICST53116.2021.9497810.

[3] Jarich, T. H. P. *Global Mobile Trends 2020. New decade, new industry?* GSMA Intelligence, 2019.

[4] Community, H. E. S. *The difference between the inter-eNodeB handover and the intra-eNodeB handover*. URL: https://forum.huawei.com/enterprise/en/the-difference-between-the-inter-enodeb-handover-and-the-intra-enodeb-handover/thread/573471-100305.

[5] Huurdeman, A. A. *The Worldwide History of Telecommunications*. John Wiley Sons, 2003, p. 529. URL: https://books.google.fi/books?hl=sv&id=SnjGRDVIUL4C&dq=GSM+1992&q=GSM+first&redir_esc=y#v=snippet&q=GSM%201992%20finnish&f=false.

[6] Ofcom. *Ofcom research shows 4G significantly outperforms 3G networks*. URL: https://www.ofcom.org.uk/about-ofcom/latest/media/media-releases/2015/4g-outperforms-3g.

[7] Trends, D. *5G vs. 4G: Key Differences Between Networks Explained*. URL: https://www.digitaltrends.com/mobile/5g-vs-4g/.

[8] Renard, B., Elayoubi, S. E. and Simonian, A. A dimensioning method for the LTE X 2 interface. *IEEE Wireless Communications and Networking Conference, WCNC* (2012), pp. 2718–2723. DOI: 10.1109/WCNC.2012.6214261.

[9] Donovan, J. *Cells, Sectors and Antenna Beamforming | CommScope*. Jan. 2017. URL: https://www.commscope.com/blog/2014/cells-sectors-and-antenna-beamforming/.

[10] *Chapter 2 – Network Science by Albert-László Barabási*. URL: http://networksciencebook.com/chapter/2#degree.

[11] Barabási, A.-L. *Network Science*. 2016. URL: http://networksciencebook.com/chapter/1#n.

[12] Albert, R. and Barabasi, A.-L. Statistical mechanics of complex networks. *Reviews of Modern Physics* 74 (1 June 2001), pp. 47–97. DOI: 10.1103/revmodphys.74.47. URL: https://arxiv.org/abs/cond-mat/0106096v1.

[13] Erdős, P. and Rényi, A. On the strength of connectedness of a random graph. *Acta Mathematica Academiae Scientiarum Hungaricae* 12 (1-2 Mar. 1964), pp. 261–267. DOI: 10.1007/BF02066689.

[14] Barabási, A.-L. and Albert, R. Emergence of Scaling in Random Networks. *Mat. Res. Soc. Symp. Proc* 74 (1995), p. 677. URL: www.sciencemag.org.

[15] Liben-Nowell, D. and Kleinberg, J. The Link Prediction Problem for Social Networks *. (2004). URL: www.arxiv.org..

[16] Ahmad, I., Akhtar, M. U., Noor, S. and Shahnaz, A. Missing Link Prediction using Common Neighbor and Centrality based Parameterized Algorithm. *Scientific Reports 2020 10:1* 10 (1 Jan. 2020), pp. 1–9. ISSN: 2045-2322. DOI: 10.1038/s41598-019-57304-y. URL: https://www.nature.com/articles/s41598-019-57304-y.

[17] López-Pérez, D., Jüttner, A. and Zhang, J. Optimisation methods for dynamic frequency planning in OFDMA networks. *NETWORKS 2008 - 13th International Telecommunications Network Strategy and Planning Symposium* (2008). DOI: 10.1109/NETWKS.2008.4763671.

[18] Ni, W., Collings, I., Lipman, J., Wang, X., Tao, M. and Abolhasan, M. Graph theory and its applications to future network planning: Software-defined online small cell management. *IEEE Wireless Communications* 22 (1 Feb. 2015), pp. 52–60. DOI: 10.1109/MWC.2015.7054719.

[19] Ceci, E. and Barbarossa, S. Graph signal processing in the presence of topology uncertainties. *IEEE Transactions on Signal Processing* 68 (2020), pp. 1558–1573. DOI: 10.1109/TSP.2020.2976583.

[20] Shen, Y., Fu, X., Giannakis, G. B. and Sidiropoulos, N. D. Topology Identification of Directed Graphs via Joint Diagonalization of Correlation Matrices. *IEEE Transactions on Signal and Information Processing over Networks* 6 (2020), pp. 271–283. DOI: 10.1109/TSIPN.2020.2984131.

[21] Frasconi, P., Gori, M. and Sperduti, A. Hidden Recursive Models. (1998), pp. 305–311. DOI: 10.1007/978-1-4471-1520-5_32. URL: https://link.springer.com/chapter/10.1007/978-1-4471-1520-5_32.

[22] Gori, M., Monfardini, G. and Scarselli, F. A new model for earning in raph domains. *Proceedings of the International Joint Conference on Neural Networks* 2 (2005), pp. 729–734. DOI: 10.1109/IJCNN.2005.1555942.

[23] Scarselli, F., Gori, M., Tsoi, A. C., Hagenbuchner, M. and Monfardini, G. The graph neural network model. *IEEE Transactions on Neural Networks* 20 (1 Jan. 2009), pp. 61–80. DOI: 10.1109/TNN.2008.2005605.

[24] Veličković, P., Cucurull, G., Casanova, A., Romero, A., Liò, P. and Bengio, Y. Graph Attention Networks. *6th International Conference on Learning Representations, ICLR 2018 - Conference Track Proceedings* (Oct. 2017). URL: https://arxiv.org/abs/1710.10903v3.

[25] Dehmamy, N., Barabási, A.-L. and Yu, R. Understanding the Representation Power of Graph Neural Networks in Learning Graph Topology. *Advances in Neural Information Processing Systems* 32 (July 2019). URL: https://arxiv.org/abs/1907.05008v2.

[26] Cirstea, R.-G., Guo, C. and Yang, B. Graph Attention Recurrent Neural Networks for Correlated Time Series Forecasting – Full version. (Mar. 2021). URL: https://arxiv.org/abs/2103.10760v2.

[27] Hamilton, W., Ying, R. and Leskovec, J. Inductive representation learning on large graphs. *Advances in Neural Information Processing Systems*. 2017, pp. 1024–1034.

[28] Sen, P., Namata, G. M., Bilgic, M., Getoor, L., Gallagher, B. and Eliassi-Rad, T. Collective Classification in Network Data. *AI Magazine* 29.3 (2008), pp. 93–106.

[29] Marinka, Z., Jure, L. and Monica, A. *BioSNAP: Network datasets: Human protein-protein interaction network*. URL: https://snap.stanford.edu/biodata/datasets/10000/10000-PP-Pathways.html.

[30] Richard, E., Gaïffas, S. and Vayatis, N. Link Prediction in Graphs with Autoregressive Features. *Journal of Machine Learning Research* 15 (2014), pp. 565–593.

[31] Silva Soares, P. R. da and Prudencio, R. B. C. Time Series Based Link Prediction. *The 2012 International Joint Conference on Neural Networks (IJCNN)* (June 2012), pp. 1–7. DOI: 10.1109/IJCNN.2012.6252471. URL: https://www.researchgate.net/publication/237091637_Time_Series_Based_Link_Prediction.

[32] Liu, X. F. and Tse, C. K. A complex network perspective of world stock markets: Synchronization and volatility. *International Journal of Bifurcation and Chaos* 22 (6 2012). DOI: 10.1142/S0218127412501428.

[33] Senin, P. Dynamic Time Warping Algorithm Review. (Jan. 2009).

[34] Petitjean, F., Ketterlin, A. and Gançarski, P. A global averaging method for dynamic time warping, with applications to clustering. *Pattern Recognition* 44 (3 Mar. 2011), pp. 678–693. ISSN: 0031-3203. DOI: 10.1016/J.PATCOG.2010.09.013.

[35] Christ, M., Braun, N., Neuffer, J. and Kempa-Liehr, A. W. Time Series FeatuRe Extraction on basis of Scalable Hypothesis tests (tsfresh – A Python package). *Neurocomputing* 307 (2018), pp. 72–77. DOI: 10.1016/j.neucom.2018.03.067.

[36] Kipf, T. N. and Welling, M. Semi-supervised classification with graph convolutional networks. *arXiv preprint arXiv:1609.02907* (2016).

[37] Chen, Q., Zhao, H., Li, L., Long, H., Wang, J. and Hou, X. A closed-loop UL power control scheme for interference mitigation in dynamic TD-LTE systems. *IEEE Vehicular Technology Conference* (July 2015). DOI: 10.1109/VTCSPRING.2015.7145591.

[38] Kamada, T. and Kawai, S. An algorithm for drawing general undirected graphs. *Information Processing Letters* 31.1 (1989), pp. 7–15. DOI: 10.1016/0020-0190(89)90102-6.

[39] Petitjean, F., Forestier, G., Webb, G. I., Nicholson, A. E., Chen, Y. and Keogh, E. Dynamic time warping averaging of time series allows faster and more accurate classification. *Data Mining (ICDM), 2014 IEEE International Conference on*. IEEE. 2014, pp. 470–479.

[40] Christ, M., Kempa-Liehr, A. W. and Feindt, M. Distributed and parallel time series feature extraction for industrial big data applications. *ArXiv e-prints* arXiv:1610.07717 (2017). URL: https://arxiv.org/abs/1610.07717.

[41] Vidal, E. and Pinto, D. Clustering time series data with dynamic time warping-based algorithms. *Journal of Intelligent Information Systems* 53.2 (2019), pp. 237–258. DOI: 10.1007/s10844-018-0523-1.

[42] Yu, B., Yin, H. and Zhu, Z. Deep graph clustering with time series representation learning. *Neurocomputing* 443 (2021), pp. 92–101. DOI: 10.1016/j.neucom.2021.06.007.

[43] Zhang, W. and Chen, Z. Time series clustering based on graph neural networks. *Proceedings of the 2020 IEEE International Conference on Big Data*. IEEE. 2020, pp. 4597–4602. DOI: 10.1109/BigData50022.2020.9377851.

[44] Bruna, J., Zaremba, W., Szlam, A. and LeCun, Y. Spectral networks and locally connected networks on graphs. *arXiv preprint arXiv:1312.6203* (2013).

# APPENDIX A: FULL NETWORK TOPOLOGY

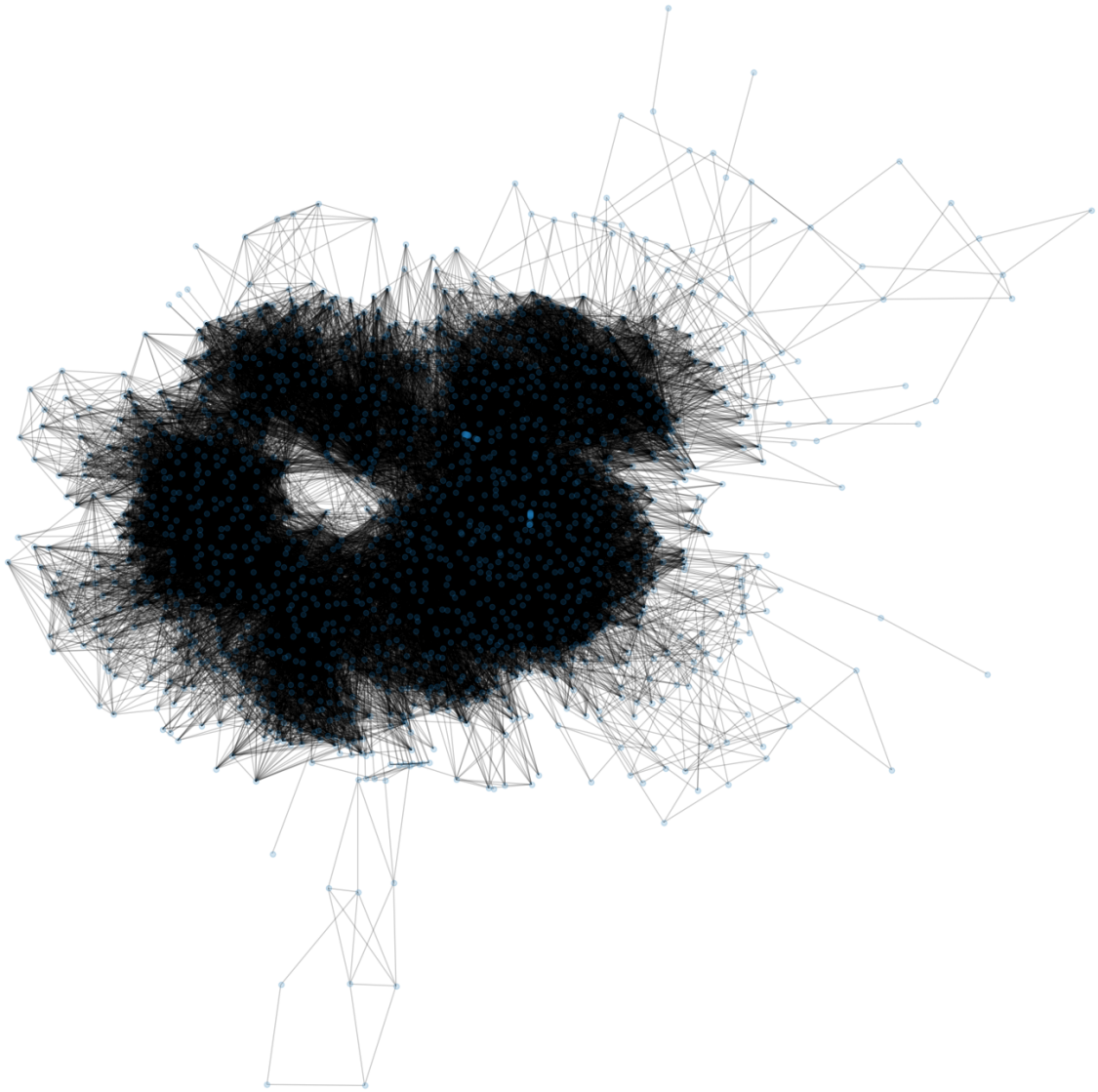


***Figure A.1.*** *Full-sized topology of the network*

# APPENDIX B: CONDA DEVELOPMENT ENVIRONMENT

```
name: myenv
channels:
  - pytorch
  - fastai
  - dglteam
  - anaconda
  - conda-forge
  - defaults
dependencies:
  - _ipython_minor_entry_point=8.7.0=h8cf3c4a_0
  - altair=4.2.2=pyhd8ed1ab_0
  - anyio=3.6.2=pyhd8ed1ab_0
  - appdirs=1.4.4=pyhd3eb1b0_0
  - appnope=0.1.3=pyhd8ed1ab_0
  - argon2-cffi=21.3.0=pyhd8ed1ab_0
  - argon2-cffi-bindings=21.2.0=py39ha30fb19_3
  - asttokens=2.2.1=pyhd8ed1ab_0
  - attrs=22.1.0=pyh71513ae_1
  - babel=2.11.0=pyhd8ed1ab_0
  - backcall=0.2.0=pyh9f0ad1d_0
  - backports=1.0=pyhd8ed1ab_3
  - backports.functools_lru_cache=1.6.4=pyhd8ed1ab_0
  - beautifulsoup4=4.11.1=pyha770c72_0
  - blas=1.0=mkl
  - bleach=5.0.1=pyhd8ed1ab_0
  - bokeh=2.4.3=pyhd8ed1ab_3
  - bottleneck=1.3.4=py39h67323c0_0
  - brotli=1.0.9=hb7f2c08_8
  - brotli-bin=1.0.9=hb7f2c08_8
  - brotlipy=0.7.0=py39ha30fb19_1005
  - c-ares=1.18.1=hca72f7f_0
  - ca-certificates=2022.12.7=h033912b_0
```

```
- catalogue=2.0.7=py39hecd8cb5_0
- certifi=2022.12.7=pyhd8ed1ab_0
- cffi=1.15.1=py39h131948b_2
- charset-normalizer=2.1.1=pyhd8ed1ab_0
- click=8.0.4=py39hecd8cb5_0
- cloudpickle=2.2.0=pyhd8ed1ab_0
- colorama=0.4.6=pyhd8ed1ab_0
- comm=0.1.2=pyhd8ed1ab_0
- contourpy=1.0.7=py39h92daf61_0
- cryptography=37.0.4=py39h9c2a9ce_0
- cycler=0.11.0=pyhd8ed1ab_0
- cymem=2.0.6=py39he9d5cce_0
- cython-blis=0.7.7=py39h67323c0_0
- cytoolz=0.12.0=py39ha30fb19_1
- dask=2023.1.0=pyhd8ed1ab_0
- dask-core=2023.1.0=pyhd8ed1ab_0
- debugpy=1.6.4=py39h7a8716b_0
- decorator=5.1.1=pyhd8ed1ab_0
- defusedxml=0.7.1=pyhd8ed1ab_0
- dgl=1.0.0=py39_0
- distributed=2023.1.0=pyhd8ed1ab_0
- entrypoints=0.4=pyhd8ed1ab_0
- executing=1.2.0=pyhd8ed1ab_0
- fastai=2.7.11=py_0
- fastcore=1.5.28=py_0
- fastdownload=0.0.7=py_0
- fastprogress=1.0.3=py_0
- fftw=3.3.9=h9ed2024_1
- flit-core=3.8.0=pyhd8ed1ab_0
- fonttools=4.38.0=py39ha30fb19_1
- freetype=2.12.1=hd8bbffd_0
- fsspec=2022.11.0=pyhd8ed1ab_0
- giflib=5.2.1=hbcb3906_2
- hdf5=1.12.1=h2b2ad87_2
- heapdict=1.0.1=py_0
- idna=3.4=pyhd8ed1ab_0
- importlib-metadata=5.1.0=pyha770c72_0
- importlib_resources=5.10.1=pyhd8ed1ab_0
- intel-openmp=2021.4.0=hecd8cb5_3538
- ipykernel=6.19.2=pyh736e0ef_0
```

```
- ipython=8.7.0=pyhd1c38e8_0
- ipython_genutils=0.2.0=py_1
- jedi=0.18.2=pyhd8ed1ab_0
- jinja2=3.1.2=pyhd8ed1ab_1
- joblib=1.2.0=pyhd8ed1ab_0
- jpeg=9e=hac89ed1_2
- json5=0.9.5=pyh9f0ad1d_0
- jsonschema=4.17.3=pyhd8ed1ab_0
- jupyter_client=7.4.8=pyhd8ed1ab_0
- jupyter_core=5.1.0=py39h6e9494a_0
- jupyter_events=0.5.0=pyhd8ed1ab_0
- jupyter_server=2.0.1=pyhd8ed1ab_0
- jupyter_server_terminals=0.4.2=pyhd8ed1ab_0
- jupyterlab=3.5.1=pyhd8ed1ab_0
- jupyterlab_pygments=0.2.2=pyhd8ed1ab_0
- jupyterlab_server=2.16.5=pyhd8ed1ab_0
- kiwisolver=1.4.4=py39h92daf61_1
- krb5=1.19.2=hcd88c3b_0
- langcodes=3.3.0=pyhd3eb1b0_0
- lcms2=2.14=h90f4b2a_0
- lerc=3.0=he9d5cce_0
- libblas=3.9.0=12_osx64_mkl
- libbrotlicommon=1.0.9=hb7f2c08_8
- libbrotlidec=1.0.9=hb7f2c08_8
- libbrotlienc=1.0.9=hb7f2c08_8
- libcblas=3.9.0=12_osx64_mkl
- libcurl=7.82.0=h6dfd666_0
- libcxx=14.0.6=h9765a3e_0
- libdeflate=1.8=h9ed2024_5
- libedit=3.1.20210910=hca72f7f_0
- libev=4.33=h9ed2024_1
- libffi=3.4.2=hecd8cb5_6
- libgfortran=5.0.0=11_3_0_h97931a8_27
- libgfortran5=11.3.0=h082f757_27
- liblapack=3.9.0=12_osx64_mkl
- libllvm11=11.1.0=hd011deb_2
- libnghttp2=1.46.0=ha29bfda_0
- libpng=1.6.37=ha441bb4_0
- libsodium=1.0.18=hbcb3906_1
- libssh2=1.10.0=h0a4fc7d_0
```

```
- libtiff=4.4.0=h2cd0358_2
- libwebp=1.2.4=hfa4350a_0
- libwebp-base=1.2.4=h775f41a_0
- llvm-openmp=15.0.7=h61d9ccf_0
- llvmlite=0.39.1=py39h8346a28_0
- locket=1.0.0=pyhd8ed1ab_0
- lz4=4.2.0=py39hd0af75a_0
- lz4-c=1.9.3=he49afe7_1
- markupsafe=2.1.1=py39ha30fb19_2
- matplotlib=3.6.2=py39h6e9494a_0
- matplotlib-base=3.6.2=py39hb2f573b_0
- matplotlib-inline=0.1.6=pyhd8ed1ab_0
- mistune=2.0.4=pyhd8ed1ab_0
- mkl=2021.4.0=hecd8cb5_637
- mkl-service=2.4.0=py39h9ed2024_0
- mkl_fft=1.3.1=py39h4ab4a9b_0
- mkl_random=1.2.2=py39hb2f4e1b_0
- msgpack-python=1.0.4=py39h92daf61_1
- munkres=1.1.4=pyh9f0ad1d_0
- murmurhash=1.0.7=py39he9d5cce_0
- nbclassic=0.4.8=pyhd8ed1ab_0
- nbclient=0.7.2=pyhd8ed1ab_0
- nbconvert=7.2.6=pyhd8ed1ab_0
- nbconvert-core=7.2.6=pyhd8ed1ab_0
- nbconvert-pandoc=7.2.6=pyhd8ed1ab_0
- nbformat=5.7.0=pyhd8ed1ab_0
- ncurses=6.3=hca72f7f_3
- nest-asyncio=1.5.6=pyhd8ed1ab_0
- networkx=2.7.1=pyhd3eb1b0_0
- notebook=6.4.11=py39hecd8cb5_0
- notebook-shim=0.2.2=pyhd8ed1ab_0
- numba=0.56.4=py39hf6b9d82_0
- numexpr=2.8.1=py39h2e5f0a9_2
- numpy=1.22.3=py39h2e5f0a9_0
- numpy-base=1.22.3=py39h3b1a694_0
- openssl=1.1.1t=hfd90126_0
- packaging=22.0=pyhd8ed1ab_0
- pandas=1.4.2=py39he9d5cce_0
- pandoc=2.19.2=h694c41f_1
- pandocfilters=1.5.0=pyhd8ed1ab_0
```

```
- parso=0.8.3=pyhd8ed1ab_0
- partd=1.3.0=pyhd8ed1ab_0
- pathy=0.10.1=py39hecd8cb5_0
- patsy=0.5.3=pyhd8ed1ab_0
- pexpect=4.8.0=pyh1a96a4e_2
- pickleshare=0.7.5=py_1003
- pillow=9.3.0=py39h81888ad_1
- pip=22.3.1=py39hecd8cb5_0
- pkgutil-resolve-name=1.3.10=pyhd8ed1ab_0
- platformdirs=2.6.0=pyhd8ed1ab_0
- pooch=1.4.0=pyhd3eb1b0_0
- preshed=3.0.6=py39he9d5cce_0
- prometheus_client=0.15.0=pyhd8ed1ab_0
- prompt-toolkit=3.0.36=pyha770c72_0
- psutil=5.9.4=py39ha30fb19_0
- ptyprocess=0.7.0=pyhd3deb0d_0
- pure_eval=0.2.2=pyhd8ed1ab_0
- pycparser=2.21=pyhd8ed1ab_0
- pydantic=1.10.2=py39hca72f7f_0
- pygments=2.13.0=pyhd8ed1ab_0
- pyopenssl=22.0.0=pyhd8ed1ab_1
- pyparsing=3.0.9=pyhd8ed1ab_0
- pyrsistent=0.19.2=py39ha30fb19_0
- pysocks=1.7.1=pyha2e5f31_6
- python=3.9.15=h218abb5_2
- python-dateutil=2.8.2=pyhd8ed1ab_0
- python-fastjsonschema=2.16.2=pyhd8ed1ab_0
- python-json-logger=2.0.4=pyhd8ed1ab_0
- python_abi=3.9=2_cp39
- pytorch=1.13.1=py3.9_0
- pytz=2022.6=pyhd8ed1ab_0
- pyyaml=6.0=py39ha30fb19_5
- pyzmq=24.0.1=py39hed8f129_1
- readline=8.2=hca72f7f_0
- requests=2.28.1=pyhd8ed1ab_1
- scikit-learn=1.2.0=py39hdbdcc14_0
- scipy=1.10.0=py39h91c6ef4_0
- seaborn=0.11.2=pyhd3eb1b0_0
- send2trash=1.8.0=pyhd8ed1ab_0
- setuptools=65.5.0=py39hecd8cb5_0
```

```
- shellingham=1.5.0=py39hecd8cb5_0
- six=1.16.0=pyh6c4a22f_0
- smart_open=5.2.1=py39hecd8cb5_0
- sniffio=1.3.0=pyhd8ed1ab_0
- sortedcontainers=2.4.0=pyhd8ed1ab_0
- soupsieve=2.3.2.post1=pyhd8ed1ab_0
- spacy=3.3.1=py39hc29d2bd_0
- spacy-legacy=3.0.9=py39hecd8cb5_0
- spacy-loggers=1.0.1=pyhd3eb1b0_0
- sqlite=3.40.0=h880c91c_0
- srsly=2.4.6=py39hcec6c5f_0
- stack_data=0.6.2=pyhd8ed1ab_0
- statsmodels=0.13.5=py39hacda100_1
- stumpy=1.11.1=pyhd8ed1ab_0
- tabulate=0.9.0=pyhd8ed1ab_1
- tblib=1.7.0=pyhd8ed1ab_0
- terminado=0.17.1=pyhd1c38e8_0
- thinc=8.0.15=py39hc29d2bd_0
- threadpoolctl=3.1.0=pyh8a188c0_0
- tinycss2=1.2.1=pyhd8ed1ab_0
- tk=8.6.12=h5d9f67b_0
- tomli=2.0.1=pyhd8ed1ab_0
- toolz=0.12.0=pyhd8ed1ab_0
- torchvision=0.13.1=cpu_py39hc47236b_0
- tornado=6.2=py39ha30fb19_1
- tqdm=4.64.1=pyhd8ed1ab_0
- traitlets=5.7.1=pyhd8ed1ab_0
- tsfresh=0.20.0=pyhd8ed1ab_1
- tslearn=0.5.3.2=py39h7cc1f47_0
- typer=0.4.1=py39hecd8cb5_0
- typing-extensions=4.4.0=py39hecd8cb5_0
- typing_extensions=4.4.0=py39hecd8cb5_0
- tzdata=2022g=h04d1e81_0
- unicodedata2=15.0.0=py39ha30fb19_0
- urllib3=1.26.13=pyhd8ed1ab_0
- wasabi=0.9.1=py39hecd8cb5_0
- watermark=2.3.1=pyhd8ed1ab_1
- wcwidth=0.2.5=pyh9f0ad1d_2
- webencodings=0.5.1=py_1
- websocket-client=1.4.2=pyhd8ed1ab_0
```

```
  - wheel=0.37.1=pyhd3eb1b0_0
  - xz=5.2.8=h6c40b1e_0
  - yaml=0.2.5=h0d85af4_2
  - zeromq=4.3.4=he49afe7_1
  - zict=2.2.0=pyhd8ed1ab_0
  - zipp=3.11.0=pyhd8ed1ab_0
  - zlib=1.2.13=h4dc903c_0
  - zstd=1.5.2=hcb37349_0
prefix: /Users/veronikablazhko/opt/anaconda3/envs/myenv
```