Imperial College London

# Solving Linear Equations using Graph Neural Networks

Author
Wenbo Duan
CID: 02255853

Supervised by
Dr Seyed Mohsen Moosavi-Dezfooli

A Thesis submitted in fulfillment of requirements for the degree of
**Master of Science in Applied Machine Learning**

Department of Electrical and Electronic Engineering
Imperial College London
2023

# Abstract

Motivated by the recent advancement in AI for Science, in this project, we studied new approaches towards a longstanding issue in the field of Applied Mathematics, known as the *Minimum fill-in problem*, which is an optimization barrier that plays a critical role in boosting the efficacy of solving linear equations. The primary goal of this project is to identify an optimal permutation order that facilitates a Symmetric and Positive Definite (SPD) coefficient matrix, potentially yielding the sparsest possible factorization outcomes.

Starting from the graph theory, we first emphasized the connections between Gaussian elimination process in linear equation resolutions and node elimination processes within graph triangulation. By introducing a graphical perspective and utilizing Graph Neural Network (GNN) techniques, we established the first step as employing GNNs to foresee the optimal node elimination sequences in input graphs - a parallel process from the minimum fill-in problem to graph theory.

The subsequent progression towards the GNN model was then divided into three main phases: devising the inference pipeline, designing the dataset, and finalizing the GNN architecture choice. During each stage, we showcased comparisons of various design choices aiming to simplify the task and enhance the network's expressiveness. Addressing challenges like the noise sensitivity of node-level regression and the limitations brought by the graph's isomorphic structure, we formulated approaches that incorporated a recurrent multi-step prediction method and enhanced feature representation, leveraging tailored encoding strategies and a Graph Transformer framework.

Upon commencing training across a range of six distinct dataset sizes, our optimized model displayed encouraging outcomes, surpassing locally pre-trained models and unveiling new permutation sequences that potentially eclipse existing heuristic algorithms for graphs up to 60-node size.

# Declaration of Originality

I hereby declare that the work presented in this thesis is my own unless otherwise stated. To the best of my knowledge the work is original and ideas developed in collaboration with others have been appropriately referenced.

# Copyright Declaration

# Acknowledgments

I wish to express my deepest gratitude to my supervisor, Dr Seyed Moosavi Dezfooli, for his consistent guidance and insights throughout this journey. Beyond providing a supportive environment and invaluable resources, he imparted significant knowledge regarding research approaches and critical analysis.

I would also like to thank Dr. Alhussein Fawzi for his valuable time and suggestions. I would like to acknowledge Prof. Hamza Fawzi for his constructive advice on topic selection.

# Contents

# List of Acronyms

**AI** Artificial Intelligence

**ML** Machine Learning

**FEA** Finite Element Analysis

**SPD** Symmetric and Positive Definite

**GNN** Graph Neural Network

**CNN** Convolutional Neural Network

**MPGNN** Message Passing Graph Neural Network

**MLP** Multi-Layer Perceptrons

**GIN** Graph Isomorphism Network

**PE** Position Encoding

**SE** Structure Encoding

**GT** Graph Transformer

**DNN** Deep Neural Network

**GINE** Graph Isomorphism Network with Edges

**RW** Random Walk

# List of Figures

# 1 Introduction

Contents



## 1.1 Background

The success of contemporary machine learning achievements has facilitated its deployment in solving issues beyond the domain of ”classical AI”. Notably, it has become a powerful tool in aiding scientific investigations. The enlargement of science knowledge databases, like AlphaFold DB [1], and ProofNet[2], along with learning frameworks such as the Graph Learning [3] and Transformer [4], are significantly enhancing the logic capacities of the inference models, paving a potential pathway to discovering new patterns from the ’Disasters of dimensionality’, and making the inspiring outcomes spanning the areas from solving Partial Differential Equations [5], predicting molecule property [1] to recognizing Higgs boson [6] in Physics.

In this project, we focus on the integration of Artificial Intelligence (AI) and Machine Learning (ML) techniques within the field of Applied Mathematics, a critical foundation in today’s digital age. Fundamental algorithms such as tensor decomposition, low-rank approximation, matrix

completion, and matrix multiplication remain pivotal in powering contemporary industrial application and digital service. However, implementing these mathematical algorithms often involves heuristic configurations and randomization techniques to manage the ever-increasing scale of data. Consequently, enhancing the efficiency of these mathematical algorithms has attracted significant attention. Vigorous efforts are underway to find the ideal equilibrium between the efficiency and precision of the algorithms. With the rise of ML and AI, integrating these academic-specific issues into the ML search framework appears to be a viable strategy for refining the resolution of these problems.

### 1.1.1 AI for Algorithms Discovery

The interdisciplinary cooperation between the ML and other academics has made it possible to discover new results outside of the conventional domain. As aforementioned potentials in the applied mathematics area, the learning-based approaches to refining algorithm designs have attracted great attention.

One representative example of enhancing mathematical algorithms through the application of deep learning is seen with AlphaTensor [7]. Highlighted on the cover of *Nature* in 2022, AlphaTensor emerged as the inaugural AI system proficient in devising new, potent, and provably correct algorithms for Matrix Multiplication, notably enhancing the efficiency of multiplying matrices of $4 \times 4$ dimensions compared to existing human-created algorithms [8]. Building on this innovation, in 2023, it further revolutionized fundamental algorithms like sorting, with the advent of AlphaDev [9] accelerating processing speeds by 70% for short sequences through a learning-centric approach, showcasing advancements in the Deep Reinforcement Learning system

Indeed, these breakthroughs in fundamental algorithms were constructed upon innovative approaches to traditional mathematical challenges. Whether it pertains to finding the rank-one tensor in matrix multiplication or identifying the optimal assembly instructions vital for enhancing sorting algorithms, these endeavors require navigating an extremely large search space. Motivated by the progress in non-convex optimization abilities and the considerable opportunities in improving essential algorithms, we are prompted to broaden the application of ML techniques to a more diverse array of mathematical problems.

### 1.1.2 Minimum Fill-in Problem

An interesting issue in optimizing solutions for sparse linear equations is known as the *Minimum Fill-in* problem, which is significantly connected to both graph theory and sparse matrix methodologies. To provide an insight into the subject, we first give a broad introduction to the problem.

In a system of linear equations $Ax = b$, Gaussian Elimination is frequently used as the direct method to obtain the solution. It solves the system by recursively performing the row reduction on the coefficient matrix, and hence, obtains a triangular matrix to compute the final answer. See Fig.1.1 for a simple example.

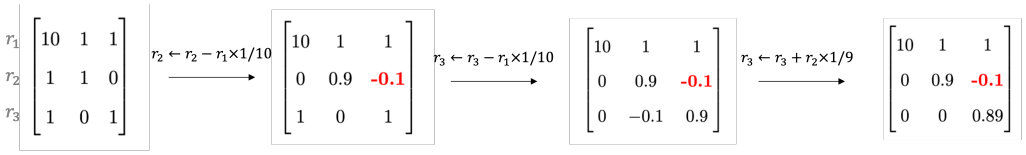


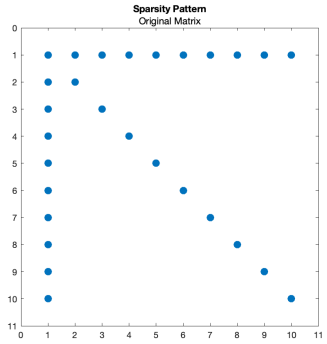

Figure 1.1: A simple example of Gaussian Elimination with Fill-in (red) constructed

After applying Gaussian Elimination, we obtain a triangular matrix, which aids in deriving the solution of the system through recursive calculations on said matrix. This process can be encapsulated in one step using the LU decomposition, expressed as $A = LU$, where both $L$ and $U$ are triangular matrices, or via the Cholesky decomposition, denoted as $A = LL^T$, applicable if the matrix is SPD

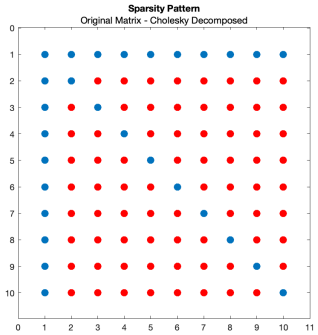
One critical problem in the sparse matrix decomposition, as the new entries labelled as red in Fig.1.1 showed, is due to the *fill-in*: After the factorization of large sparse coefficient matrices $A$, new nonzero elements called fill can replace original zeros. The sparsity patterns of results become increasingly important in terms of computational complexity, storage, and time consumption especially when the system has thousands of equations. To reduce the fill-ins, most of the existing procedures take advantage of the freedom for the selection of the pivots, namely, *permutation*.

It is widely acknowledged that fill-in can substantially vary depending on the chosen permutation order, as depicted in Fig.2. This illustration demonstrates that a simple rearrangement in the coefficient ordering of a linear system can result in significantly distinct sparsity patterns post-factorization. Nonetheless, finding the optimal permutation order has been established as an NP-complete task since 1981, as proven by Yannakakis [10]. Recent years have resulted a vast development of heuristic search methods, accompanied by the progression of theoretically precise algorithms with a time complexity of $\mathcal{O}(1.8899^n)$ [11].

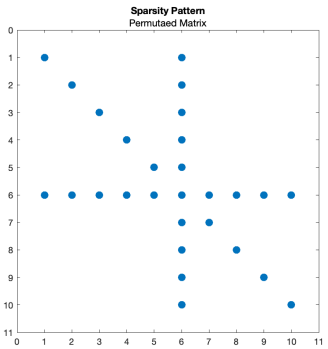
1



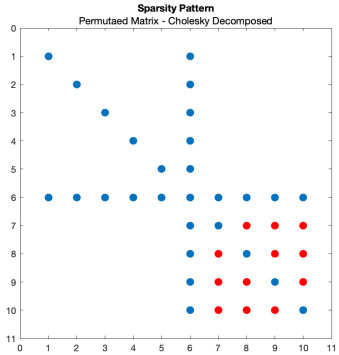

(a) Original Matrix




(b) Original Matrix after Factorization




(c) Permuted Matrix




(d) Permuted Matrix after Factorization

Figure 1.2: The Sparsity Pattern Comparison between the one matrix and it's permuted version. The red colour stands for the new fill-in after the matrix factorization.

## 1.2 Motivation

The approach to sparse linear systems is pivotal in engineering simulations. For instance, in Finite Element Analysis (Finite Element Analysis (FEA)), real-world physical phenomena like forces, vibrations, and fluid flows can be broken down into significant linear systems. Solving these equations can reveal patterns in complex situations across various industry sectors. This necessitates the development of highly efficient algorithms and software for solving linear systems, with a focus on computational complexity, memory utilization, and time efficiency. Besides FEA, other applications encompass database management, VLSI system assessment, and knowledge-based systems [12]. It is anticipated that enhancements in solving the minimum fill-in problem would augment the efficacy of all reliant applications, creating subtle yet substantial impacts.

On the other hand, there has been a vast advancement in unstructured data learning in the recent decade, such as Graph Learning and GNN. These techniques are proven to show a powerful ability to extract rich information from the complex connections between nodes and the topological

structure of graphs, and more importantly, the creation of operators like Convolution on Graphs in the GNN brings the predictive power of deep learning to rich data structures. Luckily, Graph has always been an important tool in the sparse matrix techniques study since the methodology developed in the 1970s by Rose [13]. It is inspiring to explore the interaction between the dual problem of the minimum fill-in in graph representation, namely, chordal completion and the advanced graph learning frameworks.

On the other hand, recent decades have witnessed significant progress in learning from unstructured data, including Graph Machine Learning and Graph Neural Networks GNN. These methodologies have demonstrated a potent capacity to extract abundant information from intricate node connections and graph topological structures, notably enhancing deep learning's predictive power for complex data structures through innovations like Graph Convolution operations in GNN. Fortunately, graphs have remained a vital instrument in the study of sparse matrix techniques, a tradition initiated in the 1970s by Rose [13]. This opens up an exciting avenue to investigate the relationship between the chordal completion, which is the dual problem of minimum fill-in represented in graphs, and the modern graph learning frameworks.

## 1.3 Research Contribution

This project yielded the following results:

- We developed a new technique that incorporates the classic minimum fill-in issues into the graph learning task. This method breaks down the search in the NP-complete problems into a multi-stage recurrent prediction task. Compared to other naive frameworks, our recurrent pipeline presented simple and effective solutions.

- We constructed a deep learning framework based on the Graph Transformer utilizing PyTorch Geometry. Through extensive experiments on pre-encoding strategies, we customized an efficient encoding scheme to the problem and trained a range of potent prediction models capable of handling graphs with up to 60 nodes.

- We discovered a group of novel solutions that outperform the existing top heuristic methods. We have released an open-source collection consisting of over 300 instances where the newly identified order information surpasses the existing heuristic approaches' best results.

1

## 1.4 Thesis Organisation

The structure of this report is arranged as follows:

In Chapter 2, we initiate a literature review from two perspectives: firstly, an examination of previous studies that integrated AI into mathematical endeavors; and secondly, a systematic overview of traditional methodologies surrounding minimum fill-in problems. The algorithms and examples mentioned in the introduction will be expanded upon in this section.

In Chapter 3, we present a formal description of the problem along with essential background information necessary for comprehending the challenges and proposed methods. We will accentuate the links between graph theory, sparse matrix techniques, and our proposed approaches.

In Chapter 4, we offer an extensive insight into the entire methodological design process leading up to the final solution. This includes the rationale behind our design choices, encompassing three primary components: the overarching pipeline design, dataset creation, and the chosen deep learning architecture. A summary of the devised method will conclude this chapter.

In Chapter 5, we delineate the experimental results derived from employing the proposed method, showcasing outcomes from six distinct training set sizes. Additionally, we will provide practical examples illustrating the real-world applications of our method.

In Chapter 6, we will encapsulate the key contributions, pivotal observations, and reflections on potential future developments, summarizing the presented method and its impacts.

# 2 Literature review

Contents



## 2.1 Learning-based approach of algorithm design

Among the algorithms in Applied Mathematics, heuristic and randomized elements are frequently integrated into algorithms to manage the increasing volumes of data . Consequently, learning-based strategies for algorithm development have garnered significant attention due their potential to enhance the efficiency of many prevalent algorithms.

Low-rank matrix approximation is one of the most widely used tools in massive data analysis, machine learning and statistics. Given an $n \times d$ matrix $A$, and a parameter $k$, the goal of low-rank approximation is to compute a rank-k matrix $A'$ that minimizes the approximation loss $||A-A'||_F$. One way to approximate the lower rank target $B, C$ in $A \approx BC$ is using the random sketching method [14], where it uses a randomized Gaussian matrix $\Omega$ to construct a subspace $A\Omega$ that captures most of the action of $A$. [15] integrated the random sketching idea into a compact singular value decomposition $A\Omega = U\Sigma V^T$, and finally output $A' = [AV]_k V^T$ as the rank-k approximation. Extended from the random sampling matrix $\Omega$, [16] proposed to replace $\Omega$ with a 'learned' matrix of the same sparsity to reduce the approximation error. The team proposed a Differentiable process of conducting the singular value decomposition, using back-propagation algorithm to compute the

stochastic gradient of $\Omega$ with the Frobenius norm loss as described before. It is shown that for multiple types of data sets, a learned sketch matrix can substantially reduce the approximation loss compared to a random matrix S, sometimes by one order of magnitude.

Matrix Multiplication sits at the core of contemporary digital age as one of the most basic operations in Linear Algebra. Standard matrix multiplication algorithm was believed the best efficient until 1969. German mathematician Strassen showed a better algorithm exists [8], bring the computation complexity of matrix multiplication from $\mathcal{O}(n^3)$ to $\mathcal{O}(n^{log7})$. Strassen algorithm achieves a better efficiency by reorganizing the computation elements and thus reducing the multiplication time. For instance, the multiplication time is reduced by 75% from the term $A^2 + B^2$ to $(A + B)(A - B)$. On the other hand, it is found that matrix multiplication algorithms can be formalized as low-rank decomposition of a specific three-dimensional tensor, called matrix tensor [17] [18], making it possible to transform the asymptotically optimal algorithm design into a regularized searching challenge. For a matrix tensor $T$ of size $n \times n \times n$, one could find a new way of representing the $n \times n$ matrix multiplication process by decomposing $T$ into R rank-one terms, namely, $T_n = \sum_i^R U_i \otimes V_i \otimes W_i$. In AlphaTensor [7], the author modelled the process of low-rank tensor decomposition as a Markov Decision Process and came up with a training model similar to AlphaZero [19], a novel deep reinforcement learning framework which trained the policy net and value net in a joint transformer-based network. AlphaTensor used both networks to guide the Monte-Carlo tree search during the training process. The final trajectory is used as the answer and the distribution of answer is refined by the quantile regression distributional loss [20].

Sorting algorithms are used trillions times of any given day. As one of the foundation algorithms in Computer Science, intensive research have carried out aiming at optimizing the algorithms' latency automatically and adeptly. This include [21], who proposed an enumerative search method to optimize the algorithms at a code level. It selectively refining the abstraction under which candidates are considered equivalent, only in the promising part of the candidate space. [22] developed a stochastic search at compiler level. They decompose the compilation into smaller synthesis sub-problems, namely, partitioning, layout, and code generation, and show that the synthesized programs are no more than 65% slower than highly optimized expert-written programs. In the work of AlphaDev in [9], a novel approach was brought to the fore by implementing deep reinforcement learning to address the problem, elevating the process by tailoring optimizations to real observed latency at the granularity of CPU instructions. Furthermore, this innovation featured a sorting network as cited in [23], structured with comparators and interconnecting wires, which facilitates the efficient exchange of augmented data through the established wiring systems.

## 2.2 Algorithms on minimum-fill in problem



As introduced in Section 1.1.2, Minimum-fill in problem plays a critical role on the sparse matrix techniques while it is impossible to solve in practice due to the NP-complete complexity. Heuristics are used to attempt to reduce fill-in and methods are mainly developed in three directions accordingly.

The band reduction were the first strategy developed in 1970s. The term bandwidth of matrix $A$ refers to the maximum distance between the first non-zero element and it's diagonal element among all rows in a matrix, denoted as $\max\{\beta_i(A)|1 \leq i \leq n\}$. As indicated in Fig. 2.1, Cuthill and McKee [24] in 1969 made the observation that to minimize the bandwidth of the row associated with $z$, node $z$ should be ordered as soon as possible after $y$. The Cuthill-McKee algorithm can thus be regarded as a method that reduces the bandwidth of a matrix via a local minimization of the $\beta_i$'s. The algorithm will start from an arbitrary node, and execute the search for the closest follow-by nodes in a Depth First Search fashion and finally reverse the outcome as the final permutation order.

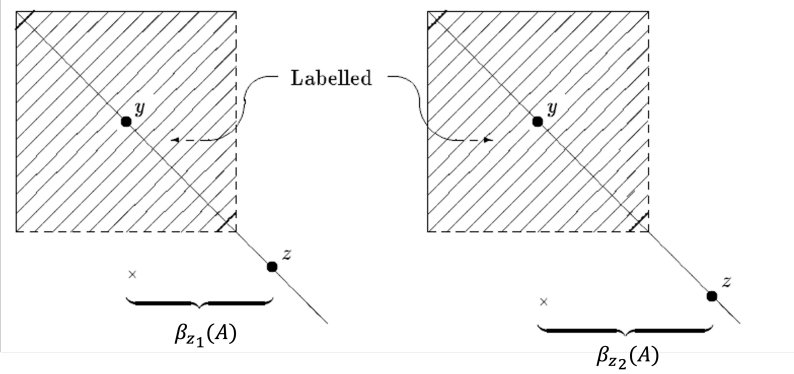



Figure 2.1: The illustration of how the distance between node $y$ and node $z$ affect the bandwidth $\beta(A)$ [25]

The Minimum Degree Algorithm [26] is extensively utilised in numerous applications due to its simplicity, cost-effectiveness, and effectiveness. As a result of the Gaussian Elimination procedure, prospective fill-ins may occur during each column elimination. Utilising a greedy strategy, an appropriate approach for minimising the nonzero values at each stage would be to consistently move the column containing the fewest nonzero values from the submatrix to the subsequent step. In essence, this involves selecting, during each elimination phase, the row and column that results in the least addition of nonzero values to the triangular components. Since the computing the exact degree is costly, Approximate Minimum Degree [27] was developed in 1990s later extended

this to compute a column ordering of $A^T A$ without forming $A^T A$ explicitly.

2

Nested Dissection [28] is a 'top-down' strategy anchored on the graph associated with the corresponding matrix. Initial graph partitioning techniques [29] were based on the concept of exchanging node pairings within the graphical representation. Since then, this method has evolved to include the swapping of larger node blocks, centralising the principal function of recursively identifying graph separators. As depicted in Fig. 2.2, the graph that is associated with the matrix is divided into two sections by the separators. Changing the node order to follow a block - block - separator sequence ensures that fill-in occurrences are restricted to the separator boundaries, which are depicted in Fig.2.2 as a narrow shaded zone.

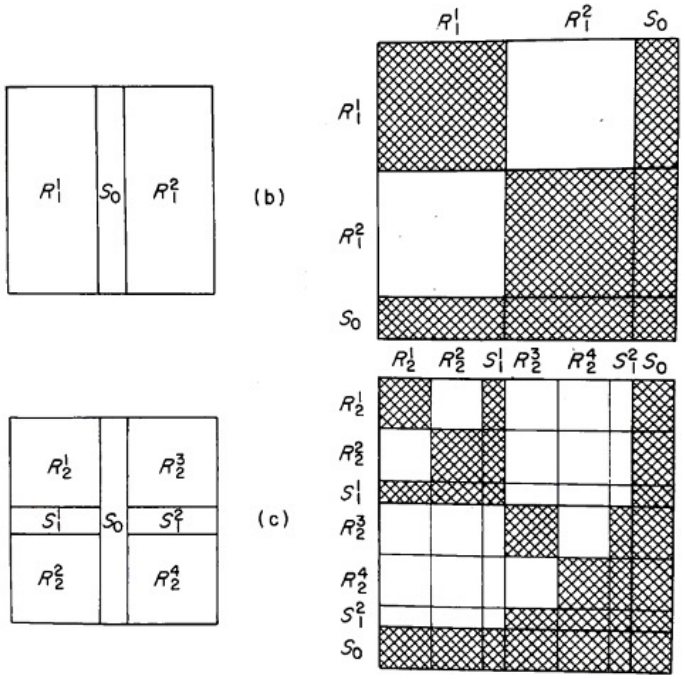



Figure 2.2: The illustration of Nested Dissection Process [30]

Recent advancements in Nested Dissections have expanded beyond 2D frameworks to incorporate 3D geometries. [31] introduces a novel partitioning technique that capitalises on the geometric positioning of nodes; concurrently, in light of advancements in parallel computation hardware, [32] reveals multi-tiered concepts that leverage the capabilities of eigenvector technologies.

# 3 Preliminaries

## Contents



In this chapter, we will begin by providing an introduction of the essential background knowledge. After establishing the core notations, we will proceed to give a clear explanation of the problem that piques our interest, with a focus on highlighting the connections and relationships between the graph and the minimum fill-in problem.

## 3.1 Graph Theory

### 3.1.1 Basic Notations

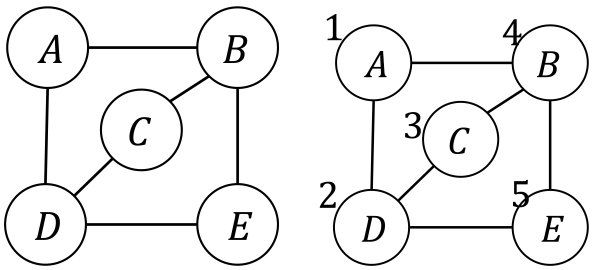
An undirected graph $G$ is denoted as a pair of sets:

$$G = (V, E)$$

The elements of $V$ are the *vertices* of the graph, the elements of $E$ are the *edges* and thus, $E$ is a subset of $\{\{v, w\}|v, w \in V\}$, $|V|$ is the number of vertices and $|E|$ is the number of edges in $G$. The terms *undirected* refers that the edge $\{v, w\}$ between vertex $v$ and vertex $w$ is unordered pairs, namely $(v, w) \equiv (w, v)$.

3

A graph with $n$ vertices is said to be ordered when all vertices have a one-to-one correspondence to the integer $1, 2, \cdots, n$. If $\alpha$ is the correspondence between the vertex index and the order, the graph can be denoted as $G_\alpha = (V, E, \alpha)$. See Fig.3.1 for a comparison




(a) Undirected Graph (b) Ordered Undirected Graph

Figure 3.1: The comparison between undirected graph and ordered undirected graph

A *chord* in a path of an undirected graph is an edge between non-consecutive vertices on the path. See Fig.3.2 for example.

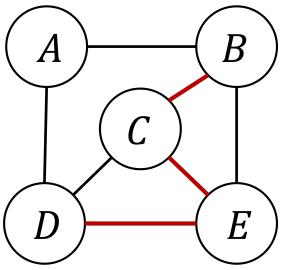
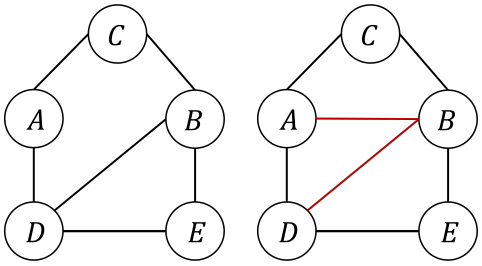



Figure 3.2: The edge $\{c, d\}$ is a chord in the path $(b, c, e, d)$

### 3.1.2 Basic definition

**Definition 1** An undirected graph is *chordal* if every cycle of length greater than three has a chord. The chordal completion of an undirected graph is a chordal graph.
It is occasionally referred to as a triangulated graph, given that it doesn't encompass an induced cycle with a length exceeding three. Refer to Fig.3.3 for illustrations.

**Definition 2** If $\alpha$ is an ordering of $N$, the *fill-in* $F(\alpha)$ produced by $\sigma$ is a set of new edges that are added when eliminating node from $\alpha(1)$ to $\alpha(n)$, namely, $F = \{\{w, v\}|w \neq v, \{w, v\} \neq E\}$

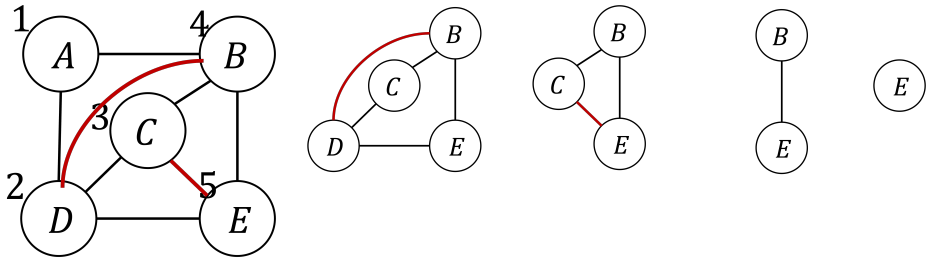



(a) It is not chordal graph since the cycle $(a, c, b, d)$ is chordless

(b) Chordal Graph

Figure 3.3: non-chordal graph and chordal graph

$G' = (V, U \cup F)$ is a chordal graph




(a) (b)

Figure 3.4: (a) The graph with fill-in resulted by the order $\alpha$, (b) The elimination process of the graph according to the order $\alpha$

**Definition 3** An ordered graph $G_\alpha = (V, E, \alpha)$ is *filled* if any two higher order neighbours $adj^+(v)$ of any vertex in $G$ induce an edge: $w, z \in adj^+(v) \rightarrow \{w, z\} \in E$

**Definition 4** $\alpha^*$ is the perfect elimination order if $G = (V, E, \alpha^*)$ is a filled graph.

This is because eliminating nodes by this order wouldn't create new chords. As a result, $F_{\alpha^*} = \phi$

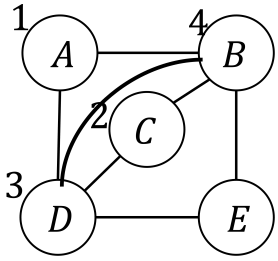



Figure 3.5: The chordal graph with the perfect elimination order. Eliminating nodes in this order would not create any new edges.

**Theorem 1** A graph is chordal if and only if it has a perfect elimination ordering [13].

## 3.2 Minimum fill in Problem Statement

### 3.2.1 Statement in Matrix View

3

As introduced in section 1.1.2, we give a formal description of the problem here and its connections to the graph theory.

For a system of linear equations

$$Ax = b$$

With $A$ sparse and symmetric. If $P$ is a permutation matrix, and thus $P^T P = I$, the system of linear equations can be written

$$PAP^T Px = Pb$$

or if $y = Px$ and $c = Pb$,

$$By = c$$

where $B = PAP^T$ is the permuted form of $A$.

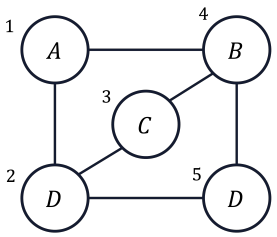
We wish to find a permutation order $P$ that could lead to the fewest amount of new non-zero entries created while factorizing $B$ depending on $P$, and if $B$ is SPD, $B = LL^T$, the target could be expressed as:

$$\begin{aligned} \arg\min_P &= ||L||_0 \\ s.t.\ B &= LL^T \\ B &= PAP^T \\ By &= c \end{aligned}$$

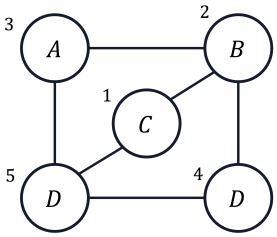
### 3.2.2 Graph Background of Symmetric Gaussian Elimination

Consider a matrix $A$. Assuming $A$ is a square matrix of order $n$, where the elements are represented by $A_{i,j}$; provided that all the diagonal elements $A_{i,i}$ are non-zero, it is possible to correlate an undirected graph with matrix $A$. This graph comprises $n$ identified vertices, denoted as $v_1, v_2, ..., v_n$. A connection in the form of an edge between $(v_i, v_j)$ exists in the graph if the corresponding matrix element $A_{i,j}$ is not equal to zero.

One of the advantages of using the graph is that the graph of a symmetric matrix remains unchanged if a symmetric permutation is performed on the matrix, only the labelling of the vertices changes. As examples in Fig.3.6, the structure (nodes and vertices) of the graph representation remains unchanged from $A$ to $PAP^T$, where $P$ is the permutation matrix. This invariance property makes the graph a convenient tool to analysis of sparse matrix structure.




(a) Graph $G = (V, E, \alpha_1)$ associated with Matrix $A$




(b) Graph $G = (V, E, \alpha_2)$ associated with Matrix $B$

| | 1 | 2 | 3 | 4 | 5 |
|---|---|---|---|---|---|
| 1 | x | x | | x | |
| 2 | x | x | x | | x |
| 3 | | x | x | x | |
| 4 | x | | x | x | x |
| 5 | | x | | x | x |

(c) Matrix $A$

| | 1 | 2 | 3 | 4 | 5 |
|---|---|---|---|---|---|
| 1 | x | x | | | x |
| 2 | x | x | x | x | |
| 3 | | x | x | | x |
| 4 | | x | | x | x |
| 5 | x | | x | x | x |

(d) Matrix $B = PAP^T$

Figure 3.6: (a) The graph with fill-in resulted by the order $\alpha$, (b) The elimination process of the graph according to the order $\alpha$

In the context of Gaussian Elimination, the appearance of new non-zero elements also maps to the new edges added to the graph. To illustrate clearly, see Fig. 3.9 for a detailed step process.

As depicted in Fig. 3.9, Gaussian Elimination is recursively implemented on the matrix rows. For instance, during the initial phase of the elimination procedure, it is necessary to eliminate the non-zero elements in the second and third rows. This is achieved through executing algorithmic manipulations between rows 1 and 2, and rows 1 and 4, respectively, thereby initiating the process of identifying the triangular coefficient matrix, where new non-zero elements, referred to as fill-ins, become apparent. Observing from the graphical representations, the graph undergoes elimination following the Parter Rule [33]: to derive $G^{k+1}$ from $G^k$, remove vertex $k$ and introduce all potential edges connecting vertices that were adjacent to vertex $k$ in $G^k$.

In relation to graph analysis, the Minimum fill-in issue can also be portrayed as a *Chordal Completion* task: given a specific graph $G$, is it possible to incorporate no more than $k$ edges to transform the graph into a chordal one? As delineated in Theorem 1 within section 3.1.3, a graph is chordal if and only if it has a perfect elimination order. Consequently, our objective can be

reformulated to identify an elimination order that minimizes the number of new edges introduced.

3

## 3.3 Graph Neural Network

Deep learning excels at identifying hidden patterns in data, commonly represented in Euclidean space, for tasks like image classification and natural language understanding. However, the rise of non-Euclidean data is characterized by complex, interconnected topological structures. GNN are specialized neural networks utilized for managing data with graph structures. They operate by analyzing data based on either node characteristics or structural attributes, facilitating hierarchical feature extraction or processing through the transfer, modification, and amalgamation of information among the vertices.

### 3.3.1 Graph Representation Learning

In the Graph Neural network, a primary objective is to derive a low-dimensional representation for every individual node, denoted as $f : u \rightarrow \mathcal{R}^d$, the goal is to associate each node with an embedding vector that encapsulates a majority of the node's structural details. This way, nodes with similarities are situated closely in the embedding area, and the similarity between node $u$ and node $v$ might be delineated via cosine distance, expressed as $similarity(u, v) \approx Z_v^T Z_u$. Following the efficient translation of nodes, the decoder can consequently revert the vectors back to nodes.

The most straightforward encoding approach could be utilizing a look-up table, whereby the feature vector is acquired simply through $ENC(v) = Z_v = Z \cdot v$, with $Z$ being a look-up table housing $d$ dimensional vectors of $|V|$. The ultimate quality of the embeddings can be gauged by the similarity metrics between pairs of nodes.

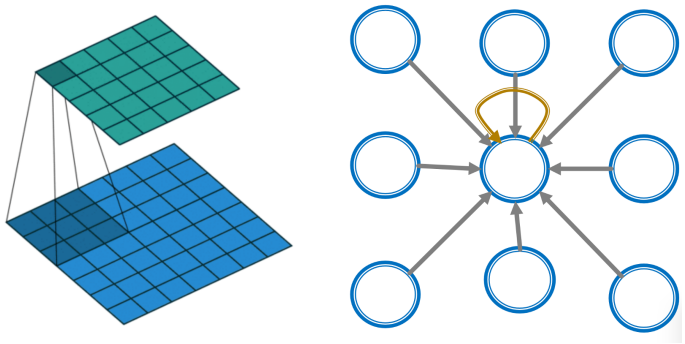
In recent times, the advent of deep learning methodologies has fostered the swift advancement of encoding strategies and similarity evaluations, transcending beyond linear transformations and becoming more adept at integrating node attributes. This evolution has given rise to deep graph encoders, which employ multiple layers of non-linear transformations anchored on graph topology.

### 3.3.2 Convolutional Graph Neural Network and Message Passing Mechanism



The graph convolution operation was introduced in [34] as a potent method to facilitate deep encoding, bridging the discrepancy prevalent in traditional deep learning toolkits, which are primarily structured for more straightforward data configurations.

The idea of a single graph convolution operation is similar to the conventional one in Convolutional Neural Network (CNN). As shown in Fig.3.7, the GNN approach is to transform the message $h_i$ from the neighbour nodes and sum them up, namely, $\sum_i w_i h_i$.



(a) (b)

Figure 3.7: The comparison between Convolution operator and Graph Convolution operator [35]

Building upon the graph convolution, the information of a graph was processed via the *Aggregation* and *Propagation* mechanism where the graph itself defines its computation graph. See Fig.3.8 for a two-layer propagation example.

Concretely, at the initial 0 th layer the embedding $h$ of node $v$ is equal to node features $x$, namely,

$$h_v^0 = x_v$$

To create higher order embedding of the node, the neighbours of node $v$ and the previous level of embedding of itself are summed up and transformed via a non-linearity activation function:

$$h_v^{l+1} = \sigma(W_l \sum_{u \in N(v)} \frac{h_u^{(l)}}{|N(v)|} + B_l h_v^{(l)}), \forall l \in \{0, \cdots, l\}$$

Where $W$ and $B$ are some linear transformations. The final representation can thus be obtained by repeating this aggregation and propagation mechanism several times.

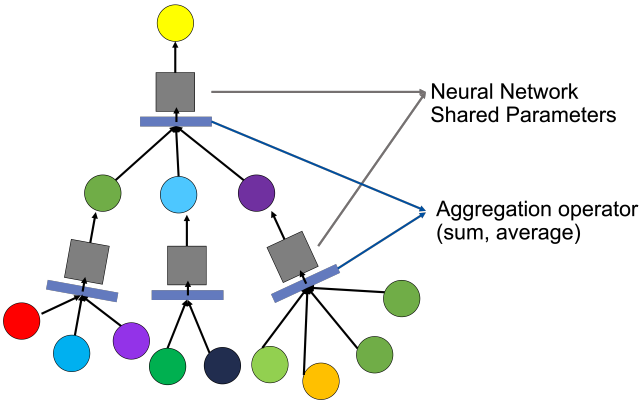



Figure 3.8: A typical computation graph of graph neural network

Apart from the GNN, the aggregation and propagation mechanism are actually capable of extending into a wider range of design space, generalized as Message Passing Graph Neural Network (MPGNN)s. The MPGNNs generalized the computation into two stages. The first stage is a *message* function, for example, the linear transformation term of $B_l h_v$ and $W_l h_u$ in the previous GNN example of computing the message of node $v$ and $u$, explicitly denoted as:

$$m_u^l = MSG_w(h_u^{l-1})$$

the second stage is *aggregation*, where it fuses all the messages together, For example, GNN takes the average of the nodes' neighbourhood message as the final information to propagate:

$$A = AGG^l(\{m_u^l, u \in N(v)\})$$

The final stage is to concatenate the neighbourhood message and itself and add the nonlinearity transformation to it, namely,

$$h_v^l = \sigma(CONCAT(A, m_v^l))$$

Synthesize the expression together, with $x_i^{k-1} \in \mathcal{R}^F$ denoting features of node $i$ in layer $(k-1)$, MPGNNs can be described as:

$$x_i^k = \gamma^{(k)}(x_i^{k-1}, \bigoplus_{j \in \mathcal{N}(i)} \phi^{(k)}(x_i^{(k-1)}, x_j^{(k-1)}))$$

Where $\bigoplus$ denotes a differentiable, permutation invariant function, sum, for example, and $\phi$ and $\gamma$ denote differentiable functions such as Multi-Layer Perceptrons (MLP)s

### 3.3.3 Graph Isomorphism Network

The term isomorphism refers to two graphs with identical structure and properties, namely, they have a one-to-one correspondence between their vertex set. In the study of representational properties of GNN [36], ideally, the upper bound of the expressive power of a GNN is expected to reach the injective mapping among the non-isomorphism graphs, that is to say, every two non-isomorphism graphs should be mapped to two points on the embedding space.

The aggregation function is the key factor affecting the injective mapping. Recall the explicit expression of aggregation in GCN, where it takes the mean pooling on the neighbourhood information: $Mean(x_{uu_{\in}N(v)})$. The multi-set function like mean-pooling in GCN can be summarized as:

$$\phi(\sum_{x\in S} f(x))$$

Where $\phi$ and $f$ are some non-linear functions. To maximize the expression power, Graph Isomorphism Network (GIN) [36] replace both linear function $\phi$ and $f$ with two distinct neural networks, taking advantage of Universal Approximation Theorem [37], and reforming the injective muti-set function as:

$$MLP_{\phi}(\sum_{x\in S} MLP_f(x))$$

As a result, the complete GIN model could be reformed from the aforementioned message-passing mechanism as follows:

$$GINConv(c^{(k)(v)}, \{c^{(k)}(u)\}_{u\in N(v)}) = MLP_{\phi}((1+\epsilon)\cdot c^{(k)}(v) + \sum_{u\in N(v)} c^{(k)}(u))$$

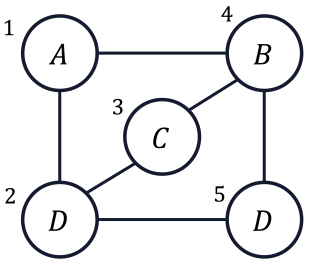
In which the term $c^{(k)}(v)$ summarizes the root node $v$, features of the k-hop neighborhood; $\{(k)(u)\}_{u\in N(v)}$ summarises the neighboring nodes features.

3

$A^{(1)}$:

| | 1 | 2 | 3 | 4 | 5 |
|---|---|---|---|---|---|
| 1 | x | x | | x | |
| 2 | x | x | x | | x |
| 3 | | x | x | x | |
| 4 | x | | x | x | x |
| 5 | | x | | x | x |

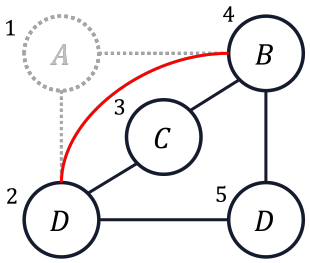
(a)




$A^{(2)}$:

| | 1 | 2 | 3 | 4 | 5 |
|---|---|---|---|---|---|
| 1 | x | x | | x | |
| 2 | | x | x | x | x |
| 3 | | x | x | x | |
| 4 | | x | x | x | x |
| 5 | | x | | x | x |

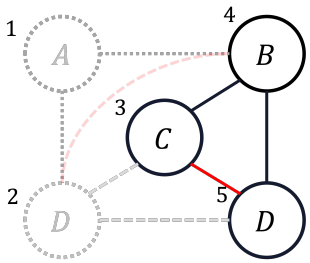
(b)




$A^{(3)}$:

| | 1 | 2 | 3 | 4 | 5 |
|---|---|---|---|---|---|
| 1 | x | x | | x | |
| 2 | | x | x | x | x |
| 3 | | | x | x | x |
| 4 | | | x | x | x |
| 5 | | | x | x | x |

(c)




⋮

| | 1 | 2 | 3 | 4 | 5 |
|---|---|---|---|---|---|
| 1 | x | x | | x | |
| 2 | x | x | x | x | x |
| 3 | | x | x | x | x |
| 4 | x | | x | x | x |
| 5 | | x | x | x | x |

(d)

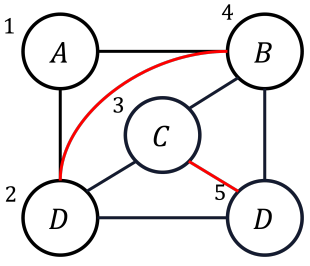



Figure 3.9: The two initial elimination steps and the corresponding elimination graphs for matrix $A$. The red entries are marked as red. The last graph shows the last result of the elimination

# 4 Graph Neural Network Method

## Contents



To the best of the author's knowledge, this is the first attempt to use deep learning to handle the minimum fill-in problem. It is worth noticing that due to the NP-hard complexity of this problem, finding the optimum solution, much alone creating a training database with optimum labels, is impractical. Nonetheless, we recognize the promising potential that Deep Neural Network (DNN)s hold in searching in nonconvex domains and are eager to investigate if this unique strategy might potentially match or even outperform currently available heuristic methods. As a result, the goal of our deep learning-based method might be stated as follows:

*Can we develop and train a deep learning model capable of forecasting the permutation order with equal or superior efficacy compared to the optimal heuristic approaches?*

Graphs, which are conventional and effective tools for analysing sparse matrices, have been used extensively for analytical purposes in sparse matrix areas for decades. This includes their use in the three most widely used heuristic algorithms, as detailed in Chapter 3. The emphasis has

recently shifted from analysing the structural topology of graphs to incorporating them with the Graph Neural Network GNN. As a result, GNN, which enables the combination of deep encoding techniques with graph configurations, was chosen as our preferred deep learning technique for handling our minimum fill-in challenge.



To integrate the practical issues into the GNN framework, the process is typically divided into four stages, as outlined in Figure 4.1. Aside from the input data setup, the computational modules, output, and loss function cover a large portion of the design space.

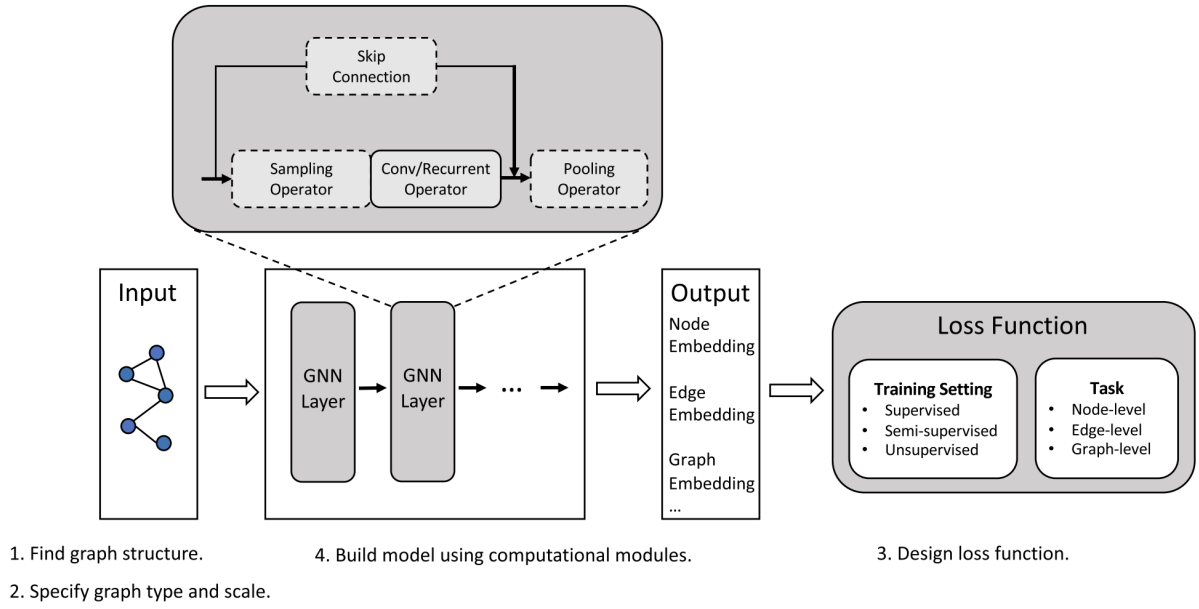



Figure 4.1: A generalized design pattern for a graph neural network problem [38]

In this section, we will discuss our method design, detailing each design module. We will demonstrate the pathway that led us to our final Graph Neural Network solutions through a comprehensive investigation of many options.

Moving to the initial stage in our configuration, the first question we need to tackle is *What does the pipeline look like?* , since it governs how the input is processed and impacts the inference process.

## 4.1 Prediction Pipeline Design

### 4.1.1 The single feed-forward scheme

Two different inference procedures were studied. The first approach is to take the prediction as a *node level regression* task. Considering the final objective was to find the best permutation sequence, the most intuitive method was to assign a new order to each individual node. Refer to Fig.4.2 for an illustration.



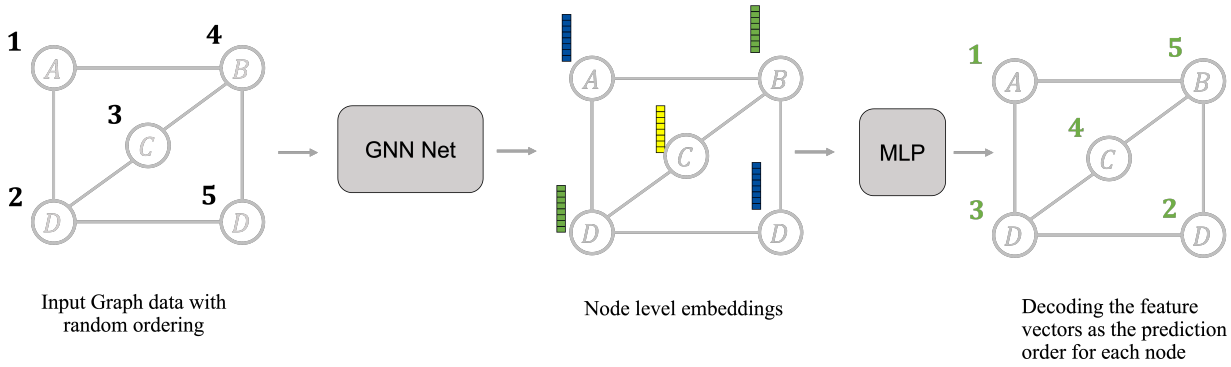



Figure 4.2: The singular feedforward strategy for determining the predictive sequence of each node

The procedure of inference in the initial scheme can be described as follows:

- Data Normalization. The first step involves converting the $n \times n$ sparse matrix into an undirected graph comprising $n$ nodes. Initially, these nodes are arranged randomly, represented by labels such as $[v_1, v_2, \ldots, v_n]$.
- Feature Embedding. In this stage, a GNN-based network is employed to encode each node into a compressed $k$-dimensional feature vector, namely, $f : \mathcal{G} \rightarrow \mathcal{R}^{n \times k}$. The anticipated outcome is a $\mathcal{R}^{n \times k}$ matrix corresponding to a graph with $n$ nodes. The network is designed to proficiently extract detailed structural information through the k-hoop vicinity of the root node. Additionally, it facilitates the distinctive identification of any two separate features through injective mapping.
- Feature Decoding: Subsequently, we utilize a Multi-layer Perceptron MLP network, denoted as $g : \mathcal{R}^{n \times k} \rightarrow \mathcal{R}^{n}$, to convert the feature vectors of each node into singular real number weights, intended for evaluation purposes.

- Comparative Ranking: To finalize the output, a reordering sequence is established by assigning new indices to each node based on their respective weights in relation to one another, culminating in the derivation of the final permutation sequence.

Following the establishment of the first naive framework, we began by validating the network's correctness using a single sample training task. The rapid convergence of training loss, as shown in Fig.4.2, confirms the effective functionality of the back-propagation and feedforward cycles.

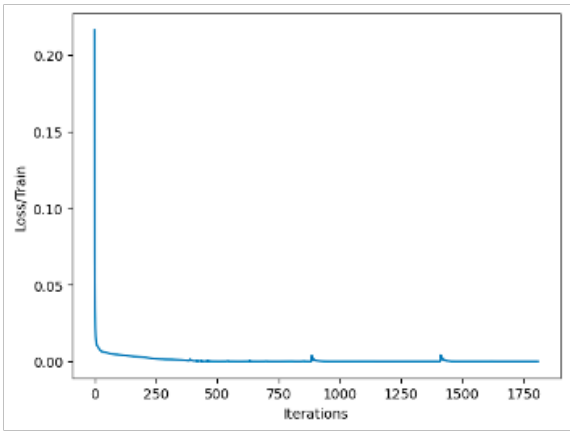
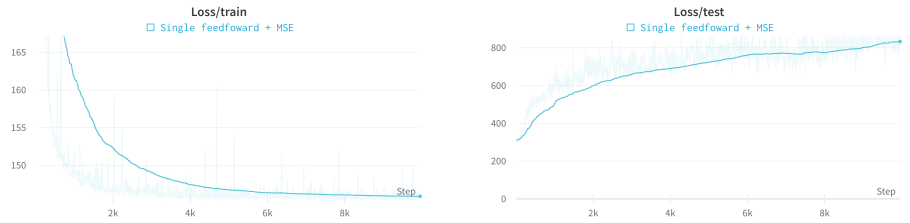



Figure 4.3: The training loss of a single-sample task to verify the correctness of the network

Upon confirming the accuracy of the initial network, the training sample size was then amplified to 10,000, allocating a 10% segment for testing purposes. During this preliminary test, we employed the Mean Square Error to evaluate the discrepancy between the ground-truth order of each node and their forecasted weight, prior to restructuring the relative ranking as delineated in the previously discussed process.

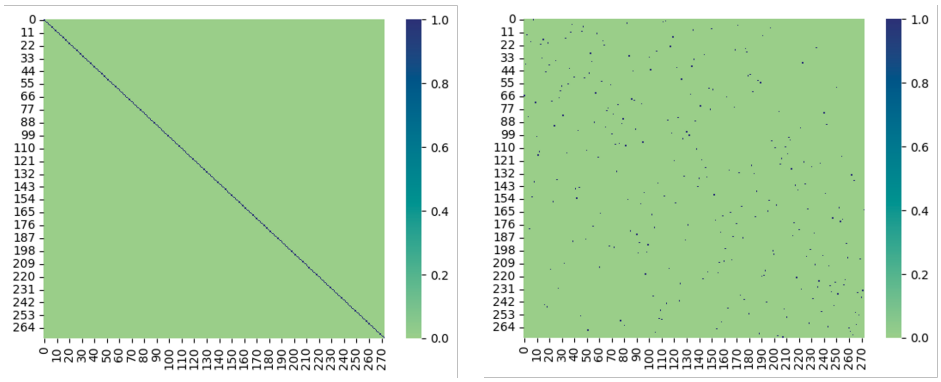
Regrettably, the basic method exhibited inadequate performance upon expanding the dataset. As depicted in Fig.4.4, the training loss is shown to be converged but gives up to severe overfitting after the early training stages. Furthermore, the training loss remains significantly different from the loss observed in the single-sample task, indicating that this technique has a limited learning potential.

To delve deeper into evaluating the ultimate quality of inference, an analysis was conducted in conjunction with the ground truth, resulting in a confusion matrix depicted in Fig. 4.5. Based on a single test graph, the test goal is to align the test result closely with the diagonal line, indicating a match between the expected and actual results. Regrettably, despite the model demonstrating a certain degree of convergence capacity, the mis-correspondence rate hovered around nearly 100%, indicating a subpar quality of inference.




(a) Train loss (b) Test Loss

Figure 4.4: The loss curve using the naive pipeline



(a) The ground-truth prediction (b) The naive pipeline prediction

Figure 4.5: The comparison of confusion matrix between ground-truth and prediction

To enhance the approach beyond the initial scheme, we have outlined the noted drawbacks in the first pipeline as follows:

- Excessive training loss
- Incorrect inference results
- Considerable overfitting

The first issue may derive from the Graph Neural Network's restricted expressive capacity discussed in section 3.3.3. Due to the injective mapping in non-isomorphic graph sets, similar tree structures yield identical feature encodings even for different nodes. To augment the GNN's capability, pre-encodings of input data may be utilized, with more information in the forthcoming sections.

The second concern may arise from the improper selection of training loss. When organizing a sequence of nodes, relative differences hold more significance than exact prediction outcomes. To prevent overfitting, implementing techniques such as Batch Norm, Dropout, adjusting the learning rate, and increasing the training batch size is advisable.

### 4.1.2 The recurrent scheme

To address the issue of the network's limited prediction capability stemming from a single instance of feed-forward, an enhanced approach could involve breaking down the entire task into several sub-tasks. Rather than amplifying the network and task complexity, the goal is to reduce task complexity through this decomposition, maintaining a framework that is both straightforward and efficient. The encoding-decoding format from the initial scheme is retained in this setup, but the task is modified to a *node-level binary classification* problem. See Fig.4.6 for an illustration of the second proposed pipeline.

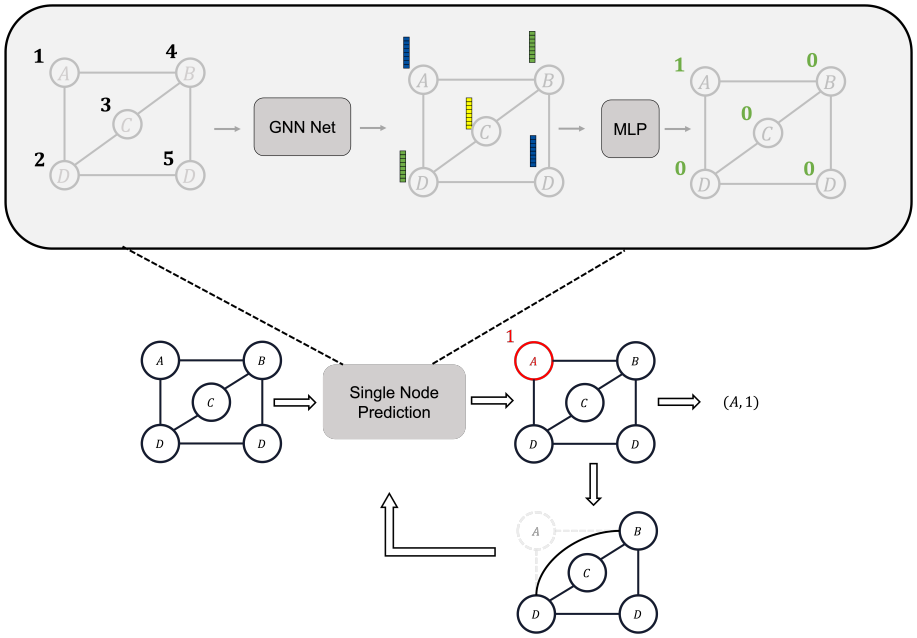



Figure 4.6: The recurrent inference scheme

The workflow in the second suggested pipeline is outlined as follows:

- Data Normalization: The input data structure remains the same as in the first scheme, represented as an n-node undirected graph denoted by $G = (V, E), |V| = n$.
- Feature Encoding: The core concept remains consistent with the first approach, focusing on extracting as much node structure information as possible and presenting it in feature vectors denoted as $R^{n \times k}$.
- Binary Classification: Diverging from the initial scheme that predicts node weights all at once, the recurrent strategy aims to gradually reconstruct the chordal completion process. Consequently, the prediction objective is simplified to identifying the most significant node

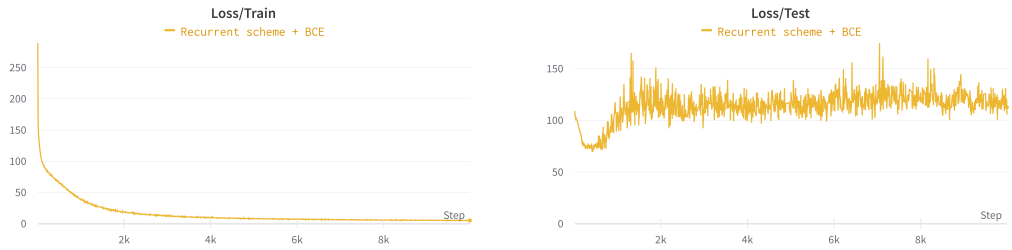
in the current graph structure. Simultaneously, the node's order will be logged in a global list, forming the basis for the final permutation order.

- Auxiliary Node Removal: As delineated in section 3.3.2, the removal criteria in chordal completion are solely dependent on node connectivity. We have developed a Python-based elimination engine that can seamlessly integrate into the inference procedure, considering the given graph and target node.

- Iterative Prediction: Following the auxiliary node removal, the remaining graph undergoes relabeling and is readied for re-entry into the same network. A global dictionary will record the transitions in the graph's index pre and post-node elimination.



To evaluate the predictive quality and determine whether the recurrent scheme improves the identified issues, a comparable experiment was carried out utilizing the same batch of 10,000 training samples and a 10% for the test set as before. The training result is shown in Fig. 4.7.



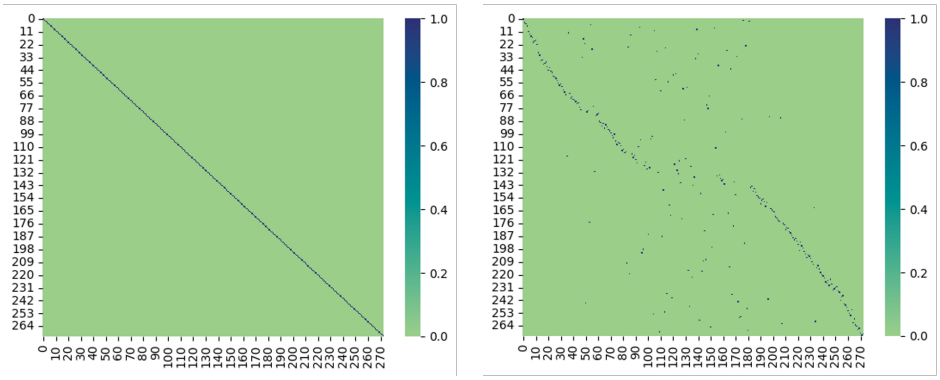

(a) Train loss (b) Test Loss

Figure 4.7: The loss curve using the recurrent pipeline

Despite overfitting in the recurrent scheme training remains, it evidently managed to converge to a near-zero training loss, unlike the single feed-forward method, which struggled to differentiate between similar graph structures. This can be attributed to the prediction task's nature, allowing for more degree of freedom clustering in the embedding space.

To further substantiate the efficacy of the second scheme, a single sample test was also performed, as shown in Fig. 4.8.

Compared to the initial scheme, the predicted order now loosely aligns with the ground truth. In the 270-node analysis, it was noticed that a specific group of nodes exhibited symmetrical order shifts. For instance, a node originally ranked 30th was predicted as the 70th, and vice versa. This phenomenon might stem from the variability in graph structures, wherein multiple valid ordering solutions can achieve optimal fill-in results. Consequently, we intend to retain the recurrent scheme



(a) The ground-truth prediction (b) The recurrent pipeline prediction

Figure 4.8: The prediction quality comparison

as a foundation for subsequent enhancements. For a detailed representation of the approach, refer to Algorithm 1.

It's crucial to note that the experiments conducted so far were primarily for qualitative comparison. The training setups, encompassing the dataset, structural configuration, and hyperparameters, were implemented in a preliminary manner without fine-tuning. This approach inevitably leads to deviations in the test outcomes. Nevertheless, the qualitative analysis has provided valuable insights into the strengths and weaknesses of the selected design, setting the stage for more refined explorations in the subsequent phases of improvement on the current choice. We may summarise this approach explicitly in the algorithm table below:

**Algorithm 1** Recurrent Node Prediction

**Ensure:** $G = (V, E, \sigma)$
**Ensure:** $f : G \to \mathcal{R}^{|V|\times 1}$
  $i \leftarrow 1$
  $P \leftarrow []$ ▷ Final ordering list
  $\sigma_i \leftarrow \{v_1 : 1, v_2 : 2, \cdots, v_n : n\}$ ▷ Node-Ordering mapping
  $D \leftarrow [\sigma_i]$ ▷ Relabel maps list
  **while** $V \neq \phi$ **do**
    $W \leftarrow f(G)$
    $v_p \leftarrow argsort(W)$
    $G \leftarrow (\complement_V v_p, E \cup F_p, \complement_\sigma \sigma_p)$ ▷ New graph after node elimination
    **for** $\sigma$ in D[::-1] **do**
      $v_p = \sigma[\sigma.value = v_p]$ ▷ recover the node index to the beginning graph
    **end for**
    $P \leftarrow P + [v_p]$
    $\sigma_i \leftarrow \{v_1 : 1, v_2 : 2, \cdots, v_{n-i} : n - i\}$ ▷ Reset graph label
    $D \leftarrow D + [\sigma_i]$
    $i \leftarrow i + 1$
  **end while**

## 4.2 Dataset Design

### 4.2.1 Data source



This section investigated two types of data sources: realistic industry-derived sparse matrices and synthesised data.

We originally aimed to discover existing open-source databases from real-world engineering challenges, which were largely prompted by engineering practices. This brought us to the University of Texas's SuiteSparse Matrix Collection [39]. This collection contains a large number of datasets from numerous fields, including structural engineering, thermodynamics, and chemical process simulations, among others. Although 235 positive symmetric matrices were found, using them on personal devices proved impractical due to their high memory and processing demands. The average graph size was 41 Mb, consisting of around $259,789 \times 259,789$ elements, with a small percentage (less than 10%) having a graph size of fewer than 1,000 nodes. On a high-performance computing node equipped with 100 AMD EPYC 7742 CPU units and 512 GB of RAM, preliminary attempts to load and deconstruct the matrix to extract ground-truth labels exceeded 96 hours of runtime.

The prohibitive computational resource requirements for tackling industrial problems make the generation of adequately labelled datasets unfeasible within the given time. Nevertheless, the insights gained from examining the structures and properties of these datasets encouraged us to develop analogous matrices on a reduced scale independently. We have outlined a method for synthesizing training data, as depicted in 4.9:

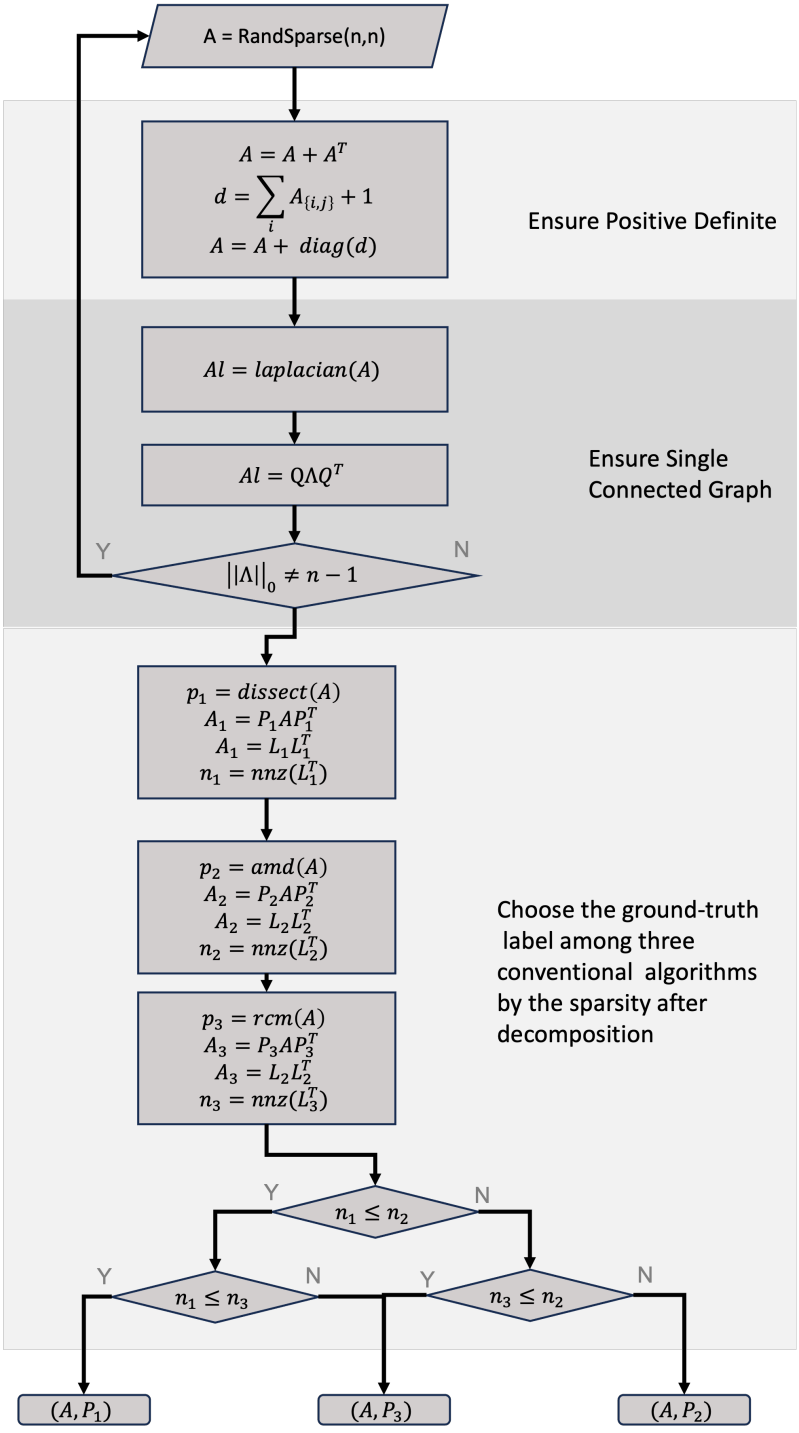



Figure 4.9: The flow chart of how a single data sample generated

In the illustrated flow chart, we employed a strategic approach to decrease data complexity by constraining the data type to singularly connected graphs, wherein the Laplacian matrix of the graph possesses a single zero in its spectrum of eigenvalues, in line with the principles of Algebraic Connectivity [40]. This facilitates the transformation of a complicated work into a number of

smaller, independent sub-tasks, each modified by individual node eliminations. Moreover, to fully leverage the capabilities of DNN, we selected the most optimal label from the results yielded by three prevalent heuristic algorithms: the Approximate Minimum Degree Algorithm [27], the Reverse Cuthill–McKee algorithm [24], and the Nested Dissection algorithm [28], thereby facilitating enhanced outcomes.



### 4.2.2 Data structure

After determining the best permutation sequence from the graph's initial data, we may proceed to create the training set by turning the raw data into a structured training set format.

Referring back to Section 4.1's prediction technique, we chose the recurrent strategy for creating predictions, classifying the work as a node classification operation. The label inside the training data would highlight the node selected for removal with a positive mark at each step while allocating negative marks to the remaining nodes. This is illustrated graphically in Fig.4.10.

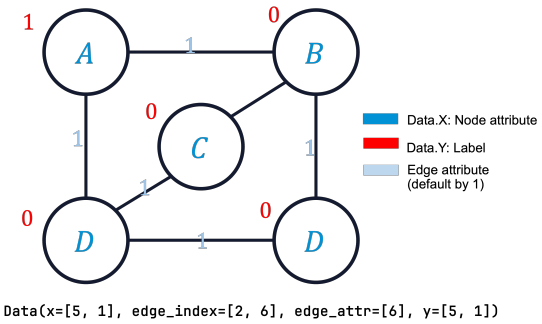



Figure 4.10: An example of an input data structure with four essential components

In the practical implementation, each graph is classified as a separate entity, characterized by four key matrices that act as its attributes: the node attributes matrix, the node label matrix, the edge matrix (specified by the originating and terminating nodes), and the edge attribute matrix. Following this, one can initiate the node elimination process as per the discovered optimal sequence, confirming the elimination guidelines outlined in section 3.3.2, with every intermediate phase logged as a separate instance

However, determining whether to include the effects before and after node elimination into the data structure is a significant decision throughout this process, as seen in Fig.4.11, which compares both cases. Given the deep learning network's unexplored expressive powers, tackling this issue is critical for striking a balance between data complexity and the depth of structural insights. As a result, we conducted two separate tests to determine the best structure for representation.

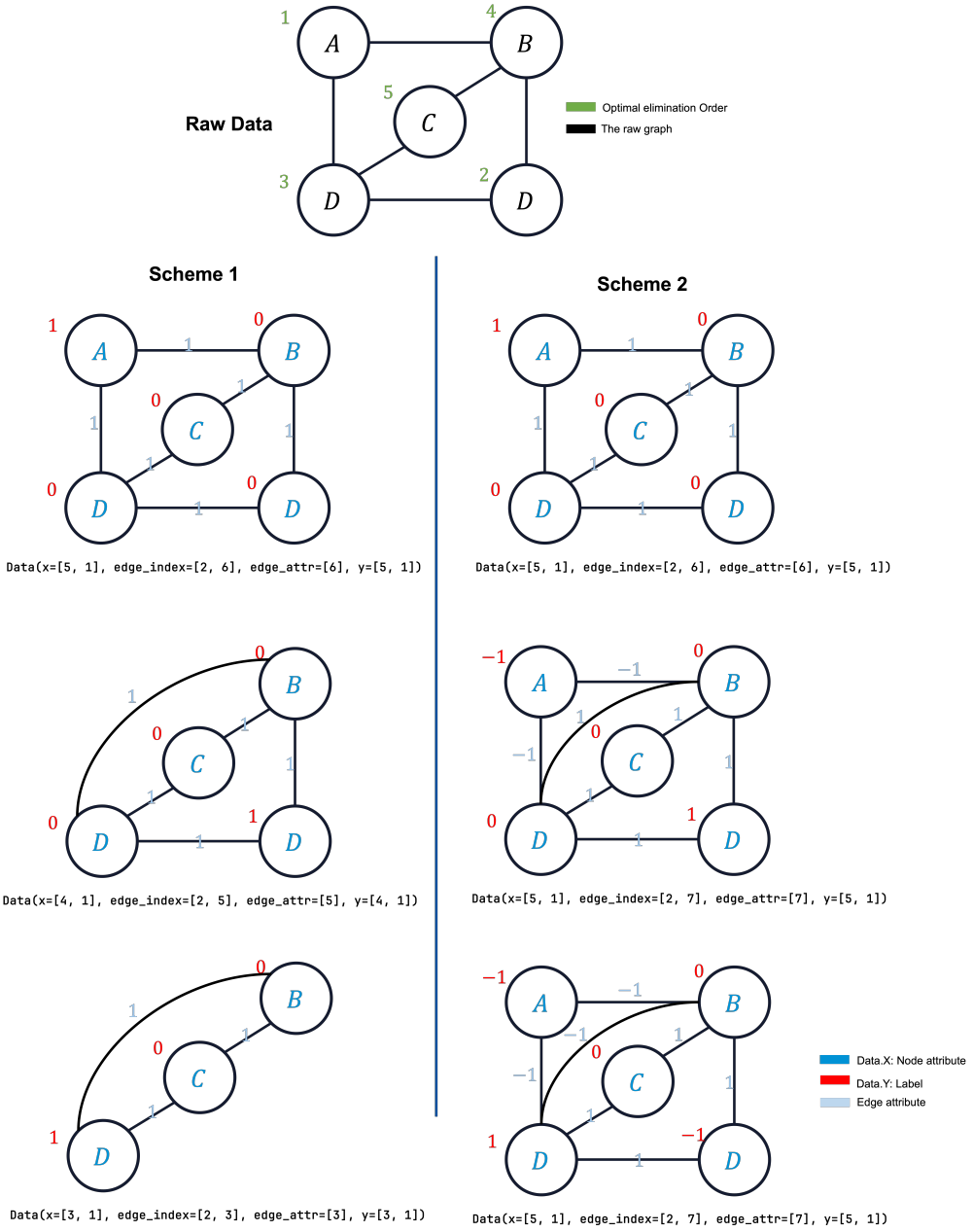


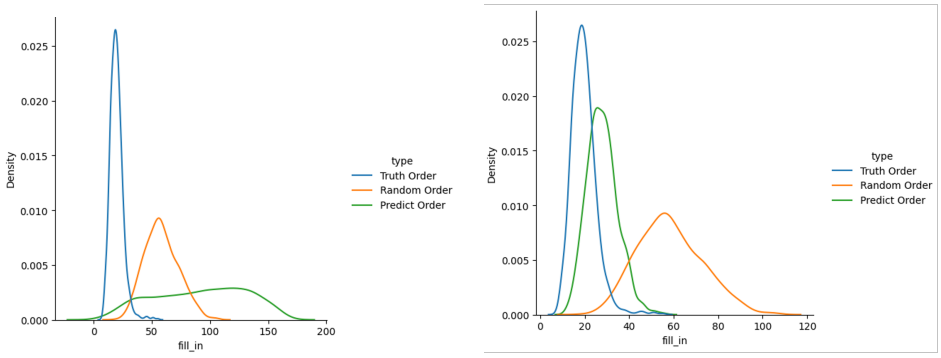

Figure 4.11: A comparison between two data structure schemes

For the initial plan, the procedure is straightforward: we record the outcome of each elimination phase, noting the present target node for elimination as positive and all other nodes as negative. Since the prediction is anticipated to rely largely on connectivity and structural data, random integers were initially allocated to node attributes, and a default value was assigned to the edge attribute. Given the importance of structural information in representation learning within GNNs, we seek to incorporate as many node elimination results as possible into the data structure. Instead of reducing the data dimensions after elimination, we keep all the information and use unique

representation to distinguish between: 1. visited and unvisited nodes, and 2. original and newly created edges using a variety of markers.

To evaluate the efficacy of these two data structure strategies, we created two separate training sets derived from the same set of 100 graphs, each consisting of 20 nodes. Consequently, the first scheme encompasses $100 \times 20$ subgraphs, with node sizes varying between 1 and 20, while the second scheme holds 200 subgraphs, all retaining a consistent size of 20 nodes. The initial 1,500 subgraphs were designated as the training set, leaving the remaining 500 subgraphs for the test set. We quantified the fill-in ( or saying the new edges added during the chordal completion ) in accordance with the elimination sequences dictated by three sources: the initial random order, the ground truth order, and the predicted order. Refer to Fig.4.12 for a comparison of the result quality between the two approaches.






(a) Scheme 1 (b) Scheme 2

Figure 4.12: The histogram of fill-in count resulted by two data structure schemes

Fig.4.12 displays the histogram of fill-in outcomes produced by the two distinct scheme-specific encoded data strategies, along with a naive predictor. For a more comprehensive comparison, we have also incorporated histograms representing the results derived from both ground-truth ordering (blue) and random ordering (blue).

It is clearly observed that scheme 2, which retained and clearly labelled all node and edge information throughout the elimination procedure, produced substantially inferior results, surpassing the lower threshold established by random order. Scheme 1, which did not entail additional node and edge attributes, performed significantly better. The fill-in distribution deviates significantly from the random order, indicating that the network has partially incorporated the elimination procedure and the attributes that facilitate the ground-truth ordering. This phenomenon could be attributed to the limited supplementary data limiting the model's search trajectory to a more focused one. As a result, we were motivated to retain Scheme 1's data structure for the completion

of the dataset construction and further model refinement.

### 4.2.3 Pre-encoding

It becomes apparent that the methodology employed in organising and representing data can substantially influence the eventual learning potential of the model. Despite the fact that Scheme 2 illustrates the hazards of augmenting data representation inadequately, it continues to be a vital path for the ongoing refinement of models.

This requirement arises from a central issue in graph learning: the *graph isomorphism problem*, which limits the capacity of graphs to represent information. Consider Fig.4.13, in which the GNN fails to differentiate between nodes $v_1$ and $v_2$. An ideal node representation learning strategy should be able to distinguish between the nodes $v_1$ and $v_2$. In the current scenario, however, the graph network allocates identical embeddings to both nodes because the tree structures starting from these roots are always identical.

Introducing more appropriate features to each node could be a viable solution for overcoming this limitation. For instance, assigning distinct attribute values to the nodes $v_1$ and $v_2$ may reveal these discrepancies. This process could broadly be termed as feature encoding.

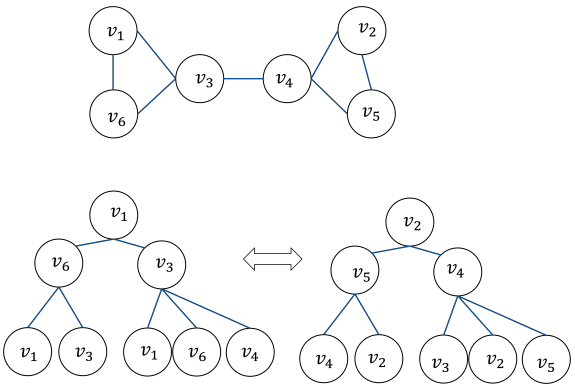



Figure 4.13: An example of isomorphism problem

Various *Positional Encoding* and *structure encoding* strategies were analyzed to amplify the expressive power of GNNs. Given that the impact of diverse encodings significantly hinges on the specific requirements of the task at hand, we investigated several encoding schemes as delineated in recent guidelines [41], and summarised as follows:

1. Local Position Encoding (PE): Partial Random Walk Matrix. We define the adjacency ma-

trix of the graph as $A$, the degree matrix as $D$, and the Random Walk matrix as $AD^{-1}$, which illustrates the transition probability between any two nodes. Consequently, the transition probability after m steps can be described as $(AD^{-1})^m$. To facilitate a node's understanding of its position and function within a local node cluster, we employ the sum of the non-diagonal elements in the rows of the random walk matrix, represented as:

$$\sum_i (AD^{-1})^m - diag((AD^{-1})^m)$$



2. Local Structure Encoding (SE): Node Degree. We aim to enable a node to comprehend the edge connection scenarios in its vicinity, making the node degree a valuable piece of information, represented as:

$$diag(D)$$

3. Global PE: Laplacian Eigenvectors. We represent the Laplacian matrix as $\phi = D - A$. Our goal is to incorporate global positional data to gauge the proximity between two nodes within the graph, symbolized as:

$$eig(D - A)$$

4. Global SE: The $k$-smallest eigenvalues of the Laplacian. This scheme seeks to equip the network with insights on the graph's overall structure, denoted by $U_k$, where:

$$L_{m\times m} = U_{m\times k} E_{k\times k} V^T_{k\times m}$$

In order to identify the most proficient strategies among the various options available, we executed an ablation study to assess the effectiveness of each distinct encoding method. The analysis was performed utilizing 10 samples, each comprising 40 nodes, in conjunction with a basic Graph Convolution Network predictor. We established the default random node value as the baseline, which acted as the exclusive piece of encoded data, aside from the edge connections. The results of this investigation can be seen in Table 4.1.

Remarkably, the Local SE encoding, which represents the node's degree, emerged as the most vital factor in amplifying expressive power. When paired with the Local PE, these two complementary encoding schemes outperformed in all the evaluations. This result suggests that positional and local data have a significant impact on the quality of the search for the optimal elimination order. Considering graphs with $N$ nodes, $E$ edges, and a random walk distance of length $m$, for

| Encoding Scheme | Loss | Improved by |
|---|---|---|
| Base | 0.134 | \ |
| **+ LocalSE** | **0.080** | **40.30%** |
| + LocalPE | 0.110 | 18.01% |
| + globalPE | 0.130 | 2.98% |
| + globalSE | 0.091 | 32.09% |
| **+ LocalSE + LocalPE** | **0.075** | **44.03%** |
| + LocalSE + LocalPE + GlobalPE | 0.092 | 31.34% |
| + LocalSE + LocalPE + GlobalPE + GlobalSE | 0.091 | 32.09% |

Table 4.1: The ablation experiment on different encoding schemes



example, we can finalise and describe our data structure as follows:

*Data(x=[N, 1], edge_index=[2, E], edge_attr=[E], y=[N, 1], degree=[N, 1], localPE=[N, m])*

## 4.3 Model Architecture

### 4.3.1 Graph Isomorphism Network

The discoveries thus far are based on the fundamental implementation of the Graph Convolutional Network. In the quest to improve feature representation, the Graph Isomorphism Network has emerged as the leading option to replace the GCN. As shown in section 2, these MPGNNs effectively represent a neighbourhood aggregation strategy, wherein the GIN aligns with the injective mapping among all non-isomorphic graphs as introduced in section 3. A node's update rule can be briefly defined within a practical context as:

$$h_i' = \theta(h_i + \sum_{j \in \mathcal{N}(i)} h_j)$$

Where $\phi$ stands for a neural network.

Following the initial experimentation on feature encoding and data structure formulation, we proceeded with training tests utilizing the developed training pipeline and framework. The entire process is depicted in Fig. 4.14. The raw data input comprises sparse symmetric matrices, representing the coefficient matrices of linear equations. We extracted two kinds of information from the graph: local positional information and local structural details. Ultimately, we implemented the GIN network supplemented with a Sigmoid activation function. The anticipated output is a weight vector primarily containing values near zero, with one significant value approximating one.

To facilitate gradient descent, we employed binary cross-entropy, expressed as:

$$l_n = -w_n[y_n \cdot logx_n + (1 - y_n) \cdot log(1 - x_n)]$$

And the total loss will thus be $L = \frac{\sum_i^n l_n}{n}$

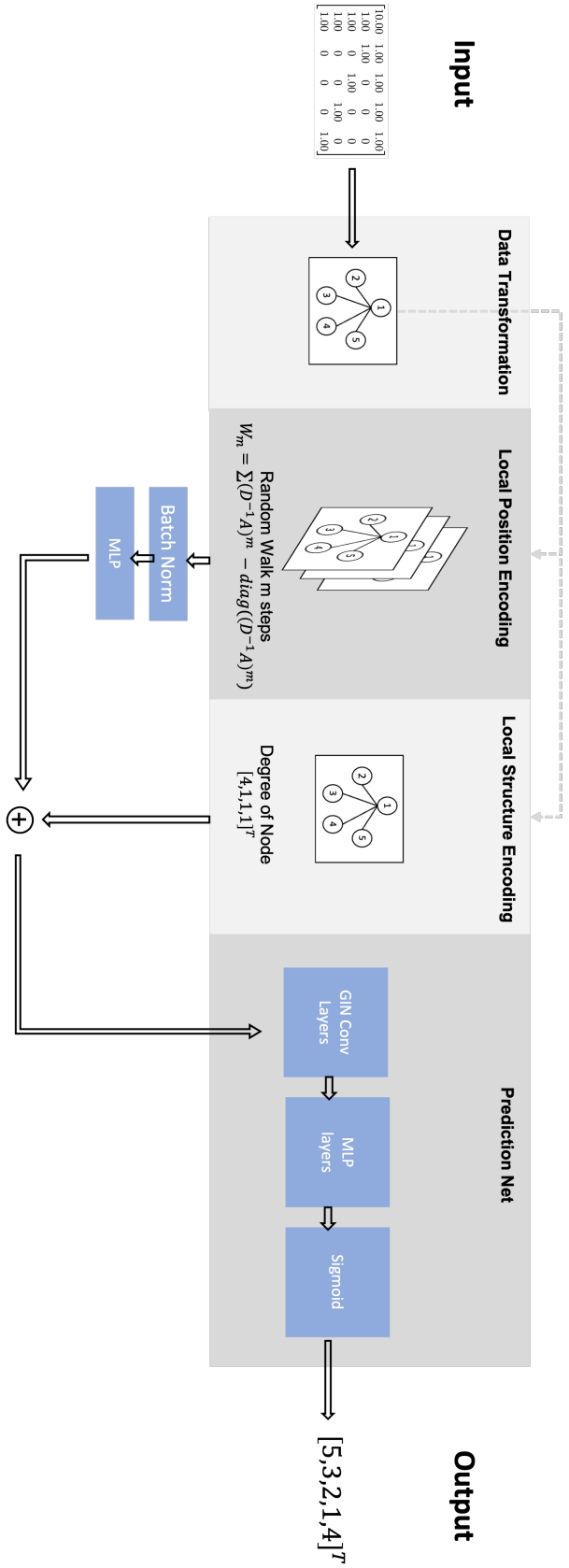



Figure 4.14: Overview of the pipeline using Graph Isomorphism Network

To acquire a thorough understanding of the architecture's efficacy, we conducted training on 800 fully-formed 20-node graphs, thereby generating 14,000 subgraph samples for training and allocating 2,000 subgraphs for testing purposes. The test outcomes are illustrated in Fig.4.15.

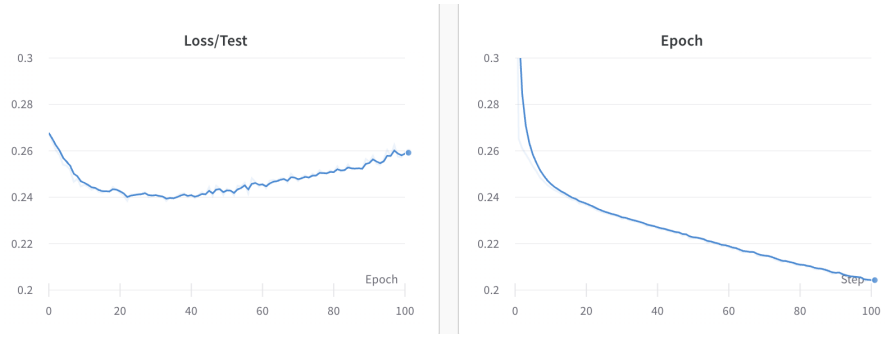



Figure 4.15: Training curve using the Graph Isomorphism Network

While fine-tuning the hyper-parameters, it was discovered that overfitting began to show at the 30-epoch threshold. Despite extensive efforts to mitigate this, including the use of BatchNorm, the incorporation of Dropout techniques, the adjustment of the learning rate, and the addition of 10,000 samples to the training dataset, the lowest recorded test error barely fell below 0.2. This means that, despite the GIN network's ability to nurture injective mappings inside non-isomorphism networks, a considerable percentage of samples remain difficult to differentiate unambiguously.

### 4.3.2 Graph Transformer

During the study of the GIN training loss, it was discovered that one issue potentially leading to increased loss and a proclivity to overfitting might be the GNN's inherent *permutation invariant property.*

Unfortunately, in our scenario, preserving the node sequence is crucial in deciding the eventual outcome. Consequently, despite employing a neural network and universal approximation to improve the GIN's optimal injective mapping capabilities, the employed aggregator maintained its permutation invariance, omitting a number of essential attributes essential to our problem. To address the challenges stemming from the permutation invariance issues, we turned our attention to the Graph Transformer (GT) architecture. When compared with the GIN, the GT embodies the subsequent enhancements:

- GT implements a global mechanism within graph data, similar to the attention mechanism observed in the Transformer architecture. In GT, this mechanism enables the dynamic weighting of node representations within a graph, allowing the model to selectively concentrate on various segments of the input graph. This, in turn, nurtures a more sophisticated comprehension of the fundamental patterns and associations present. Critically, by integrating position-aware encoding, this mechanism is anticipated to overcome the permutation invariance, thereby endowing each node with more vibrant and diverse representations.
- The GT architecture retains the use of the Message Passing Graph Neural Network MPGNN, but augments it with an enhanced version of GIN that also considers edges, namely, Graph Isomorphism Network with Edges (GINE). Specifically, it incorporates edge features into the MPGNN, fostering a heightened awareness and understanding of the graph structure.

The Upgrade function of a Graph Transformer could be summarized as below:

$$X^{(l+1)}, E^{(l+1)} = GPS^{(l)}(X^{(l)}, E^{(l)}, A)$$

In which the computation can be broken down into separate components: the determination of attention weights and the computation of embedding messages, respectively.:

$$\begin{aligned} X_M^{(l+1)}, E^{(l+1)} &= MPNN_e^{(l)}(X^{(l}E^{(l)}, A) \\ X_T^{(l+1)} &= GlobalAttn^{(l)}(X^{(l)}) \\ X^{(l+1)} &= MLP^{(l)}(X_M^{(l+1)} + X_T^{(l+1)}) \end{aligned}$$

The data flow within the architecture can be summarized in Fig.4.16:

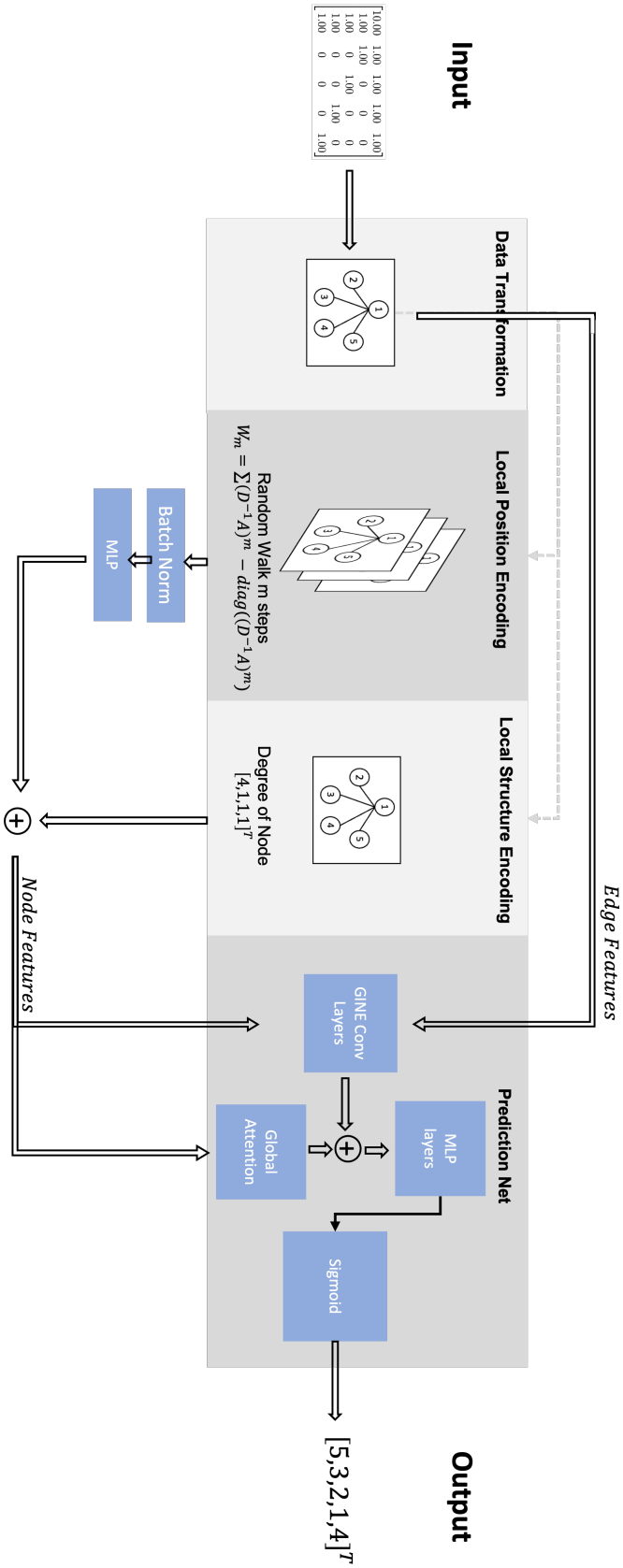



Figure 4.16: Overview of the pipeline using Graph Transformer

Contrary to the GIN methodology, the newly updated MPGNN module in the GT setup integrates edge attributes. Furthermore, before progressing to the MLP decoder, the concluding feature vectors are enhanced with supplementary attention weights, enabling the network to more accurately discern node sequences.

This time, we furthered our exploration with a training initiative using the GT structure and the same dataset previously examined in the GIN experiment. In detail, we employed 4,000 subgraph samples, each containing up to 20 nodes, for the training stage, and allocated 2,000 subgraphs for the evaluation segment. The initial outcomes of the training segment, encompassing 100 epochs, are illustrated in Fig 4.17.

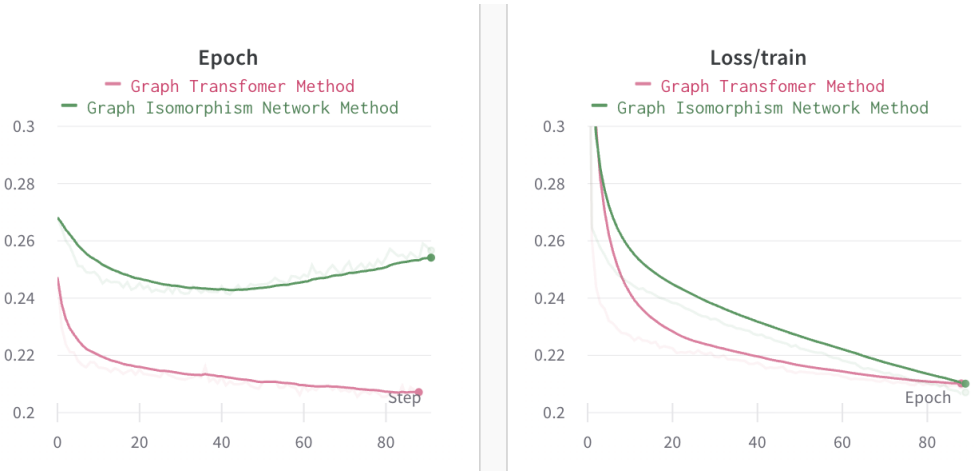



Figure 4.17: Training curve using the Graph Isomorphism Network and Graph Transformer

The inclusion of the global attention mechanism and edge features in the GT framework has notably enhanced the network's performance using the same training data and encoding information. Despite both the GT and GIN frameworks showing similar training losses after the first 100 epochs, the GT framework demonstrated superior test results without signs of potential overfitting. This suggests that while training losses are comparable, the GT framework's generalization ability for new tasks is significantly more robust. It reaffirms the vital role of each node's position in determining the optimal elimination rule in our case, emphasizing the improved effectiveness of the GT method.

# 5

# Experiment & Analysis



Contents



## 5.1 Pre-training

After deciding on the optimal training framework, we conducted a series of experiments to evaluate the effectiveness of the proposed method. In this section, we will detail the specifics of the experiments conducted with different sizes and configurations and emphasise the most effective configuration.

### 5.1.1 Dataset Configuration

We employed the process outlined in Algorithm 1 to create our dataset. All the authentic labels were selected from the top performers of the three heuristic algorithms: the Approximate Minimum Degree Algorithm [27], the Reverse Cuthill–McKee algorithm [24], and the Nested Dissection algorithm [28], utilizing the built-in implementation in MATLAB. We classified the dataset into 6 levels, grouped according to their node size. Refer to Table.5.2 for an overview.

| Dataset Name | Complete Graph | Sub Graph | avg nonzeros | Graph Size |
|---|---|---|---|---|
| NC_10 | 1,000 | 10,000 | 12.38 | 10 |
| NC_20 | 500 | 10,000 | 35.33 | 20 |
| NC_30 | 333 | 9,990 | 65.17 | 30 |
| NC_40 | 250 | 10,000 | 101.67 | 40 |
| NC_50 | 200 | 10,000 | 136.35 | 50 |
| NC_60 | 167 | 10,020 | 187.15 | 60 |

Table 5.1: The summary of dataset configurations

The term 'complete graph' refers to the initial graph produced by algorithm 1. Given the structure of our prediction pipeline and the learning objectives described in Section 3.1, the actual data input into the network consisted of subsets of graphs, with each intermediate graph documented during node elimination serving as a single training instance. In order to maintain a constant number of instances of actual data for training, variable quantities of complete graphs were generated based on the graph's size. Additionally, the term 'Avg Nonzero' represents the average number of edges present in a single complete graph, which correlates with the density and sparsity of the graph. Lastly, the size of the graph is determined by the number of nodes in a complete graph.

### 5.1.2 Training Configuration

We began the first training with NC_20 to determine the optimal hyper-parameters. A comprehensive search was conducted to identify the optimal learning rate, batch size, epoch count, and number of layers for the MLP utilised in the core of the Graph Transformer. Figure 5.1 summarises the search's findings:

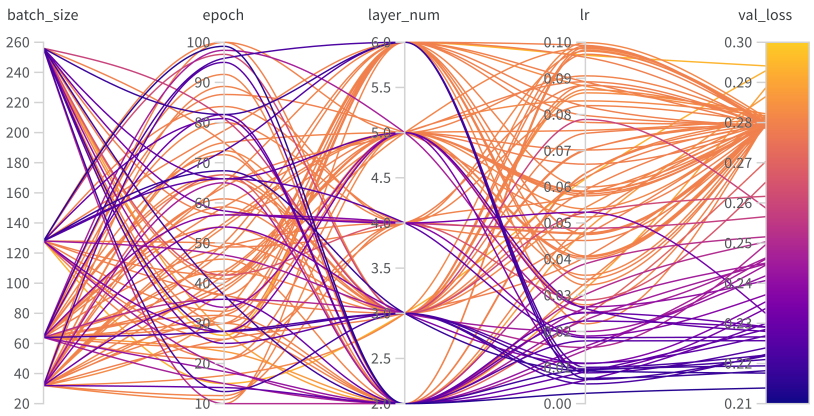



Figure 5.1: Summary of hyper-parameter searching

In accordance with the search results that yielded the lowest test loss, we established our

hyper-parameters as follows:

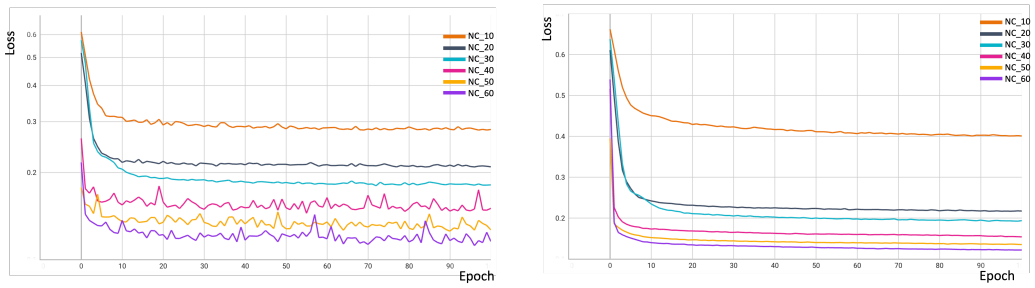
| Hyper-parameters | |
|---|---|
| # GT Layers | 3 |
| Hidden Dim | 64 |
| MPNN | GINE |
| GlobAttention | Transformer |
| # Heads | 4 |
| Dropout | 0.5 |
| Positional Encoding | Random Walk (RW) |
| PE dim | 10 |
| PE encoder | linear |
| Batch Size | 128 |
| Learning Rate | 3.00E-04 |
| # Epoch | 100 |
| Weight Decay | 1.00E-05 |
| # Parameters | 1282593 |

Table 5.2: Summary of hyper-parameters Settings



### 5.1.3 Result

Using the aforementioned generated six types of datasets, we carried out a training task on each using the previously described configuration, resulting in six models. The outcomes of the training are summarized in Fig.5.2.




(a) Test Loss (b) Train Loss

Figure 5.2: The training curve using on 6 different sizes datasets

All training sessions effectively converged within 100 epochs without any indications of overfitting. Intriguingly, the magnitude of the ultimate convergence point correlates directly with the size of the training set. To gain a deeper understanding of how loss values affect prediction quality, we expanded our testing to include the final prediction results for each of the six models.

For each of the six models, the prediction quality test procedures can be outlined as follows:

1. Selecting 200 complete graphs from the dataset and executing the node elimination individually on each graph.
2. For every graph, feed it into the respective models to acquire a predicted elimination order.
3. Implement node elimination as per the predicted sequence, keeping track of the new edges created until the last node is reached.
4. Repeat the node elimination process on the same graph, this time adhering to the ground-truth order and a random order, respectively, and recording the results.



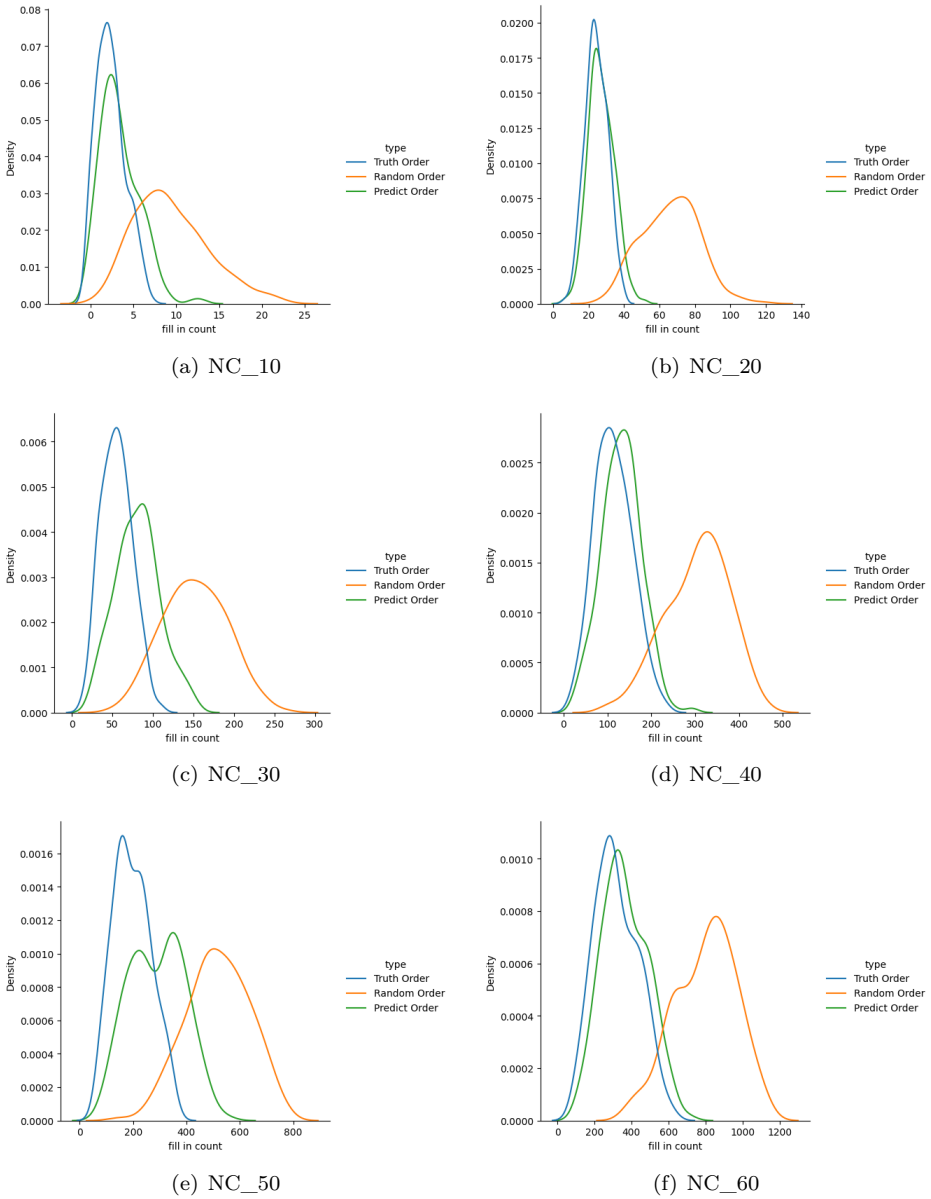
We conducted the prediction quality evaluation across all six models, yielding an overview of the fill-in outcomes generated by various methods. Refer to Table.5.3.

| Test Dataset | **NC_10** | **NC_20** | **NC_30** | **NC_40** | **NC_50** | **NC_60** |
|---|---|---|---|---|---|---|
| Ground-truth Order | | | | | | |
| Mean | 2.36 | 24.31 | 55.24 | 113.63 | 196.84 | 319.42 |
| Variance | 2.74 | 38.45 | 352.53 | 1,084.34 | 4,886.44 | 13,486.43 |
| Random Order | | | | | | |
| Mean | 9.38 | 66.62 | 151.71 | 304.38 | 517.76 | 799.13 |
| Variance | 18.43 | 262.24 | 1,590.48 | 5,240.6 | 13,959.85 | 27,007.31 |
| **KL Divergence** | **1.49** | **1.44** | **1.03** | **1.09** | **0.61** | **0.31** |
| Predict Order | | | | | | |
| Predictio Model | NC_10.pth | NC_20.pth | NC_30.pth | NC_40.pth | NC_50.pth | NC_60.pth |
| Mean | 3.42 | 27.41 | 81.42 | 133.63 | 289.83 | 366.42 |
| Variance | 6.51 | 53.43 | 745.83 | 1,804.34 | 10,319.73 | 13,830.31 |
| **KL Divergence** | **0.47** | **0.09** | **0.36** | **0.18** | **0.35** | **0.0006** |

Table 5.3: The distributions of fill-in resulted by different methods and on different datasets

The KL Divergence value in the table signifies the discrepancy between the ground-truth distribution and both the Random Order Distribution and the predicted distribution. Here, smaller values indicate a closer resemblance between the distributions, considering them as normal distributions.

Notably, all six models exhibited improved performance, shifting the fill-in distribution noticeably closer to the ground-truth distribution from the random one, indicating that the models learned the optimal algorithm's elimination rules and patterns to a certain extent. Despite expectations, data size increase didn't significantly enhance performance, diverging from the trends noted in the training loss curve. This could be attributed to the Binary Cross Entropy loss function structure: each subgraph, irrespective of its size, contains only one positive label in our setup, making larger subgraphs yield sparser ground-truth matrices and consequently smaller cross entropy values in the loss. Yet, as shown in Table.5.2, this didn't affect the final prediction quality. Further analysis could involve visualizing these distributions in histograms.




(a) NC_10 (b) NC_20

(c) NC_30 (d) NC_40

(e) NC_50 (f) NC_60

Figure 5.3: The distributions of fill-in

As shown in Fig.5.3, despite using the same strategy across all six models, prediction quality varied significantly between different sizes. To identify the best model and reduce instability, we further analyzed each model's predictive performance. We classified the prediction results into three categories: 1) cases where prediction quality surpassed the ground-truth, 2) cases lagging behind the ground-truth, and 3) total instances performing better than the random method, as detailed in Table.5.4.

In Table.5.4, the "discrepancy count" signifies the difference between the fill-in values derived from predictions and the ground-truth, while the "percentage" illustrates the proportion of this

| Graph Size | Fill-in discrepancy count | | | | | |
|---|---|---|---|---|---|---|
| | Better or equivalent to the standard solution. | | Behind the standard solution | | | Better than Random |
| | <0 | =0 | <=10% | <=20% | >20% | |
| NC_10 | 2 | 122 | 0 | 1 | 75 | 71 |
| **NC_20** | **22** | **53** | **47** | **29** | **49** | **125** |
| NC_30 | 0 | 0 | 1 | 12 | 187 | 200 |
| NC_40 | 4 | 1 | 51 | 77 | 67 | 195 |
| NC_50 | 0 | 0 | 2 | 7 | 191 | 200 |
| NC_60 | 1 | 0 | 46 | 64 | 56 | 166 |

Table 5.4: The discrepancy of fill-in count



difference relative to the fill-in from the prediction. The outcomes in each row are generated from predictions using various models.

As emphasized in the second row, it is apparent that the model trained on graphs with 20 nodes delivers the most precise results, with a substantial number of predictions surpassing the ground-truth benchmarks. Nonetheless, it also reveals that in 75 out of 200 cases, the predictive model's outcomes were behind those achieved through random ordering. Conversely, the NC_50 model, despite the noticeable discrepancy in fill-in distribution depicted in Fig.5.3, consistently predicts a more favourable ordering compared to a random sequence.

The reason for the diverse performances exhibited by these six models is the varying complexities and structures present in the respective training datasets. One of them might have become overly accustomed to a particular graph structure, resulting in overfitting. However, despite these considerations, the $NC_20.pth$ model retains its position as our preferred choice for ongoing enhancements, due to its demonstrated high adaptability to the training data with the existing setup.

## 5.2 Curriculum Learning

Following a detailed search and analysis in the initial training phase, we have successfully established the model *NC_10.pth* as the baseline, initially developed on graphs comprising 20 nodes.

In order to further increase the model's efficacy and adaptability to manage more complex data, we turned our attention to implementing Curriculum Learning. This corse-to-fine strategy in which learning evolves from basic to more complex concepts, has the potential to keep the advantage e.g. prediction accuracy of our baseline NC_20.pth model but also is promising to enable the model to adapt to a more complete graph structure.

In accordance with this, we chose to include the NC_50 dataset in the training regimen. Previous evaluations have indicated that models adjusted with this dataset demonstrate a high degree of generalisation, thereby functioning as a supplemental asset to bolster the capabilities of the NC_20.pth models.

It is pleasing to note, as depicted in Fig.5.4, that the pre-trained NC_20.pth model not only retains its foundational strengths but also exhibits significant enhancements when trained with the intricately structured 50-node size data. This endeavour is consistent with the tenets of Curriculum Learning, as the model is deftly navigating through more complex terrains, thereby indicating substantial improvements in its predictive capabilities.



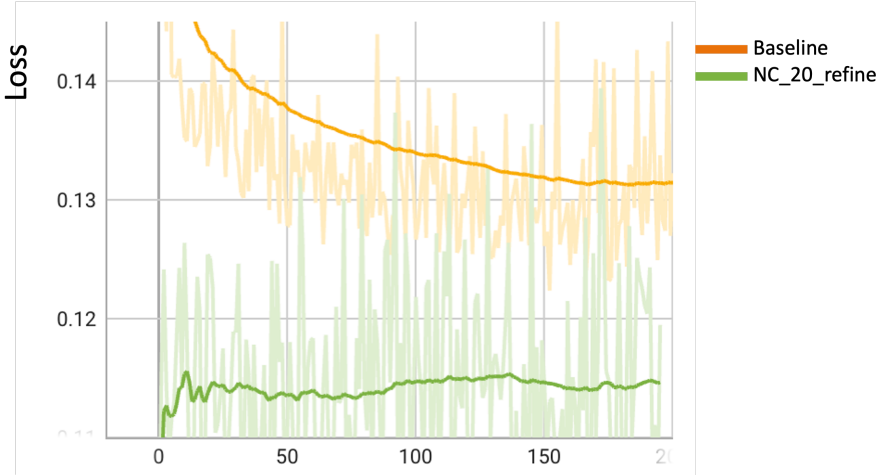



Figure 5.4: The curriculum learning for NC_20ṗth on NC_50 data

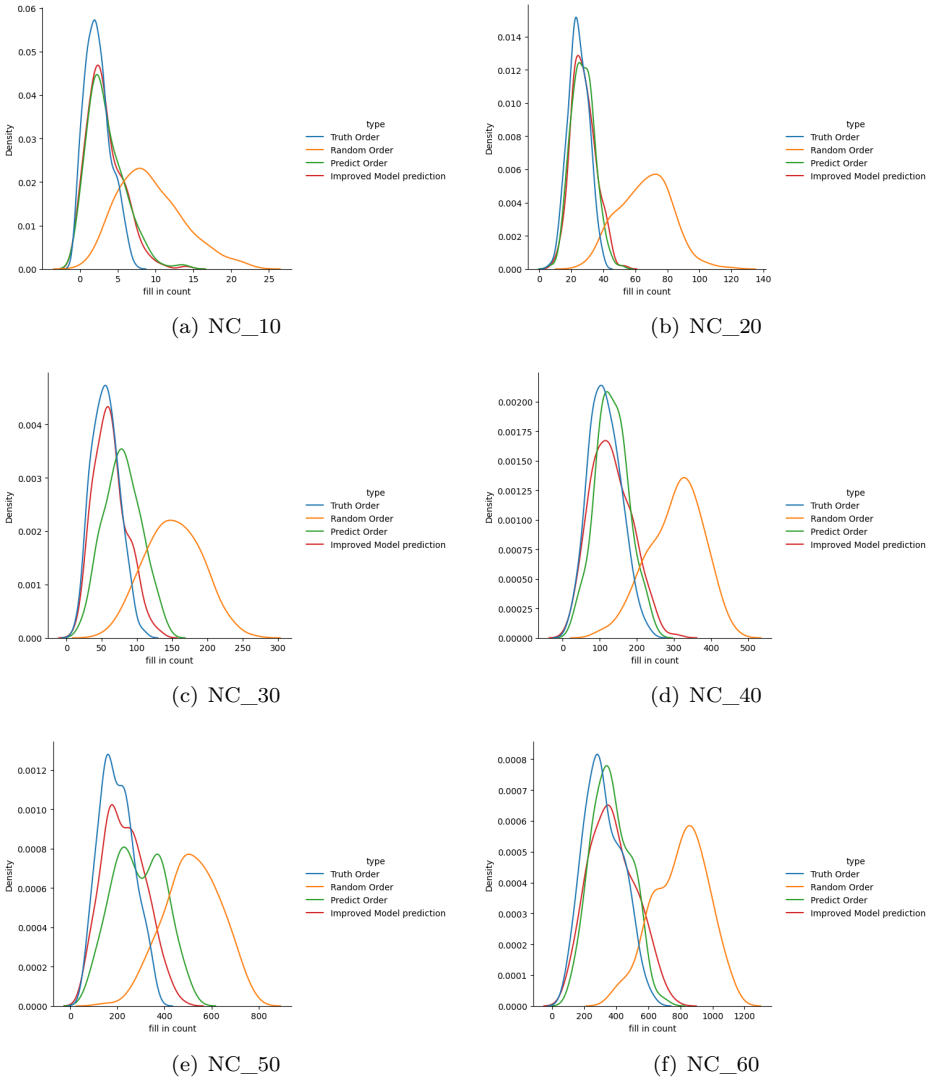
Upon securing the refined final model, we proceeded to evaluate it on each of the six previously mentioned datasets, comparing its performance to that of the models originally trained on their respective datasets.

As illustrated in Fig. 5.5, the upgraded model exhibits clear advancements post the implementation of curriculum learning. We analyzed the fill-in outcomes originating from four different sources for each of the six datasets:

1. The outcomes derived from the ground-truth order (Blue)

2. Those resulting from a random order (Orange)

3. The outcomes achieved through predictions from pretrained models, which were originally trained on their respective local datasets (Green)




(a) NC_10 (b) NC_20

(c) NC_30 (d) NC_40

(e) NC_50 (f) NC_60

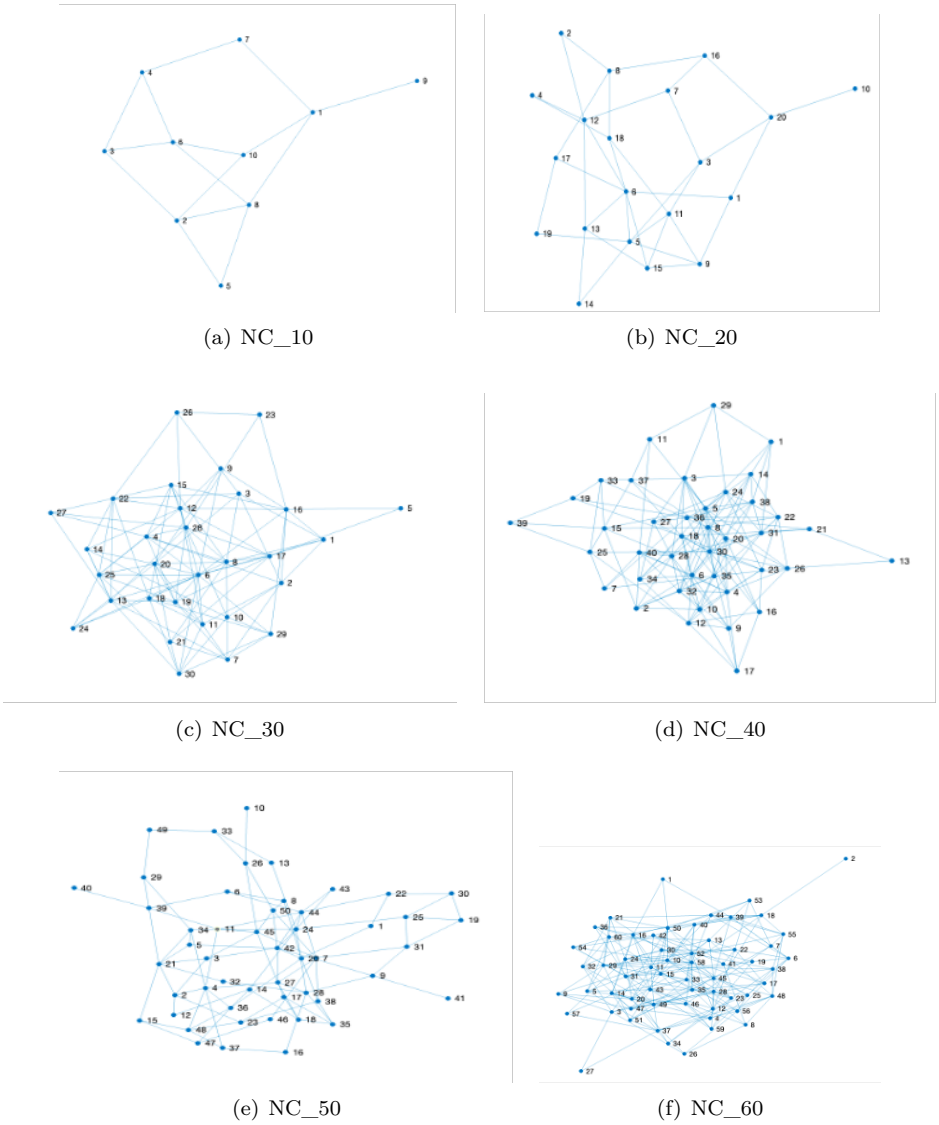
Figure 5.5: The fill-in distribution comparison between 1. the ground-truth order, 2. the random order, 3. the order from local trained model, 4. the order from the fine-tuned model

4. The outcomes achieved through predictions from the revamped model, NC_20_refine.pth, which was developed based on the pre-trained model NC_20.pth (Red).

Remarkably, across all dataset test cases, the enhanced model consistently yielded predictions that were comparable or superior to those generated by the locally pre-trained models. This was particularly evident in case (c) and case (e), where the refined model demonstrated a significantly improved distribution. Moreover, in all test cases, the refined model managed to elevate the quality of the results without compromising the efficacy of the pre-trained outcomes.

Taking a closer look at the datasets, Fig.5.6 displays a randomly selected sample from each of the six datasets, presenting the distinct differences in both size and graph density. While the

refined model was primarily honed using the NC_20 dataset for pre-training and further fine-tuned with the NC_50 dataset, it wasn't familiarized with the remaining datasets. Nonetheless, it demonstrated impressive performance across all tests, even when scaling up to larger sizes like the NC_60. This consistent performance across varying datasets underscores the refined model's robustness and generalization capabilities.



(a) NC_10 (b) NC_20

(c) NC_30 (d) NC_40

(e) NC_50 (f) NC_60

Figure 5.6: Samples with different sizes

5

# Conclusions

In this study, we initiated research into a persistent puzzle within the field of applied mathematics, known as the *mimimum fill-in* problem. The aim of our study was to discover new methodologies for solving this puzzle, which has a significant impact on the underlying mathematical algorithms that are limited by it. Enhancing the solution to the minimum fill-in problem can have a direct impact on the effectiveness of solving linear equations. This advancement is particularly significant in the context of the information age when a single application has the potential to create a vast number of linear equations.

In spite of the enduring barrier posed by its NP-complete complexity, the recent efforts in using artificial intelligence (AI) for scientific pursuits have shed light on novel approaches. In recent years, there has been a considerable increase in endeavours to leverage artificial intelligence (AI) and machine learning (ML) approaches in addressing these mathematical challenges. Noteworthy examples include the application of deep reinforcement learning techniques to discover new patterns in matrix multiplication and sorting algorithms. Motivated by the observed progress and potential outcomes in these endeavours, we were inspired to undertake research aimed at utilising artificial intelligence (AI) and machine learning (ML) techniques to address this specific challenge.

In order to comprehend the complexities of the problem, we first examined its deep connections to graph theory. This analysis revealed a clear parallel between the Gaussian Elimination method in linear equations and the process of node elimination in graph triangulation. As a result, we proceeded to translate the matter into its corresponding problem in the field of graph theory, with a particular focus on determining the most efficient sequence of node eliminations during the chordal completion phase.

Graph Neural Networks (GNNs) then become the preferred framework for addressing this intricate difficulty because of their track records in processing unstructured data. The ability to effectively learn feature representations is crucial in the use of GNN. In this particular scenario, our objective is to utilise GNN in order to predict an ideal sequence for eliminating nodes in each given input graph.

Subsequently, we elaborated on our approach to establishing a GNN pipeline. Recognising the

considerable design possibilities offered by the GNN architecture and recognising the significant influence of practical contexts on GNN models, we organised our methodology into three phases: formulating the inference pipeline, constructing the dataset pipeline, and ultimately determining the GNN architectural selection.

During this developmental journey, we encountered and addressed two significant challenges: First, our experimental findings indicate that the attempt to forecast the sequence of all nodes simultaneously, referred to as node-level regression, exhibited a significant vulnerability to noise, leading to completely impractical outcomes. As a result, we devised a strategy to streamline the effort by breaking down the process into a sequence of iterative prediction phases. In this approach, the model's objective is to determine the most likely node to choose at each decision point, therefore transforming the problem into a binary classification task at the node level.

Moreover, we realized that the feature representation's expressiveness was greatly hindered by the graph's isomorphic structure, as evidenced by the model's inability to differentiate between a certain subset of nodes during single-graph experiments. To enhance functionality, we devised two approaches:

1. Through the ablation experiment, we formulated custom feature encoding schemes, utilizing the final fill-in outcome as a selection benchmark. This allowed us to validate an initial encoding scheme that incorporated local positional encoding through the Random Walk Matrix, coupled with a local structure encoding technique utilizing node degree information.

2. Parallel to the positional encoding, we devised a Graph Transformer structure, which empirical comparisons demonstrated to be the most expressive network structure, thereby bolstering our system's performance and efficacy.

After finalising our choices for the dataset and architectural designs, we began the training procedure for our Graph Transformer-based network. The training routine may be divided into two key parts.

1. Initially, we created an extensive array of datasets encompassing a range of sizes, from 10 nodes up to 60 nodes. We undertook training exercises individually on each set, eventually determining that the model honed with the 20-node-sized dataset served as an optimal foundational model.

2. In the succeeding step, we persisted in the training of this foundational model, progressively incorporating datasets with escalating complexity. This approach culminated in the development of our ultimate model.

In comparison to locally pre-trained models, our final iteration showcased good generalization and robustness across a range of six distinct dataset sizes. Additionally, our techniques assisted in revealing new permutation sequences, indicating a potential path for continued progress in this domain.

# A

A

# Additional Information

## A.1 Software and hardware specifications

| Hardware | |
|---|---|
| CPU | Intel(R) Xeon(R) Gold 5320 CPU @ 2.20GHz |
| GPU | NVIDIA GeForce RIX 4090 |
| Video Memory | 24,564 MB |
| RAM | 128,416 MB |
| Software | |
| MATLAB | 2023a |
| PyTorch | 2.0.1 |
| PyG | 2.3 |
| Python | 3.8 |

## A.2 Data flows in the complete network

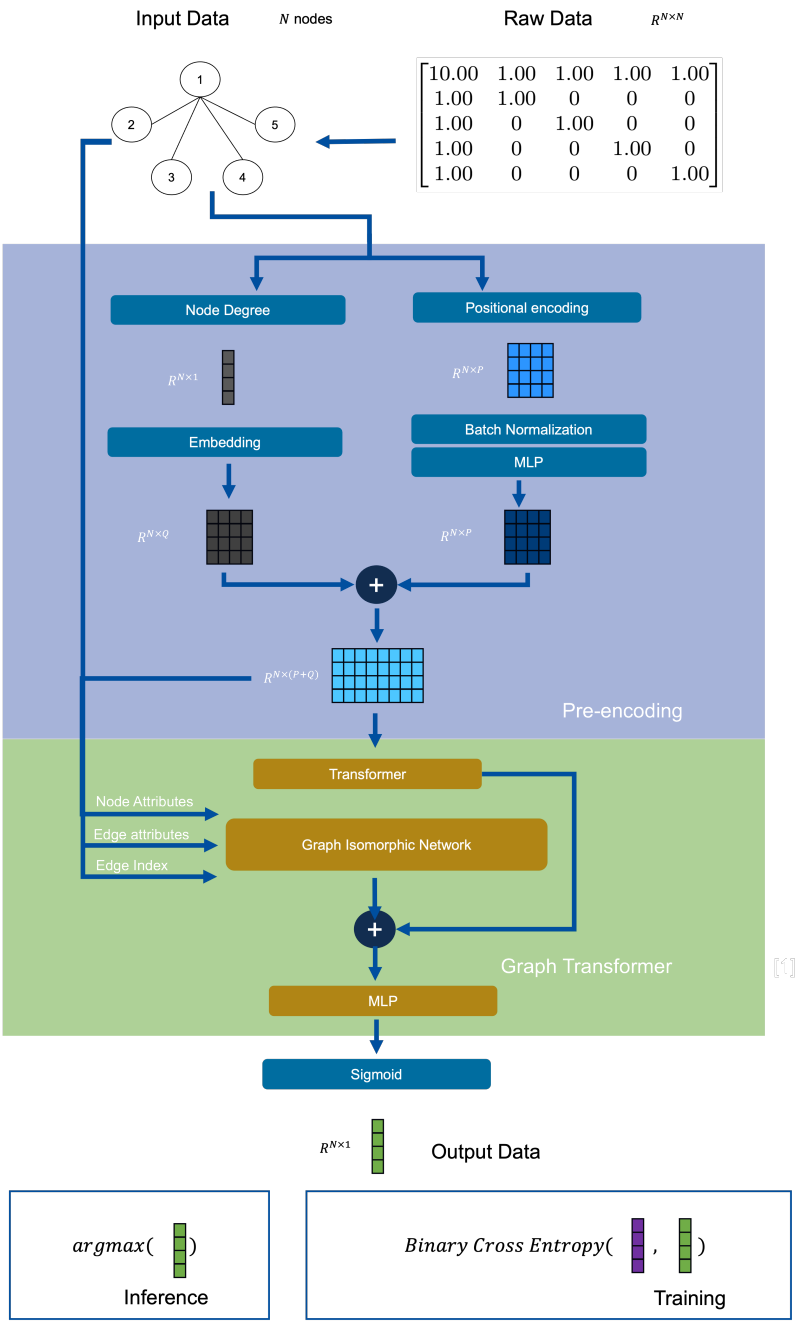
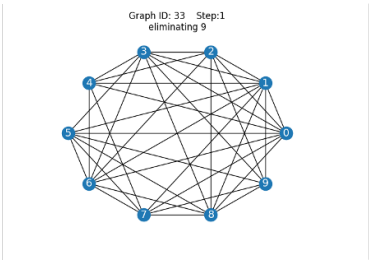



Figure A.1: Data flows in the complete network

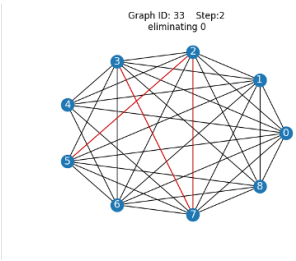
## A.3 An end-to-end example

| | |
|---|---|
| Inout Node Size | 10 |
| Ground-truth Order | [9, 8, 4, 3, 1, 7, 0, 2, 6, 5] |
| Predicting Order | [9, 0, 1, 2, 3, 4, 5, 6, 8, 7] |
| Fill in | |
| Approximate Minimum Degree | 5 |
| Nested Dissection | 5 |
| Reverse Cuthill-McKee | 5 |
| **Graph Transformer Prediction** | **3** |

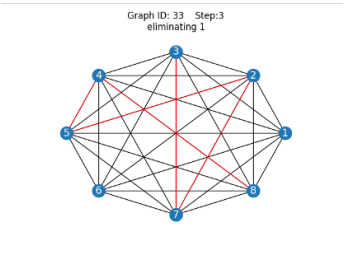
### A.3.1 Node Elimination by ground-truth order



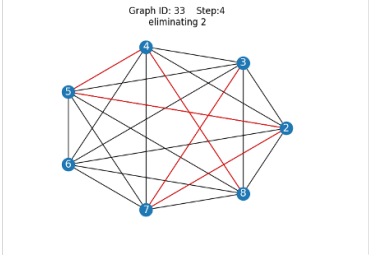

(a) step 1



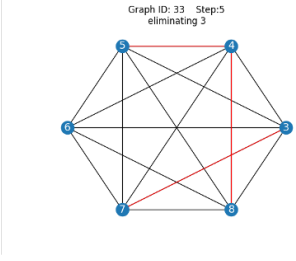

(b) step 2



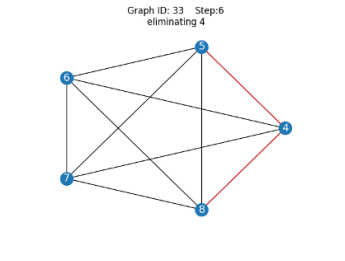

(c) step 3



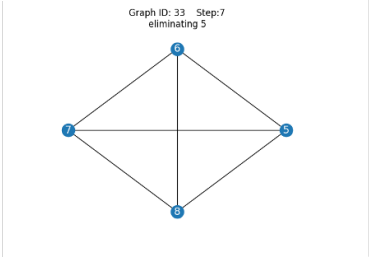

(d) step 4



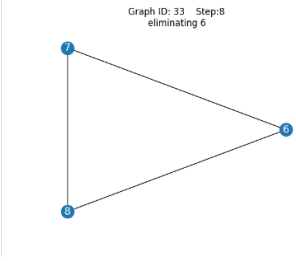

(e) step 5



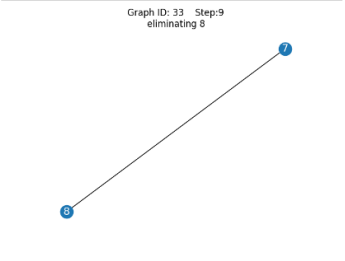

(f) step 6



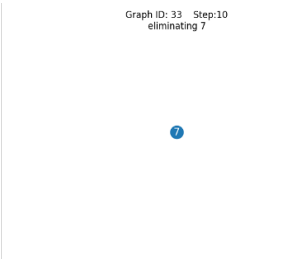

(g) step 7




(h) step 8

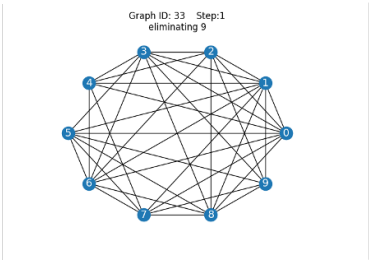



(i) step 9

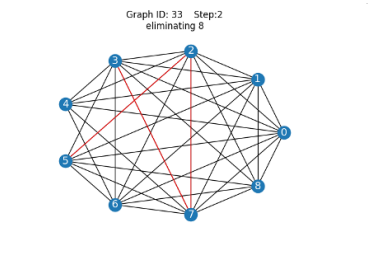



(j) step 10

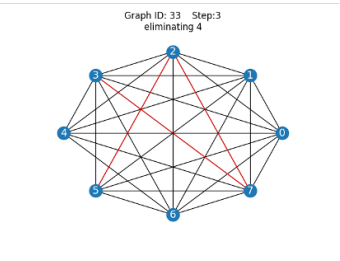
### A.3.2 Node Elimination by the predict order



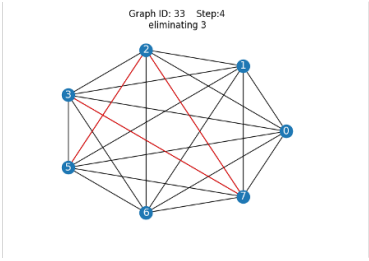

(k) step 1



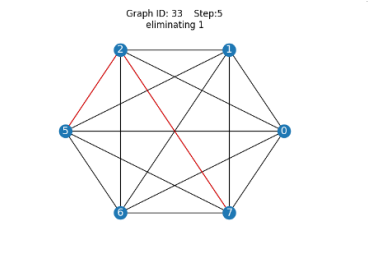

(l) step 2



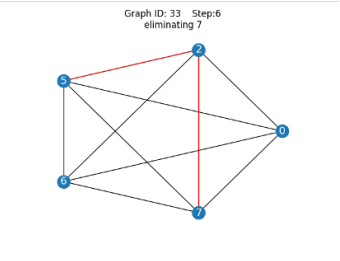

(m) step 3



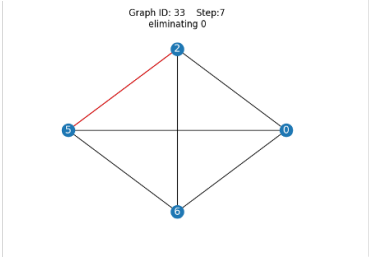

(n) step 4



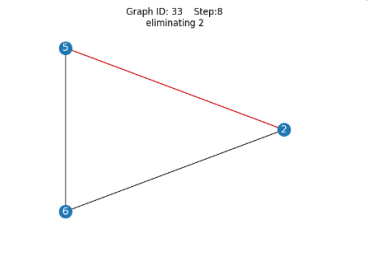

(o) step 5



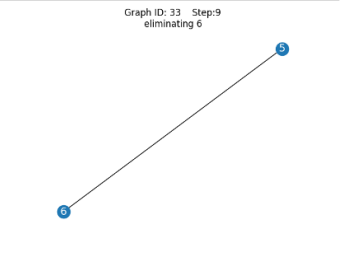

(p) step 6



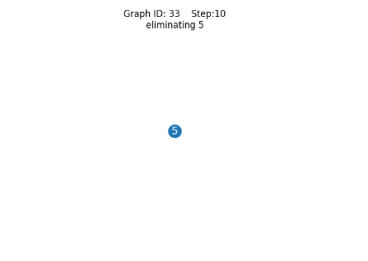

(q) step 7




(r) step 8




(s) step 9




(t) step 10

## A.4 Useful Links

- The collection of search results that outperforms the best heuristic algorithms
- The Project Code

A

# Bibliography


[1] J. Jumper, R. Evans, A. Pritzel, *et al.*, "Highly accurate protein structure prediction with alphafold," *Nature*, vol. 596, no. 7873, pp. 583–589, 2021.

[2] Z. Azerbayev, B. Piotrowski, H. Schoelkopf, E. W. Ayers, D. Radev, and J. Avigad, "Proofnet: Autoformalizing and formally proving undergraduate-level mathematics," *arXiv preprint arXiv:2302.12433*, 2023.

[3] Y. Rong, T. Xu, J. Huang, *et al.*, "Deep graph learning: Foundations, advances and applications," in *Proceedings of the 26th ACM SIGKDD international conference on knowledge discovery & data mining*, 2020, pp. 3555–3556.

[4] A. Vaswani, N. Shazeer, N. Parmar, *et al.*, "Attention is all you need," *Advances in neural information processing systems*, vol. 30, 2017.

[5] J. Sirignano and K. Spiliopoulos, "Dgm: A deep learning algorithm for solving partial differential equations," *Journal of computational physics*, vol. 375, pp. 1339–1364, 2018.

[6] Y. Chen, E. Huerta, J. Duarte, *et al.*, "A fair and ai-ready higgs boson decay dataset," *Scientific Data*, vol. 9, no. 1, p. 31, 2022.

[7] A. Fawzi, M. Balog, A. Huang, *et al.*, "Discovering faster matrix multiplication algorithms with reinforcement learning," *Nature*, vol. 610, no. 7930, pp. 47–53, 2022.

[8] V. Strassen *et al.*, "Gaussian elimination is not optimal," *Numerische mathematik*, vol. 13, no. 4, pp. 354–356, 1969.

[9] D. J. Mankowitz, A. Michi, A. Zhernov, *et al.*, "Faster sorting algorithms discovered using deep reinforcement learning," *Nature*, vol. 618, no. 7964, pp. 257–263, 2023.

[10] M. Yannakakis, "Computing the minimum fill-in is np-complete," *SIAM Journal on Algebraic Discrete Methods*, vol. 2, no. 1, pp. 77–79, 1981.

[11] F. V. Fomin, D. Kratsch, I. Todinca, and Y. Villanger, "Exact algorithms for treewidth and minimum fill-in," *SIAM Journal on Computing*, vol. 38, no. 3, pp. 1058–1079, 2008.

[12] S. L. Lauritzen and D. J. Spiegelhalter, "Local computations with probabilities on graphical structures and their application to expert systems," in *Readings in uncertain reasoning*, 1990, pp. 415–448.

[13] D. J. Rose, "A graph-theoretic study of the numerical solution of sparse positive definite systems of linear equations," in *Graph theory and computing*, Elsevier, 1972, pp. 183–217.

[14] J. A. Tropp, A. Yurtsever, M. Udell, and V. Cevher, "Practical sketching algorithms for low-rank matrix approximation," *SIAM Journal on Matrix Analysis and Applications*, vol. 38, no. 4, pp. 1454–1485, 2017.

[15] K. L. Clarkson and D. P. Woodruff, "Low-rank approximation and regression in input sparsity time," *Journal of the ACM (JACM)*, vol. 63, no. 6, pp. 1–45, 2017.

[16] P. Indyk, A. Vakilian, and Y. Yuan, "Learning-based low-rank approximations," *Advances in Neural Information Processing Systems*, vol. 32, 2019.

[17] V. Y. Pan, "Fast feasible and unfeasible matrix multiplication," *arXiv preprint arXiv:1804.04102*, 2018.

[18] L.-H. Lim, "Tensors in computations," *Acta Numerica*, vol. 30, pp. 555–764, 2021.

[19] D. Silver, T. Hubert, J. Schrittwieser, *et al.*, "Mastering chess and shogi by self-play with a general reinforcement learning algorithm," *arXiv preprint arXiv:1712.01815*, 2017.

[20] W. Dabney, M. Rowland, M. Bellemare, and R. Munos, "Distributional reinforcement learning with quantile regression," in *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 32, 2018.

[21] P. M. Phothilimthana, A. Thakur, R. Bodik, and D. Dhurjati, "Scaling up superoptimization," in *Proceedings of the Twenty-First International Conference on Architectural Support for Programming Languages and Operating Systems*, 2016, pp. 297–310.

[22] P. M. Phothilimthana, T. Jelvis, R. Shah, N. Totla, S. Chasins, and R. Bodik, "Chlorophyll: Synthesis-aided compiler for low-power spatial architectures," *ACM SIGPLAN Notices*, vol. 49, no. 6, pp. 396–407, 2014.

[23] D. Bundala and J. Závodnỳ, "Optimal sorting networks," in *International Conference on Language and Automata Theory and Applications*, Springer, 2014, pp. 236–247.

[24] E. CUTHILL, *Several strategies for reducing the bandwidth of matrices, sparse matrices and their applications, dj rose and ra willoughby, eds*, 1972.

[25] A. George, J. Liu, and E. Ng, "Computer solution of sparse linear systems," *Oak Ridge National Laboratory*, 1994.

[26] H. M. Markowitz, "The elimination form of the inverse and its application to linear programming," *Management Science*, vol. 3, no. 3, pp. 255–269, 1957.

[27] P. R. Amestoy, T. A. Davis, and I. S. Duff, "An approximate minimum degree ordering algorithm," *SIAM Journal on Matrix Analysis and Applications*, vol. 17, no. 4, pp. 886–905, 1996.

[28] R. J. Lipton, D. J. Rose, and R. E. Tarjan, "Generalized nested dissection," *SIAM journal on numerical analysis*, vol. 16, no. 2, pp. 346–358, 1979.

[29] B. W. Kernighan and S. Lin, "An efficient heuristic procedure for partitioning graphs," *The Bell system technical journal*, vol. 49, no. 2, pp. 291–307, 1970.

[30] S. Pissanetzky, *Sparse matrix technology-electronic edition*. Academic Press, 1984.

[31] M. T. Heath and P. Raghavan, "A cartesian parallel nested dissection algorithm," *SIAM Journal on Matrix Analysis and Applications*, vol. 16, no. 1, pp. 235–253, 1995.

[32] M. Fiedler, "Algebraic connectivity of graphs," *Czechoslovak mathematical journal*, vol. 23, no. 2, pp. 298–305, 1973.

[33] S. Parter, "The use of linear graphs in gauss elimination," *SIAM review*, vol. 3, no. 2, pp. 119–130, 1961.

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

[35] J. Leskovec, *Cs224w*, 2021.

[36] K. Xu, W. Hu, J. Leskovec, and S. Jegelka, "How powerful are graph neural networks?" *arXiv preprint arXiv:1810.00826*, 2018.

[37] K. Hornik, M. Stinchcombe, and H. White, "Multilayer feedforward networks are universal approximators," *Neural networks*, vol. 2, no. 5, pp. 359–366, 1989.

[38] J. Zhou, G. Cui, S. Hu, *et al.*, "Graph neural networks: A review of methods and applications," *AI open*, vol. 1, pp. 57–81, 2020.

[39] S. P. Kolodziej, M. Aznaveh, M. Bullock, *et al.*, "The suitesparse matrix collection website interface," *Journal of Open Source Software*, vol. 4, no. 35, p. 1244, 2019.

[40] C. W. Wu, "Algebraic connectivity of directed graphs," *Linear and multilinear algebra*, vol. 53, no. 3, pp. 203–223, 2005.

[41] L. Rampášek, M. Galkin, V. P. Dwivedi, A. T. Luu, G. Wolf, and D. Beaini, "Recipe for a general, powerful, scalable graph transformer," *Advances in Neural Information Processing Systems*, vol. 35, pp. 14 501–14 515, 2022.