Information-Theoretic Analysis of the Complexity and Compression of Network Topologies



Amirmohammad Farzaneh

Mansfield College

Department of Engineering Science, University of Oxford

Supervised by Professor Justin P. Coon

A thesis submitted to the University of Oxford for the degree of Doctor of Philosophy in Engineering Science

Hilary Term 2024

Abstract

Thesis Title: Information-Theoretic Analysis of the Complexity and Compression of Network Topologies
Name: Amirmohammad Farzaneh
College: Mansfield College
Degree: Doctor of Philosophy
Term and Year of Submission: Hilary Term 2024

Networks around us are growing larger than ever. The rapid growth of communication networks can clearly be seen in the emergence of 5G and the movement towards 6G, in which the connection density can be as high as ten million devices per square kilometer. This growth is not limited to communication networks and can be seen in other types of networks such as social networks etc. Graph theory has always been the primary mathematical tool for simulating networks. Traditional ways of storing graphs are known to be very complex, even for small graphs. When faced with graphs with millions of nodes, we are inevitably faced with questions about the information content inherent in their topology. We need to answer questions such as how complex these giant graphs are, and how much we are able to compress them. To answer questions like this, we use the help of information theory. Information theory is one of the most powerful tools to analyse the information content of random variables. Treating network graphs as random variables, we start by categorizing them into three main groups based on their behaviour in real-life scenarios. While Shannon's entropy is studied for classic random graph generators, we use Kolmogorov complexity to study the complexity of static networks, and entropy rate to model and study dynamic networks. After studying the complexity of graphs, we move on to study their compression to their theoretical limit. Because of their vast application in different domains, we mainly study optimal tree compression. Finally, the application of our findings to two major engineering problems is highlighted: network routing, and graph neural networks. It is shown how both of these fields can benefit from graph compression and an information-theoretic analysis of the complexity of graphs.

Acknowledgements

First and foremost, I extend my profound gratitude to Professor Justin Coon, my DPhil supervisor, whose unwavering guidance and support served as a cornerstone throughout my doctoral journey. From our initial collaboration during the first interview, where we jointly tackled a research problem directly tied to my DPhil, to the culmination of my studies, Professor Coon consistently demonstrated a genuine commitment to my academic growth. His dedicated attention to each of my concerns, coupled with his efforts to promptly address them, greatly contributed to the successful milestones I achieved during my DPhil studies. I am unequivocally indebted to his invaluable aid and unwavering support.

I must also highlight the invaluable assistance and support provided by Dr. Mihai-Alin Badiu. His consistent guidance and insightful advice were instrumental in shaping the trajectory of many of my research projects. A substantial portion of the progress achieved during my DPhil studies can be attributed to his unwavering support and expertise. I would also like to thank Dr. Ferhat Yarkin, Dr. Zijing Wang, and my other fellow research group colleagues for their continuous support and assistance throughout my academic journey.

I would like to express my deepest gratitude to my two examiners, Dr. Jakob Foerster, my internal examiner, and Dr. Vincenzo Nicosia, my external examiner, for their invaluable feedback and insightful comments throughout the review process. Their constructive critiques and thoughtful suggestions have significantly improved the quality and clarity of this thesis. I am truly appreciative of the time and effort they dedicated to this work, and their guidance has been instrumental in refining my

research.

My sincere gratitude extends to Mansfield College and Moogsoft Inc. for their generous financial support during my DPhil studies. The continuity of my academic pursuits would have been unattainable without their benevolent contributions, which played a pivotal role in facilitating my research journey.

I extend my heartfelt appreciation to the cherished friends I made in Oxford, who crafted a second home away from home for me. Their constant care and kindness alleviated the anxieties of being away from home, transforming Oxford into my newfound sanctuary. To Elliot, Thizi, Chris, Caroline, and all my other dear Oxford friends, thank you for seamlessly evolving into my second family. Each one of you contributed to making my time in Oxford an enriching experience filled with camaraderie and support.

Ultimately, I wish to convey my deepest gratitude to my parents and my two sisters for showering me with their unwavering love and support throughout this transformative journey. Their unconditional commitment has been my steadfast anchor, ensuring that not a single moment was marred by loneliness or lack of support. Despite the physical distance, they consistently went above and beyond to ensure the smoothness of my life and studies, and for that, I am eternally indebted to them.

List of Publications

Journal Publications

1. **A. Farzaneh,** J. P. Coon, and M. -A Badiu. 2021. "Kolmogorov Basic Graphs and Their Application in Network Complexity Analysis" *Entropy* 23, no. 12: 1604. **Featured as the issue's cover story**

2. **A. Farzaneh,** J. P. Coon, "An information theory approach to network evolution models," *Journal of Complex Networks*, Volume 10, Issue 3, June 2022.

Conference and Workshop Publications

1. **A. Farzaneh,** "An Information-Theoretic Analysis of Temporal GNNs," accepted to 2024 IEEE Information Theory Workshop (ITW), Shenzhen, China.

2. **A. Farzaneh** and J. P. Coon, "An information-theoretic analysis on temporal graph evolution," In Temporal Graph Learning Workshop @ NeurIPS 2023, 2023.

3. **A. Farzaneh,** M. -A. Badiu and J. P. Coon, "Practical Random Tree Generation: Information-Theoretic Analysis and Applications to IoT," 2023 IEEE Conference on Standards for Communications and Networking (CSCN), Munich, Germany, 2023. **Winner of student travel grant**

4. **A. Farzaneh** and J. P. Coon, "Practical random tree generation using spanning trees: Entropy and compression," In ICML 2023 Workshop Neural Compression: From Information Theory to Applications, 2023.

5. **A. Farzaneh**, M. -A. Badiu and J. P. Coon, "LEAST: a Low-Energy Adaptive Scalable Tree-based routing protocol for Wireless Sensor Networks," presented at 2023 Wireless Telecommunications Symposium, Boston, US, 2023.

6. **A. Farzaneh,** M. -A. Badiu and J. P. Coon, "TreeExplorer: a coding algorithm for rooted trees with application to wireless and ad hoc routing," 2022 IEEE 96th Vehicular Technology Conference (VTC2022-Fall), London, United Kingdom, 2022, pp. 1-5.

Acronyms

AEP	Asymptotic Equipartition Property
AMIR	Alternative Mutual Information Rate
BA	Barabási–Albert
BFS	Breadth-First Search
BS	Base Station
CDMA	Code Division Multiple Access
CH	Cluster Head
CTMC	Continuous Time Markov Chain
DFS	Depth-First Search
ER	Erdős–Rényi
GNN	Graph Neural Network
HMM	Hidden Markov Model
HN	Host Node
K-complexity	Kolmogorov complexity
LEACH	Low-Energy Adaptive Clustering Hierarchy
LEAST	Low-Energy, Adaptive, Scalable, and Tree-based
LZ	Lempel-Ziv
LZW	Lempel-Ziv-Welch
MAC	Multiple Access Communications
MIR	Mutual Information Rate
ML	Machine Learning
NEC	Network Evolution Chain
PC	Pit-Climbing
SBM	Stochastic Block Model
SGT	Simply Generated Tree
SN	Sink Node
TD	Tunnel-Digging
TDMA	Time Division Multiple Access
TPC	Ternary Pit-Climbing
TTD	Ternary Tunnel-Digging
WSN	Wireless Sensor Network

Contents

Abstract i

Acknowledgements ii

List of Publications iv

Acronyms vi

List of Figures xi

List of Tables xiii

List of Mathematical Notations xiv

1 Introduction 1
- 1.1 Complexity analysis of networks . . . 3
  - 1.1.1 Deterministic approaches . . . 4
  - 1.1.2 Probabilistic approaches . . . 5
- 1.2 Optimal graph compression . . . 6
- 1.3 Applications of network complexity analysis and compression . . . 7
  - 1.3.1 Routing protocols in WSNs . . . 7
  - 1.3.2 Temporal GNNs . . . 8
- 1.4 Thesis contributions and organisation . . . 9

2 Encoding Graphs 12
- 2.1 Overview of random graph models . . . 13
- 2.2 Background . . . 15
  - 2.2.1 Entropy of graphs . . . 15
  - 2.2.2 Graph compression . . . 18
    - 2.2.2.1 Classic graph representation methods . . . 20
    - 2.2.2.2 Rearranging the adjacency matrix . . . 21
    - 2.2.2.3 Exploring the graph . . . 21
    - 2.2.2.4 Dictionary-based coding . . . 22
    - 2.2.2.5 Query-based compression . . . 22
    - 2.2.2.6 Structure-based compression . . . 22
- 2.3 Information-theoretic basics and definitions . . . 23

2.3.1 Graph entropy . . . 23
2.3.2 Graph encoders . . . 25
2.4 Entropy estimation of random graph sources . . . 27
2.4.1 Approach . . . 28
2.4.2 Zero-order model . . . 30
2.4.3 First-order model . . . 32
2.4.4 Second-order model . . . 34
2.4.5 Higher order models . . . 35
2.4.6 Simulation results . . . 35
2.5 Entropy and compression of random graph sources . . . 36
2.5.1 Erdős–Rényi model . . . 37
2.5.1.1 Compression method . . . 37
2.5.1.2 Mean length of the encoder . . . 40
2.5.1.3 Time complexity analysis . . . 42
2.5.2 Stochastic block model graphs . . . 45
2.5.2.1 The respective distributions approach . . . 46
2.5.2.2 The single distribution approach . . . 46
2.5.2.3 The hybrid approach . . . 47
2.6 Summary . . . 49

3 Kolmogorov Complexity of Graphs 51
3.1 Background . . . 52
3.2 Basics and definitions . . . 53
3.2.1 Kolmogorov basic graphs . . . 54
3.2.2 Kolmogorov random and non-random graphs . . . 60
3.3 Kolmogorov graph covering . . . 63
3.4 Simulation results . . . 68
3.4.1 Traditional graph representation methods . . . 69
3.4.2 Zachary's karate club . . . 69
3.4.3 Existing graph compression algorithms . . . 70
3.5 Summary . . . 72

4 Network Evolution Models 74
4.1 Background and basic definitions . . . 75
4.2 Network Evolution Chain . . . 78
4.3 Entropy rate and typical sequences . . . 84
4.4 Functions of Network Evolution Chains . . . 87
4.5 Continuous time modelling of Network Evolution Chains . . . 90
4.5.1 The $s(g)$ method . . . 91
4.5.2 The Continuous Time Markov Chain method . . . 91
4.6 Erdős–Rényi Network Evolution Chains . . . 93
4.6.1 Stationary distribution of the number of nodes . . . 98
4.6.2 Probability distribution of individual graphs . . . 101
4.6.3 Entropy rate . . . 102
4.6.4 Functions of Erdős–Rényi Network Evolution Chains . . . 103

4.7 Simulation results . . . 104
4.7.1 Overview of the code . . . 105
4.7.2 Stationary distribution . . . 105
4.7.3 Stationary distribution of graphs . . . 106
4.7.4 Entropy rate . . . 107
4.7.5 Number of triangles in a network . . . 109
4.8 Summary . . . 111

5 Complexity and Compression of Trees 113
5.1 Background and basic definitions . . . 114
5.2 Entropy of tree sources . . . 116
5.2.1 Entropy of the uniform source . . . 118
5.2.2 Entropy of Simply Generated Trees . . . 121
5.2.3 Conditioned Galton-Watson Trees . . . 128
5.2.4 The spanning tree model . . . 133
5.3 Universal compression of tree structures . . . 139
5.3.1 Compressing uniform tree sources . . . 142
5.3.1.1 Pit-climbing algorithm . . . 142
5.3.1.2 Tunnel-digging algorithm . . . 146
5.3.1.3 TreeExplorer . . . 148
5.3.1.4 Comparison with entropy . . . 149
5.3.1.5 Comparison with adjacency list . . . 149
5.3.1.6 The Newick format . . . 152
5.3.1.7 Other succinct tree coding methods . . . 152
5.3.2 Universal compression of SGTs . . . 152
5.3.3 Universal compression of the Erdős–Rényi spanning tree model 157
5.4 Summary . . . 162

6 Applications 164
6.1 Routing protocols in Wireless Sensor Networks . . . 165
6.1.1 Preliminaries and network model . . . 166
6.1.2 Protocol description . . . 168
6.1.2.1 Selection of host nodes . . . 168
6.1.2.2 Selection of heir nodes . . . 169
6.1.2.3 Relocation of first-level nodes . . . 169
6.1.3 Energy consumption analysis . . . 169
6.1.4 Simulation results . . . 173
6.1.5 Application of tree compression . . . 178
6.2 Temporal Graph Neural Networks . . . 179
6.2.1 The information bottleneck problem for temporal GNNs . . . 181
6.2.2 Alternative Mutual Information Rate (AMIR) . . . 184
6.2.3 Benefits of AMIR . . . 188
6.2.3.1 Rate of convergence . . . 188
6.2.3.2 Number of sample sequences needed . . . 189
6.2.4 AMIR of Markov chains . . . 189

6.2.5 Simulation results . . . 191
6.2.6 Gaussian processes . . . 191
6.2.7 Hidden Markov Model . . . 193
6.2.8 Application of NEC to temporal GNNs . . . 194
6.3 Summary . . . 195

7 Conclusions and Future Work **197**
7.1 Thesis summary . . . 197
7.2 Future works . . . 200
7.2.1 Universal compression of graphs . . . 200
7.2.2 Improving the performance of Kolmogorov graph covering algorithm . . . 201
7.2.3 Enhancing NECs . . . 201
7.2.4 Applications of the proposed research . . . 202
7.3 Conclusions . . . 202

Bibliography **204**

List of Figures

2.1 Comparison of different types of graphs 14
2.2 The categorization of different methods for lossless graph compression 19
2.3 Classic graph representation methods 21
2.4 Example of redundancy in graphs 30
2.5 Example of second order dependency 34
2.6 Entropy and mean compressed length of Bernoulli processes with length 50 as a function of p 43
2.7 Entropy and mean compressed length of Bernoulli processes with $p = 0.1$ as a function of the length of the Bernoulli sequence, l 43
2.8 A comparison between the 3 proposed approaches to compressing SBM graphs 50

3.1 A graph with shift coefficient of 2 57
3.2 An example of enhanced Kolmogorov graph covering 67
3.3 Two sample graphs shown with their K-complexity estimates using three methods: adjacency matrix, adjacency list, and Kolmogorov graph covering 70
3.4 Zachary's karate club graph, along with its K-complexity estimation using adjacency matrix, adjacency list, and Kolmogorov graph covering 71
3.5 Comparing the results of Kolmogorov graph covering and GraphZip . 72

4.1 Example of four consecutive graphs in an NEC 80
4.2 The estimated entropy rate for four instances of the same ERNEC . . 109
4.3 The estimated entropy rate for the number of triangles in an NEC . . 111

5.1 Example of double-node transformation 120
5.2 Upper bound to the entropy of the conditioned Galton-Watson model 131
5.3 Steps of the spanning tree model 134
5.4 Entropy upper bound for ER spanning trees for graphs with 100 nodes as a function of the ER parameter p 139
5.5 Entropy per node upper bound and maximum entropy per node for ER spanning trees as a function of node number n in the giant component regime threshold 140
5.6 Running TPC on a sample tree 143
5.7 Running TTD on a sample tree 147

5.8 Comparing the performance of TreeExplorer on a uniform unlabelled unordered rooted tree source with its entropy 150
5.9 Comparing the performance of TreeExplorer with adjacency list . . . 151
5.10 Comparing the performance of TreeExplorer with the Newick format 153
5.11 An example SGT and its corresponding SGT sequence 154
5.12 Proposed algorithm for compressing ER Spanning Trees 159

6.1 An example of a Wireless Sensor Network 165
6.2 LEACH routing map 167
6.3 Setup phase of LEAST 170
6.4 A snapshot of the network map for a sample network at a random around in the LEAST protocol 170
6.5 Comparison of LEAST with LEACH 174
6.6 Comparison of LEAST with other protocols 175
6.7 Comparison of LEAST with LEACH: total remaining energy per round 176
6.8 Effect of p_{HN} on the lifetime of the network 177
6.9 Simulation result for LEAST averaged over 50 random sensor node location distribution 177
6.10 Compression ratio of the spanning tree model in the giant component regime boundary compared to adjacency list representation as a function of number of nodes in the WSN 179
6.11 A comparison between GNNs and temporal GNNs 180
6.12 Model of an ML algorithm 181
6.13 Estimating MIR and AMIR 192
6.14 Estimating AMIR of an HMM 194

List of Tables

2.1 Simulation results for the Barabási–Albert model 36
2.2 Simulation results for the $G(n, p)$ model 36
2.3 Comparison of different algorithms for compressing a Bernoulli process of length l and parameter p 44

3.1 Examples of basic graphs 59

4.1 Simulation results for a random ERNEC with 5 nodes: stationary distribution of the number of nodes 106
4.2 Simulation results for a random ERNEC with 8 nodes: stationary distribution of the number of nodes 106
4.3 Stationary distribution of graphs in an example ERNEC 107
4.4 Simulation results for a random ERNEC: entropy rate 108

5.1 Unlabelled unordered rooted trees of up to four nodes 122
5.2 Example of Simply Generated Trees and their probabilities 123

List of Mathematical Notations

$\log(\cdot)$	Logarithm function with base two
$\ln(\cdot)$	Natural logarithm
$\lceil \cdot \rceil$	Ceiling operator
$\lfloor \cdot \rfloor$	Floor operator
$\binom{\cdot}{\cdot}$	Binomial coefficient
$\sim$	Asymptotically equal to
$\approx$	Approximately equal to
$\lvert \cdot \rvert$	Cardinality of a set
$\{X_\cdot\}$	Stochastic process
$X_{\cdot:\cdot}$	Subset of a stochastic process
$p(\cdot)$	Probability operator
$\mathbb{E}[\cdot]$	Expectation operator
$P_\cdot$	Empirical probability distribution of a sequence
$H(\cdot)$	Entropy / Entropy rate operator
$H_\cdot$	Entropy of a source
$h(\cdot)$	Entropy of a Bernoulli random variable with the given parameter
$I(\cdot)$	Information in a random sample
$I(\cdot;\cdot)$	Mutual information
$D(\cdot\|\|\cdot)$	Kullback–Leibler (KL) divergence
$l_C(\cdot)$	Codeword length
$L(\cdot)$	Average codeword length of a graph encoder
$R(\cdot)$	Coding redundancy per symbol
$\mathscr{R}$	Average coding redundancy
$\mathcal{U}$	Universal computer
$O(\cdot)$	Big O operator
$K(\cdot)$	Kolmogorov complexity operator
$\delta(\cdot)$	Randomness deficiency
G	Graph space
T	Tree space
$V(\cdot)$	Set of vertices of a graph
$E(\cdot)$	Set of edges of a graph
$S(\cdot)$	Shift coefficient of a graph
$\mathrm{B}(\cdot,\cdot)$	Binomial distribution
$G(\cdot,\cdot)$	ER model
$\tau(\cdot)$	Number of spanning trees of a graph

C_n	nth Catalan number
$a(\cdot)$	Addition probability in NEC
$r(\cdot)$	Deletion probability in NEC
$s(\cdot)$	Same probability in NEC

Chapter 1

Introduction

Networks are all around us. We can find networks almost anywhere if we look carefully enough. We have networks such as protein-protein interaction networks at the tiniest of scales. Extremely big networks such as the internet can have more than 50 billion nodes. All in all, networks form an inseparable part of our lives.

Historically, graph theory has been the main mathematical tool to model networks. The use of graphs to model different types of network is observable in almost every area that deals with networks and their topology. These fields include, but are not limited to, communication systems [1], social sciences [2], earth sciences [3], and biology [4]. Furthermore, graphs exhibit remarkable versatility as data structures, and can be used to store various data types, including images and text [5]. This multifaceted utility underscores the paramount significance of graph theory and its extensive applicability across different engineering problems.

It must be noted that the terms *graph* and *network* might convey different meanings in different contexts. The term network is often used to refer to a real-world realisation of a series of interconnected entities, whereas the term graph is an abstract and mathematical realisation of networks. In this thesis, these two terms are used interchangeably, as we will be using graphs to model networks.

Information theory provides us with tools to analyse sources of information and quantify their complexity. This motivates us to use the same tools to analyse graphs as information-containing entities. Information theory is not only a powerful tool in analysing the complexity of data sources within itself, but also provides us with bounds as to how compressible different data sources are. Since its introduction by Shannon back in the 1940s, information theory has served as a powerful tool for quantifying and analysing the amount of information content within data. Rather than waning, the relevance and significance of information theory have only grown stronger with time. However, in today's world, several issues that lend themselves to formal analysis through information theory have been overshadowed by more fashionable approaches, like artificial intelligence. Consequently, the importance of classical methodologies is often overlooked. In today's data-rich environment, information theory can be a pivotal tool for addressing numerous challenges. Therefore, an information-theoretic analysis on graph sources can provide us with a novel insight on the complexity of graphs and networks around us.

Given the highlighted significance of both domains, it becomes evident why adopting an information theory approach to address challenges in graph theory can prove highly advantageous. Graphs that exist in today's scientific problems are bigger than ever, and growing with time. For instance, with the emergence of 5G and the movement towards 6G, the number of devices present in the global network has significantly grown. 6G will be engineered to accommodate as many as ten million devices per square kilometer [6]. This is where information theory comes in. Quantifying the amount of information in networks and compressing them can be achieved by applying information theory to graph theory. This holds immense potential, as it leads to conserving storage space and communication bandwidth.

Most of the research done in this thesis is dedicated to establishing a robust mathematical foundation within the scope of the proposed research area. This emphasis

was imperative because, despite a limited body of literature in this domain, there had been no prior endeavor to approach this problem from a more comprehensive perspective. As a result, I divided my research into three main phases, as listed below.

1. Information-theoretic complexity analysis of networks
2. Production of optimal graph compression algorithms
3. Applications of network complexity analysis and compression

Each chapter in this thesis is written by having at least one of these phases as its primary objective. Some chapters might tackle more than one phase. In this chapter, we will provide a brief description and background on each of the phases of the project. More detailed literature reviews on each subject are provided to the reader at the beginning of each chapter.

1.1 Complexity analysis of networks

Despite their widespread use, graphs are known to be hugely complex data structures. One important reason for this is the overwhelming number of possible ways that connections can be formed between the nodes in a graph. For instance, the number of possible graphs on 24 nodes exceeds the number of all atoms in the universe. The complexity of graphs grows even more when moving from simple graphs to more specialised models such as directed graphs, weighted graphs, or multi-graphs. For these reasons, measuring the complexity of graphs has been the interest of many scientists in different fields such as mathematics, physics, and chemistry throughout history. In fact, one of the oldest methods of measuring a graph's complexity is introduced by a physicist, John R. Platt, back in 1947 [7]. In his work, Platt measures the complexity of a graph by counting the number of different two-edge subgraphs

within the given graph. Later on, Rashevsky introduced a complexity measure that considers the asymmetries within a graph as a signal of higher complexity [8]. Afterwards, Mowshowitz continued the work and published a series of papers on this subject in 1968 [9–12]. Since 1968, the attempt to quantify the complexity of graphs has continued through different methods and techniques. Reference [13] divides these approaches into two categories: deterministic and probabilistic. We will provide a description of each of these categories in turn.

1.1.1 Deterministic approaches

When given a graph as an input, these approaches perform a deterministic analysis on the input without any assumptions on the distribution of the graph or its subgraphs, and return the calculated complexity for the graph as their output. These methods can take on different forms. Some of the most prominent categories within the deterministic domain are described below.

Calculating the Kolmogorov complexity (K-complexity) [14] of the input graphs is one of the main approaches in the deterministic domain. Kolmogorov complexity of a string of characters is the shortest possible program length that when fed into a universal Turing machine, outputs that string. Since its introduction by Kolmogorov, this concept has been used as a measure of complexity and randomness [15]. Kolmogorov complexity of a string is incomputable [16], but upper bounds to it can be found [17]. Many studies have argued K-complexity to be the superior method of estimating a graph's complexity [18, 19]. Because of these reasons, several papers have been published on ways of estimating the K-complexity of graphs.

Another form of deterministic methods are those that focus on the number of substructures or subgraphs within a network. Generally, these methods assume that the more unique substructures exist within a network, the more complex that network is. Many scientists have worked on producing such measures so far, each of which is

based on counting specific subgraphs. Some examples of the considered substructures include: number of spanning trees [20], number of two-edge subgraphs [21], and the number of all possible subgraphs [22].

The last category of deterministic approaches is based on methods to generate a graph. These approaches first introduce a set of generative blocks, and then try to define the graph at hand using the defined blocks and a set of elementary operators [13]. One example of such measures is introduced in [23], which uses boolean operators such as union and intersection to describe graphs.

It is worth mentioning that the substructure and generative approaches are mainly heuristic. We are more interested in a systematic definition of complexity, from which conclusions can be made regarding the compressibility of graphs. K-complexity has direct ties with compression methods, and is considered as an offspring of information theory. Therefore, from the approaches in the deterministic domain, we will only focus on K-complexity in this thesis.

1.1.2 Probabilistic approaches

Probabilistic approaches to measuring the complexity of graphs assign a probability distribution to particular elements of a graph or the graphs themselves, and then measure the resulting entropy. Even though Shannon's entropy is usually known as the standard approach, the entropy measure used in these approaches is not necessarily the one introduced by Shannon. Reference [24] discusses the use of more generalised entropy measures such as Rényi entropy [25].

Probabilistic approaches to measuring the complexity of graphs are one of the main objectives of Chapter 2. Therefore, a more detailed literature review on the works done in this field is provided in that chapter.

1.2 Optimal graph compression

The underlying topology of a network, modeled using a graph, is of fundamental importance in countless network protocols and applications. This importance necessitates the storage, communication, and processing of these graphs. As the aim of compression is to reduce the overhead in both storage and communication purposes, as well as facilitating data processing, the need for graph compression algorithms becomes more intelligible. If we look at graph compression as a source coding method, it is clear that having an effective graph compression algorithm will use the available bandwidth more efficiently. Moreover, networks all around us are growing. For instance, a typical 5G cell has the capacity to support 50,000 IoTs or narrowband IoT devices [26]. Additionally, the number of IoT devices is expected to reach 25 billion by 2025 [26]. These network expansions are additional reasons for the importance of graph compression, and the need for an instant and formalised tackling of this problem.

Graph compression has taken many different shapes and forms so far. Scientists have developed numerous graph compression algorithms based on specific application scenarios or topological characteristics. Some examples of these methods are SlashBurn [27] for community graphs, and GraphZip [28] for sparse graphs. In addition to most of the existing approaches being application-specific, they are also mainly heuristic. The typical way to assess the performance of such algorithms is by comparing their performance with other existing methods. Because of these reasons and the undeniable importance of graph compression in many domains, we are motivated to look for optimal graph compression algorithms. This optimality will be based on comparing the mean code length of the developed algorithms with the metrics found for measuring network complexity. This mirrors the approach that Shannon took by comparing the mean code length of coding algorithms with the entropy of the source [29]. A more detailed overview on the existing graph compression algorithms

is provided in Chapter 2.

1.3 Applications of network complexity analysis and compression

We believe that the discoveries emerging from our proposed research field hold significant potential for widespread application across various domains. Beyond its direct utilization in graph storage databases, the application of graph complexity analysis and compression extends to diverse domains, showcasing its versatility and impact. To understand this, we can look at the example of internet topology discovery. Discovering the topology of the internet is crucial in applications such as security, routing, network management, protocol design, and improving quality of service (QoS) [30]. In the majority of the existing topology discovery algorithms, such as the distributed Bellman-Ford algorithm [31], nodes need to communicate different paths in the network, or their local perception of the topology of the network to other nodes. Therefore, in such algorithms, the messages that are being communicated among nodes are essentially graphs, which can benefit from graph compression.

In this thesis, we focused on two main application domains to test out the results of our research on: network routing protocols in Wireless Sensor Networks (WSNs), and temporal Graph Neural Networks (GNNs). Below, we provide an overview of these two domains and discuss them in more detail in Chapter 6.

1.3.1 Routing protocols in WSNs

Routing is of fundamental importance in almost every communication network realisable. Packets not only must be able to find their way to their destination, but also do so in an efficient manner. Even though the concept of routing is always the same, the algorithms take on different shapes and forms owing to the different characteristics

of the network they are designed for. In the context of WSNs, routing is still a very active and ongoing area of research. The main restrictions in WSNs are considered to be limited power supply and computation power of the sensor nodes, which affect how routing protocols must be designed for such networks. Due to these key reasons and other limitations, routing in WSNs is different from traditional routing protocols such as those for IP routing, or ad-hoc routing.

The concept of routing tables is an inseparable part of many routing protocols in general. It is possible to define a table for every node in the network that stores all the hops in the shortest path between itself and other reachable nodes of interest. This table is known as a routing table [32]. It can be easily proven that a routing table essentially forms a rooted tree, with the root being the source node. Extending the same concept to WSNs, it will be discussed that an important routing table is that of the sink node of the network. This is due to the typical pattern of data flow in WSNs. Additionally, this tree can be looked at as an expansion and improvement to hierarchical routing protocols for WSNs [33]. These protocols are already considered to be the most energy-efficient, and therefore widely used, routing protocols for WSNs [34]. These reasons motivate us to focus on tree compression algorithms, and their role in improving existing hierarchical routing protocols for WSNs. We show that this could lead to a significant increase in the power efficiency of routing protocols for WSNs.

1.3.2 Temporal GNNs

GNNs have attracted a lot of attention in the past few years. After the great success of neural networks, it was noticed that a significant number of systems and datasets take the form of graphs, which gave rise to the birth of GNNs. This field studies neural networks in which the input takes on the form of a graph. GNNs have been extensively analysed, and a survey of the works existing in this field can be found

in [35].

Later on, it was noticed how the input graphs to GNNs are usually dynamic, and evolve over time. This gave raise to temporal GNNs, which is a field dedicated to studying and introducing GNNs that take on dynamic graphs as their inputs. Even though much more limited than GNNs, numerous studies have been conducted on this field in the past few years. A great example of these attempts is a model called Temporal Graph Networks [36]. Despite the existence of effective algorithms and models in this domain, there is still a lack of a theoretical framework for formal analysis of these networks. We believe that information theory can act as an effective tool for such analysis, and therefore attempt to study the behaviour of temporal GNNs using this tool in our research.

1.4 Thesis contributions and organisation

The three phases of this research project indicate that our work on this subject initiates from a strong and general mathematical basis, and slowly moves towards more practical aspects and solving real-world challenges. In this section, we provide an overview of the main contributions of this thesis and explain the organization of its chapters.

The main contributions of this thesis are as follows:

- **Information-theoretic framework:** We develop a novel information-theoretic approach for analyzing the complexity of graphical structures, providing a foundational perspective for subsequent analysis.

- **Comprehensive complexity analysis:** Leveraging this framework, we conduct a thorough complexity analysis of networks from different aspects, offering new information-theoretic insights into their structural properties and behaviour.

- **Graph compression optimization:** We demonstrate how the information-theoretic perspective informs graph compression, proposing a family of optimal compression algorithms for different graph types.

In **Chapter 2**, we study the encoding of graphs from a classical information theory point of view. After providing an extensive literature review on this subject, we provide the fundamentals of measuring graph complexity using entropy and analyse the traditional methods of storing graphs. We then move on to provide optimal compression algorithms for some of the most well-known random models for graph generation.

In **Chapter 3**, we study the complexity of static networks. The term static is used to refer to networks that do not undergo any significant change or evolution through time. After describing such networks and providing clear practical examples, we argue why the best tool for analysing the complexity of such networks is Kolmogorov complexity. We then move on to introduce a method to estimate an upper bound to the Kolmogorov complexity of any given graph that can be viewed as static.

We shift our focus to the complexity of evolving networks in **Chapter 4**. Firstly, it is reasoned that most networks around us are actually dynamic networks. Secondly, a novel model is introduced that can be used to model dynamic networks and study their behaviour and evolution using information theory. This model then branches out into different categories, each of them suited for a specific kind of analysis of dynamic networks.

In **Chapter 5**, we narrow down our focus to an important subset of graphical data structures: trees. Trees represent a specific category of graphs characterised by their lack of loops within their structure. Unique characteristics of trees, alongside their various applications and presence in different fields, necessitated a deeper study on the subject of their information-theoretic properties. This chapter discusses random models for trees, analyses their complexity, and studies their optimal compression.

Subsequently, in **Chapter 6**, we focus on some applications of our findings to real-world problems. Routing in WSNs and an information-theoretic analysis of temporal GNNs are chosen as our main focal points of the chapter. It will be discussed how our theoretical findings in this thesis can lead to an improvement in both domains with respect to performance, efficiency, and resource utilization.

Ultimately, **Chapter 7** concludes the thesis and introduces potential directions for future research on the discussed topics.

Chapter 2

Encoding Graphs

Classical information theory, in the same format introduced by Shannon back in 1948, is still very widely used. This is perhaps because of the generality of the theorems, their applicability to various fields and domains, and their strength in the analysis of information content of data. Because of this, it is comprehensible for our first step to be the analysis of graphical data structures from the perspective of Shannon's information theory. In this chapter, we take on this approach, and view graphs as random variables drawn from a probability space. This view will allow us to analyse the information-theoretic properties of graphs. Our analysis will be based on calculating Shannon's entropy [29], and using source coding techniques to compress graphs to their theoretical limit.

This chapter starts with some background information on graph theory and random graph generators. We then provide an overview on the works done on measuring the complexity of graphs using information theory, as well as a brief survey on existing graph compression algorithms. We then go through the information-theoretic definitions and basics that will be used throughout this chapter and the entire thesis. We end the chapter by providing some novel work on estimating the entropy of graphs and providing compression algorithms for some of the most well-known random graph

generators.

2.1 Overview of random graph models

Graph theory is a powerful mathematical tool that allows us to model and simulate the behaviour of networks around us. Throughout this thesis, we will use graphs and graph theory to help us model networks. We start by providing a formal definition for graphs.

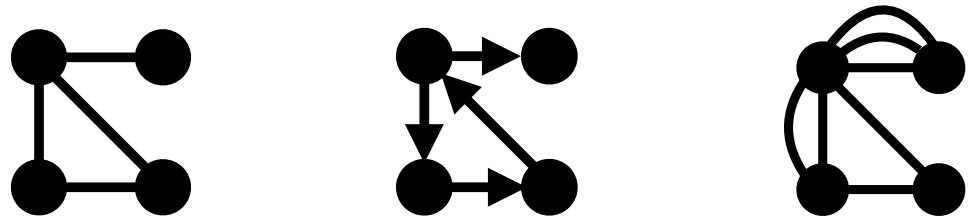
Definition 2.1.1 (Graphs)**.** A graph g is defined as a set of vertices (or nodes) $V(g)$, and a set of edges $E(g)$. Elements in $E(g)$ are ordered pairs of elements in $V(g)$, where each pair shows a connection between two nodes.

There are many different ways to model networks using graphs. Throughout this thesis, we will use simple graphs. Despite their simplicity, simple graphs are one of the most widely used graphs for simulating networks. Simple graphs are defined as below.

Definition 2.1.2 (Simple graphs)**.** A simple graph is one whose edges are undirected and unweighted, and does not contain any self-loops (edge from a node to itself), or multi-edges (multiple edges between the same pair of nodes).

Fig. 2.1 illustrates simple graphs in contrast to other graph models such as directed graphs, and multi-graphs. Even though each of these categories of graphs have their own applications, our primary focus is on simple graphs. Furthermore, our work can be extended to cover other types of graphs.

Additionally, graphs can be labelled or unlabelled. The nodes of a labelled graph have unique identifiers, whereas there is no difference between the nodes of an unlabelled graph. In this research project, we work with both cases and it will be mentioned whether the graphs at hand are labelled or unlabelled.



(a) Simple graph (b) Directed graph (c) Multi graph

Figure 2.1. Comparison of different types of graphs

When modelling a network using graphs, it is important that the mathematical features of that graph mimic the features observed in the real-world network that we are aiming to model. In order to do this, we often use random graph sources. To define a random graph source, we first need to define a sample space. As throughout this thesis our random variables are graphs, we use the term *graph space* for this purpose, and define it as below.

Definition 2.1.3 (Graph space)**.** A graph space, shown with G, is a finite set containing all the graphs that are possible to be generated.

Having the definition of a graph space, a random graph generator can simply be defined as a probability space on a given graph space. The formal definition of a random graph generator is given below.

Definition 2.1.4 (Random graph generator)**.** Let G be a graph space, and let $\mathcal{A} = 2^G$ be the power set of G (the set containing all possible subsets of G). Furthermore, let p be a probability measure on G. Then, the discrete probability space $(G, \mathcal{A}, p)$ is called a random graph generator.

When studying specific types of networks, it is ideal to use a random graph generator whose corresponding distribution resembles what happens in practice. For instance, many networks around us exhibit a property known as the small-world property. The small-world property has been extensively studied by network scientists [37], and can be observed in many networks around us. A graph is small-world

if the average distance between any two nodes in the graph scales as $\log n$, where n is the number of nodes in the graph. Note that the distance between nodes in this context is defined as the minimum number of hops that need to be taken for going from one node to the other. This behaviour can be observed in many different graphs around us, such as social networks, genetic networks, and even communication networks. Therefore, if we are to study such networks using a random graph generator, we expect it to be likely for the outputs of the random graph generator to exhibit the same behaviour as well. This shows the importance of matching the chosen random graph generator to the networks that we aim to study. Several popular random graph generators whose ability to model networks has been proven with time will be studied in the following sections.

2.2 Background

Our investigations in this chapter can be divided into two main categories: measuring graphs' entropy, and compressing graphs to their theoretical limit. There have been works on both of these subjects in the past. Even though most of these works are not fully aligned with what we are seeking in this research, it is still beneficial to familiarise ourselves with their methodology. In this section, we will provide an overview on the literature existing on these subjects.

2.2.1 Entropy of graphs

As stated in Chapter 1, estimating the complexity of graphical data structures has been of importance to scientists in different fields, and the approaches in this regard can be categorised into either the deterministic or the probabilistic domain. Measuring the entropy of graphs falls into the probabilistic approaches to calculating graph complexity. While early approaches have been mainly heuristic, some approaches have

used information-theoretic tools in their analysis. A review of the existing methods on the entropy measures for graphs has been performed in [38], which shows that there have been numerous approaches that try to measure entropy of graphs with different definitions. These works can be categorized into the following groups:

1. Those that measure the entropy of invariants within a graph

2. Those that measure the entropy of a graph itself

The majority of the works on entropy of graphs fall into the first category. These works were initially studied by physicists and chemists to quantify the complexity of molecular structures that were modelled using graphs. Perhaps one of the first information-theoretic approaches to solve this problem after the introduction of information theory by Shannon was done by Rashevsky. In his paper, titled *Life, Information Theory, and Biology* [8], he tackles the problem of finding the complexity of biological structures, modeled using graphs. His approach was to apply a probability distribution to the vertices of the graph, and then calculate the Shannon entropy of that distribution. A year later, Trucco followed Rashevsky's approach of realizing a discrete information source on the vertices of graphs. He did this by grouping vertices and seeing them as letters from an alphabet [39]. Following in the footsteps of Rashevsky and Trucco, Mowshowitz extended this notion of complexity for graphs by considering various distributions on graphs throughout his series of papers [9–12]. These papers culminated in his recent survey on this subject, where he discusses the history of this subject and the different metrics used in different fields of science [38].

Even though the approaches in the first category use Shannon's entropy and information-theoretic metrics as their tool, they are not what we are looking for in our analysis. In all of the approaches mentioned above, a probability distribution is first defined on a single fixed graph, and then the complexity of that graph is calculated as the entropy of the defined distribution. However, we want to look at

graphs themselves as random variables, drawn from a random source. This means that rather than defining a distribution within graphs, we want to have a distribution *on* graphs. Even though the other approaches do introduce a measure of complexity on graphs, our approach is closer to the original definition of entropy, especially if the ultimate goal is graph compression.

The second category consists of approaches that are of more interest in this research. In these approaches, a probability distribution is first defined on the possible graphs, and then the entropy of the distribution is calculated. This is in accordance with Shannon's definition of entropy, which can give rise to a better understanding of the topologies, as well as providing a lower bound for their compression. For instance, [40] studies and calculates Shannon's entropy for an ensemble of graphs from the Stochastic Block Model (SBM), and [41] studies the structural entropy of graphs generated using the Erdős–Rényi model. These approaches are precisely what we are looking to do for all graphs models, and have acted as one of the main guidelines in our research. More recently, [42] used the same approach to quantify the structural entropy of 1D random geometric graphs, and [43] calculated the structural entropy for Stochastic Block Models.

Another important group of studies that fall into the second category are those that introduce entropy measures for complex networks. Complex networks can be roughly defined as networks that exhibit certain topological characteristics. Most research done in this category use concepts from statistical mechanics to define notions of entropy for complex networks. For instance, [44] studies microcanonical network ensembles using Gibbs entropy. Other notable studies in this area are [45–47]. Shannon entropy is also used in this context in [48], which studies the entropy of canonical and microcanonical network ensembles.

In this chapter, we aim to study the complexity of network ensembles through Shannon entropy. It was seen that there are works that have been done in this

domain. However, we would like to look for a more generalised approach towards quantifying the entropy of graphs using Shannon entropy. Because of this, one of our main motivations in this chapter is to extend prior work on graph entropy to cover more models and distributions defined on graphs.

2.2.2 Graph compression

In addition to measuring the complexity of graphs using entropy, we are also interested in optimal encoding of graphs. From an information theory point of view, it is clear why the first step towards compression is to calculate the entropy. Entropy acts as the lower bound for compression, and as it will be seen in this chapter, the average length of any given encoding algorithm can not be less than the entropy of the random variable.

Extensive work has been done on the compression of graphs with many different approaches. In this section, we will mention some of the different approaches that have been taken in order to achieve graph compression.

Existing graphs compression algorithms can be divided into two main categories: lossless methods and lossy methods. In this research, we are mainly concerned with compression algorithms that let us recreate the original graph without any error. Therefore, our main focus in the scope of this research is lossless compression. Below, we will provide a description on some of the most prominent methods of lossless graph compression.

Reference [49] provides a detailed survey on lossless graph compression algorithms. This paper categorises compression algorithms based on their application areas, techniques used in each algorithm, and the features of each algorithm. Fig. 2.2 represents most of the categorization done in [49]. It can be seen that graph compression has a wide range of applications, and there are currently many methods being used that attempt to achieve maximum compression.

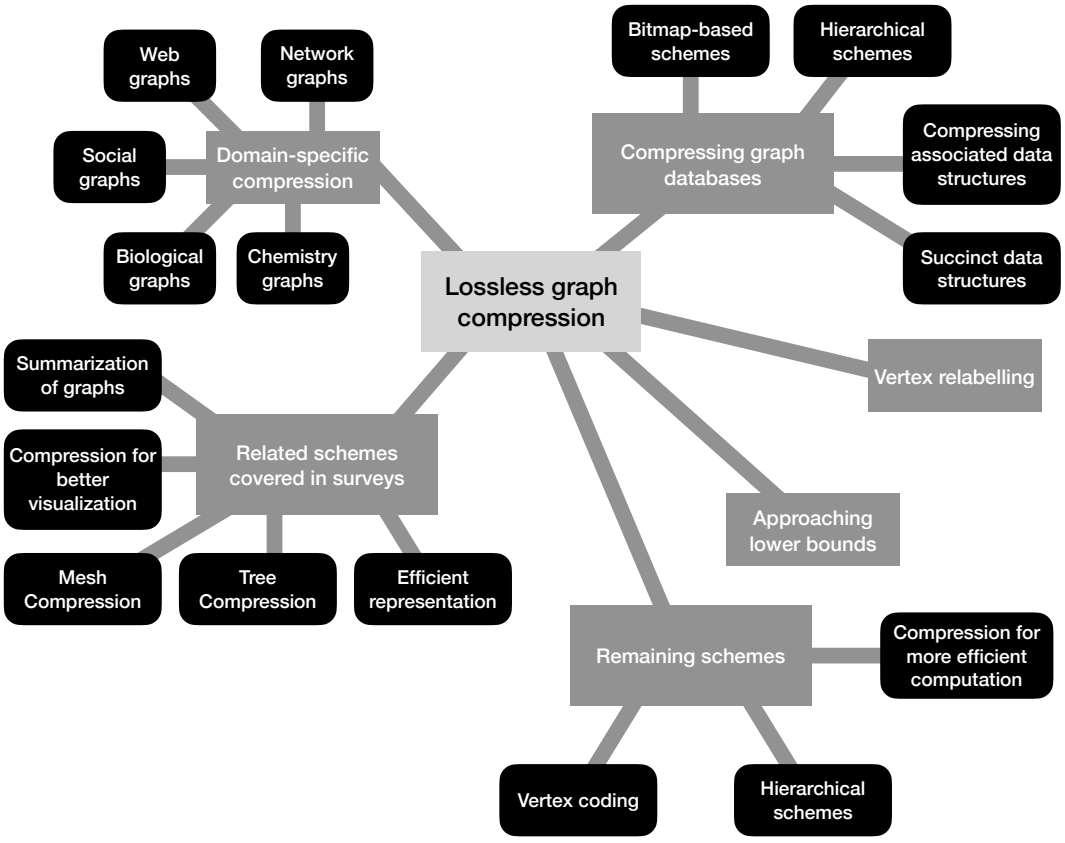


Figure 2.2. The categorization of different methods for lossless graph compression

We will start this section by providing a brief overview of classic graph representation methods. We will then categorise graph compression algorithms based on their approaches, and introduce some of the notable attempts in each category.

2.2.2.1 Classic graph representation methods

In this section we will briefly mention the most common ways that a graph g with n nodes and k edges is represented, without any form of compression algorithm applied to it.

- **Adjacency matrix:** In this method, graph g is represented by a $n \times n$ matrix, in which the element (i, j) is set to 1 if and only if nodes i and j are connected. This method requires n^2 bits to store the graph. It will be shown in this chapter that if g is a simple graph, its adjacency matrix representation requires only $\binom{n}{2}$ bits.

- **Edge list:** In this format, we represent each edge by specifying the endpoints of it. The existence of the node pair (i, j) in the list would mean that there exists an edge between nodes i and j. This will provide us with a variable length code for graphs. Each node pair requires $2\lceil \log n \rceil$ bits to be represented. Therefore, the code length for g will have a length of $2k\lceil \log n \rceil$.

- **Adjacency list:** This method is a hybrid of adjacency matrix and edge list. For each node in g, its set of neighbour nodes are listed using their labels. In addition to the endpoints of each edge being listed once, each node label is also mentioned once before its neighbours are listed. Therefore, the number of bits needed for the storing the adjacency list of g will be $(2k + n)\lceil \log n \rceil$.

Examples for each of these methods are illustrated in Fig. 2.3.



(a) Example graph

0	1	1	0
1	0	0	1
1	0	0	1
0	1	1	0

(b) Adjacency matrix

(1,2)
(1,3)
(2,4)
(3,4)

(c) Edge list

1	2	3
2	1	4
3	1	4
4	2	3

(d) Adjacency list

Figure 2.3. Classic graph representation methods

2.2.2.2 Rearranging the adjacency matrix

A great number of graph compression algorithms are based on rearranging the adjacency matrix of graphs. In other words, they try to relabel the nodes of a graph in a way that makes the adjacency matrix more compressible. One of the algorithms that uses this approach is SlashBurn [27]. Slashburn looks for communities in the input graph. It then rearranges the adjacency matrix such that the nodes that are part of the same community are near together. It then uses this partitioning to compress the graph. SlashBurn has proved to be very effective in compressing many graphs. However, it performs poorly when given a graph that can not be easily divided into communities.

2.2.2.3 Exploring the graph

Another form of graph compression methods are those that randomly explore the graph, and then compress it based on their observations. These methods compress the graph based on one or a series of random walks on the graph. Many compression techniques designed for web graphs use such methods [50]. A more recent algorithm

uses BFS (Breadth-First Search) to explore a graph and then compress it [51]. The limitation to these methods is inherent in their random nature. Even for one graph, different walks on the graph may result in different compressed lengths.

2.2.2.4 Dictionary-based coding

A subset of graph compression algorithms are based on dictionary-based source coding techniques. In these methods, the substructures within the graphs play the role of words. One of the most successful methods in this domain is grammar-based graph compression [52]. This compression method uses a dictionary-based compression algorithm named RePair [53]. The essence of grammar-based graph compression is to detect repeated substructures and represent them using RePair's grammar-based technique. The result is a fast compression algorithm. However, as stated by the paper itself, they do not guarantee a better compression ratio compared to other algorithms.

2.2.2.5 Query-based compression

Many approaches to graph compression are not interested in being able to reconstruct the original graph from its compressed version. Instead, they look for compressed versions that are able to answer a range of queries regarding the original graph. This approach has been of major interest to network scientists. A good example of such algorithms is [54]. Despite the huge amount of interest in this area, it will not be our primary focus when it comes to graph compression, as we are interested in being able to reconstruct the topology of the network.

2.2.2.6 Structure-based compression

These approaches use the structural features within a given graph to compress it. One of the earliest attempts in graph compression uses a structural approach, and presents

a compression algorithm based on finding cliques within a graph [55]. Afterwards, [56] provided a new approach to use the structural properties of a graph to compress it. This paper divides the set of possible graphs into different graph families based on their structure, and provides a compression scheme for one of these families. One of the latest novelties in the area of structure-based compression is an algorithm called GraphZip [28]. This algorithm is also based on cliques within the graph. It uses more simple representations for cliques within a graph to acquire a more compressed format. However, this compression technique is only effective for sparse graphs.

The main advantage of looking at graph compression as an information-theoretic source coding problem is that it allows us to measure how optimally the algorithm is performing. This can be done by comparing the average codeword length with the entropy of the source. However, none of the existing graph compression algorithms claim that they are performing optimally, and they simply compare their results with other existing methods. One of our goals in this research project is to provide graph compression algorithms that can claim that they are optimal or near-optimal with respect to information-theoretic metrics.

2.3 Information-theoretic basics and definitions

In this section, we will introduce some information-theoretic background with regards to random graph sources and their analysis. We start by providing an overview on the entropy of graphs. Additionally, we will provide formal definitions for graph encoding and compression.

2.3.1 Graph entropy

The first and the most important information-theoretic definition that we need to state is the definition of entropy. In this thesis, we rely on the classical entropy

defined by Shannon [29].

Definition 2.3.1 (Entropy of a random graph source)**.** The entropy of a random graph source $(G, \mathcal{A}, p)$ is shown with $H_{G,p}$, and is defined as

$$H_{G,p} = -\sum_{g \in G} p(g) \log_2 p(g). \tag{2.1}$$

We can use Definition 2.3.1 to calculate the entropy of a given random graph source. Note that if the probability distribution on the graph space is clear, we simply use the notation H_G to refer to the entropy of the graph source. Furthermore, note that the base of the logarithm used in Definition 2.3.1 is two, and therefore the entropy is measured in bits. Therefore, if the base of a logarithm is not specified, we ask the reader to consider the logarithm to be taken in base two as we will be using bits as the unit for entropy throughout this thesis. Converting the entropy to other units such as nats or dits can be done by changing the base of the logarithm.

It is known that the notion of entropy, defined according to Definition 2.3.1, shows the amount of uncertainty within a random source. Essentially, higher entropy for a random graph source would indicate a higher amount of uncertainty, or more surprise, about the output of the random graph source.

We have now established that the random variables in this study are graphs. However, it might be difficult to imagine graphs as random variables. We might be more accustomed to representations of graphs, such as adjacency matrices and adjacency lists. In the following theorem, we prove that as long as we use a one-to-one mapping, we can use any graph representation method and the entropy of the random graph source will not be affected.

Theorem 2.3.1. *Consider a one-to-one measurable map $X : G \to \Omega$. The entropy defined on the sample space Ω is equal to that of the original random graph source.*

Proof. Under X, the graph source $(G, \mathcal{A}, p)$ is mapped to $(\Omega, \mathcal{A}', p')$. The image

measure of p under the map X is the measure $p \circ X^{-1}$ on $(\Omega', \mathcal{A}')$ that is defined by

$$p \circ X^{-1} : \mathcal{A}' \to [0,1], \qquad \mathcal{A}' \mapsto p(X^{-1}(\mathcal{A}')).$$

Therefore, the probability measure p for any $\mathcal{A}' \subset \Omega$ is equal to $p(\mathcal{A})$ where $\mathcal{A} = X^{-1}(\mathcal{A}')$. We now calculate the entropy of p' and show that it is indeed equal to the entropy of p.

$$\begin{aligned} H(p') &= -\sum_{m \in \Omega} p'(m) \log p'(m) && \text{(2.2a)} \\ &= -\sum_{m \in \Omega} p(X^{-1}(m)) \log p(X^{-1}(m)) && \text{(2.2b)} \\ &= -\sum_{g \in G} p(g) \log p(g) && \text{(2.2c)} \\ &= H(p) && \text{(2.2d)} \end{aligned}$$

□

Theorem 2.3.1 states that as long as a one-to-one mapping is used to represent graphs, the entropy of the random graph source does not change. Therefore, we can utilize the most suitable graph representation method for the application scenario in mind and study the graph entropy using that representation.

2.3.2 Graph encoders

In this section, we will discuss graph encoders. We use this term to refer to functions that take a graph as an input, and return an encoded version of that graph. An information-theoretic definition of graph encoders is provided below.

Definition 2.3.2 (Single graph encoder)**.** A single graph encoder C is a mapping from a graph space G to $\mathcal{B}^*$, the set of finite-length strings of symbols from a binary alphabet.

Note that the alphabet used for an encoding does not necessarily need to be binary. However, as most of the systems around us are digital and work with a binary alphabet, that is the focus of this thesis. Generalisation to non-binary alphabet is trivial and is left to the reader. Additionally, even though the binary alphabet of choice can be anything, we prefer to use the standard set of $\{0, 1\}$.

Generally, not all mappings that conform to Definition 2.3.2 are of interest to us. There are several classes of codes that can be produced. However, the most practical one is the class of instantaneous, or prefix, codes. A code C_1 is a prefix of another code C_2 if C_2 starts with C_1. Using this notion, prefix codes are defined as below.

Definition 2.3.3 (Instantaneous (prefix) graph encoder)**.** A graph encoder C is called instantaneous or prefix in G if there exists no pair $\{(g_1, g_2)|g_1 \in G, g_2 \in G, g_1 \neq g_2\}$ such that $C(g_1)$ is a prefix of $C(g_2)$.

In this thesis, we are primarily interested in single graph encoders that are instantaneous.

Note that the inequality operator used in Definition 2.3.3 is not still well-defined for graphs. This depends on the types of graphs that we have in our graph space, and the networks that we are analysing using graph theory. For instance, if we are working with labelled graphs, equality of two graphs g_1 and g_2 will be equivalent to the equality of their respective adjacency matrices. If we are working with unlabelled graphs, any two isomorphic graphs will also be considered equal. Therefore, the definition of equality in graphs heavily depends on the networks and graphs that we are working with.

Having defined an encoder C on a graph space G, we use the notation $l_C(g)$ to show the length of the encoded version of graph g using C in bits. Having the length of the coded version of our graphs, we can define the expected length of C.

Definition 2.3.4 (Expected length)**.** The expected length of single graph encoder C,

shown with $L(C)$, defined over the random graph source $(G, \mathcal{A}, p)$ is given by

$$L(C) = \sum_{g \in G} p(g) l_C(g) \tag{2.3}$$

Having defined the basics of graph entropy and encoding, we move on to state one of the most fundamental theorems in information theory. This theorem, as stated below, provides us with a lower bound on the expected code length for a random graph source.

Theorem 2.3.2. *The expected length of any instantaneous single graph encoder is lower-bounded by the entropy of the random graph source. That is,*

$$L(C) \geq H_G. \tag{2.4}$$

Proof. The proof of this theorem can be found in [57, Ch. 5]. □

Theorem 2.3.2 provides the basis for defining optimal graph encoders. Optimality of a given graph encoder can be checked by measuring the expected codeword length of the encoder and comparing it with the entropy of the random graph source. A smaller gap between the expected codeword length of an encoder and the entropy of the random graph source indicates a better performance for the graph encoder. Based on this, one of our aims in this chapter is to study some existing random graph sources and their entropy, and then introduce single graph encoders that perform near-optimally for each of them.

2.4 Entropy estimation of random graph sources

In this section, we aim to estimate the entropy of graphs with a general approach. As in many practical scenarios the underlying distribution of the random graph source

is not known, we introduce a method that is able to provide an upper bound to the entropy of the source without any knowledge about the distribution. Having Theorem 2.3.2 in mind, an upper bound on the entropy of a graph will provide us with a better insight as to how compressible the graphs at hand are.

2.4.1 Approach

To find an upper bound for the entropy of a graph source, we want to use ideas similar to what Shannon used to predict the entropy of printed English text [58]. However, characteristics of graphs prevent us from using that idea directly. Unlike English text, graphs do not have a natural ordering and a unique way to be read from left to right. Therefore, we will try to adjust it so that we can use it to estimate the entropy of different graphs.

Firstly, we form our graph space with graphs that have a maximum number of n nodes. We then simply look at each graph as a set of edges. This way, representing the graph would be equivalent to indicating the set $\{E_1, E_2, ..., E_{\binom{n}{2}}\}$, where $E_i \in \{0, 1\}$ and it represents the existence or non-existence of the ith edge. Notice that there is no natural ordering on the possible edges of a graph. The label for each edge is only there to distinguish different edges. Secondly, we look at the set $\{E_1, E_2, ..., E_{\binom{n}{2}}\}$ as a set of correlated random variables. Each E_i can depend on any number of other variables in the set. Furthermore, we are also interested in defining a form of stationarity for graphs. However, stationarity is defined for processes in which there is a variable of time. In a graph, time is not a factor, but position is. To define a notion of stationarity for graphs, we will use the concept of isomorphism in graphs.

Definition 2.4.1 (Isomorphic graphs)**.** Two graphs g_1 and g_2 are called isomorphic if there exists a bijective function $f : V(g_1) \to V(g_2)$ such that for every pair of vertices u and v in g_1, we have $(u, v) \in E(g_1)$ if and only if $(f(u), f(v)) \in E(g_2)$.

Using Definition 2.4.1, we define isomorphic subsets as below.

Definition 2.4.2 (Isomorphic subsets)**.** Consider a random graph generator $(G, \mathcal{A}, p)$, where G consists of possible graphs on n labelled nodes. Let the set E contain all possible pairs of nodes, which indicates possible edges that can exist within the generated random graph. Now consider two subsets $S_1, S_2 \subset E$. The graphs g_1 and g_2 are constructed by having all the edges in S_1 and S_2 to exist, respectively. The two subsets S_1 and S_2 are called isomorphic subsets if the two graphs g_1 and g_2 are isomorphic.

Ultimately, using Definition 2.4.2, we define a notion of stationarity within graphs.

Definition 2.4.3 (Stationarity in graphs)**.** A random graph generator $(G, \mathcal{A}, p)$ is called stationary if for any two isomorphic subsets $S_1, S_2 \subset E$ we have

$$p(S_1) = p(S_2).$$

A more informal way of stating Definition 2.4.3 would be that the distribution of a group of edges within a network is independent from the location of those edges within the graph.

Although assuming a graph generator to be stationary based on this definition seems to be a very strong assumption, it provides an approximation for many networks around us. Firstly, if the underlying random graph source is known, then its stationarity can be mathematically checked. Additionally, it can be a valid assumption in many real-life networks. For example, in a social network, different friendships around the world seem to follow roughly the same process to determine whether they exist or not. Note that the formalism below relies on this definition of stationarity, and this assumption needs to be checked before applying the proposed models.

The principal idea in the proposed method is that the structures inside a graph have a certain degree of redundancy. Meaning, we can guess whether an edge exists or not based on the existence of other edges. In other words, the entropy of a single

edge usually decreases when the state of nearby edges are known. For example, one can, with a high probability, successfully guess the existence of the dotted edge in Fig. 2.4 given the state of the other edges present in the graph.

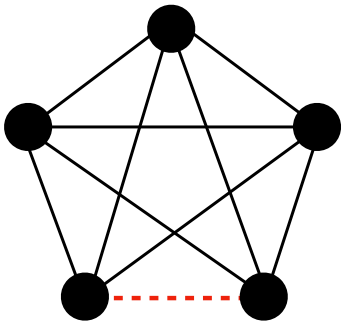


Figure 2.4. Example of redundancy in graphs

We are now ready to introduce models to estimate the entropy of graphs. In all of the models, we assume that the graphs are stationary according the Definition 2.4.3.

2.4.2 Zero-order model

In this model, we assume that we do not have any prior information about the graph, and consider independent edges for which the probability of existing and not existing is equal. In other words, we have

$$\forall E_i \in \{E_1, E_2, ..., E_{\binom{n}{2}}\} \quad p(E_i = 1) = p(E_i = 0) = 0.5.$$

The entropy of the graph source can therefore be calculated as the following.

$$H_G = H(E_1, E_2, ..., E_{\binom{n}{2}}) \tag{2.5a}$$

$$= H(E_1) + H(E_2) + ... + H(E_{\binom{n}{2}}) \tag{2.5b}$$

$$= \binom{n}{2} \tag{2.5c}$$

Equality 2.5a comes from defining a graph as the set of its possible edges. Equality 2.5b is due to the independence assumption in the zero-order model. Finally, equality 2.5c is because for all edges we have

$$H(E_i) = -0.5\log(0.5) - 0.5\log(0.5) = 1,$$

and we have $\binom{n}{2}$ edges.

This result is not surprising. It is well known that any simple graph with n nodes can be represented using $\binom{n}{2}$ bits by simply stating which edges exist and which ones do not. This is precisely what the adjacency matrix representation of a graph does. The interesting thing is that this value is the maximum entropy possible for a graph. This is stated in the following theorem.

Theorem 2.4.1. *For any graph with n vertices, the maximum entropy is $\binom{n}{2}$.*

Proof.

$$\begin{aligned} H_G &= H(E_1, E_2, ..., E_{\binom{n}{2}}) && \text{(2.6a)} \\ &= H(E_1) + H(E_2|E_1) + ... + H(E_{\binom{n}{2}}|E_{\binom{n}{2}-1}, ..., E_1) && \text{(2.6b)} \\ &\leq H(E_1) + H(E_2) + ... + H(E_{\binom{n}{2}}) && \text{(2.6c)} \\ &\leq \binom{n}{2} && \text{(2.6d)} \end{aligned}$$

Equality 2.6b comes from the chain rule in entropy. Inequality 2.6c is due to the fact that conditioning reduces entropy [57][Thm. 2.6.5]. Finally, inequality 2.6d is because the maximum entropy of a source with 2 alphabets is 1 bit [57][Thm. 2.6.4], and we have $\binom{n}{2}$ such sources. □

2.4.3 First-order model

In this model, we still assume that the edges are independent, but their distribution is not assumed to be uniform anymore. The stationarity assumption implies that all edges have the same probability distribution. Therefore, we can assume that each edge is present with a probability of q and is not present with a probability of $1-q$, independent from the other edges. This is the exact same model which was introduced by Edgar Gilbert back in 1959 [59], and is usually known as the Erdős–Rényi (ER) model. The entropy of this model can be calculated using the following equations.

$$
\begin{aligned}
H_G &= H(E_1, E_2, ..., E_{\binom{n}{2}}) && \text{(2.7a)}\\
&= H(E_1) + H(E_2|E_1) + ... + H(E_{\binom{n}{2}}|E_{\binom{n}{2}-1}, ..., E_1) && \text{(2.7b)}\\
&= H(E_1) + H(E_2) + ... + H(E_{\binom{n}{2}}) && \text{(2.7c)}\\
&= \binom{n}{2}(-q\log q - (1-q)\log(1-q)) && \text{(2.7d)}
\end{aligned}
$$

It can be seen that for calculating the entropy of this model, we need to know the value for q. Therefore, if q is unknown, we need to estimate it in order to estimate the entropy of the graph.

Assume $\mathbf{x}$ to be an observed sequence of m edges $\{x_1, ..., x_m\}$ in a large graph, and assume $P_{\mathbf{x}}$ to be the empirical probability distribution of $\mathbf{x}$ [57, pg. 347]. We can apply the Shannon-McMillan-Breiman theorem [57, Thm. 16.8.1] to $\mathbf{x}$.

$$
-\frac{1}{m}\log p(x_1, .., x_m) \rightarrow H \qquad \text{with probability 1.} \tag{2.8}
$$

In Eq. 2.8, H is the entropy rate of the process, which in the case of the first-order model, is equal to the entropy of a single edge. It is known that as the random variables in $\mathbf{x}$ are drawn i.i.d, the probability of the sequence, $p(\mathbf{x})$, depends only on

its type (number of 1s) and can be written as follows [57, Thm. 11.1.2].

$$p(\mathbf{x}) = 2^{-m(H(P_{\mathbf{x}})+D(P_{\mathbf{x}}||q))}, \tag{2.9}$$

where $D(P_{\mathbf{x}}||q)$ shows the Kullback-Leibler divergence or relative entropy [60], between the two distributions. After applying Eq. 2.9 to Eq. 2.8, we end up with the following equation.

$$H(P_{\mathbf{x}}) + D(P_{\mathbf{x}}||q) \to H \qquad \text{with probability 1.} \tag{2.10}$$

As we are treating edges as independent random variables in this model, we have the following equation from [57, Thm. 11.2.1].

$$D(P_{\mathbf{x}}||q) \to 0 \qquad \text{with probability 1} \tag{2.11}$$

Consequently, we have

$$H(P_{\mathbf{x}}) \to H \qquad \text{with probability 1.} \tag{2.12}$$

Eq. 2.12 provides us with a method to estimate the entropy of a large graph. After observing the existence of a large number of edges, we can estimate the entropy of the graph in the following manner. observed a sequence of m edges $\mathbf{x}$, where 1 means that the corresponding edge exists and 0 means that it does not. We choose $\hat{q}$ to be the number of observed 1s divided by m. This way, the entropy of the graph can be estimated as the following.

$$\hat{H} = -\binom{n}{2}(\hat{q}\log\hat{q} + (1-\hat{q})\log(1-\hat{q})) \tag{2.13}$$

Eq. 2.12 guarantees that $\hat{H}$ will tend towards the actual entropy of the model as

n grows large.

2.4.4 Second-order model

In this section, we will improve the model so that it captures more of the correlation between different edges. In this model, the probability of the existence of each edge depends on the structure of edges that are directly connected to it. For example, the dotted edges in Fig. 2.5 may have different probabilities of existence, as the structure of the edges connected directly to them are different. Note that by structure here, we mean the degree of both of the vertices of the edge, alongside the number of common neighbours that those two vertices might have.

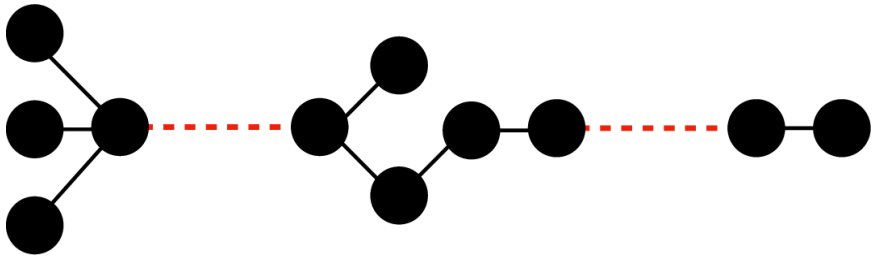


Figure 2.5. Example of second order dependency

The challenge with this model is that when describing a graph edge by edge, we may not have all the information about each edge's second-order approximation. Suppose we want to estimate the probability of existence of edge e. Also, assume the set K_e to contain edges from the neighbours of e for which we already know whether they exist or not, and the set U_e to contain those neighbours of e whose condition is still not known. We can write

$$p(e|K_e) = \sum_{u \in 2^{|U_e|}} p(u|K_e)p(e|K_e, u). \tag{2.14}$$

Based on the discussions made in Section 2.4.3, we can use the law of large numbers to estimate probability. The function N of a set of edges will show the number of subgraphs observed so far, which form the same structure as its input edges. Using

this notation and the law of large numbers for calculating probabilities we can estimate $p(e|K_e)$ in the following way.

$$\begin{aligned}\hat{p}(e|K_e) &= \sum_{u\in 2^{|U_e|}} \frac{N(u\cup K_e)}{N(K_e)} \frac{N(e=1\cup u\cup K_e)}{N(u\cup K_e)} \\ &= \sum_{u\in 2^{|U_e|}} \frac{N(e=1\cup u\cup K_e)}{N(K_e)} \\ &= \frac{N(e=1\cup K_e)}{N(K_e)}\end{aligned} \tag{2.15}$$

To estimate the entropy of the graph source, we can use Eq. 2.6b. To calculate $H(E_i|E_1,...E_{i-1})$, we first estimate $p(E_i|E_1,...E_{i-1})$ using Eq. 2.15 on a set of training graphs, and then we have

$$\begin{aligned}\hat{H}(E_i|E_1,...E_{i-1}) = &- \hat{p}(E_i|E_1,...E_{i-1}) \log \hat{p}(E_i|E_1,...E_{i-1}) \\ &- (1-\hat{p}(E_i|E_1,...E_{i-1})) \log(1-\hat{p}(E_i|E_1,...E_{i-1})).\end{aligned} \tag{2.16}$$

2.4.5 Higher order models

We can continue expanding the model to higher orders by taking into account the dependencies of each edge on its second neighbours, third neighbours, and so on. This will of course result in a better approximation of the graph, and a lower upper bound on the graph's entropy. However, by increasing the order of the model, its complexity will also increase. Therefore, we will end the discussion on different models and move on to inspecting the introduced models.

2.4.6 Simulation results

In this section, we will estimate the entropy of graphs created using different graph generation model.

Table 2.1 illustrates entropy estimations using first and second order models for

graphs created using the Barabási–Albert model [61]. Note that calculating the entropy of the Barabási–Albert model is not straightforward and therefore not available in Table 2.1.

Table 2.1. Simulation results for the Barabási–Albert model

Graph size	**Maximum entropy**	**1st order estimate**	**2nd order estimate**
10	45	43.0406	41.3420
15	105	100.6647	99.5422

Table 2.2 shows the estimations for graphs that are created using the ER model [59]. This table has one very important result; it is not always better to use higher order models. Because of having more states, higher order models require a longer sequence to converge to the actual entropy. Consequently, if certain graphs are created from distributions which are closer to the lower order models, it will be more efficient to use lower order models to estimate their entropy. As a result of this, the second order estimates in Table 2.2 are always higher than the first order estimate, as the graphs are created using the $G(n, p)$ model, which perfectly aligns with the first order estimates.

Table 2.2. Simulation results for the $G(n, p)$ model

Model	**Maximum entropy**	**Actual entropy**	**1st order estimate**	**2nd order estimate**
$G(10, 0.05)$	45	12.8879	13.0594	17.7477
$G(30, 0.07)$	435	159.1768	159.5907	199.6200
$G(100, 0.07)$	4950	1811.3	1810.7	1903.7

2.5 Entropy and compression of random graph sources

In this section, we introduce and study the entropy and compression of two well-known random graph sources: the Erdős–Rényi model, and the Stochastic Block Model.

2.5.1 Erdős–Rényi model

The Erdős–Rényi (ER) model is one of the most simple, yet widely-used, random graph generation models. This model, also known as the $G(n, p)$ model, was first introduced by Edgar Gilbert in 1959 [62]. Since then, this model has been used to study the behaviour of certain networks and graphs, and has been applied to areas such as modelling the web [63], biology [64], and game theory [65].

There are two parameters to be set in this model: n and p. n sets the number of nodes in the graph. Therefore, all the graphs that the model generates will have the same number of nodes. p determines the probability of the existence of each edge in the graph. In other words, each possible edge in the graph exists with a probability of p, independently from others. Because of this, any given graph generated from the $G(n, p)$ model can be represented with a sequence of $\binom{n}{2}$ random variables, with each one representing the state of one edge in the graph. Additionally, note that each of these random variables is a Bernoulli random variable with a parameter of p. Therefore, the entropy of an Erdős–Rényi model with n nodes and edge probability p can be calculated as below, using the results from Section 2.4.3.

$$H_{G(n,p)} = \binom{n}{2} h(p), \tag{2.17}$$

where $h(p)$ shows the entropy of one single Bernoulli variable with parameter p and can be found using the following equation.

$$h(p) = -p \log p - (1 - p) \log(1 - p) \tag{2.18}$$

2.5.1.1 Compression method

Now that we have calculated the entropy of the Erdős–Rényi model, we move on to find a near-optimal graph encoder for it. As mentioned before, an Erdős–Rényi

graph is essentially a sequence of Bernoulli random variables, which we call a Bernoulli process or a Bernoulli sequence. To be more general, we study the compression of Bernoulli processes with any length and not just those in the form of $\binom{n}{2}$.

Consider the set $X = \{X_1, X_2, ..., X_l\}$, for which we have

$$X_i \sim \text{Bernoulli}(p) \qquad \forall i \in \{1, .., l\}.$$

This set can also be called a Bernoulli process with parameters l and p. We will use q throughout this section to represent $1 - p$. Furthermore, consider S to represent the number of 1s in X. In other words, we have

$$S = \sum_{i=1}^{l} X_i. \tag{2.19}$$

We can use conditional entropy to write

$$H(X, S) = H(X) = H(S) + H(X|S). \tag{2.20}$$

The idea of compressing a binary sequence by conditioning it on its number of 1s is not new [57, Sec. 13.2]. This idea has been proven to be effective in many scenarios. In this section, we propose a novel approach for compressing Bernoulli processes using a variation of this idea. We divide our proposed compression algorithm into two steps: compressing $X|S$ and compressing S.

Compressing $X|S$ Given $S = k$, all the possible sequences are equiprobable and we have

$$H(X|S = k) = \log \binom{l}{k}. \tag{2.21}$$

To compress X given $S = k$, we can simply use enumerative source encoding [66]. Using this method, we can compress $X|S$ to its entropy. The essence of the method

is that we use $\lceil \log \binom{l}{k} \rceil$ bits to encode the lexicographic order of X among all the $\binom{l}{k}$ binary sequences of length l that have k 1s.

Compressing S We now have to compress S in an optimal manner. Notice that as S is the sum of a number of Bernoulli random variables, it in fact has a binomial distribution $\mathrm{B}(l, p)$. To achieve optimal compression, we could use Huffman coding [67] to assign codewords to the $l+1$ possible values of S. But instead, we will provide a much faster method. We could also simply use $\log n$ bits to represent S. However, we will show that our method for compressing S has a smaller mean length than $\log l$.

Suppose that we want to compress $S = k$. We take the following steps to find $C(k)$, the codeword for k.

1. Set $F = 0$ if $k \leq lp$, or $F = 1$ if $k > lp$.

2. Calculate $t = \lfloor \log(|k - \lfloor lp \rfloor| + 1) \rfloor$, and let T be its binary representation using $\lceil \log \log l \rceil$ bits.

3. Calculate $u = |k - \lfloor lp \rfloor| + 1 - 2^t$, and let U be its binary representation using t bits.

4. $C(k) = F + T + U$, where $+$ shows concatenation of the bit sequences.

To reconstruct k having $C(k)$, we identify F, t, and u from the code. Then, we calculate $d = 2^t + u - 1$. If $F = 0$, we have $k = \lfloor lp \rfloor - d$. Otherwise, we have $k = \lfloor lp \rfloor + d$.

The intuition behind this method of compression is that we are describing the distance of k from the mean value of the distribution, and assigning shorter codewords to values closer to the mean, which have a higher probability of occurrence. To this end, the binary representation of this distance is coded in the following manner. Instead of simply using $\log n$ bits, the position of the most significant bit of this number (t) is presented first using $\log \log l$ bits. Then, t bits follow to provide us with

the rest of the binary number. This way, smaller distances will be paired with shorter codewords. Additionally, the existence of F indicates if k is lower than the mean or higher. Knowing F, in addition to the distance, will provide us with the exact value for k.

Based on the discussions, the codeword for X is the code for its number of 1s using the described method, followed by the lexicographic order of X among all sequences with the same Hamming weight. We use the name BernoulliZip for the proposed compression algorithm. Notice that BernoulliZip always produces a prefix code. After observing the initial $1 + \lceil \log \log l \rceil$ bits of the code, t will be known. The following t bits will help us identify k, and then $\lceil \log \binom{l}{k} \rceil$ bits will follow to determine the lexicographic ordering of the sequence.

2.5.1.2 Mean length of the encoder

In this section, we will calculate an upper bound for the mean length of the proposed compression scheme. The codeword consists of two parts: S and the index of X given S. If we show the length of the codeword for X with L and the length of $C(k)$ with L_k, we have the following equation.

$$\mathbb{E}[L] = \sum_{k=0}^{l} \binom{l}{k} p^k q^{l-k} \left(L_k + \left\lceil \log \binom{l}{k} \right\rceil \right) \tag{2.22}$$

As we already know that the encoding of X given S is optimal, we are interested in calculating the mean length of our encoding method for S. Firstly, notice that based on the described algorithm, we have the following equation for the length of $C(k)$.

$$L_k = 1 + \lceil \log \log l \rceil + \lfloor \log(|k - \lfloor lp \rfloor| + 1) \rfloor \tag{2.23}$$

Taking into account the ceiling and floor functions in Eq. 2.23, we can write

$$\mathbb{E}[L_k] \leq 2 + \log\log l + \mathbb{E}[\log(|k - lp| + 1)]. \tag{2.24}$$

As the logarithm is a concave function, we can use Jensen's inequality [57, Thm. 2.6.2] and write

$$\mathbb{E}[\log(|k - lp| + 1)] \leq \log(\mathbb{E}[|k - lp|] + 1) \tag{2.25}$$

As k comes from a $\mathrm{B}(l, p)$ distribution, $\mathbb{E}[|k - lp|]$ is actually the mean absolute deviation (MAD) of the binomial distribution, for which we have the following inequality [68].

$$\mathbb{E}[|k - lp|] \leq \sqrt{lp(1-p)} \tag{2.26}$$

By inserting Eq. 2.26 into Eq. 2.25, we will have

$$\mathbb{E}[\log(|k - lp| + 1)] \leq \log(\sqrt{lpq} + 1). \tag{2.27}$$

Ultimately, Eq. 2.23 and Eq. 2.27 will provide us with the following upper bound on the mean length of our code for the binomial distribution.

$$\begin{aligned} \mathbb{E}[L_k] &\leq \log(\sqrt{lpq} + 1) + \log\log l + 1 \\ &\leq \log(\sqrt{lpq}) + \log\log l + 2 \end{aligned} \tag{2.28}$$

We can now calculate an upper bound for the mean code length of BernoulliZip. The entropy of the original sequence is in fact the entropy of l independent Bernoulli trials and equals $lh(p)$. The difference between BernoulliZip's mean code length and the entropy of the sequence is due to the difference between $\mathbb{E}[L_k]$ and the entropy of $\mathrm{B}(l, p)$. Additionally, there is a difference of at most 1 bit between BernoulliZip's compression of $X|S$ and $H(X|S)$. As the entropy of $\mathrm{B}(l, p)$ is known to be $\frac{1}{2}\log(2\pi elpq) + O(\frac{1}{l})$ [69], we will have the following upper bound on BernoulliZip's

mean code length for large ls.

$$\begin{aligned}\mathbb{E}(L) &\leq lh(p) + \mathbb{E}[L_k] - \frac{1}{2}\log(2\pi elpq) + 1 \\ &\leq lh(p) + \log\log l + \log\sqrt{\frac{1}{2\pi e}} + 4\end{aligned} \tag{2.29}$$

As per Eq. 2.29, the mean code length of BernoulliZip is at most of order $\log\log l$ bits longer than the entropy of the sequence. Therefore, the mean codeword length of BernoulliZip is asymptotically optimal and we have

$$\mathbb{E}(L) \sim lh(p). \tag{2.30}$$

Throughout the rest of this section, we will use c_B to show the constant term in Eq. 2.29, $\log\sqrt{\frac{1}{2\pi e}} + 4$, which is approximately 1.9529.

Some simulation results for comparing the entropy of the process and the mean code length of BernoulliZip are illustrated in Figures 2.6 and 2.7. It can be observed in these figures that the difference between the codeword length of BernoulliZip and the entropy of the source is small.

2.5.1.3 Time complexity analysis

In this section, we will calculate the time complexity of different parts of BernoulliZip, when run on a sequence of length l.

- **Finding the number of 1s in the sequence:** Finding the Hamming weight of a given sequence is the first step we take when compressing the sequence. This is a basic summation of the sequence, and is of $O(l)$ complexity.

- **Lexicographic ordering:** Given the sequence and its number of 1s, we must calculate its lexicographic order using the method provided in [66]. It can be seen that this order can be found using a summation on the sequence, and is at

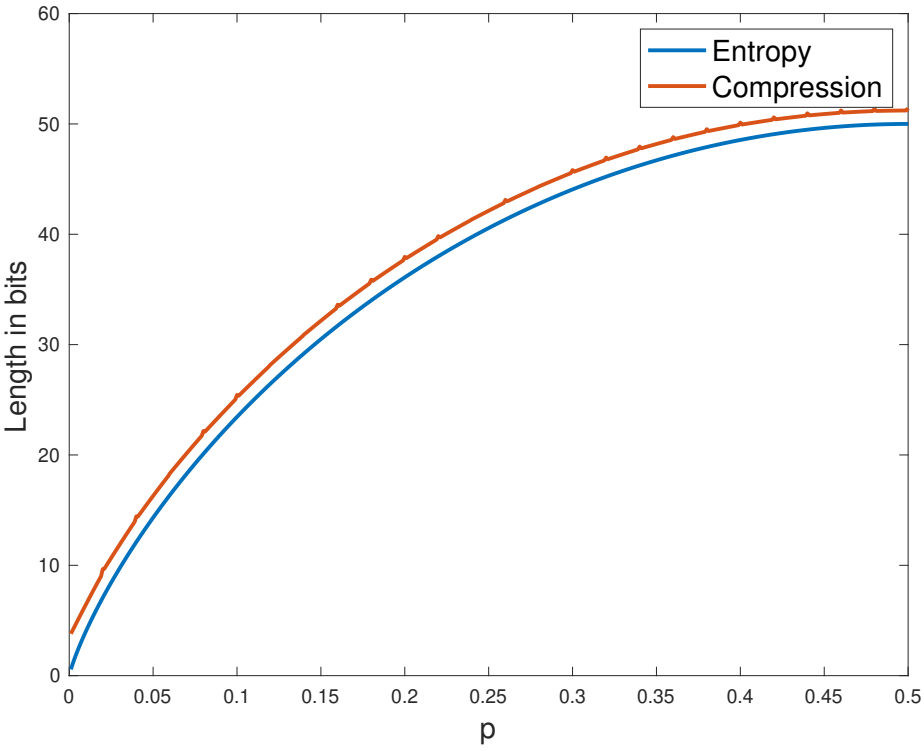


Figure 2.6. Entropy and mean compressed length of Bernoulli processes with length 50 as a function of p

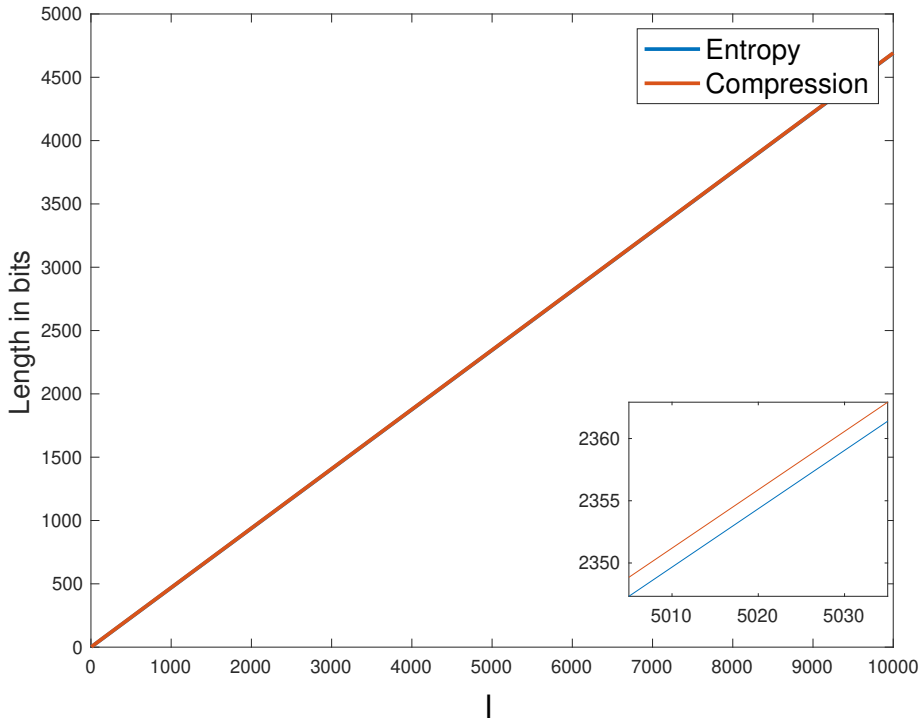


Figure 2.7. Entropy and mean compressed length of Bernoulli processes with $p = 0.1$ as a function of the length of the Bernoulli sequence, l

most of $O(l)$ complexity. However, this is only if we assume the values of the binomial coefficients as given, so that there is no need for calculating them in each step of the summation. This is a valid assumption if l is not very large.

- **Calculating t:** Calculating t requires the calculation of the floor of a logarithm, with its maximum input being equal to l. This can be done using at most $\log l$ multiplications by 2. Therefore, this step is at most of $O(\log l)$ complexity.

- **Calculating u:** To calculate u, a power function needs to be calculated, with the exponent being at most equal to $\log l$. Simple implementations of the power function will do this using at most $\log l$ multiplications. Consequently, this step is also of $O(\log l)$ time.

- **Finding the binary representation of t and u:** t is a number less than $\log l$, and u is a number less than l. Therefore, representing each of them in binary can be done using at most $\log l$ divisions, and is therefore of at most $O(\log l)$ complexity.

As seen above, all steps of the compression process are of $O(l)$ complexity or less. Therefore, BernoulliZip has a time complexity of at most $O(l)$. Table 2.3 compares BernoulliZip with Huffman coding and Shannon-Fano-Elias coding [57, Ch. 5] in terms of time complexity and mean code length. The results illustrate that BernoulliZip is providing us with a much faster compression algorithm while still providing us with a near-optimal mean code length.

Table 2.3. Comparison of different algorithms for compressing a Bernoulli process of length l and parameter p

Algorithm	**Time Complexity**	**Upper Bound to Mean Length**
BernoulliZip	$O(l)$	$lh(p) + \log\log l$
Huffman	$O(l2^l)$	$lh(p) + 1$
Shannon-Fano-Elias	$O(2^l)$	$lh(p) + 2$

2.5.2 Stochastic block model graphs

In this section, we will study the entropy and compression of another important group of random graphs, those generated using the stochastic block model (SBM) [70]. This model divides the n nodes of a graph g into r disjoint partitions $C_1, \ldots, C_r$, also known as communities. The existence probability of an edge that is connecting nodes $v \in C_i$ and $u \in C_j$ is P_{ij}, where P_{ij} is an element from a predefined $r \times r$ matrix, P. This can be seen as dividing the edges within the graph into $r + \binom{r}{2}$ distinct Bernoulli sequences. Our proposed methodology in this section is to use this equivalency to both calculate the entropy, and to apply BernoulliZip to compress graphs generated using the SBM.

The entropy of SBM graphs is straightforward to calculate when the communities are known. This is because the edges can be seen as independent Bernoulli random variables. Identifying the communities is a requirement for efficient compression of SBM graphs [71]. There exist methods which are able to exactly recover these communities or estimate them with a small error [72]. Throughout this section, we use $H(G)$ to show the conditional entropy of an SBM graph source G when the communities are known. The following equation calculates the entropy of a graph source G created using the SBM in terms of its communities and matrix P.

$$H(G) = \sum_{i=1}^{r} \binom{|C_i|}{2} h(P_{ii}) + \sum_{i=1}^{r-1} \sum_{j=i+1}^{r} |C_i||C_j| h(P_{ij}) \tag{2.31}$$

Based on Eq. 2.31, it can be seen that the entropy of an SBM source can be calculated by simply looking at it as multiple ER sources. After having the entropy of the SBM model, we move on to finding an optimal compression algorithm for it. We first discuss two initial approaches to this problem, and then propose a third and more efficient approach. In all of these approaches, it is assumed that the communities $C_1, \ldots, C_r$ and the matrix P are known to both the encoder and the decoder. This is a very strong assumption, but algorithms such as that introduced in [72] can be used

to detect communities with high accuracy.

2.5.2.1 The respective distributions approach

In this approach, we use BernoulliZip to compress groups of edges that have the same distribution. In other words, the edges are divided into $r + \binom{r}{2}$ distinct groups, and BernoulliZip is applied to each of these groups individually. The resulting codes are then concatenated. In order for the code to be uniquely decodable, the groups are sorted based on their Bernoulli parameter. Additionally, the edges within each group are sorted lexicographically according to their respective pair of nodes. As BernoulliZip produces prefix codes, the resulting code of this approach will also be a prefix code.

The redundancy in this approach will be the sum of BernoulliZip's redundancy for each of the $r + \binom{r}{2}$ groups. Using the redundancy value calculated in Section 2.5.1.2, we have the following bound for the resulting total redundancy $\mathscr{R}$ of the respective distributions approach.

$$\begin{aligned}\mathscr{R} \leq & \sum_{i=1}^{r} \log\log \binom{|C_i|}{2} \\ & + \sum_{i=1}^{r-1} \sum_{j=i+1}^{r} \log\log |C_i||C_j| \\ & + \left(r + \binom{r}{2}\right) c_B\end{aligned} \tag{2.32}$$

2.5.2.2 The single distribution approach

The idea in this approach is to assume a single probability distribution for all of the $\binom{n}{2}$ edges of the graph, and then apply BernoulliZip to the entire sequence, assuming the new distribution for all the variables in the sequence. For each bit, the code will have a redundancy equal to the relative entropy of the actual distribution of the bit and the one that we have assumed [57, Ch. 13]. If we assume a Bernoulli parameter

of p for all the variables in the sequence, the redundancy of the single distribution approach as a function of p can be calculated as below.

$$\begin{aligned} R(p) = & \sum_{i=1}^{r} |C_i| D(P_{ii}||p) \\ & + \sum_{i=1}^{r-1} \sum_{j=i+1}^{r} |C_i||C_j| D(P_{ij}||p) \end{aligned} \tag{2.33}$$

The task is to find the optimal value for p, which is the one that minimizes $R(p)$. We do this by taking the derivative of $R(p)$ with respect to p and setting it to zero. It is then shown that the optimal value for p is actually the weighted mean of all the $r + \binom{r}{2}$ Bernoulli parameters, with the weight of each Bernoulli parameter being the number of edges that have a Bernoulli distribution with that parameter in the graph. Eq. 2.34 shows the value of p^*, which minimizes $R(p)$.

$$p^* = \frac{\left(\sum_{i=1}^{r} |C_i| P_{ii} + \sum_{i=1}^{r-1} \sum_{j=i+1}^{r} |C_i| P_{ij} \right)}{\binom{n}{2}} \tag{2.34}$$

The proposed approach of this section is to first calculate p^*, and then compress the whole sequence using BernoulliZip, with p^* as the Bernoulli parameter. Therefore, the total redundancy $\mathscr{R}$ will be the sum of $R(p^*)$ and the error of BernoulliZip as shown by Eq. 2.35.

$$\mathscr{R} \leq R(p^*) + \log\log \binom{n}{2} + c_B \tag{2.35}$$

2.5.2.3 The hybrid approach

As the name suggests, this approach is a hybrid of the previous two approaches. It is reasonable to ask whether we can find an optimal grouping of edges, where the edges in each group are compressed using the single distribution approach, and the resulting codes are concatenated to form the code for the whole graph. The proposed approach of this section finds such a grouping, and the result is a code that has a

shorter average length than the previous two approaches.

To tackle this problem in its most general form, we assume that each edge has its own Bernoulli parameter. Clearly, this parameter can be the same for a number of edges. Therefore, we have a sequence of $\binom{n}{2}$ Bernoulli parameters, and we want to group them in a form that minimizes compression redundancy. In order to minimize the error that arises from relative entropy, we need to choose random variables with close Bernoulli parameters to be in the same group. Therefore, it is rational to assume that the optimal grouping of the Bernoulli random variables for minimizing the introduced error must be in the form of splitting the sorted sequence of the Bernoulli parameters into a number of disjoint groups. Based on this, we will work with the sorted sequence of the $\binom{n}{2}$ Bernoulli parameters $(p_1, p_2, \ldots, p_{\binom{n}{2}})$, where $p_1 \leq p_2 \leq \ldots \leq p_{\binom{n}{2}}$.

Our proposed method is to study the average code redundancy per bit in order to find a better grouping for the random variables. We examine the random variables one by one starting from p_1, updating the value of average redundancy per bit as we move forward. The average redundancy takes into account both BernoulliZip's redundancy and the relative entropy redundancy using the mean of the parameter of all the observed and ungrouped Bernoulli random variables. As soon as we observe a rise in the average redundancy per bit, we add a boundary, meaning we end the current grouping and start a new one. Assume that we observe this rise at p_i. In this case, all the variables without a group with labels less than i will be assigned to the same group. The whole process is then started again from p_i. A pseudocode for this method is shown in Algorithm 2.5.1.

In Algorithm 2.5.1, the function BernoulliZip_redundancy(X) returns the compression redundancy caused by BernoulliZip and relative_entropy_redundancy(X) returns the compression redundancy caused by the mismatch between the true and assumed distributions. Notice that this compression redundancy is measured when mean prob-

Algorithm 2.5.1: The Hybrid Approach

```
1  X = input sequence;
2  L = length of input sequence;
3  start_index = 1;
4  last_group = 1;
5  old_mean_redundancy = +∞;
6  for i = 1, ..., L do
7  |   mean_redundancy = (BernoulliZip_redundancy(X[start_index :
   |    i]) + relative_entropy_error(X[start_index : i]))/(i − start_index + 1);
8  |   if mean_redundancy > old_mean_redundancy then
9  |   |   X[start_index : i − 1] are grouped as group number last_group;
10 |   |   last_group = last_group + 1;
11 |   |   start_index = i;
12 |   end
13 |   old_mean_redundancy = mean_redundancy;
14 end
```

ability distribution is used as the Bernoulli parameter as suggested by Eq. 2.34. The result of running Algorithm 2.5.1 on an example Bernoulli sequence can be seen in Fig. 2.8. The figure shows the difference between the mean length produced by each method and the entropy of the sequence. The simulation is performed on a sequence of 500 Bernoulli random variables, with a random parameter from a set of 20 parameters. For each point on the x-axis, the result for compressing the sequence from the beginning up to that point is illustrated.

As seen in Fig. 2.8, the hybrid approach is performing much better than the single distribution approach, and is closer to the optimal than the respective distribution approach as the length of the sequence grows large. Therefore, our proposed solution to compressing stochastic block model graphs is the hybrid approach.

2.6 Summary

In this chapter, we provided the information-theoretic fundamentals for studying random graph sources. We started the chapter by providing background works on

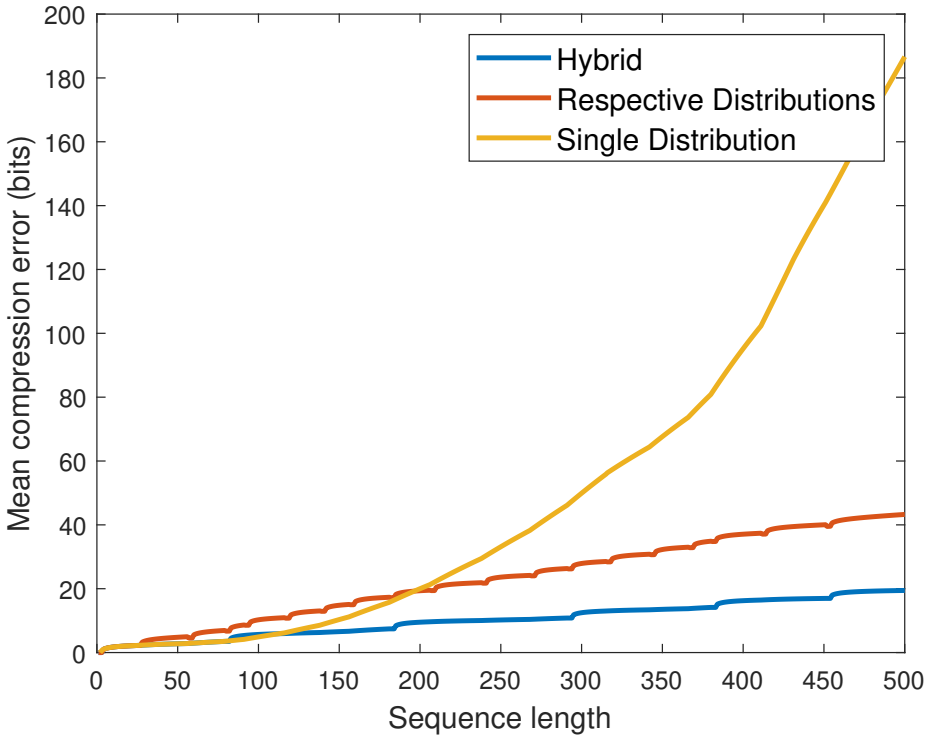


Figure 2.8. A comparison between the 3 proposed approaches to compressing SBM graphs

both measuring the entropy of graphs and graph compression. We then moved on to formally define random graph sources, their entropy, and their encoding. It was noted that in many real-world scenarios, the random graph model for best simulating the networks at hand is not necessarily known. Therefore, we provided an approach to estimate the entropy of graphs and find upper bounds to it. We then continued our discussions by studying the entropy and compression of two of the most well-known random graph models that exist to simulate networks: the Erdős–Rényi model and the stochastic block model. We quantified the entropy of these two models and provided near-optimal encodings for each of them. In the following chapters, we study other scenarios that can arise when studying networks and discuss the best approaches that can be undertaken to analyse them using information-theoretic tools.

Chapter 3

Kolmogorov Complexity of Graphs

To date, there have been numerous attempts to capture and measure the complexity of graphs [13]. These attempts have used different methods to provide a complexity measure for networks. As stated in Chapter 1, these approaches can be divided into two categories: deterministic and probabilistic. Kolmogorov complexity and Shannon's entropy [29] are the dominant approaches in the deterministic and probabilistic domains, respectively. In Chapter 2, we studied the entropy of random graph sources. We argued that whenever the behaviour of a real-life network can be modeled using a random graph source, we can study its complexity and compressibility using Shannon's entropy. However, this might not always be the case.

It is known that entropy is the dominant approach for calculating the complexity of a random variable, especially when doing so as a metric for that variable's compressibility. Notice that entropy is defined for an ensemble, and it needs a probability distribution defined on the random variable at hand. However, when studying graphs, we are sometimes faced with a single one rather than an ensemble, and the probability distribution for a single graph can not be known or estimated. Consider networks that undergo minimal changes through time. We use the term static to refer to such networks. Some examples of static networks are road networks and protein-protein

interaction networks. As a probability distribution on these graphs can not be well defined, we use the measure of Kolmogorov complexity to study this type of networks.

In this chapter, we will study the Kolmogorov complexity of graphs. After going through some of the background work, we start from basic definitions of Kolmogorov complexity, and then present a method to estimate the value of Kolmogorov complexity of graphs. Additionally, we also provide an encoding Algorithm that exhibits a near-optimal performance with respect to Kolmogorov complexity.

3.1 Background

Kolmogorov complexity (K-complexity) [14] of a string of characters is the length of the shortest possible program that when fed into a universal Turing machine, outputs that string. Since its introduction by Kolmogorov, this concept has been used as a measure of complexity and randomness [15]. It is proven that Kolmogorov complexity of a string is incomputable [16]. Therefore, we can only estimate it by finding upper bounds to it [17]. Having a more accurate estimate of the upper bound of Kolmogorov complexity has been of great importance, as it will help in areas such as data compression and data mining [73, 74].

As seen in section 2.4.2, a simple graph on n vertices can be fully represented using a sequence of $\binom{n}{2}$ bits, which can be seen as a reshaping of the graph's adjacency matrix. As a result, $\binom{n}{2}$ is often seen as the upper bound for the K-complexity of all graphs with n nodes [13]. Additionally, for some graphs, other representation methods such as adjacency lists might result in codes shorter than $\binom{n}{2}$ bits, which will result in a lower upper bound for the K-complexity of these graphs. Reference [75] obtains a canonical code for the structure of a graph by reordering the rows and columns of its adjacency matrix, as an estimate of its K-complexity. From a more general viewpoint, the codeword length of any lossless compression method for graphs can

be considered as an upper bound to the K-complexity of graphs [76]. There exist numerous compression and summarisation algorithms focusing on graphs [77,78], but these algorithms are not directly tested for calculating the complexity of graphs. Other methods have a more general approach and utilize the notion of algorithmic probability to feed random programs into a Turing machine and count the number of times that the machine produces the desired output [76], [18]. However, we strongly believe that a formalised method for estimating the K-complexity of graphs based on their structural properties is very much needed. Based on this, the main goal of this chapter is to provide a formal analysis of the K-complexity of graphical data structures.

3.2 Basics and definitions

Our discussions commence with the presentation of relevant definitions and foundational concepts to familiarise the readers with the concept of K-complexity, particularly in the context of graphs. This introductory step is intended to facilitate a comprehensive understanding of the achieved results. This section also contains some new theorems and lemmas that will be used later on to prove the main findings of this chapter.

Before beginning our discussions, we first provide the mathematical definition of K-complexity.

Definition 3.2.1 (Kolmogorov Complexity)**.** [57, Sec. 14.2] The Kolmogorov complexity (K-complexity) $K_{\mathcal{U}}$ of a string x with respect to a universal computer $\mathcal{U}$ is defined as

$$K_{\mathcal{U}}(x) = \min_{p:\mathcal{U}(p)=x} l(p),$$

which shows the minimum length over all programs p that print x and halt.

As seen above, the definition of K-complexity depends on having a universal com-

puter $\mathcal{U}$. In all of our discussions in this chapter, we will consider $\mathcal{U}$ to be the universal Turing machine. This machine, introduced by Alan Turing back in 1937 [79], is able to output any computable sequence if the right program is inputted. It is now known that all recently developed computational systems can be simplified to the level of a Turing machine [57, Ch. 14]. Therefore, in this thesis, we consider $\mathcal{U}$ to be the deterministic universal Turing machine, and simplify the notation of K-complexity of x to $K(x)$.

Additionally, note that there is usually an assumption about the prior knowledge that $\mathcal{U}$ has about x, before the program p is fed into it. For example, it might be assumed that the length of x is known to $\mathcal{U}$. In this case, we will use conditional K-complexity. $K(x|y)$ shows the K-complexity of x when y is known to $\mathcal{U}$. In the application of K-complexity to graphs, we often assume that n, the number of nodes of graph g, is known to $\mathcal{U}$. We can show the K-complexity of g in this case with $K(g|n)$. If it is clear that n is known by $\mathcal{U}$, we will use the simpler notation $K(g)$ to refer to the conditional K-complexity. Moreover, as K-complexity is defined for a string x, we can assume x to be the sequence of bits in the adjacency matrix of the graph. This is because of the fact that for labelled simple graphs there exists a one-to-one mapping between the adjacency matrices and the graphs, as shown in Section 2.3.1.

3.2.1 Kolmogorov basic graphs

In this section, we look for the least complex graphs possible with n vertices. We assume that the vertices of an n-node graph are labelled from 1 to n, and that the value of n is known to the machine. We first provide the definition for randomness deficiency, and then basic graphs are introduced as the easiest graphs to describe.

Definition 3.2.2. [15, Def. 6.4.2] A labelled graph g on n nodes has randomness

deficiency at most $\delta(n)$, and is called $\delta(n)$-random, if it satisfies

$$K(g|n,\delta) \geq \binom{n}{2} - \delta(n),$$

where $K(g|n,\delta)$ shows the K-complexity of the binary sequence of g's adjacency matrix given n and δ.

Definition 3.2.3 (Basic graphs)**.** A graph is basic if given n, its Kolmogorov complexity is of $O(1)$. In other words, basic graphs have a random deficiency of $\binom{n}{2} - O(1)$.

An example of a basic graph is the complete graph. If n is known, the entire graph can be described by simply stating that it is a complete graph using a constant number of bits. The number of possible basic graphs is limited, as stated by the following theorem.

Theorem 3.2.1. *The number of basic graphs is of* $O(1)$.

Proof. It is known that a fraction of at most $1/2^{\delta(n)}$ of all labelled graphs on n nodes is $\delta(n)$-random [15, Lemma 6.4.1]. Consequently, a fraction of at most $1/2^{\binom{n}{2}-O(1)}$ of the $2^{\binom{n}{2}}$ possible labelled graphs on n nodes is basic. Therefore, the number of possible basic graphs is at most $2^{O(1)}$, which is a constant number. □

In order to provide the conditions under which a graph is basic, we will use the following definitions.

Definition 3.2.4 (Circular shift)**.** A circular shift in a graph with n nodes, where the nodes are labeled from 1 to n, is an operation that reassigns the node labels such that node 1 receives the new label n, and each node labeled i (for $2 \leq i \leq n$) receives the new label $i-1$.

Definition 3.2.5 (Shift coefficient)**.** The shift coefficient of a graph is the minimum positive number of circular shifts that needs to be applied to its node labels in order

to reach a graph with the same adjacency matrix again. We use $S(g)$ to show the shift coefficient of graph g.

Notice that in Definition 3.2.5 the direction in which the circular shift is applied to the labels is not important. This is because if we reach the same graph with a number of shifts in one direction, applying the same number of shifts in the opposite direction will also provide us with the same graph. Additionally, the maximum shift coefficient possible for a graph is n, as n shifts will definitely provide us with the same graph again. The following lemma provides us with a better insight into the possible values of shift coefficient for a graph.

Lemma 3.2.1. *The shift coefficient of a graph g with n nodes is always a divisor of n.*

Proof. It is easy to check that n circular shifts will always provide us with the same graph. If $S(g) = n$, the claim of the lemma holds. If not, then we must have $S(g) < n$. Consider q and r to be the quotient and remainder of dividing n by $S(g)$, respectively. Notice that as both $qS(g)$ and n circular shifts will result in the same graph, then applying $r = n - qS(g)$ circular shifts should also result in the same graph. However, we have $0 \leq r < S(g)$. r can not be a positive value because if it was, it would contradict with the definition of $S(g)$. Therefore, we must have $r = 0$, and $S(g)$ should be a divisor of n. □

Fig. 3.1 provides an example of finding the shift coefficient for a graph. It can be seen that we have the same graph after 2 circular shifts in the node labels.

It is notable to mention the similarities between the definition of the shift coefficient and that of circulant graphs [80]. Circulant graphs can be defined as graphs with a shift coefficient of 1.

We are now equipped to state and prove the following theorem.

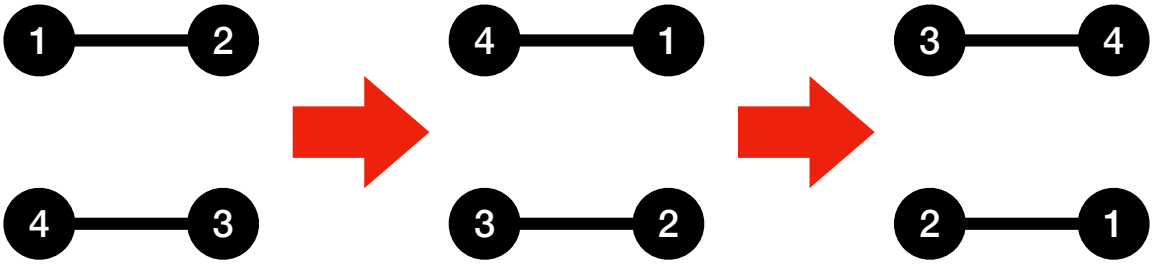


Figure 3.1. A graph with shift coefficient of 2

Theorem 3.2.2. *A graph g with n nodes is basic if and only if it satisfies the following two conditions when given n.*

1. *It has a shift coefficient with $O(1)$ Kolmogorov complexity.*
2. *The first $S(g)$ columns of its adjacency matrix have a Kolmogorov complexity of $O(1)$.*

Proof. Assume that graph g satisfies the two conditions. This graph can be described only by representing the connections of its first $S(g)$ vertices. The connections of other vertices can easily be inferred by shifting the first $S(g)$ nodes, based on the definition of shift coefficient. Additionally, describing the connections of the first $S(g)$ nodes is of $O(1)$ complexity. Therefore, representing $S(g)$ and the connections is also of $O(1)$ complexity, hence the constant Kolmogorov complexity of the whole graph.

We prove the second side of the theorem by contradiction. Suppose there exists a graph g with $O(1)$ Kolmogorov complexity that does not satisfy one of the two conditions of the theorem, meaning either the shift coefficient or the first $S(g)$ columns of the adjacency matrix of g have a K-complexity larger than $O(1)$. As both $S(g)$ and the first $S(g)$ columns of the adjacency matrix can be inferred from the graph and the graph is of $O(1)$ K-complexity, then they must have $O(1)$ K-complexity themselves. This is because any representation of g is also a representation of $S(g)$ and the first $S(g)$ columns of the adjacency matrix. As a result, a graph can not be basic unless it satisfies both conditions of the theorem. □

We can also state and prove the following corollaries to Theorem 3.2.2. The first corollary is about the complement of a graph, which is created by reversing the state of all $\binom{n}{2}$ edges in the graph. The second corollary discusses the isomorphisms of basic graphs, where isomorphism is defined according to Definition 2.4.1.

Corollary 3.2.2.1. *The complement graph of any basic graph is also a basic graph.*

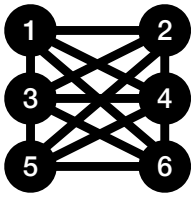
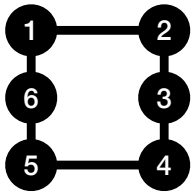
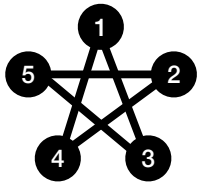
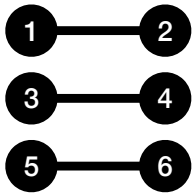
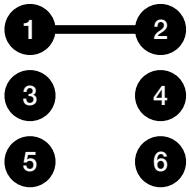
Proof. The description of any basic graph can be followed by an instruction of length $O(1)$ to complement the graph. Therefore, the complement of a basic graph can be considered a basic graph itself. □

Corollary 3.2.2.2. *Isomorphisms of basic graphs are of at most $O(n \log n)$ K-complexity.*

Proof. If a graph is an isomorphism of a basic graph, it can be represented in the following manner. Firstly, one can describe the basic graph using $O(1)$ bits. Additionally, the relabelling of each node can be described using at most $\lceil \log n \rceil$ bits that describe the new label of that node. Therefore, the relabelling of all nodes can be described using at most $n\lceil \log n \rceil$ bits. Consequently, the whole graph can be described using at most $n\lceil \log n \rceil + O(1)$ bits, which is of $O(n \log n)$ K-complexity. □

Theorem 3.2.2 provides us with a powerful tool to identify basic graphs. Many graphs are easily ruled out from being a basic graph using the first condition, as the shift coefficient can easily be calculated for any graph. The second condition is harder to check, as not all binary sequences with constant Kolmogorov complexity are known. However, we are still able to provide a list of graphs that fulfil both conditions and are therefore basic graphs. Table 3.1 shows some of the graphs that fulfill the conditions of Theorem 3.2.2. It must be noted that Table 3.1 only provides some examples of basic graphs. As their exact number is not known, there may exist other basic graphs. Therefore, the set of basic graphs used throughout this chapter is only a subset of all possible basic graphs. The problem of identifying all basic graphs remains open.

Table 3.1. Examples of basic graphs

Name	Shift coefficient	Connection description	Example graph
Complete graph	1	Each node connects to all other nodes.	
Cycle graph	1	Each node connects to the nodes directly before and after itself.	
Star polygon graph with a density of 2	1	Each node connects to the nodes directly two positions before and after itself.	
Ladder rung graph	2	Odd nodes connect to the node after, even nodes connect to the node before.	
One-edge graph	n	There is an edge between the first two nodes.	

Notice that even though the definition of shift coefficient can usually help us in identifying basic graphs, it may not add any additional information for some cases. Based on Lemma 3.2.1, for a graph with n nodes, the possible set of shift coefficients consists of n and its divisors. Therefore, if n is prime or the set of its divisors has $O(1)$ members, then all possible shift coefficients of the graph are of $O(1)$ K-complexity. For these graphs, the shift coefficient condition does not provide us with any additional information about the K-complexity of the graph and we must only look at the connections between nodes.

3.2.2 Kolmogorov random and non-random graphs

In the context of Kolmogorov complexity, randomness has a specific definition, which is different from what was discussed in Chapter 2. In this section, we will distinguish between random and non-random graphs with respect to Kolmogorov complexity, and will provide a methodology to detect which graphs are random and which are not.

Definition 3.2.6 (Random graphs)**.** A graph with n nodes is a random or incompressible graph if it can not be described to a universal Turing machine using less than $\binom{n}{2}$ bits. In other words, a random graph is a graph with a randomness deficiency equal to zero. Any other graph will be called non-random or compressible.

It is interesting to search for the maximum possible number of non-random graphs that exist on n nodes. Notice that in order for a graph to be considered non-random, it should be able to be described using $\binom{n}{2} - 1$ bits or less. Using $\binom{n}{2} - 1$ bits, one can index at most $2^{\binom{n}{2}-1}$ graphs. Therefore, a fraction of at least half of all possible graphs will remain to be indexed using $\binom{n}{2}$ bits or more. Consequently, at least half of all possible graphs with n nodes are random graphs.

The following definition will aid us in our search for the properties of non-random graphs.

Definition 3.2.7 (Basic subgraph)**.** A subgraph g' of graph g is called a basic subgraph if g' is a basic graph given its number of vertices k, and an appropriate relabelling of its nodes using the labels $\{1, \ldots, k\}$.

We will now prove an important theorem about random graphs using the incompressibility method [81].

Theorem 3.2.3. *Random graphs do not contain any basic subgraphs with more than* $2 \log n + 4$ *vertices.*

Proof. Suppose that there is random graph which includes a basic subgraph of size $2\log n + 5$ or larger. We provide a method to represent such a graph using fewer than $\binom{n}{2}$ bits, which proves the theorem by contradiction.

Suppose that a graph with n nodes contains a basic subgraph with k nodes. To describe a basic subgraph with k nodes, we first need to specify k. We can do this using $\lceil \log n \rceil$ bits. Additionally, we need to describe the nodes that form this subgraph. The nodes of the graph can be labelled using $\lceil \log n \rceil$ bits. Therefore, $k\lceil \log n \rceil$ bits are needed to specify the k nodes of the basic subgraph. Notice that for some basic graphs, the order in which the label of these nodes are listed may be of importance. For example, if the basic graph is a cycle, the labels of its nodes need to be listed in the order of their connection. Additionally, $O(1)$ bits are needed to specify the basic graph at hand. Using this method, instead of using $\binom{k}{2}$ to represent the edges of a basic subgraph, we are using $k\lceil \log n \rceil + \lceil \log n \rceil + O(1)$ bits. As the constant term can be ignored as n grows large, we can use the following inequality to determine the values of k for which this method of representation provides us with a code with less than $\binom{n}{2}$ bits.

$$\binom{k}{2} > (k+1)\lceil \log n \rceil \tag{3.1}$$

On the other hand, the following inequality holds for the right hand side of Eq. 3.1.

$$(k+1)\lceil \log n \rceil \leq (k+1)(\log n + 1) \tag{3.2}$$

Therefore, we search for values of k that satisfy

$$\binom{k}{2} > (k+1)(\log n + 1). \tag{3.3}$$

Solving the quadratic inequality of Eq. 3.3 provides us with the following positive

integer values for k.

$$k > 2\log n + 4 \tag{3.4}$$

Therefore, a random graph can not have a basic subgraph larger than $2\log n + 4$. Otherwise, it can be described with the explained method and have a positive randomness deficiency. □

Notice that the graph representation method used in the proof of Theorem 3.2.3 results in descriptions with the same length for isomorphic graphs. However, there are techniques that can be used to reduce the codeword length even more for some graphs, depending on their node labelling. For example, we can use techniques such as variable-length encoding and gap encoding [78] to describe the node labels of a basic subgraph more efficiently. An optimal method for describing the node labels of a basic subgraph should also use a code of length $O(1)$ when all the nodes of a graph form a basic graph, and the node ordering is not important (such as a complete graph). Throughout the rest of this paper, we will assume that we use one of such techniques to code a set of ordered node labels. The function label_coding(L) returns a coded version of its input L, which is an ordered node list. Notice that according to the method used in the proof of Theorem 3.2.3, the code length for a list of k nodes should not be longer than $(k+1)\lceil\log n\rceil$ bits.

Because of the importance of the subgraphs discussed in Theorem 3.2.3, we provide a definition for them below.

Definition 3.2.8 (Semi-basic graphs)**.** A graph on n nodes is called semi-basic if it contains at least one basic subgraph with more than $2\log n + 4$ nodes.

Theorem 3.2.3 states an important feature of random graphs. Additionally, it also specifies an important family of non-random graphs, which we have defined as semi-basic graphs in Definition 3.2.8. This theorem acts as the basis of our method

for detecting non-randomness in graphs. We will introduce our proposed method in full detail in the following section.

3.3 Kolmogorov graph covering

In this section, we will provide an algorithm that we call Kolmogorov graph covering. The aim of this algorithm is to provide an estimate to the K-complexity of its input graphs, by breaking it down into simple building blocks. In our proposed method, we use semi-basic graphs as these building blocks. As discussed in the previous section, semi-basic graphs provide a reliable measure of detecting non-randomness in graphs. Therefore, we use these graphs as the basis of our proposed method for calculating an upper bound on the Kolmogorov complexity of graphs. Kolmogorov graph covering therefore tries to cover its input graph with basic graphs of size $2\log n + 5$ or larger, thus reducing the complexity of the input graph. Algorithm 3.3.1 demonstrates Kolmogorov graph covering.

Algorithm 3.3.1: The Kolmogorov graph covering algorithm

```
1  g = input graph;
2  complexity = (n choose 2);
3  for i = n, n − 1, ..., 2 log n + 5 do
4  |   for s ⊂ V(g), |s| = i do
5  |   |   if the nodes from s form a basic graph in g then
6  |   |   |   L = ordered list of all nodes in s;
7  |   |   |   complexity = complexity − (|s| choose 2) + length (label_coding(L)) + O(1);
8  |   |   |   Remove s from g;
9  |   |   end
10 |   end
11 |   if |V(g)| < 2 log n + 5 then
12 |   |   Break the loop;
13 |   end
14 end
```

The essence of Algorithm 3.3.1 is that it finds the biggest basic subgraphs of its input graph, and uses those subgraphs to reduce the complexity of its input if

possible. If it encounters a basic subgraph larger than $2\log n$+4, the algorithm uses the explained method for describing basic subgraphs to reduce the complexity of the input graph. Afterwards, it will remove the nodes of the basic subgraph from the original graph, and will search for other basic subgraphs. The reason for this is to avoid having any duplicate edges when covering the graph with basic subgraphs. Ultimately, the algorithm outputs its estimation for the upper bound of the K-complexity of its input graph.

The graph coding method corresponding to the complexity calculated in Algorithm 3.3.1 is as follows. First, the basic subgraphs within the original graph are presented, using $O(1)$ bits each. Then, the nodes that are part of these subgraphs are specified in order. Finally, the code ends with the simple adjacency matrix representation of unspecified edges in a lexicographical order.

Note that Algorithm 3.3.1 has a very high time complexity. In its worst case scenario, the algorithm goes through all possible subgraphs larger than $2\log n + 4$, which is of $O(n^2)$ time complexity. Additionally, at each step, it checks if a given graph is a basic graph. This requires at least going through all the nodes of each subgraph, which is of $O(n)$ time complexity. Consequently, Algorithm 3.3.1 is of at least $O(n^3)$ time complexity and is not suitable to be used as a graph compression algorithm itself. However, if computation time was to be ignored, Algorithm 3.3.1 would have been a powerful graph compression algorithm.

The main purpose of Algorithm 3.3.1 is to act as a tool for calculating an upper bound on the K-complexity of graphs. This can have many applications, especially in graph compression. One of the most important application scenarios of Algorithm 3.3.1 is to provide an evaluation metric for graph compression algorithms. We can test the performance of a graph compression Algorithm for a particular graph by inputting that graph into both the compression algorithm and Algorithm 3.3.1, and then comparing the output length of the compression algorithm with the complexity

estimation that Algorithm 3.3.1 has provided. If the compression algorithm has a larger output length than the output of Algorithm 3.3.1, it is inferred that the input graph could have been compressed more efficiently. Otherwise, we conclude that the compression algorithm is performing well in compressing that particular graph, and it has found a better upper bound on the K-complexity of the input graph. Notice that this method is only valid for assessing a compression method when the method performs the compression solely based on its input graph, and does not assume a probability distribution on an ensemble of graphs.

It is not claimed that Algorithm 3.3.1 provides the most compressed version of the graph possible. One of the most important examples of this is sparse graphs. For sparse graph, the adjacency list representation of graphs may provide us with a lower K-complexity for these graphs. Similarly, very dense graphs may also be represented using fewer bits than what Algorithm 3.3.1 provides us with by only listing the missing edges. The main claim is that semi-basic graphs can be used to distinguish between random and non-random graphs, and that Algorithm 3.3.1 provides us with a relatively good upper bound for the K-complexity of different graphs. Additionally, one of the key advantages of Algorithm 2.5.1 is that unlike most algorithms, it does not only depend on the existence of edges, and is able to capture simplicities that are caused by the absence of edges as well.

Algorithm 3.3.1 is ignoring the simplicities that may exist in the connection between pairs of basic subgraphs after they are found. The algorithm is assuming that the adjacency matrix representation is used to show the connections between the basic subgraphs. However, these connections may be further simplified and be used to reduce the K-complexity of the whole graph even more. To this end, Algorithm 3.3.2 is presented as an optional expansion of Algorithm 3.3.1.

Algorithm 3.3.2 can be run on the input graph after Algorithm 3.3.1 to reduce the complexity of the connections between the basic subgraphs found by the Kolmogorov

Algorithm 3.3.2: Kolmogorov graph covering enhancement add-on

```
1  B = set of found basic subgraphs;
2  for ∀b1, b2 ∈ B do
3      b'1 = b1;
4      b'2 = b2;
5      repeat
6          q = the biggest unseen basic subgraph using the nodes in b'1 and b'2
             (Edges within b'1 and b'2 can be chosen arbitrarily to facilitate this.);
7          v1 = set of nodes from b'1 in q;
8          v2 = set of nodes from b'2 in q;
9          if (|v1| + |v2|) log n < |v1||v2| then
10             L = ordered list of all nodes in v1 and v2;
11             complexity = complexity−|v1||v2| + label_coding(L) + O(1);
12             remove v1 from b'1;
13             remove v2 from b'2;
14         end
15     until there remains no basic subgraph in b'1 ∪ b'2;
16 end
```

graph covering algorithm. Because the edges within each basic subgraph have once been taken into account by Algorithm 3.3.1, there is no need to list them again. Instead, the state of those edges can be chosen in a way that contributes towards building a bigger basic subgraph using the nodes of pairs of basic subgraphs found by Algorithm 3.3.1. This will ultimately result in a lower K-complexity for the graph. The condition checked in the if statement of Algorithm 3.3.2 is to ensure that using the basic graph representation for the selected vertices will result in a reduction in the K-complexity of the graph.

Note that Algorithm 3.3.2 also has a very high time complexity. Given that it loops through all possible pairs of basic subgraphs, and goes through each one again in a manner similar to Algorithm 3.3.1 to find new basic subgraphs, its worst case time complexity can be estimated as $O(n^4)$.

Fig. 3.2 illustrates the procedure of Algorithm 3.3.2 using a simple example. The first step is a basic Kolmogorov graph covering where nodes 2,3, and 4 have been detected as a basic graph (complete graph). To describe the connections between

node 1 and the complete graph, the enhancement algorithm has detected another basic graph (null graph) with nodes 1,3, and 4 by considering the edge between nodes 3 and 4 to be non-existent. Notice that the aim of this figure is to illustrate the method used in Algorithm 3.3.2, and does not necessarily reduce the complexity of this particular graph (because of its small size).

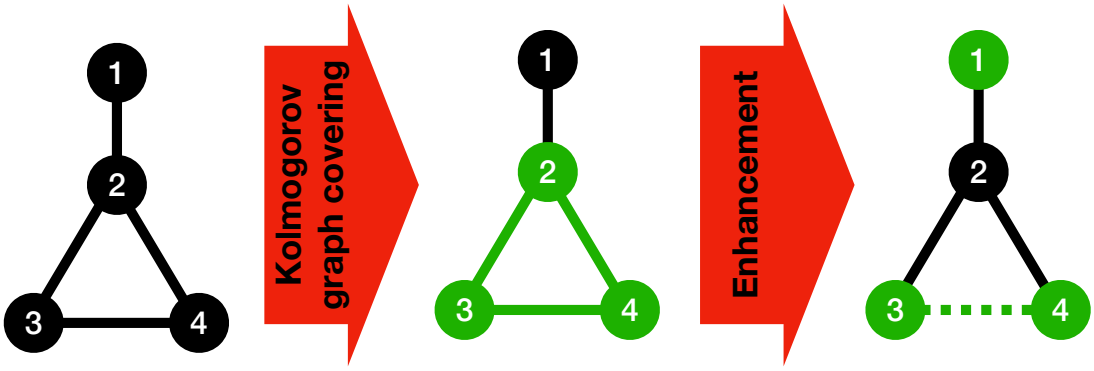


Figure 3.2. An example of enhanced Kolmogorov graph covering

We have discussed the evaluation of the performance of graph compression algorithms on single graphs, but we are often interested in assessing the compression algorithms in general. To achieve this, the following method is proposed. Instead of using a single graph, one can have a set of k graphs $\{g_1, g_2, ..., g_k\}$. Ideally, we need to have all possible graphs on n nodes in this set to completely capture the performance of the compression algorithm, but this may not be feasible. Let $L_A(g_i)$ and $K'(g_i)$ return the length of the compressed version of g_i using graph compression algorithm A, and the estimated Kolmogorov complexity of g_i using Kolmogorov graph covering, respectively. We can use the following formula as a metric to evaluate the performance of the compression algorithm.

$$M(A) = \frac{1}{k}\sum_{i=1}^{k} K'(g_i) - L_A(g_i) \tag{3.5}$$

$M(A)$ in Eq. 3.5 acts as a variable that can show the performance of the graph compression algorithm A. If $M(A)$ is a positive value, it means that the compression

algorithm is providing a shorter code length than the complexity values returned by algorithm 3.3.1 on average. If $M(A)$ is a negative value, it means that the average code length for graphs present in the set can be less than what A achieves. All in all, $M(A)$ can be used as a metric to assess the performance of different graph compression methods and compare them together. Note that, this formula is only an example of the possible ways in which our estimate for the K-complexity can be used to assess graph compression algorithms.

3.4 Simulation results

In this section, some applications of Kolmogorov graph covering are explored and simulation results using MATLAB are presented.

We have built a pool of basic graphs to be used in our code for the Kolmogorov graph covering algorithm. As mentioned earlier, we can not be sure about the number of possible basic graphs. Therefore, we are only using a subset of them in our code. The following graphs and their complements are included in our pool of basic graphs: complete graph, cycle graph, ladder rung graph, and one-edge graph. These graphs are all present in Table 2.1. It is interesting to observe that even with such limited number of basic graphs, one can detect elements of non-randomness in the input graphs.

The reported results for the Kolmogorov graph covering algorithm contain an extra term of $O(1)$. This term is added to remind the reader of the $O(1)$ bits that need to be used for representing the discovered basic subgraphs. Even though the exact number of bits that are going to be used for representing basic graphs can not be precisely known and may vary based on the coding technique, it suffices to note that it does not grow with n.

3.4.1 Traditional graph representation methods

In this section, the results of our method for estimating the complexity of graphs is compared with code lengths from traditional graph coding techniques, namely adjacency matrix and adjacency list.

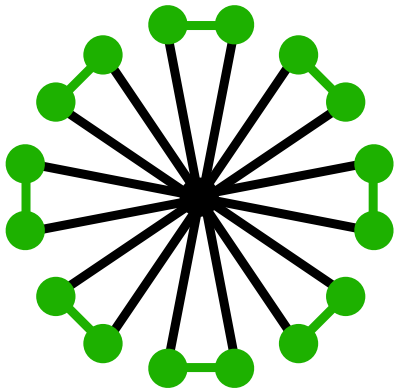
Fig. 3.3 illustrates two sample graphs, alongside with their K-complexity estimate using three approaches. Adjacency matrix and adjacency list have been used as coding methods that will provide us with an upper bound on the K-complexity of graphs. The Kolmogorov graph covering algorithm has also been run on both graphs. The green nodes and edges show the basic subgraphs found by the Kolmogorov graph covering algorithm. The basic subgraphs in figures 3.3a and 3.3b are a ladder rung graph and a null graph, respectively. Kolmogorov graph covering has presented a much lower K-complexity for the first graph compared to its adjacency list and adjacency matrix representation. For the second graph, all three approximations are returning nearly the same result.

3.4.2 Zachary's karate club

Zachary's karate club is a graph based on the social interactions of the members of a karate club that was studied by Wayne W. Zachary [82]. This graph is often used as a data set in studies related to network science. In this section, we apply the Kolmogorov graph covering algorithm to this 34-node graph and measure its complexity. The result is illustrated in Fig. 3.4[1]. The nodes in green are detected by the algorithm as forming a basic subgraph (null graph).

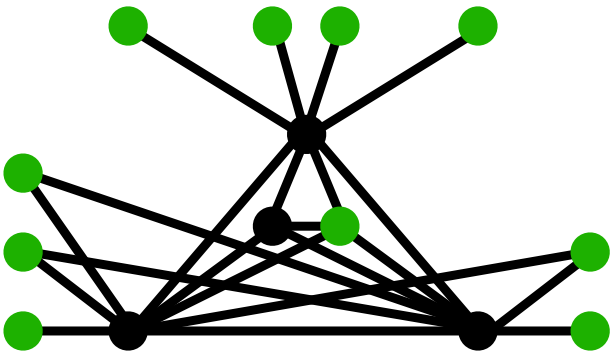
Fig. 3.4 shows that the complexity of this graph is much less than its adjacency matrix and adjacency list representations, and this is done by detecting only one basic subgraph.

[1]Original graph picture source: `https://www.learndatasci.com/tutorials/k-means-clustering-algorithms-python-intro/`



Adjacency matrix	136 bits
Adjacency list	120 bits
Kolmogorov graph covering	100 + O(1) bits

(a) Example graph 1



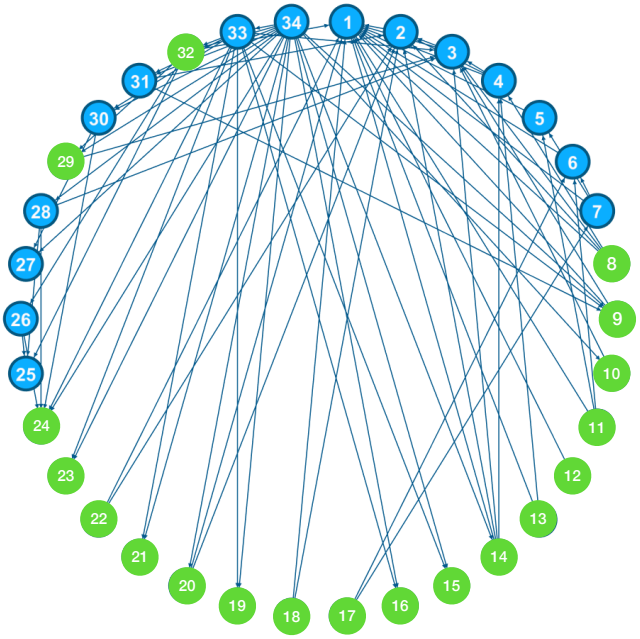
Adjacency matrix	91 bits
Adjacency list	88 bits
Kolmogorov graph covering	90 + O(1) bits

(b) Example graph 2

Figure 3.3. Two sample graphs shown with their K-complexity estimates using three methods: adjacency matrix, adjacency list, and Kolmogorov graph covering

3.4.3 Existing graph compression algorithms

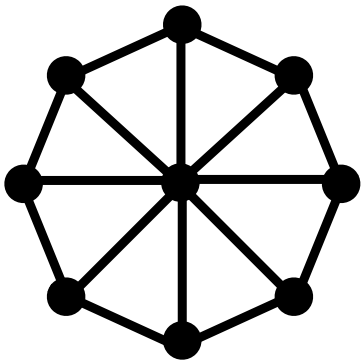
As stated, one important application of the Kolmogorov graph covering algorithm is to test and evaluate existing graph compression methods. In this section, we will evaluate the performance of GraphZip [28] on a number of graphs. GraphZip was introduced back in 2018 as a lossless graph compression method for sparse graphs. This graph compression method identifies the biggest cliques within the input graph, and defines a new pseudo vertex for each of them. Ultimately, it represents the graph using an adjacency list approach, by considering the membership of a node in a clique as an edge between that node and the clique's pseudo vertex. Fig. 3.5 shows the results of applying Kolmogorov graph covering and GraphZip to two example



Adjacency matrix	561 bits
Adjacency list	468 bits
Kolmogorov graph covering	396 + O(1) bits

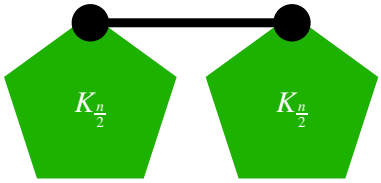
Figure 3.4. Zachary's karate club graph, along with its K-complexity estimation using adjacency matrix, adjacency list, and Kolmogorov graph covering

graphs. Fig. 3.5a shows a wheel graph of size n, in which Kolmogorov graph covering detects a ring graph of size $n-1$, and GraphZip detects a clique of size 3 and $(n-3)/2$ cliques of size 2. Fig. 3.5b illustrates two complete graphs of size $n/2$ connected by a single edge. In this case, enhanced Kolmogorov graph covering detects two complete graph of size $n/2$, and a null graph of size $n-2$ to describe the connections between the two complete graphs. GraphZip detects two cliques of size $n/2$. For one graph, Kolmogorov graph covering is providing a lower estimate for the K-complexity of the graph, whereas for the other one GraphZip is providing a shorter code length. This means that the graph of Fig. 3.5a could have been compressed more efficiently, but GraphZip is already performing well for the graph of Fig. 3.5b. It can also be observed that for both cases, the complexities are from the same order of magnitude. These examples show how Kolmogorov graph covering can be used to evaluate the performance of lossless graph compression algorithms on single graphs.



Adjacency matrix	$\binom{n}{2}$
Adjacency list	$4(n-1)\log_2 n$
Kolmogorov graph covering	$n\log_2 n + n - 1 + O(1)$
GraphZip	$(2n-5)\log_2 n$

(a) Graph 1



Adjacency matrix	$\binom{n}{2}$
Adjacency list	$\left(2\binom{n/2}{2}+1\right)\log_2 n$
Kolmogorov graph covering	$2n\log_2 n + O(1)$
GraphZip	$(n+2)\log_2 n$

(b) Graph 2

Figure 3.5. Comparing the results of Kolmogorov graph covering and GraphZip

3.5 Summary

In this chapter, the complexity of static graphs was explored using Kolmogorov complexity. Basic graphs were introduced as the least complex graphs possible. It was shown that because of their properties, basic graphs can be used as a tool to distinguish between random and non-random graphs. Based on this observation, the Kolmogorov graph covering algorithm was presented as a method to estimate an upper bound for the K-complexity of its input graph. It was also discussed that the algorithm may be unable to capture some non-randomness in the input graph. To overcome this, an extension to the algorithm was proposed. Additionally, it was shown how Kolmogorov graph covering can be used to evaluate the performance of graph compression algorithms. Finally, the results of applying Kolmogorov graph

covering on a number of example graphs was presented, and it was discussed how these results can be interpreted.

Chapter 4

Network Evolution Models

In the previous chapters, we discussed different scenarios for networks, ways to model those scenarios using information theory, measure their complexity, and finally compress their topologies. However, there is still one important family of network models that we are yet to study: network evolution models.

Many networks that we observe in our daily lives are dynamic entities. Even though they might maintain a certain general topology, they undergo changes through time. Consider an online social media network like Facebook as an example. The connections of a person will most likely change and evolve over time. New connections will be formed, and existing connections might be removed. Additionally, new users can join the network, and old ones might leave. Kolmogorov complexity is best suited as a measure of complexity for static networks that do not undergo any changes, as it does not consider the underlying distribution. Additionally, random graph generators are not a good way to model the evolution of such networks. This is because random graph generators usually produce i.i.d samples of graphs, whereas their is a strong correlation between consecutive samples of a dynamic network. There exists special models that are able to capture the dynamics of evolving networks. However, our goal is to have a model that is analysable using information theory. Therefore, the main

goal of this chapter is to introduce a novel information-theoretic network evolution model and analyse its behaviour and properties.

We start the chapter by providing an overview of the background work. After stating some basic information-theoretic concepts for the analysis of stochastic processes, we introduce our proposed model for dynamic networks and go through its properties. We then introduce three important extensions to this model, where each one is well-suited for a specific type of analysis of dynamic networks. The chapter is concluded by providing some important simulation results, which further back up our findings and claims.

4.1 Background and basic definitions

As most of the networks that we observe around us are dynamic entities, the evolution of networks has been greatly studied throughout the years. Most of the studies done in this area have been trying to provide mathematical models for the growth or evolution of graphs, in a way that mimics the behaviour of real-life networks. A very powerful and important example of such models is the Barabási–Albert (BA) model [83]. This model, also known as the preferential attachment model, defines a probability distribution on how to add new nodes and edges to the graph, and can therefore be considered a growing network evolution model. This will result in networks that are known as scale-free, which means that their degree distribution follows a power law. Most of the networks around us exhibit this property, and this is why the BA model has gained a lot of attention ever since its introduction. After the BA model, there have been other methods that introduce link creation rules that try to preserve real-world network behaviours. Some more recent models include [84], which uses a fitness-based attachment process for nodes, and [85], which introduces network growth models based on the notion of specialisation.

The study of the evolution of social networks is one of the other main areas for which the subject of network evolution models has been extensively studied. For instance, [86] studies a number of network evolution models in the context of social networks and compares them with each other. These models are defined on a fixed set of nodes, with the connection probabilities being modeled in a way that best simulates the behaviour of social networks. Reference [87] is a more recent study on this subject, and provides an overview of the existing models of social network evolution. In the models studied in that paper, new links are added to the graph based on the network structure. These models have proven to be very useful in simulating the behaviour of evolving networks, and have been able to answer many important questions in studying social networks. Moreover, many of them have also been used in other contexts such as the spread of epidemics [88], and the formation of mobile networks [89]. Refernce [90] also studies some of these models that have been designed for modelling network evolution.

Based on these studies, the question of how best to mathematically model and simulate the behaviour of dynamic networks has been extensively researched. However, we want to not only model such evolutions, but also study them using information theory. Information theory can provide us with quantitative answers to questions such as how much we expect the network to evolve over time or how compressible the network is. This can be used in applications in which we need to transmit the topological states of the network over a communication channel in an efficient manner. Moreover, most of the existing models only consider the growth of networks and do not consider the scenarios in which the network shrinks. The shrinkage of networks is observable in many scenarios, such as the departure of people from online social networks.

We introduce a network evolution model that can be examined using an information theory point of view, and is powerful in capturing the structural dynamics

of real-life networks. We believe that this model can easily simulate the temporal evolution of data networks, social networks, biological networks and any other form of network that can be represented by a graph. As an example, the evolution of metabolic pathways in molecular and cellular evolution can be simulated with this model [91]. This would allow for an information-theoretic analysis on the topological dynamics of these networks in order to better understand the evolution process of the organisms.

The problem of finding the more probable sequences of a particular random variable is a fundamental principle of data compression in information theory. Typical sequences are defined as a result of the Asymptotic Equipartition Property (AEP), and are proven to essentially represent sequences that have a very high probability of occurrence. The concept of typical sequences motivates us to search for a similar approach in relation to graphs. There are many application scenarios in which the topology of the network needs to be transmitted and we need to compress it as much as possible. In many of these scenarios, we have the current state of the network and want to compress the future states. This motivates us to define a network evolution model that both simulates the network well and can be analysed using information theory. Additionally, analysing the dynamics of graphs by means of information theory will result in calculating a value for the entropy rate of the process. [92] uses a similar approach, and is very successful in calculating the entropy rate of nonequilibrium growing networks, and then analysing them. However, there are differences between the interpretation of entropy in our model and the one introduced by [92], which is further discussed in Section 4.3.

The main goal of this chapter is to introduce Network Evolution Chains. This is a novel model designed to not only simulate the dynamics of networks, but also do so in a way that is fully analysable using information theory. Furthermore, this model is designed such that any of the mentioned link creation rules, such as the BA

model and its extensions, can be incorporated into it in order to mimic and simulate different types of networks and make use of the vast amount of models that already exist in this field.

4.2 Network Evolution Chain

In this section, we introduce a stochastic process that manages to capture and model the way networks change over time. In information theory, the Asymptotic Equipartition Property (AEP) is the basis of defining typical sequences and compressing random variables. However, the stochastic process at hand needs to fulfil certain conditions for the AEP to be applied to it. In order for our model to be as inclusive as possible, we use a more general version of the AEP which is known as the Shannon-McMillan-Breiman theorem. Using this theorem one can estimate the entropy rate of stochastic processes that fulfil its conditions. The entropy rate of a stochastic process quantifies the average amount of uncertainty when going from one random variable in the stochastic process to the next, and is defined as below.

Definition 4.2.1 (Entropy Rate)**.** The entropy rate of a stochastic process $\{X_i\}$ is defined by

$$H(X) = \lim_{l\to\infty} \frac{1}{l} H(X_1, X_2, \ldots, X_l), \tag{4.1}$$

when the limit exists.

Having defined the entropy rate of a process, we can use the general AEP, as stated below, to estimate the entropy rate.

Theorem 4.2.1 (AEP: Shannon-McMillan-Breiman Theorem)**.** *[57, Thm. 16.8.1] If $H(X)$ is the entropy rate of a finite-valued stationary ergodic process $\{X_i\}$, then*

$$-\frac{1}{l} \log p(X_1, X_2, ..., X_l) \to H(X) \quad \textit{with probability } 1.$$

Based on Theorem 4.2.1, the conditions that need to be satisfied for the AEP to be applicable to the process at hand are the stationarity and ergodicity of the process. A process is stationary if its statistical properties do not change over time, and is ergodic if its long-term statistical properties over time converge to the ensemble's statistical properties.

We now define a stochastic process that matches our desired model and prove that it satisfies the conditions of Theorem 4.2.1. We can then define typical sequences of the model.

Definition 4.2.2 (Network Evolution Chain)**.** A stochastic process $\{g_i\}$ is called a Network Evolution Chain (NEC) if

- g_1 is a graph with exactly 1 node.
- g_i $(i > 1)$ is a graph created from g_{i-1}. To this end, a transition scheme is first chosen among the following options: deletion, addition, and same. The probability of choosing deletion, addition, or same is $r(g_{i-1})$, $a(g_{i-1})$, and $s(g_{i-1})$, respectively. Therefore, we have $r(g_{i-1})+a(g_{i-1})+s(g_{i-1}) = 1$. Based on the chosen transition scheme, one of the following procedures will be done for creating g_i:
  - **Deletion:** g_i is g_{i-1} with one of its nodes deleted according to a probability distribution.
  - **Addition:** g_i is generated by adding a node to g_{i-1}. The edges between the new node and the existing ones are added according to a random graph generation model. In this random graph generation model, all possible edge connections between the new node and the old ones should have a non-zero probability.
  - **Same:** g_i is the same as g_{i-1}.

- There exists an integer $n_{\max} \geq 1$ such that if g_i is a graph with $n_{\max}$ nodes then we have $a(g_i) = 0$. For all other graphs $a(g_i) \neq 0$.
- If g_i is a graph with only one node, then we have $r(g_i) = 0$. For all other graphs $r(g_i) \neq 0$.
- For all graphs we have $s(g_i) \neq 0$.

Fig. 4.1 illustrates four possible consecutive steps of an instance of an NEC defined based on Definition 4.2.2.

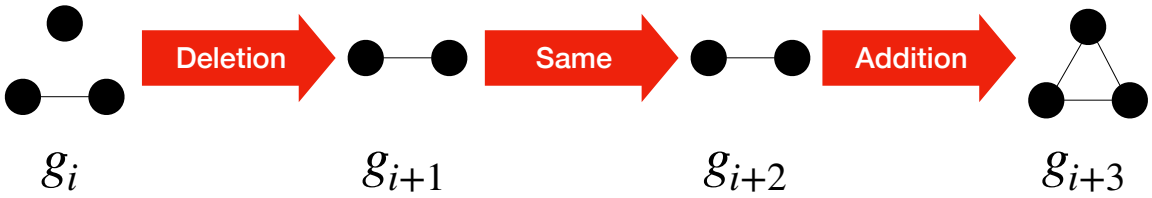


Figure 4.1. Example of four consecutive graphs in an NEC

According to Definition 4.2.2 for NECs, the graph space G for an NEC consists of simple graphs of size 1 to $n_{\max}$.

The following statements will aid us in proving that a Network Evolution Chain satisfies the conditions of Theorem 4.2.1.

Definition 4.2.3 (Irreducible Markov chain)**.** [57, Ch. 4] "If it is possible to go with positive probability from any state of a Markov chain to any other state in a finite number of steps, the Markov chain is said to be irreducible."

Definition 4.2.4 (Aperiodic Markov chain)**.** [57, Ch. 4] "If the largest common factor of the lengths of different paths from a state to itself in a Markov chain is 1, the Markov chain is said to be aperiodic."

Lemma 4.2.1. *[57, Ch. 4] "If the finite-state Markov chain $\{X_i\}$ is irreducible and aperiodic, the stationary distribution is unique, and from any starting distribution, the distribution of X_l tends to the stationary distribution as $l \to \infty$."*

Corollary 4.2.1.1. *[93, Thm. 2] A finite-state, irreducible, and aperiodic Markov chain will form an ergodic process.*

Proof. Theorem 2 from [93] provides a condition under which an irreducible and aperiodic Markov chain is ergodic. It can be easily shown that if the Markov chain is finite-state, it fulfils the specified condition in the mentioned theorem and the Markov chain will be ergodic. □

The following theorem states and proves that a Network Evolution Chain will eventually satisfy the conditions of Theorem 4.2.1.

Theorem 4.2.2. *A Network Evolution Chain tends towards a stationary and ergodic stochastic process as* $l \to \infty$.

Proof. Using Lemma 4.2.1 and Corollary 4.2.1.1, it suffices for us to show that an NEC is a finite-state, irreducible and aperiodic Markov chain. We will talk about each of these in turn. Firstly, based on its definition, an NEC is in fact a Markov chain, as g_i is a random graph that only depends on g_{i-1}. Furthermore, this Markov chain is also finite-state. Notice that the definition of an NEC does not allow us to have graphs with more than $n_{\max}$ nodes in the chain. Therefore, our graph space consists of all graphs whose number of nodes is less than or equal to $n_{\max}$ and greater than or equal to 1, which is finite. An NEC is also irreducible. To prove this, we must present a manner in which we can go from any state to another. Suppose we want to go from graph g_1 to graph g_2, and both of these graphs are in the graph space G. If $g_1 = g_2$, we are done. Otherwise, we first remove nodes from g_1 one by one until we reach a graph with only one node. Notice that this is possible because $r(g) \neq 0$ for $\{g | g \in G, |V(g)| \neq 1\}$. Afterwards, we can add nodes to our one node graph one by one until we build g_2. This is also possible because $a(g) \neq 0$ for $\{g | g \in G, |V(g)| \neq n_{\max}\}$, and the fact that recreating the exact edge connections of g_2 has a non-zero probability. To prove that an NEC is aperiodic, notice that the

shortest path we can take from a graph to itself is of length 1. This is because for all $g \in G$ we have $s(g) \neq 0$. Since the largest common factor of any number and 1 is 1, we have proved that this chain is aperiodic. Now that we have proved that an NEC fulfils all the conditions in Lemma 4.2.1 and Corollary 4.2.1.1, we can conclude that an NEC will tend towards a stationary and ergodic process as $l \to \infty$.

□

The NEC provides us with a model that generates a chain of random graphs. The name Network Evolution Chain is due to the fact that an NEC can beautifully describe the manner in which networks evolve over time. Every network initially begins with a single node, just like an NEC. Afterwards, at any point in time, a node can be added or deleted, or the network can stay the same as it was. Additionally, we may need to add or remove edges, but not nodes. These cases can be simulated by first deleting a node from one end of the edge we want to add or remove and then add that node again with our desired change in its edge connections. Note that even though edge deletion can be modeled in the described format, it still can not be considered equivalent to having edge deletion modeled directly. Modelling edge deletion directly will have an impact on the assumptions and proofs about NEC, and is therefore not considered in this thesis. However, it is a potential venue for future work on this subject.

To model the evolution of a network using an NEC, there are three important factors that need to be chosen according to that specific network. The first one is the model according to which nodes and edges are added to the graph in case we are adding a node. The ER model [59] and the BA model [94] are examples of the models that can be used. Additionally, any link creation rule that has been studied in terms of simulating real-life networks can be used here in order to create NECs that simulate those networks. For instance, link creation rules mentioned in [95] for social networks can be used to determine the link creation model. The second factor is the

probabilities of transition. In other words, $r(g)$, $a(g)$, and $s(g)$ need to be defined for all $g \in G$. There are many ways to set these probabilities. They can only depend on the number of nodes in g or can be unique to each graph. The final factor is the probability distribution according to which the deleted node is chosen in case of a deletion event. For example, the node to be deleted can be chosen uniformly among all the nodes in the graph, or nodes with fewer edges can have a higher probability of getting deleted. Undoubtedly, the best way to chose these parameters is according to the data observed from the behaviour of the network we want to simulate. Parameter estimation techniques such as maximum likelihood estimation (MLE) or maximum a posteriori estimation (MAP) can be used to estimate the transition probabilities.

It is notable to mention the scalability of simulating NECs as a function of the maximum number of nodes in the network ($n_{\max}$). At each step of the evolution, a series of random parameters need to be generated, which is of constant time complexity. If *same* operation is chosen, nothing needs to done. Calculating the time complexity of the *Addition* and *Deletion* is more challenging, as it heavily depends on how the graphs are stored. Additionally, for *Addition*, this depends on the method chosen for forming the new edges. For instance, if an adjacency matrix is used to store the graphs and an ER model is used to form the new edges, *Addition* can be done in $O(n_{\max})$. This is because the algorithm simply needs to go through all existing nodes and decide if it wants to connect to them, and then finally add a new row to the adjacency matrix. Similarly, for the adjacency matrix representation, *Deletion* can also be done in at most $O(n_{\max})$, as we would only need to remove the corresponding row and column of the deleted node from the adjacency matrix. Overall, the time complexity of the *Addition* and *Deletion* steps need to be calculated according to the graph representation method used in the model, as well as the model for adding new edges when *Addition* is chosen.

4.3 Entropy rate and typical sequences

We have now defined Network Evolution Chains, which are stochastic processes that can be considered ergodic and stationary for large l. Assume the process $\{g_i\}$ to be an NEC generated using Definition 4.2.2. According to Theorem 4.2.1, this chain will tend towards its stationary distribution as l grows large. Therefore, we can use Theorem 4.2.1 and write

$$\lim_{l\to\infty} -\frac{1}{l}\log p(g_1,\ldots,g_l) = H(g), \tag{4.2}$$

where $H(g)$ shows the entropy rate of the stochastic process $\{g_i\}$.

Based on Eq. 4.2, we can define the typical sets of an NEC.

Definition 4.3.1. [57, Ch. 3] The typical set $A_\epsilon^{(l)}$ with respect to a defined NEC is the set of all sequences $(g_1, g_2, ..., g_l) \in G^l$ with the property

$$2^{-l(H(g)+\epsilon)} \leq p(g_1, g_2, ..., g_l) \leq 2^{-l(H(g)-\epsilon)}.$$

It can be proved that "the typical set has probability nearly 1, all elements of the set are nearly equiprobable, and the number of elements in the set is nearly $2^{lH(g)}$." [57, Ch. 3].

We now need to calculate the entropy rate of NEC. To this end, we use a result from [57], which is stated in the following lemma.

Lemma 4.3.1. *[57, Thm. 4.2.1] For a stationary stochastic process $\{X_i\}$, the following limit exists and is equal to the entropy rate of the process:*

$$H(X) = \lim_{l\to\infty} H(X_l|X_{l-1}, X_{l-2}, ..., X_1)$$

The entropy rate defined in Lemma 4.3.1 can be interpreted in two ways. Firstly,

based on its definition, it shows the conditional entropy of the last observed random variable given all the symbols that came before it. Secondly, [57] proves that the defined entropy rate also indicates the per symbol entropy of the l observed random variables.

As an NEC is a stationary stochastic process for large l, we can use Lemma 4.3.1 to calculate its entropy rate. We can write the following chain of equations.

$$H(g) = \lim_{l\to\infty} H(g_l|g_{l-1}, g_{l-2}, ..., g_1) \tag{4.3a}$$

$$= \lim_{l\to\infty} H(g_l|g_{l-1}) \tag{4.3b}$$

$$\begin{aligned} &= \lim_{l\to\infty} H(g_l|g_{l-1}, X) \\ &\quad + H(X|g_{l-1}) - H(X|g_{l-1}, g_l) \end{aligned} \tag{4.3c}$$

$$= \lim_{l\to\infty} H(g_l|g_{l-1}, X) + H(X|g_{l-1}) \tag{4.3d}$$

Eq. 4.3a is the definition of entropy rate based on Lemma 4.3.1. Eq. 4.3b holds because of the fact that g_l is only dependent on g_{l-1}. Eq. 4.3c comes from taking into account the three states of addition, deletion, and remaining the same. X is defined as a random variable that indicates which of the progression methods we have chosen for going from g_{l-1} to g_l. In other words, we have $X \in \{\text{addition}, \text{deletion}, \text{same}\}$. Finally, Eq. 4.3d holds because having both g_{l-1} and g_l, there will be no uncertainty about which transition scheme has been chosen for going from g_{l-1} to g_l and therefore we will have $H(X|g_{l-1}, g_l) = 0$. For calculating $H(X|g_{l-1})$, we can write

$$H(X|g_{l-1}) = -\sum_{g\in G} p(g)\left(a(g)\log a(g) + r(g)\log r(g) + s(g)\log s(g)\right). \tag{4.4}$$

In Eq. 4.4, $p(g)$ shows the stationary distribution of graph g. Now that $H(X|g_{l-1})$ has been calculated, we move on to calculate $H(g_l|g_{l-1}, X)$.

$$
\begin{aligned}
H(g_l|g_{l-1}, X) = &\sum_{g\in G} p(g)a(g)H(g_l|g_{l-1} = g, X = \text{addition}) \\
&+ \sum_{g\in G} p(g)r(g)H(g_l|g_{l-1} = g, X = \text{deletion}) \qquad (4.5a) \\
&+ \sum_{g\in G} p(g)s(g)H(g_l|g_{l-1} = g, X = \text{same}) \\
= &\sum_{g\in G} p(g)a(g)H(g_l|g_{l-1} = g, X = \text{addition}) \\
&+ \sum_{g\in G} p(g)r(g)H(g_l|g_{l-1} = g, X = \text{deletion}) \qquad (4.5b)
\end{aligned}
$$

Eq. 4.5a comes from conditional entropy and Eq. 4.5b comes from the fact that we have $H(g_l|g_{l-1} = g, X = \text{same}) = 0$ because there remains no uncertainty when a graph stays the same as it was before. Notice that Eq. 4.5b can not be more simplified because the node addition and deletion models are not known.

We can now provide the following equation for calculating the entropy rate of a Network Evolution Chain.

$$
\begin{aligned}
H(g) = &-\sum_{g\in G} p(g)(a(g)\log a(g) + r(g)\log r(g) + s(g)\log s(g)) \\
&+ \sum_{g\in G} p(g)a(g)H(g_l|g_{l-1} = g, X = \text{addition}) \\
&+ \sum_{g\in G} p(g)r(g)H(g_l|g_{l-1} = g, X = \text{deletion}) \qquad (4.6)
\end{aligned}
$$

We observe that calculating the exact value of the entropy rate depends on the factors below:

1. $a(g)$, $r(g)$, and $s(g)$ for all $g \in G$.

2. Model of choice for adding a node and its edges to g for all $g \in G$.

3. Model of choice for deleting a node from g for all $g \in G$.

4. The stationary probability distributions of all $g \in G$.

After knowing these factors about a Network Evolution Chain, the entropy rate for the chain can be calculated using Eq. 4.6.

It is notable to mention the differences between the entropy rate calculated in this section with that of [92]. Firstly, [92] does not take into account the shrinkage of a network and the entropy is calculated asymptotically as the network grows large. Secondly, [92] has taken into account only two possible ways of connecting nodes: one based on nodes having a constant mean degree, and the other one based on the network being a tree. This is in contrast to NECs, which can simulate any type of connection between nodes. Finally, the network size in [92] is not bounded, whereas in an NEC the network size is limited to $n_{\max}$. All in all, these two approaches are different in their own ways and each one must be utilized in its appropriate application scenario.

4.4 Functions of Network Evolution Chains

We have introduced Network Evolution Chains in order to simulate the way networks evolve over time. When a certain network is simulated using an NEC, we have the whole graph of the network at each step as our random variable and there will be no information loss. However, if we want to only look at certain properties of the network, it may not be efficient to store the whole graphs in our chain. In this section, we introduce functions of Network Evolution Chains as a method for extracting specific information from the networks in the chain. To this end, we define Network Property Chains.

Definition 4.4.1 (Network Property Chain)**.** Consider the Network Evolution Chain $\{g_1, g_2, ..., g_l\}$. $Y = \{Y_1, Y_2, ..., Y_l\}$ is called a Network Property Chain (NPC) of this

NEC if there exists a function f such that we have

$$\forall i \in \{1, 2, ..., l\} \qquad Y_i = f(g_i).$$

An NPC is therefore defined using an NEC and a function f. We can use Network Property Chains to look for certain properties and behaviours of a network. Specifically, we choose f so that it returns the network properties that we are interested in and want to study. As such, the co-domain of function f is determined based on the application and can include real numbers, other graphs etc.

Notice that a function of a Markov chain is not necessarily a Markov chain. However, the following theorem will aid us in analysing Network Property Chains without needing them to be Markov chains.

Theorem 4.4.1. *A Network Property Chain is a stationary and ergodic stochastic process.*

Proof. We first prove that an NPC is a stationary process. As f is not necessarily a bijection, consider P_i as the set of all $g \in G$ for which we have $f(g) = Y_i$. We can calculate the probability $p(Y_i, ..., Y_{i+k})$ for an integer k using the following equation.

$$\begin{aligned} p(Y_i, Y_{i+1}, ..., Y_{i+k}) &= p(Y_i) \prod_{j=i+1}^{i+k} p(Y_j | Y_{j-1}, ..., Y_i) \\ &= p(P_i) \prod_{j=i+1}^{i+k} \sum_{g_1 \in P_{j-1}} \sum_{g_2 \in P_j} p(g_2 | g_1) \end{aligned} \tag{4.7}$$

As we know that an NEC is a stationary process, there are no terms in Eq. 4.7 which are time-variant and therefore an NPC is also stationary.

To prove that an NPC is ergodic, we use Birkhoff's ergodic theorem [96]. For using Birkhoff's theorem to prove ergodicity, we need to show that all functions of an NPC satisfy Birkhoff's condition. Firstly, all functions of an NPC are also a function of its

respective NEC. To see this, notice that if the function f which is used for building the NPC is injective, then this statement is easily concluded and if f is surjective, then any function of the NPC can be seen as a function of its respective NEC where the function takes on equal values for all g_i and g_j for which we have $f(g_i) = f(g_j)$. Additionally, we know that an NEC is in fact ergodic and all functions of it satisfy the condition of Birkhoff's theorem. Therefore, an NPC is also an ergodic process.

□

Theorem 4.4.1 shows that Network Property Chains are stationary and ergodic processes. Therefore, we can apply the Asymptotic Equipartition Property to these chains as well. We will end up with the following equation after applying the AEP to NPC.

$$\lim_{l\to\infty} -\frac{1}{l}\log p(Y_1, Y_2, ..., Y_l) = H(Y) \tag{4.8}$$

Applying the AEP to an NPC gives us the opportunity to define typical sequences for Network Property Chains. This means that we will be able to obtain the typical ways that properties of the network will evolve over time. For example, assume that we are interested in the number of triangles in the network. We will define $f(g)$ on G such that it returns the number of triangles in g. The typical set of $\{Y_i\}$ will then contain the most probable evolution chains of the number of triangles in the network.

Notice that as an NPC is not necessarily a Markov chain, calculating its entropy rate can not be done in the same way as NECs. However, we will be able to estimate its entropy rate using the AEP. To this end, we first need $p(Y_1, Y_2, ..., Y_l)$, which can be calculated using Eq. 4.7. However, Eq. 4.7 may be very difficult to compute, particularly for NECs with a large $n_{\max}$. To assist us with the computation, we use Hidden Markov Models (HMM) [97]. HMMs are very useful for simulating Network Property Chains. As mentioned earlier, one of the goals for introducing NPCs was

to remove the need for storing the whole graphs. In an HMM, the assumption is that the original Markov chain is not available (it is hidden), and we can only observe another chain that is based on our original Markov chain. The transition probabilities for the Hidden Markov Model (g_l, Y_l) are the same as the transition probabilities for the NEC $\{g_i\}$. Assume the set P to be the image of G under f. For the emission probabilities of the HMM for all $g \in G$ and $y \in P$ we have

$$p(y|g) = \begin{cases} 1, & \text{if } f(g) = y \\ 0, & \text{otherwise} \end{cases}.$$

Now that we have defined our HMM, we can make use of its properties to efficiently calculate the probability of the observed sequence. HMMs provide us with the forward algorithm [98] which is used to efficiently calculate the probability of an observed sequence. Therefore, our solution for calculating $p(Y_1, Y_2, ..., Y_l)$ is to first define the HMM (g_l, Y_l) and then use the forward algorithm. This will provide us with the sufficient tool to calculate the probability of observed sequences, estimate the entropy rate of the NPC, and also recognise typical sequences.

4.5 Continuous time modelling of Network Evolution Chains

We have observed that Network Evolution Chains have the ability to simulate the topological dynamics of networks. However, they fail to provide us with a simulation for the network in continuous time, as the changes to the network topology in an NEC happen in discrete steps. While continuous time models will not be the primary focus of this thesis, we provide two methods to overcome this limitation in case continuous time modelling is required.

4.5.1 The $s(g)$ method

The first method uses the value of $s(g)(g \in G)$ to simulate the continuous time domain. The idea is that we first quantise time and consider the time between each step of the chain to be a constant τ. Now, for each $g \in G$, we can set $s(g)$ according to how much time we expect g to remain unchanged. Consider $L(g)$ to be the random variable that shows the time that graph g stays unchanged and $Q(g)$ to be the random variable that shows the number of consecutive steps that staying the same is chosen for graph g. We can write

$$\begin{aligned}\mathbb{E}[L(g)] &= \tau(\mathbb{E}[Q(g)] + 1) \\ &= \tau\left(1 + \sum_{i=0}^{\infty} i s(g)^i (1 - s(g))\right) \\ &= \frac{\tau}{1 - s(g)}.\end{aligned} \tag{4.9}$$

After choosing our desired value for the time quantization variable τ, we can use Eq. 4.9 to set the values for $s(g)$ for all $g \in G$. This way, the expected time that each network stays unchanged will match our desired value.

4.5.2 The Continuous Time Markov Chain method

Continuous Time Markov Chains (CTMC) [99, Ch. 5] are an extension to Markov chains with the goal to simulate Markov chains in the continuous time domain. Therefore, we can use CTMCs to simulate an NEC in continuous time.

Definition 4.5.1 (Continuous Time NEC)**.** A Continuous Time Network Evolution Chain (CTNEC) is a Continuous Time Markov Chain $N(t)$ in which the states and the transition probabilities are the same as a Network Evolution Chain and additionally, $\lambda(g)$ is chosen for all $g \in G$ as the parameter for the exponential distribution used

for determining the time we stay at state g.

We can see that the states are generated in the same manner as explained for the discrete time NEC. Furthermore, we remain in g_i for τ_i seconds. τ_i is generated using an exponential distribution with parameter $\lambda(g_i)$. We therefore have to choose a value $\lambda(g)$ for all $g \in G$. This value is what determines how much time we wait in each state according to an exponential distribution with its expected value equal to $1/\lambda(g)$.

Suppose that we label the graphs in G from 1 to $|G|$ according to an arbitrary ordering. Let p_{ij} denote the probability of going from state i to j in the respective NEC. We use the following notation from chapter 5 of [99] to provide a result about the stationary distribution of CTNECs.

$$p_{ij}(t) = p(N(t+s) = j | N(s) = i) \tag{4.10}$$

$$r_{ij} = \begin{cases} \lambda(i)p_{ij}, & \text{if } i \neq j \\ -\lambda(i), & \text{if } i = j \end{cases} \tag{4.11}$$

In the above notation, $N(t)$ is used to show the state of the CTMC at time t and p_{ij} represents the probability that the process goes to state j when it leaves state i. Based on the definition, $p_{ij}(t)$ represents the probability of the process with current state of i moves to state j after time t has passed, and r_{ij} represents the rate of going from state i to state j. To assist us with the calculations, the matrices $\boldsymbol{R}$, $\boldsymbol{P}(t)$, and $\boldsymbol{P}'(t)$ are defined. The elements in row i and column j of $\boldsymbol{R}$, $\boldsymbol{P}(t)$, and $\boldsymbol{P}'(t)$ are r_{ij}, $p_{ij}(t)$, and $p'_{ij}(t)$, respectively [99, Ch. 5]. It is stated in [99] that the following equation is the solution for finding the stationary distribution of a CTMC.

$$\boldsymbol{P}(t) = e^{\boldsymbol{R}t} = \sum_{k=0}^{\infty} \frac{(\boldsymbol{R}t)^k}{k!}, \tag{4.12}$$

where $\boldsymbol{R}^0$ is the identity matrix.

Eq. 4.12 provides us with the solution to find the stationary continuous time distributions. However, it may be computationally infeasible to calculate the distribution using this equation. To overcome this challenge, [99] presents a method to estimate the answer to Eq. 4.12 [99, pg. 250].

Both the $s(g)$ method and the CTMC method are providing us with ways to simulate NECs in the continuous time domain. The $s(g)$ method is easier to use and implement, as it only requires the addition of one new parameter. The CTMC method is more accurate for the continuous time domain, as it does not discretise time. However, a new parameter needs to be chosen for every graph in the graph space, which might make its implementation more complex. All in all, the preferred method can be chosen based on the application at hand.

4.6 Erdős–Rényi Network Evolution Chains

The $G(n, p)$ model [59], often called the Erdős–Rényi model, is a mathematical model used in graph theory for generating random graphs. This model was introduced and discussed in section 2.5.1. In this section, we will introduce NECs whose distribution matches that of the Erdős–Rényi model. This NEC can then be used to model the dynamics of networks whose stationary distribution can be simulated using the Erdős–Rényi model.

If we create a Network Evolution Chain whose addition step is based on $G(n, p)$, we are interested for the stationary conditional probability of observing a graph g given its number of nodes $|V(g)|$ to match the probability of the same graph if it was generated by the $G(|V(g)|, p)$ model. Providing a necessary and sufficient condition for an NEC to achieve this property is challenging. Instead, we introduce a subset of NEC as Erdős–Rényi Network Evolution Chains and then prove that it satisfies our

desired condition.

We will use the notation G_n to show the subset of graphs in G that have n nodes. Furthermore, to avoid confusion in calculations in which p is used to show the probability of an event, we will use q instead of p as the parameter for the Erdős–Rényi model.

Definition 4.6.1 (Erdős–Rényi Network Evolution Chain (ERNEC))**.** We define an Erdős–Rényi Network Evolution Chain as a Network Evolution Chain with the following additional properties:

- for all $1 \leq n \leq n_{\max}$ and $g_1, g_2 \in G_n$ we have:
  - $a(g_1) = a(g_2)$
  - $r(g_1) = r(g_2)$
  - $s(g_1) = s(g_2)$
- The node in the first graph of the chain is labelled as 1.
- If at any point in time, addition is chosen, a new node with its label equal to the highest existing label plus one gets added to the graph. The edges between the new node and the existing nodes are added using the Erdős–Rényi model.
- If at any point in time, deletion is chosen, one of the existing nodes gets deleted according to a uniform distribution on the existing nodes. Consider the label of the deleted node to be i. After the deletion of node i, the labels of all existing nodes whose label is greater than i are reduced by one.

We now move on to prove that an ERNEC has the $G(n, q)$ distribution when conditioned on its number of nodes. We first prove the following lemma.

Lemma 4.6.1. *For an Erdős–Rényi Network Evolution Chain created based on the $G(n, q)$ model, The stationary distribution $p(g)$ for $g \in G_n, n \geq 2$ is in the form of*

$Cq^k(1-q)^{(n-1-k)}p(g')$, *with* g' *being* g *without node* n *(its newest node),* C *being a constant value among all* $g \in G_n$ *and* k *being the number of nodes in* g' *to which node* n *is connected.*

Proof. We will prove this lemma using backward induction on the number of nodes in the graph. Therefore, we start with graphs with $n_{\max}$ nodes. For $g \in G_{n_{\max}}$ we can write

$$p(g) = s(g)p(g) + p(g')a(g')q^k(1-q)^{n_{\max}-1-k} \tag{4.13}$$

$$= \frac{p(g')a(g')q^k(1-q)^{n_{\max}-1-k}}{1-s(g)}. \tag{4.14}$$

In Eq. 4.14, g' is the graph g after its node with label $n_{\max}$ is deleted from the graph and k is the number of edges connected to node $n_{\max}$. We can see that as $a(g')/(1-s(g))$ is a constant value for all $g \in G_{n_{\max}}$, our base case is proved.

We will now move on to our induction step. Suppose that we know that the statement holds for all G_k where $k > n$. We now have to prove the case for G_n. For all $g \in G_n$ we can write

$$\begin{aligned} p(g) = s(g)p(g) &+ a(g')p(g')q^k(1-q)^{n-1-k} \\ &+ r(g^+)\sum_{i=1}^{n+1}\frac{1}{n+1}\sum_{j=1}^{2^n}p(g_{ij}^+). \end{aligned} \tag{4.15}$$

In Eq. 4.15, the first term accounts for the case where the network is created using a *same* step and the next two account for the *addition* and *deletion* cases respectively. The first summation in the *deletion* term is due to the fact that the deleted node could have been any of the $n+1$ nodes in the previous graph with probability $1/(n+1)$. The second summation is to account for all possible ways that the deleted node could have been connected to the remaining nodes. g_{ij}^+ indicates the jth possible way that edges could have been connected to the deleted node which was labelled as i. $r(g^+)$

shows $r(g_i^+)$ for all i, as they are all equal.

Notice that the deleted node could have been any of the $n+1$ nodes in the previous graph with equal probability, and no matter its label, it could have been connected to the existing n nodes in any manner with its respective Erdős–Rényi probability. Because of this symmetry in the deletion step between different nodes for all $g \in G_{n+1}$, we can simplify the *deletion* term in the following manner.

$$\begin{aligned} & r(g^+) \sum_{i=1}^{n+1} \frac{1}{n+1} \sum_{j=1}^{2^n} p(g_{ij}^+) \\ & = r(g^+)(\sum_{j=1}^{2^n} p(g_{(n+1)j}^+))(\sum_{i=1}^{n+1} \frac{1}{n+1}) \\ & = r(g^+) \sum_{j=1}^{2^n} p(g_{(n+1)j}^+) \\ & = r(g^+) \sum_{j=1}^{2^n} p(g_j^+) \end{aligned} \tag{4.16}$$

where the notation in the last equality is changed for simplicity.

By induction, we can write $p(g_j^+)$ in the following manner.

$$p(g_j^+) = C^+ p(g) q^{k_j^+} (1-q)^{n-k_j^+}$$

and because in $\sum_{j=1}^{2^n} p(g_j^+)$ we are including all possible ways that edges can get connected to node $n+1$, we can further simplify Eq. 4.16.

$$\begin{aligned} r(g^+) \sum_{j=1}^{2^n} p(g_j^+) &= r(g^+) C^+ p(g) \sum_{j=1}^{2^n} q^{k_j^+} (1-q)^{n-k_j^+} \\ &= r(g^+) C^+ p(g) \end{aligned} \tag{4.17}$$

By inserting Eq. 4.17 into Eq. 4.15 we will have

$$(1 - s(g) - r(g^+)C^+)p(g) = a(g')p(g')q^k(1-q)^{n-1-k}.$$

And therefore we end up with the following equation.

$$\begin{aligned} p(g) &= \frac{a(g')}{1 - s(g) - r(g^+)C^+} q^k(1-q)^{n-1-k} p(g') \\ &= Cq^k(1-q)^{n-1-k} p(g') \end{aligned} \tag{4.18}$$

Notice that $a(g')$, $s(g)$, $r(g^+)$, and C^+ have a constant value for all $g \in G_n$. Therefore, the value $(a(g'))/(1 - s(g) - C^+)$ can be considered as a constant C for all graphs in G_n. Our lemma is therefore proved by backward induction. □

We are now equipped to prove the following theorem.

Theorem 4.6.1. *The conditional distribution of graphs in an Erdős–Rényi Network Evolution Chain when conditioned on their number of vertices is equivalent to the same Erdős–Rényi distribution used in the addition step of the NEC.*

Proof. We will use induction to prove this theorem. It is easy to see that as there exists only one graph with one node, its conditional distribution matches that of the Erdős–Rényi model. This will act as the basis of our induction.

Now, suppose that the theorem holds for the subset $\{g \in G | |V(g)| < n\}$. For the induction step, we need to prove that the theorem holds for all $g \in G_n$. We can write

$$p(g|\ |V(g)| = n) = \frac{p(g, |V(g)| = n)}{p(|V(g)| = n)} = \frac{p(g)}{p(|V(g)| = n)}.$$

Therefore, if the probability ratio for all possible pairs of graphs from this subset are equal to the same ratio from the Erdős–Rényi distribution, we can conclude that the

theorem holds for all $g \in G_n$. For $g_1, g_2 \in G_n$, we can write from Lemma 4.6.1

$$\frac{p(g_1)}{p(g_2)} = \frac{Cq^{k_1}(1-q)^{(n-1-k_1)}p(g_1')}{Cq^{k_2}(1-q)^{(n-1-k_2)}p(g_2')}.$$

let E_1, E_2, E_1' and E_2' be the number of edges in g_1, g_2, g_1' and g_2' respectively. From induction, we know that

$$\frac{p(g_1')}{p(g_2')} = \frac{q^{E_1'}(1-q)^{\binom{n-1}{2}-E_1'}}{q^{E_2'}(1-q)^{\binom{n-1}{2}-E_2'}},$$

and therefore we can write

$$\frac{p(g_1)}{p(g_2)} = \frac{q^{k_1}(1-q)^{(n-1-k_1)}q^{E_1'}(1-q)^{\binom{n-1}{2}-E_1'}}{q^{k_2}(1-q)^{(n-1-k_2)}q^{E_2'}(1-q)^{\binom{n-1}{2}-E_2'}}$$

$$= \frac{q^{k_1+E_1'}(1-q)^{\binom{n-1}{2}+(n-1)-E_1'-k_1}}{q^{k_2+E_2'}(1-q)^{\binom{n-1}{2}+(n-1)-E_2'-k_2}}.$$

We can rewrite the equation in the following format.

$$\frac{p(g_1)}{p(g_2)} = \frac{q^{E_1}(1-q)^{\binom{n}{2}-E_1}}{q^{E_2}(1-q)^{\binom{n}{2}-E_2}} \tag{4.19}$$

As Eq. 4.19 holds for all $g_1, g_2 \in G_n$, it proves that the conditional distribution of graphs in G_n matches that of the Erdős–Rényi model used in the addition step and our proof is complete. □

Now that we have proven that Erdős–Rényi Network Evolution Chains satisfy our desired property, we will move on to examine their other features.

4.6.1 Stationary distribution of the number of nodes

One interesting feature to examine in Network Evolution Chains is the stationary distribution of the number of nodes of the graphs present in the chain. We know

that such a distribution exists and is unique, because Theorem 4.4.1 guarantees it. This is because the chain containing the number of nodes for each element of an NEC is actually an NPC. The definition of an ERNEC makes the analysis of the stationary distribution of the number of nodes much easier. The following matrix can be defined as the transition probability matrix between different numbers of nodes in the ERNEC.

$$\boldsymbol{P} = \begin{bmatrix} s(1) & a(1) & 0 & ... & 0 & 0 \\ r(2) & s(2) & a(2) & ... & 0 & 0 \\ 0 & r(3) & s(3) & ... & 0 & 0 \\ \vdots & \vdots & \vdots & ... & \vdots & \vdots \\ 0 & 0 & 0 & ... & r(n_{\max}) & s(n_{\max}) \end{bmatrix} \tag{4.20}$$

There is a change of notation in Eq. 4.20, and $a(i)$, $s(i)$ and $r(i)$ show their respective values for any graph in the graph space whose number of nodes is equal to i. Let π_i be the stationary probability of observing a graph with i nodes. For finding the stationary distribution $\pi = (\pi_1, \pi_2, ...\pi_{n_{\max}})$, we need to solve the equation $\pi = \pi\boldsymbol{P}$ or equivalently, $(1 - \boldsymbol{P})\pi = 0$ constrained on $\sum_{i=1}^{n_{\max}} \pi_i = 1$. Notice that based on our constraint, if we find π_i/π_1 for all i, we will have found $(\pi_1, \pi_2, ...\pi_{n_{\max}})$. After some substituting $s(i)$ with $1 - a(i) - r(i)$ in equations present in $(1 - \boldsymbol{P})\pi = 0$, we end up with the following system of equations.

$$\begin{cases} \pi_1 a(1) - \pi_2 r(2) = 0 \\ -\pi_1 a(1) + \pi_2(r(2) + a(2)) - \pi_3 r(3) = 0 \\ \vdots \\ -\pi_{n_{\max}-1} a(n_{\max} - 1) + \pi_{n_{\max}} r(n_{\max}) = 0 \end{cases} \tag{4.21}$$

We now provide the following theorem for calculating $\frac{\pi_i}{\pi_1}$ for all i.

Theorem 4.6.2. *For all* $1 < i \leq n_{max}$ *we have*

$$\frac{\pi_i}{\pi_1} = \frac{\prod_{j=1}^{i-1} a(j)}{\prod_{j=2}^{i} r(j)}.$$

Proof. We will prove this theorem by induction. For our base case, we use the first equation in 4.21 and we have

$$\frac{\pi_2}{\pi_1} = \frac{a(1)}{r(2)},$$

which matches the theorem. Now, suppose that we know that the theorem holds for $2 \leq i \leq k$. For $k+1$, we use the kth equation in 4.21. We have

$$-\pi_{k-1}a(k-1) + \pi_k(a(k) + r(k)) - \pi_{k+1}r(k+1) = 0.$$

If we divide the equation by π_{k-1}, we end up with the following equation.

$$\frac{\pi_{k+1}}{\pi_{k-1}} = \frac{-a(k-1) + \frac{\pi_k}{\pi_{k-1}}(a(k) + r(k))}{r(k+1)} \tag{4.22}$$

Notice that from the assumption of the induction, we can write

$$\frac{\pi_k}{\pi_{k-1}} = \frac{a(k-1)}{r(k)}.$$

By plugging this into Eq. 4.22 we have

$$\frac{\pi_{k+1}}{\pi_{k-1}} = \frac{\frac{a(k)a(k-1)}{r(k)}}{r(k+1)} = \frac{a(k)a(k-1)}{r(k)r(k+1)}.$$

And because we have $\frac{\pi_{k-1}}{\pi_1} = \frac{\prod_{j=1}^{k-2} a(j)}{\prod_{j=2}^{k-1} r(j)}$ we will have

$$\frac{\pi_{k+1}}{\pi_1} = \frac{\prod_{j=1}^{k} a(j)}{\prod_{j=2}^{k+1} r(j)}.$$

□

Based on Theorem 4.6.2, for finding the stationary distribution of the number of nodes, we can choose $\pi_{1,temp} = 1$, find $\pi_{i,\text{temp}}$ for $2 \leq i \leq n_{\max}$ using the theorem and then divide $(\pi_{1,\text{temp}}, ..., \pi_{n_{\max},\text{temp}})$ by $\sum_{i=1}^{n_{\max}} \pi_{i,\text{temp}}$. It is interesting to observe that the distribution does not depend on the values set for $s(i)$. Therefore, for reaching our desired distribution on the number of nodes, we can set the values for $a(i)$ and $r(i)$ accordingly. Theorem 4.6.2 also provides us with the following corollary.

Corollary 4.6.2.1. *For all* $1 \leq k < m \leq n_{max}$ *we have*

$$\frac{\pi_m}{\pi_k} = \frac{\prod_{i=k}^{m-1} a(i)}{\prod_{i=k+1}^{m} r(i)}.$$

It is notable to mention that the model used in this section and the proved results can be seen as an analogue of the model and the results for birth-death processes [100].

4.6.2 Probability distribution of individual graphs

We have proved that the conditional distribution of graphs in an Erdős–Rényi Network Evolution Chain matches the Erdős–Rényi model. We have also provided a method for calculating the stationary distribution of the number of nodes. Having both of these values for all $g \in G$, we can calculate the probability of observing each individual graph. For $N > 1$ we can write

$$p(g) = p(g|\ |V(g)| = N)p(|V(g)| = N) = \pi_N q^{|E(g)|}(1-q)^{\binom{N}{2}-|E(g)|}, \tag{4.23}$$

where q is the parameter of the desired Erdős–Rényi model. For $N = 1$ we simply have

$$p(g) = p(g|\ |V(g)| = 1)p(|V(g)| = 1) = \pi_1. \tag{4.24}$$

Notice that as $p(|V(g)| = N)$ does not depend on $s(i)$, $p(g)$ does not depend on

$s(i)$ either. This gives us the freedom to choose $s(i)$ to match our desired continuous time model as explained in Section 4.5.1.

4.6.3 Entropy rate

For calculating the entropy rate of Erdős–Rényi Network Evolution Chains, we can use Eq. 4.6. We first start by simplifying Eq. 4.4 for an ERNEC.

$$H(X|g_{l-1}) = -\sum_{i=1}^{n_{\max}} \pi_i \left(a(i)\log a(i) + r(i)\log r(i) + s(i)\log s(i)\right) \tag{4.25}$$

We then move on to simplify Eq. 4.5b. Notice that for Erdős–Rényi Network Evolution Chains we have $H(g_l|g_{l-1} = g, X = \mathit{deletion}) = \log|V(g)|$. This is because in case of deletion, the deleted node is chosen uniformly among all the nodes of the graph. Therefore, we can write

$$H(g_l|g_{l-1}, X) = -\sum_{i=1}^{n_{\max}-1} \pi_i a(i) \sum_{j=0}^{i} \binom{i}{j} q^j (1-q)^{i-j} \log(q^j(1-q)^{i-j}) + \sum_{i=2}^{n_{\max}} \pi_i r(i) \log i. \tag{4.26}$$

We can now calculate the entropy rate of ERNEC by inserting the results from equations 4.25 and 4.26 into Eq. 4.6.

$$\begin{aligned} H(g) = &-\sum_{i=1}^{n_{\max}} \pi_i \left(a(i)\log a(i) + r(i)\log r(i) + s(i)\log s(i)\right) \\ &-\sum_{i=1}^{n_{\max}-1} \pi_i a(i) \sum_{j=0}^{i} \binom{i}{j} q^j (1-q)^{i-j} \log\left(q^j(1-q)^{i-j}\right) + \sum_{i=2}^{n_{\max}} \pi_i r(i) \log i \end{aligned} \tag{4.27}$$

4.6.4 Functions of Erdős–Rényi Network Evolution Chains

In Section 4.4, we defined a method that can be used for analyzing the topology dynamics of a function of the network using a Hidden Markov Model. Notice that the number of states in the proposed Hidden Markov Model equals the number of graphs that can be made with a maximum of $n_{\max}$ nodes. This number grows with $\mathcal{O}(e^{n_{\max}^2})$. To reduce the number of states in the HMM proposed in section 4.4, we propose a new HMM for an ERNEC, which significantly reduces the number of states.

Consider $\{g_1, g_2, ..., g_l\}$ to be an ERNEC. Also, consider the chain $X = \{X_1, X_2, ..., X_l\}$ to be a chain in which X_i shows the number of nodes in g_i. The new chain, $\{X_i\}$, is also a Markov chain. This is because the state of X_i only depends on X_{i-1} and its respective node transition probabilities. Specifically, the transition probabilities for $\{X_i\}$ can be written as the following.

$$p(X_i|X_{i-1}) = \begin{cases} a(X_{i-1}), & \text{if } X_i = X_{i-1} + 1 \\ r(X_{i-1}), & \text{if } X_i = X_{i-1} - 1 \\ s(X_{i-1}), & \text{if } X_i = X_{i-1} \\ 0, & \text{otherwise} \end{cases}$$

Now, consider another chain, $Y = \{Y_1, Y_2, ..., Y_l\}$. In this new chain, we have $Y_i = f(g_i)$ for a desired function f. We can build our HMM by using $\{X_i\}$ as the original Markov chain and $\{Y_i\}$ as the observed sequence. This way, emission probabilities can be written as

$$p(Y_i|X_i) = \frac{\sum_{g \in G_{X_i}} I(f(g) = Y_i)}{2^{\binom{X_i}{2}}},$$

where I is a function which returns 1 if the condition inside it is satisfied and 0 otherwise.

This way, the number of states of the Markov chain will increase linearly with

$n_{\max}$ and not exponentially. This will make the implementation of the HMM computationally more feasible. The transition probabilities are easily calculated. However, for calculating the emission probabilities, we still need to calculate $f(g)$ for all $g \in G$, where $|G|$ grows exponentially with $n_{\max}$ as mentioned earlier. To overcome this issue, we estimate the emission probabilities using an HMM estimation algorithm such as the Baum–Welch algorithm [98] and do not calculate them directly. In other words, we take the following steps:

1. Build our random ERNEC.
2. Build our NPC from our ERNEC.
3. Create the $n_{\max}$ by $n_{\max}$ transition probability matrix for transition between different numbers of nodes in our ERNEC (Eq. 4.20).
4. Use an HMM estimation algorithm to estimate the emission probabilities of the HMM (X_i, Y_i).
5. Run the forward algorithm to calculate the probability of the observed sequence.
6. Estimate the entropy rate of the NPC using the AEP.

The above algorithm will help us estimate the probability of an observed sequence of an NPC in an efficient manner. We can first run the described algorithm on several instances created from the same NPC to have an estimate of its entropy rate and then use that value to indicate if a sequence is typical or not.

4.7 Simulation results

For observing the results of our model, we simulated Erdős–Rényi Network Evolution Chains using a MATLAB program. We will show that the results of the simulation agree with the theorems stated in this chapter.

4.7.1 Overview of the code

The maximum number of nodes in the graph ($n_{\max}$), the Erdős–Rényi parameter, and the length of the random chain to be generated are specified by the user. $a(i)$, $r(i)$, and $s(i)$ are chosen randomly in the following manner. For each i $(1 < i < n_{\max})$, two points are chosen uniformly in $(0, 1)$. These two points divide $[0, 1]$ into three areas which will provide us with three positive numbers whose sum equals one. We assign the first one to $a(i)$, the second one to $r(i)$ and the third one to $s(i)$. For $i = 1$, we divide $[0, 1]$ into two sections and assign the first one to $a(1)$ and the second one to $s(1)$. For $i = n_{\max}$, we divide $[0, 1]$ into two sections and assign the first one to $r(n_{\max})$ and the second one to $s(n_{\max})$. After these parameters are set, the chain of graphs is randomly created in the manner described in Definition 4.6.1. Please note that in the following simulations, the value for $n_{\max}$ is deliberately chosen to be small for simplicity in presenting the results and lowering the complexity of the simulations.

4.7.2 Stationary distribution

In this section, we compare the stationary distribution of the number of nodes calculated from Theorem 4.6.2 and the one observed in the simulated chain for a number of random chains.

Table 4.1 and Table 4.2 show the simulation results for two random ERNECs, and compare the calculated value for the stationary distribution of the number of nodes with their observed values in the simulations. These results clearly show that the observed values for the stationary distribution of nodes match their corresponding calculated value. The slight difference that can be seen in both tables between the corresponding calculated and observed values can be attributed to the finite length of the chain. Stationarity of the process guarantees that these values tend to be equal to each other as the length of the chain tends to infinity.

Table 4.1. Simulation results for a random ERNEC with 5 nodes: stationary distribution of the number of nodes

ERNEC with $n_{\max} = 5$, $q = 0.7$ and length= 100000					
i	1	2	3	4	5
Calculated π_i	0.4589	0.2717	0.2274	0.0409	0.0011
Observed π_i	0.4582	0.2744	0.2251	0.0411	0.0013

Table 4.2. Simulation results for a random ERNEC with 8 nodes: stationary distribution of the number of nodes

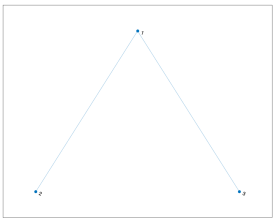
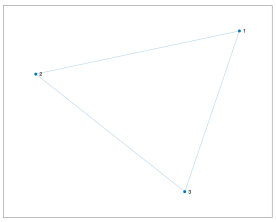
ERNEC with $n_{\max} = 8$, $q = 0.7$, and length= 100000				
i	1	2	3	4
Calculated π_i	0.0486	0.0934	0.1405	0.2743
Observed π_i	0.0494	0.0937	0.1410	0.2732
i	5	6	7	8
Calculated π_i	0.2564	0.1780	0.0052	0.0035
Observed π_i	0.2547	0.1772	0.0063	0.0047

4.7.3 Stationary distribution of graphs

In this section, we want to observe the most important property of ERNEC; the fact that the stationary distribution of graphs conditioned on their number of nodes matches the Erdős–Rényi distribution. To this end, as a toy example, an ERNEC is defined with an $n_{\max}$ of 3 and Erdős–Rényi parameter of 0.7. An instance of length 1,000,000 is then created from it. We counted the occurrence of 3 specific graphs with 3 nodes and divided those numbers by the number of occurrence of all of the graphs with 3 nodes. This provided us with an estimation of the graphs' conditioned stationary distribution. We also calculated their expected probability of occurrence based on the $G(3, 0.7)$ model. The results can be seen in Table 4.3. The estimated probabilities are very close to their expected ER probabilities and therefore the simulation backs up Theorem 4.6.1. The small difference between the observed and the calculated probabilities can again be attributed to the finite length of the

chain.

Table 4.3. Stationary distribution of graphs in an example ERNEC

ERNEC with $n_{\max} = 3$, $q = 0.7$, and length= 1000000		
Graph	Estimated Probability	ER Probability
	0.1453	0.1470
	0.0263	0.0270
	0.3372	0.3430

4.7.4 Entropy rate

One of the ultimate goals of the concept of Network Evolution Chains is to calculate the entropy rate of the process and be able to define typical sequences of the network. In this section, we examine the properties of entropy rate of ERNEC using two simulations.

For the first simulation, an ERNEC was first defined. Afterwards, its entropy rate was calculated using Eq. 4.27. Then, five chains of length 200,000 were created from the ERNEC. For each chain, we calculated $-\frac{1}{l}\log p(g_1, ..., g_l)$ and then compared it to the calculated entropy rate. The results are shown in Table 4.4.

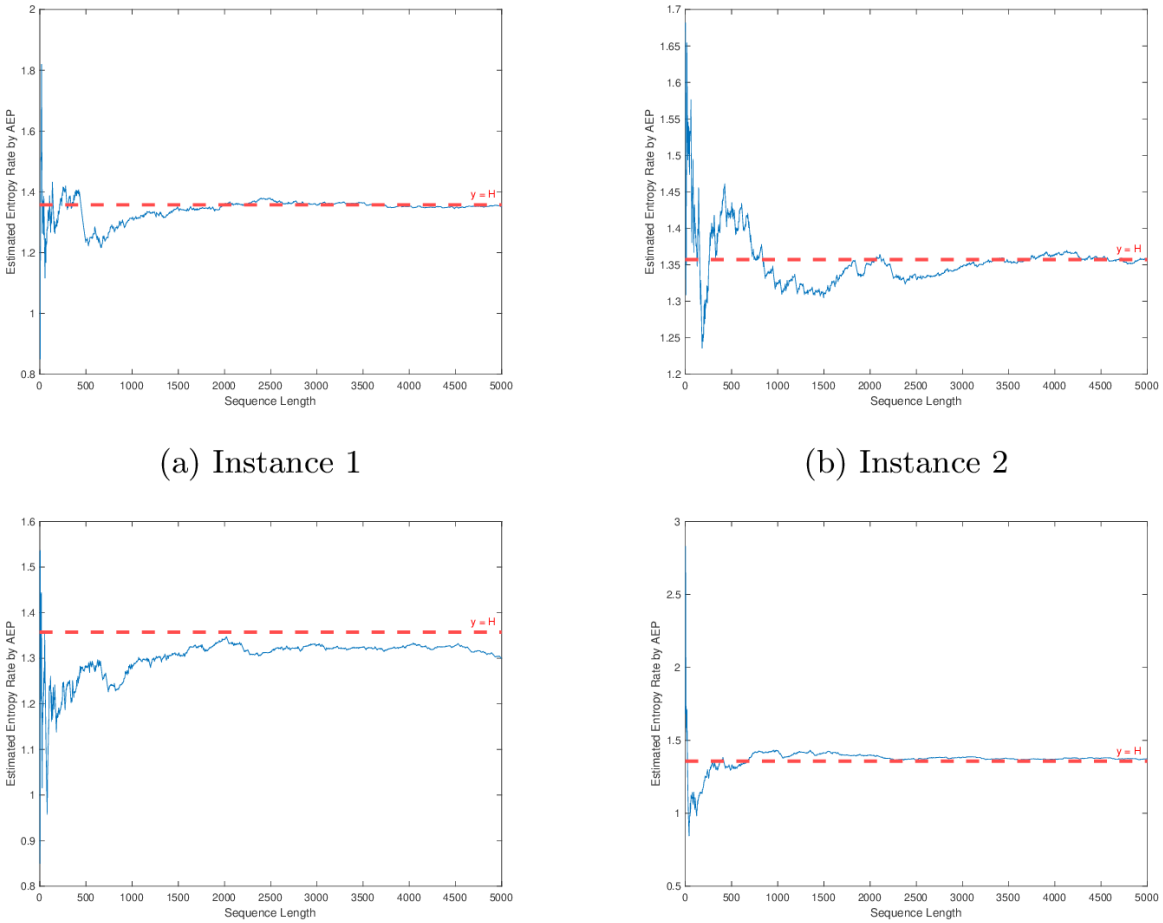
Table 4.4. Simulation results for a random ERNEC: entropy rate

ERNEC with $n_{\max} = 5$, $q = 0.7$, and $l = 200000$		
Calculated Entropy Rate for ERNEC		2.6876 b
instance	$-\frac{1}{l}\log p(g_1, ..., g_l)$	$\lvert H(g) + \frac{1}{l}\log p(g_1, ..., g_l)\rvert$
1	2.7218	0.0342
2	2.7180	0.0304
3	2.7242	0.0366
4	2.7101	0.0225
5	2.7082	0.0206

The calculated entropy rate measures the amount of uncertainty we expect when the network goes from one step to the next measured in bits. We expect most of the chains that are randomly created using the model to nearly be a typical sequence. We know that a typical sequence is a sequence whose observed mean entropy is near the actual entropy. Table 4.4 shows that the observed mean entropy for all of the instances is near the actual entropy rate and therefore those instances represent some of the typical sequences of length 200000 of graphs that are created using the specified ERNEC.

For the second simulation, we first defined an ERNEC with $n_{\max} = 5$ and an Erdős–Rényi parameter of 0.7. Then, we created four instances of length 5000 from it. Fig. 4.2 illustrates the results of this simulation. In each of the subfigures, the estimated entropy rate is plotted as a function of the number of the samples used from the chain to estimate the entropy rate using the AEP. The AEP states that as the length of the chain tends towards infinity, it is expected for the estimated entropy to tend towards the actual entropy rate. This result can be observed for all subfigures in Fig. 4.2, except for 4.2c. A gap can be observed between the calculated and estimated entropy rate of Fig. 4.2c. This gap shows that based on the value chosen for ϵ in Definition 4.3.1, there exist typical sets in which the respective sequences of all subfigures in Fig. 4.2 are considered to be in the typical set, except for the

respective sequence of Fig. 4.2c. In other words, the respective sequence of Fig. 4.2c is not as typical a sequence as the other three sequences.



(a) Instance 1

(b) Instance 2

(c) Instance 3

(d) Instance 4

Figure 4.2. The estimated entropy rate for four instances of the same ERNEC

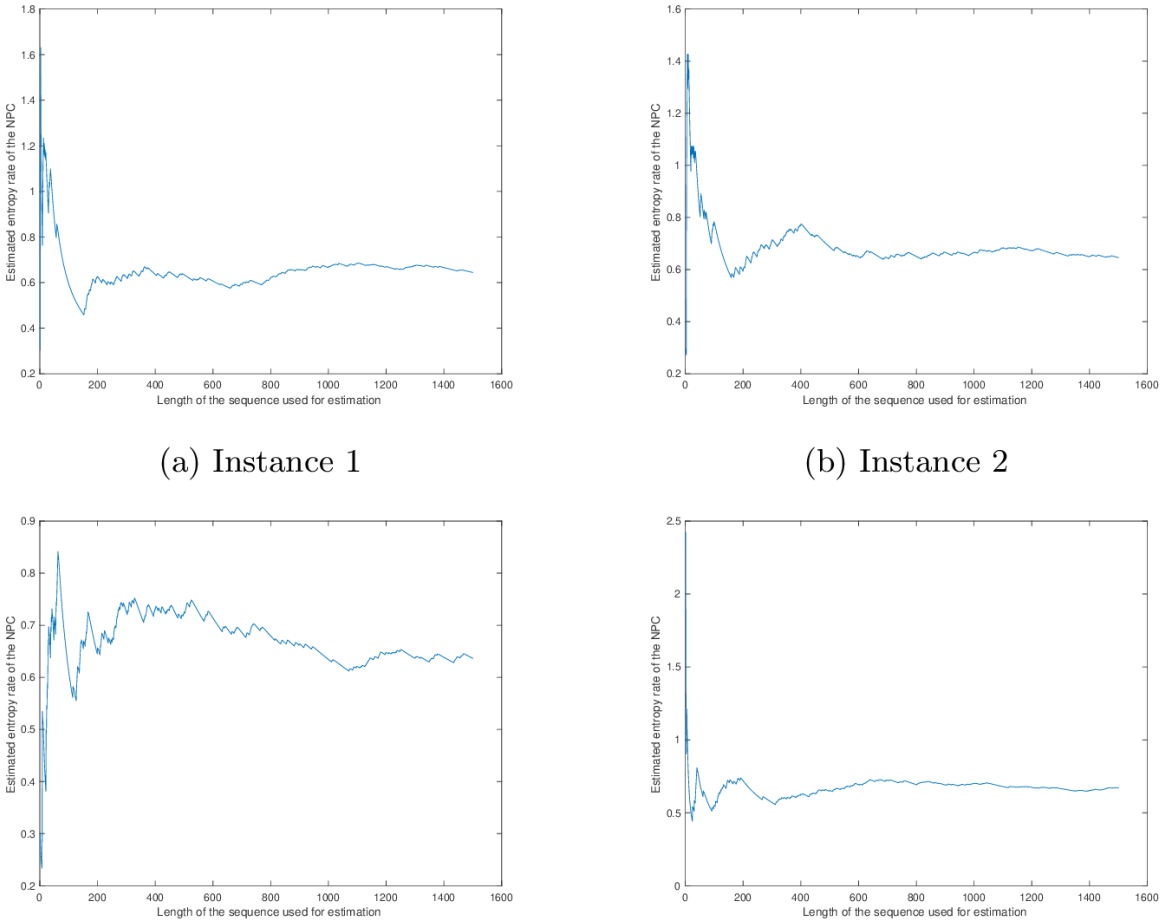
4.7.5 Number of triangles in a network

In this section, in addition to simulating an ERNEC using the described method, we also simulate a Network Property Chain. For this simulation, we are interested in the number of triangles formed in the network. Notice that using graph theory, if A is the adjacency matrix of a network, the number of triangles in the network can be calculated using the following equation.

$$\text{Number of triangles in the network} = \text{Trace}\, A^3/6 \tag{4.28}$$

First, we defined our ERNEC and then built its respective NPC by counting the number of triangles in each network. Afterwards, we formed the HMM defined in Section 4.6.4 and then estimated the transition and emission probabilities by having the NPC and the respective states. For this matter, the function **hmmestimate** in MATLAB was used, which uses a maximum likelihood estimation to estimate the parameters of an HMM [98]. Having the transition and emission probabilities, the forward algorithm calculates the probability of the observed sequence. We can then use the probability of the observed sequence to estimate the entropy rate of the Network Property Chain by the AEP. We chose $n_{\max} = 3$. This way, the number of triangles in the network is either zero or one. The Erdős–Rényi probability was set as 0.8 and four instances of length 100000 were created from the defined ERNEC. For each of these four instances, we created its NPC by counting the number of triangles in each graph. We kept the last 1500 samples of each NPC for calculating its entropy rate using the AEP. The results of this simulation are plotted in Fig. 4.3. We have plotted the estimated entropy rate as a function of the number of NPC samples used in Eq. 4.8 for estimating the entropy rate. We can observe that, for all of the four subfigures of Fig. 4.3, the entropy rate is tending towards the same value. Among these four instances, we can see that for Fig. 4.3c the convergence is slower than the others. This can be interpreted as the NPC of Fig. 4.3c not being as typical a sequence as the others. For all of the plots in Fig. 4.3, we can observe that the estimated entropy rate for the NPC is tending towards a value between 0.6 and 0.7. This means that for this specific ERNEC, knowing the number of triangles at step i, there is around 0.7 bits of uncertainty about the number of triangles at time $i + 1$. It also means that the average uncertainty about each symbol in our NPC is around 0.7 bits. Notice that because we only have 2 possible symbols in this NPC,

the maximum amount of uncertainty is 1 bit per symbol. However, we can see that having information about the network has reduced this uncertainty to around 0.7 bits.



(a) Instance 1

(b) Instance 2

(c) Instance 3

(d) Instance 4

Figure 4.3. The estimated entropy rate for the number of triangles in an NEC

4.8 Summary

Network Evolution Chains were introduced as models that can be used to simulate the evolution of dynamic networks over time, while providing us with an information-theoretic overview of the behaviour of such networks. The NEC model uses stochastic processes to simulate dynamic networks in a way that satisfies the conditions of the

Shannon-McMillan-Breiman theorem. This leads to the definition of typical sequences of an NEC, which will act as the basis of compressing the evolution of a network. This will help us in applications in which the state of a network needs to be transmitted in an efficient manner, when the previous states are assumed to be known. Knowing the entropy rate of the process can lead to better compression and transmission of the future states. Sometimes, we are interested in certain properties of the network, such as the occurrence of a certain structure. To this end, Network Property Chains were introduced and analysed as processes that only focus on our desired properties. Additionally, two methods were introduced for simulating the concept of the NEC in the continuous time domain, one of which is based on continuous time Markov chains. Furthermore, Erdős–Rényi Network Evolution Chains were defined as a subset of NECs in a way that their conditional distribution matches that of the Erdős–Rényi random graph generation model and can be used in scenarios in which the Erdős–Rényi model is used for simulating networks.

One of the most important features of the NEC model is its versatility in being applicable to most of the existing methods on modeling the growth and evolution of networks. We would like to highlight that NEC is not a new model for network growth and dynamics, but a novel information-theoretic framework to allow the analysis of such models in a more formal manner. The potential contributions of this model is vast, as it can be used in applications such as network state prediction, network fault detection, topology compression etc.

Chapter 5

Complexity and Compression of Trees

In the previous chapters, we discussed the work done so far on quantifying the complexity of graphical data structures. As an important subset of graphs, trees are widely used in different areas of science. One of their biggest application area is the field of network science to model different structures, patterns, and behaviours. Some networks are formed specifically as a tree, such as the design used in the ZigBee specification [101]. Additionally, trees are often encountered in practice as subsets of complex networks. An example of this can be seen in routing tables in networks, which are essentially a tree structure [102]. Other notable fields in which trees are used for modelling data include phylogenetic trees [103], parse trees in natural language processing [104], and Barnes-Hut trees in astrophysics [105].

Compared to the numerous studies on this area for general graphs, measuring the complexity of tree structures has barely been studied before. Some tree models for which entropy has been studied before are random binary trees [106] and specific cases of plane trees [107]. Additionally, some studies such as [108] define their own entropy measure and assess their compression algorithm based on that, which is not

our approach in this thesis. This shows that most of the studied models either lack generality, or lack the ability to simulate trees observed in real networks. Even though trees can be seen as a subset of graphs, their unique characteristics and features can be utilized to extract more accurate bounds and results, which can prove to be more useful. Consequently, we focus on the complexity and compression of trees in this chapter.

This chapter starts with an overview of the background work done on random generation of trees, the study of their entropy, and tree compression algorithms. We continue the chapter by quantifying the entropy of a number of existing random tree models. We then introduce and analyse the spanning tree model, which is a novel model for generating random trees. We end the chapter by providing efficient and near-optimal coding methods for trees generated using the models discussed in the chapter.

5.1 Background and basic definitions

Despite their vast applications, there are very few models for random generation of trees. As discussed in Chapter 2, there are numerous models for creating random graphs, such as the ER and SBM models. However, this variety can not be seen in random models for trees. The existing models for trees are very limited, and most of them rely solely on a uniform distribution among the possible trees. For example, a random generation of a Prüfer sequence [109] can result in a random tree. Other models focus only on specific type of tree, such as binary trees [110]. One of the most detailed studies on random trees can be found in [111], where several random tree models are introduced and analysed. The trees created using most of the models introduced in [111], such as Polya trees [112] and Galton-Watson trees [113] can grow indefinitely. This makes them only useful in limited application scenarios. For

instance, recursive trees are useful in the analysis of the spread of epidemics [114], family trees of preserved copies of ancient manuscripts [115], and chain letters or pyramid games [116], all of which have the possibility of growing indefinitely.

One of the most simple, yet powerful, models that has been introduced so far for random tree generation is called Simply Generated Trees (or unconditioned Galton-Watson trees). This model is simple, but still effective to capture the dynamics of trees. It has been shown that Simply Generated Trees can act as a generation model for many different types of random trees that are seen in practice [117]. One example of this is modelling branching processes [118]. We start by focusing on this model, and extract its information theoretic content in terms of Shannon's entropy. This can be beneficial in analysing the situations in which Simply Generated Trees are used to model the generation of trees.

Another way of randomly generating trees, which has not been worked on in detail in the past, is the extraction of a random tree from an already existing underlying network. This is a scenario that happens a lot in practice. For instance, as stated earlier, the tree corresponding to the routing table of a node in a network is simply a spanning tree of the original network. Therefore, it will be beneficial to look at this method of developing random trees, and analyse it using information theory. In this chapter, we formalize this method of creating random trees, and then find an upper bound to its entropy.

Ultimately, we focus on compressing trees to their theoretical limit, which is the entropy of the source. Tree compression can be used in many applications that involve the use of trees. For instance, applications of tree compression in syntax-directed compression of program files and pixel trees are studied in [119]. If traditional methods such as adjacency matrices or adjacency lists are used for coding trees, the storage space and bandwidth required for communicating trees grow exponentially large with the size of the tree. There also exists tree coding methods such as the Prüfer code, the

Neville codes, and the Demo and Micikevičius code have a fixed codeword length for same trees even if they are generated from different distributions [120], and therefore are not entropy-optimal. Additionally, survey methods that do study information-theoretic optimality of the introduced coding methods often do so by comparing the average codeword length with the entropy of the uniform source only [119]. Recently, powerful tree compression algorithms such as tree compression with top trees [121] and tree structure compression using RePair [122] have been introduced. However, the performance of these algorithms is not analysed using information theory, which always leaves the question of whether or not the structures could be compressed more efficiently. For these reasons, we seek compression algorithms for trees that are both easy to implement and are shown to have a near optimal performance with respect to information-theoretic measures. The family of dictionary-based compression methods are among the most preferred lossless compression algorithms for different sources and data types [123]. They are widely used in applications such as text compression [124] and image compression [125]. Because of the advantages of dictionary-based compression methods, we want to be able to use them to optimally compress tree structures as well. However, the issue is that a tree structure is not made out of a consecutive series of symbols. Because of this, our main approach for compressing trees will be to first decompose the trees at hand into a sequence of symbols, and then apply a dictionary-based compression algorithm on that sequence.

5.2 Entropy of tree sources

In this section, we focus on quantifying the entropy of tree structures. If we look at trees as a random variable drawn from a pool of possible trees, we need a probability distribution on these trees to be able to calculate Shannon's entropy. This necessitates the existence of a random model for creating the tree structures at hand. Having a

random model would entail having a probability distribution on the trees that models the dynamics of creating trees in real-life scenarios. Unfortunately, the issue is that there does not exist an adequate variety of random models for creating tree structures. Existing models are very limited in terms of the scenarios they can simulate. For instance, there is no parameter to set the number of nodes in Galton-Watson trees, and the number of nodes in the tree can be anywhere from one to infinity. We start with a very simple method for creating random trees, which is to choose a tree uniformly among all possible trees with the same number of nodes. Additionally, One of the few existing models for creating random trees, called Simply Generated Trees, is studied. We will then introduce another model for creating a random tree. This model is based on creating the underlying graph first, and then choosing one of its spanning trees as the output of the random generator. We call this model the spanning tree model, and then study its entropy.

As the focus of this section is on tree sources, we start by providing a more formal definition for them.

Definition 5.2.1 (Random tree source)**.** A random tree source $(T, p_T(t))$ is defined by specifying the following parameters.

- T: set of all possible trees that can be generated by the source. The size of this set can be finite or infinite.

- $p_T(t)$: A probability distribution on the trees in T. We may only use the notation $p(t)$ if it is clear which tree source we are talking about. Generally, this distribution can be time-dependent or time-independent.

Definition 5.2.1 mirrors the definition of random graph sources, as provided in Section 2.1.

5.2.1 Entropy of the uniform source

In this section, we calculate the entropy of a uniform source for various types of tree structures. By a uniform source we mean that having T, the probability of observing any $t \in T$ is simply $1/|T|$. This way, the entropy of the source will simply be $\log |T|$ bits.

We first start with unlabelled unordered rooted trees. The sequence of the number of unlabelled unordered rooted trees with n nodes is listed on the on-line encyclopedia of integer sequences [126, A000081]. Table 5.1 shows the possible trees of this kind that can be built with up to four nodes. It is known that the asymptotic limit of this sequence is $cd^n n^{-3/2}$ [127], where c and d are constants that can be found at [126, A187770] and [126, A051491], respectively. Consequently, the asymptotic uniform entropy for this model, H_T, can be calculated using the following equation.

$$\begin{aligned} H_T &\sim n \log d - 1.5 \log n + \log c \\ &\approx 1.5635n - 1.5 \log n - 1.1846 \end{aligned} \tag{5.1}$$

Eq. 5.1 shows that the growth rate of the entropy of a uniformly distributed unlabelled unordered rooted tree source is asymptotically linear.

To study the same for unlabelled ordered rooted trees, we refer to the fact that their number matches the sequence of Catalan numbers [128, Ch. 8]. In other words, the nth Catalan number, shown with C_n, equals the number of possible ordered rooted trees that can be built on n nodes. The following equality is well-known about Catalan numbers [129].

$$\begin{aligned} C_n &= \frac{1}{n+1}\binom{2n}{n} \\ &= \frac{4^n}{\sqrt{\pi n^3}}\left(1 + O\left(\frac{1}{n}\right)\right) \end{aligned} \tag{5.2}$$

Based on Eq. 5.2, the uniform entropy of the source can be calculated using the following equation.

$$\begin{aligned} H_T &= \log C_n \\ &= \log \frac{1}{n+1}\binom{2n}{n} \\ &= \log \frac{4^n}{\sqrt{\pi n^3}}\left(1 + O\left(\frac{1}{n}\right)\right) \end{aligned} \tag{5.3}$$

Additionally, the asymptotic behaviour of this entropy can also be analysed using Eq. 5.3, which would provide us with the following result.

$$H_T \sim 2n \tag{5.4}$$

An interesting thing to note is that [106] achieves the same result for the uniform entropy, but for rooted full binary trees with $2n-1$ nodes. This suggests the existence of a one-to-one mapping between ordered rooted trees with n nodes, and rooted full binary trees with $2n-1$ nodes.

It is already known that a transformation called child-sibling representation maps ordered rooted trees to binary trees [130]. Even though this transformation does not create full binary trees and is not a bijection, we can create a bijection between ordered rooted trees and rooted full binary trees using a similar idea. We call this transformation the double-node transformation, as every node in the original rooted tree, except for the root, will be mapped into two nodes in the binary tree. This is why the n nodes of the original tree will be mapped into $2n-1$ nodes in the binary tree. This transformation is a bijection as every rooted ordered tree can be mapped into a unique rooted full binary tree and vice-versa. In this transformation, each node in the original tree is transformed into two nodes in the binary tree: a child node and a sibling node, which are connected to the same parent in the transformation. The

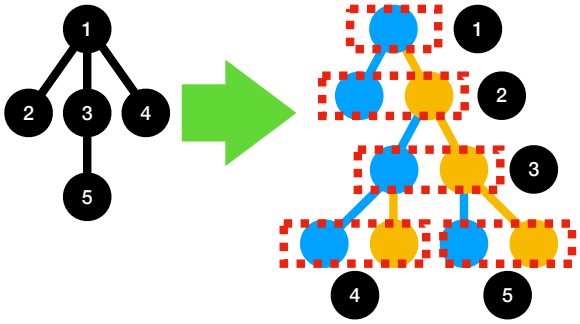


Figure 5.1. Example of double-node transformation

steps of this transformation are explained below, and Fig. 5.1 shows an example of this transformation.

Double-node transformation

Input: a rooted ordered tree with n nodes

Output: a rooted full binary tree with $2n - 1$ nodes

1. The root of the input is mapped to the root of the output. As the root does not have any siblings, it is only considered a child node.
2. The nodes of the input are traversed using BFS.
3. Every observed node will be transformed into two nodes, which will both be attached to the same node in the binary tree. We call the left one the child node, and the right one the sibling node.
4. If the observed node is the first child of its parent, its corresponding child and sibling nodes will be attached to the child node of its parent. If not, its child and sibling nodes will be attached to the sibling node of its closest sibling to its left.
5. The mapping continues until all the nodes in the input tree have been traversed.

Double node transformation provides a bijection between rooted ordered trees with n and rooted full binary trees with $2n-1$ nodes. Therefore, the uniform entropy results for these two families of trees are the same.

Finally we study labelled rooted and unrooted trees. The number of labelled unrooted trees is shown to be n^{n-2} [131, p. 26], which implies that the uniform entropy can be calculated using the following equation.

$$H_T = (n-2)\log n \tag{5.5}$$

Additionally, for any given labelled unrooted tree with n nodes, we can pick any of its n nodes as the root. Therefore, the number of labelled rooted trees is n^{n-1}, and their uniform entropy can be calculated using the following equation.

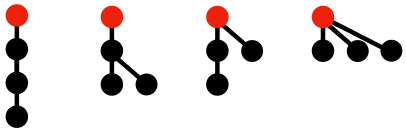
$$H_T = (n-1)\log n \tag{5.6}$$

5.2.2 Entropy of Simply Generated Trees

Simply Generated Trees (SGT) are a popular family of random trees. They were first introduced in [132], and since then have been used to model random trees. The reason for the popularity of this model lies in its simplicity, and the fact that it is powerful enough to model many scenarios. They are also used to generate more complex tree generation models. In this section, we will calculate the entropy of this family of random tree models.

Simply Generated Trees are generated based on a probability distribution on the number of children of each node. They are rooted and ordered trees. To build an SGT, we need a distribution on the set of whole numbers $\{0, 1, 2, \ldots\}$. We call this distribution the children distribution of the SGT, and show it with $p_C(c)$. The children distribution is essentially a distribution on the number of children that each

Table 5.1. Unlabelled unordered rooted trees of up to four nodes

n	Possible trees	Count
1		1
2		1
3		2
4		2

node can have, independently from other nodes. The only condition that we impose on the children distribution is for $p_C(0)$ to be nonzero. This is so that the trees created using this model can be finite. For example, we can use a geometric distribution, binomial distribution, or generally any discrete distribution on whole numbers that satisfies $p_C(0) \neq 0$. After this distribution is chosen, we are ready to generate the random tree. The following steps show how an SGT is created using its underlying children distribution.

Creating an SGT

1. Create the root of the tree. This will be level 0.
2. Create the children of the root, based on a number acquired from the children distribution.
3. For all $i > 0$, go through the nodes on level i, and choose the number of

children for each of them, based on the children distribution.

4. The algorithm is terminated once the number of children for all the nodes is chosen, and there is no more node to explore (Note that once a node is decided to have zero children, the branch corresponding to that node will be terminated, and the node will become a leaf node).

Note that the resulting tree will have ordered branches, as the number of children for each node is chosen sequentially. Generating a tree this way will result in a tree whose probability is equal to the product of the probability of the number of children of all of its nodes. The following example illustrates SGTs that can be made using a specific children distribution.

Example 5.2.1. Assume that we have a children distribution for which $p_C(0) = p$, and $p_C(1) = 1 - p$. In the trees generated using this distribution, nodes can therefore have no child, or have only one child. Table 5.2 illustrates the possible trees that can be made using this model, alongside their probabilities.

Table 5.2. Example of Simply Generated Trees and their probabilities

Graph	Probability
	p
	$p(1-p)$
	$p(1-p)^2$
Line with n nodes	$p(1-p)^{n-1}$

We now want to use information theory to measure the information content of SGTs. We will first start by studying the possible number of nodes in an SGT. It can easily be seen that in case $p_C(0) \neq 1$, the number of nodes in an SGT can theoretically

be unlimited. This is because as soon as the probability of having more than zero children is nonzero, the branches can keep growing infinitely. However, we can still study the average number of nodes in SGTs.

Assume that we have an SGT model, with a children distribution $p_C(c)$, where $p_C(0) \neq 1$. Additionally, assume T to be the set of all possible trees that can be generated by this model. We are interested in calculating the average number of nodes of trees in T. Based on the average number of children, Simply Generated Trees can be categorized into subcritical, critical, and supercriticial. This categorization can by calculating the expected number of children $\mathbb{E}[c]$, which we show with $\bar{C}$, based on the children distribution. The model is in the subcritical, critical, and supercritical if and only if $\bar{C} < 1$, $\bar{C} = 1$, $\bar{C} > 1$, respectively [117]. It will be shown that the average number of nodes for an SGT model is limited only in the subcritical phase. This number goes to infinity for the critical and super critical phases.

Let n be the random variable that shows the number of nodes in a tree, and let n_t show the number of nodes in a particular tree t. We can use the following equation to calculate the average number of nodes in the trees created using this model.

$$\mathbb{E}[n] = \sum_{t \in T} n_t p(t) \tag{5.7}$$

For each tree t, n_t can be written as the sum of the nodes in different levels, with level 0 being the root node. If $n_{t,i}$ shows the number of nodes in the ith level of tree t, Eq. 5.7 can be rewritten as

$$\mathbb{E}[n] = \sum_{t \in T} p(t) \sum_{i} n_{t,i} \tag{5.8a}$$

$$= \sum_{i} \sum_{t \in T} p(t) n_{t,i}. \tag{5.8b}$$

Based on Eq. 5.8b, the average number of nodes can be rewritten as the sum of the

average number of nodes in each level among all possible trees. Therefore, we need to calculate the average number of nodes for each level individually. We will show the random variable for the number of nodes in level i with n_i. Firstly, as all the trees will have only one node in level 0, we have $\mathbb{E}[n_0] = 1$. Additionally, if a tree t has $n_{t,i-1}$ nodes in level $i-1$, its expected number of nodes in level i will simply be $n_{t,i-1}\bar{C}$. This is because the number of children of individual nodes is independent. Therefore, if we show the probability of a generated tree having i nodes in level n with $p_n(i)$, we can write the following recursive equation for $i > 0$.

$$\mathbb{E}[n_i] = \sum_{j=0}^{\infty} p_{i-1}(j) j \bar{C} \tag{5.9a}$$

$$= \bar{C} \sum_{j=0}^{\infty} p_{i-1}(j) j \tag{5.9b}$$

$$= \bar{C}\mathbb{E}[n_{i-1}] \tag{5.9c}$$

Based on Eq. 5.9c and the fact that $\mathbb{E}[n_0] = 1$, we have

$$\mathbb{E}[n_i] = \bar{C}^i. \tag{5.10}$$

Combining Eq. 5.8b and Eq. 5.10, we get the following equation for the average number of nodes in the SGT model.

$$\mathbb{E}[n] = \sum_{i=0}^{\infty} \bar{C}^i \tag{5.11}$$

As Eq. 5.11 is a geometric series, we know that it will only converge when we have $|\bar{C}| < 1$. This means that the SGT model needs to be in the subcritical phase for the average number of nodes of the model to converge. From now on, we will assume the SGT models that we work with to be in the subcritical phase, so that the corresponding trees have a finite average node number. For the subcritical phase, the

average number of nodes per tree can therefore be found using the following equation.

$$\mathbb{E}[n] = \frac{1}{1 - \bar{C}} \tag{5.12}$$

Now, we move on to quantifying the entropy of an SGT model based on its children distribution. Assume the corresponding children distribution of the SGT model to have an entropy of H_C. In other words, we have

$$H_C = -\sum_{i=0}^{\infty} p_C(i) \log p_C(i). \tag{5.13}$$

We want to represent the entropy of the SGT model based on H_C. If we show the entropy of the SGT model with H_T, we can write

$$H_T = -\sum_{t \in T} p(t) \log p(t). \tag{5.14}$$

To simplify Eq. 5.14, note that we can write $p(t)$ as the product of the probabilities of the number of children of each node in t. We use this to write $\log p(t)$ in the form of a sum. We use n_t to show the number of nodes in t, and $c_{t,i}$ to show the number of children of node i in t.

$$H_T = -\sum_{t \in T} p(t) \sum_{i=1}^{n_t} \log p(c_{t,i}) \tag{5.15}$$

To simplify Eq. 5.15, we open it up based on the number of children. We use $n_{t,i}$ to show the number of nodes in t that have i children. This way we can write

$$H_T = \sum_{i=0}^{\infty} H_{T,i}, \tag{5.16}$$

where

$$H_{T,i} = -\log p_C(i) \sum_{t \in T} p(t) n_{t,i}. \tag{5.17}$$

To calculate $H_{T,i}$, we note that $\sum_{t\in T} p(t)n_{t,i}$ essentially represents the average number of nodes that have i children in a random tree. To calculate this sum, we perform the same trick as when we calculated the average number of nodes in a tree, we open up the sum on the level of nodes in the tree. The average number of nodes that have i children in a specific level of the tree is simply $p_C(i)$ times the number of nodes on that level. Therefore, we can use Eq. 5.9a and write

$$\sum_{t\in T} p(t)n_{t,i} = p_C(i)\sum_{j=0}^{\infty}\bar{C}^i = \frac{p_C(i)}{1-\bar{C}}. \tag{5.18}$$

Inserting Eq. 5.18 into Eq. 5.17, we get the following equation.

$$H_{T,i} = \frac{-p_C(i)\log p_C(i)}{1-\bar{C}} \tag{5.19}$$

Finally, to calculate H_T, we insert Eq. 5.19 into Eq. 5.16, and we get the following equation.

$$H_T = -\sum_{i=0}^{\infty}\frac{p_C(i)\log p_C(i)}{1-\bar{C}} \tag{5.20a}$$

$$= \frac{H_C}{1-\bar{C}} \tag{5.20b}$$

Eq. 5.20b gives us the entropy of the SGT model. The entropy can easily be calculated from the entropy of the underlying children distribution. Additionally, note that in order for this entropy to be computable, the sum in Eq. 5.18 needs to converge, and therefore the SGT needs to be in the subcritical phase. Additionally, note that this entropy can be seen as the number of nodes times H_C, which makes sense intuitively. This is because the number of children of each node is independent from the others, and adds an entropy of H_C to the ensemble. This is a very interesting result, as it relates the entropy of the underlying children distribution to the tree model using a simple equation.

Eq. 5.20b shows that both the entropy of the children distribution and the average number of children have an effect on the entropy of the trees. An increase in the entropy of the children distribution or the average number of children per node results in an increase in the entropy of the trees. Additionally, as $0 \leq \bar{C} \leq 1$, it is concluded that $H_T \geq H_C$, with equality holding if and only if the the only possible tree that the SGT model can create is a tree with only one node.

5.2.3 Conditioned Galton-Watson Trees

In this section, we will attempt to quantify the entropy of conditioned Galton-Watson trees. Conditioned Galton-Watson trees are simply Galton-Watson trees that are conditioned on their number of nodes. In other words, we condition the trees generated using the Galton-Watson model such that $|V(t)| = n$. This will change the probability distribution of the original Galton-Watson trees.

If we show the original Galton-Watson and the new probability distribution with $p(t)$ and $\pi_n(t)$, respectively, we can write

$$\pi_n(t) = \frac{p(t)}{y_n}, \tag{5.21}$$

where

$$y_n = \sum_{|V(t)|=n} p(t). \tag{5.22}$$

Based on Eq. 5.21, the first thing that we need to know in order to calculate the entropy of this model is the value for y_n. In order to find y_n, a generating function for it is first defined as below.

$$y(x) = \sum_{n \geq 1} y_n x^n \tag{5.23}$$

It is shown in [111] that the generating function $y(x)$ satisfies the following equation.

$$y(x) = x\Phi(y(x)). \tag{5.24}$$

where $\Phi(x)$ is the following generating series.

$$\Phi(x) = \sum_{i=0}^{\infty} p_C(i)x^i \tag{5.25}$$

Having the children distribution of the underlying Galton-Watson tree, we can form the corresponding generating function $\Phi(x)$, extract the generating function for y_n using Eq. 5.24, and then calculate the distribution of the conditioned Galton-Watson tree using Eq. 5.21. However, this is only if the equation formed by Eq. 5.24 is easy enough to solve. Assuming that Eq. 5.24 is solvable and we have found y_n, we proceed to calculate the entropy of the conditioned Galton-Watson model. If we use H_n to show the entropy of the conditioned Galton-Watson trees with n nodes, we can write the following equation based on Eq. 5.21.

$$\begin{aligned} H_n &= -\sum_{|V(t)|=n} \pi_n(t) \log \pi_n(t) \\ &= -\sum_{|V(t)|=n} \frac{p(t)}{y_n} \log \frac{p(t)}{y_n} \\ &= \frac{-\sum_{|V(t)|=n} p(t) \log p(t)}{y_n} + \log y_n \end{aligned} \tag{5.26}$$

Having y_n, the only term that needs to be calculated in Eq. 5.26 is $-\sum_{|V(t)|=n} p(t) \log p(t)$. Looking at this term, it looks very much like the entropy of the unconditioned model, with the only difference being that the sum is not taken over the entire sample space. As calculating the precise value of this term is very challenging, we focus on finding an upper bound to it. It is known that the expected information of a set of outcomes is maximized when they have equal probabilities. Additionally, we know from section

5.2.1 that the number of possible rooted ordered trees on n nodes is the nth Catalan number. Therefore, we have the following upper bound on $-\sum_{|V(t)|=n} p(t)\log p(t)$.

$$\begin{aligned} -\sum_{|V(t)|=n} p(t)\log p(t) &\leq y_n \log C_n \\ &= y_n \log \frac{\binom{2n}{n}}{n+1} \end{aligned} \tag{5.27}$$

By plugging Eq. 5.27 into Eq. 5.26, we get the following upper bound on the entropy of the conditioned Galton-Watson model.

$$H_n \leq \log \frac{\binom{2n}{n}}{n+1} + \log y_n \tag{5.28}$$

Eq. 5.28 provides us with an upper bound to the entropy of the conditioned Galton-Watson model when y_n can be calculated from Eq. 5.24. Below, we provide an example of finding this upper bound for a specific family of Galton-Watson trees.

Example 5.2.2. In this example, we study conditioned Galton-Watson trees that have a uniform distribution. This family of Galton-Watson trees is studied in [111, Example 1.5]. For these, we have $p_C(i) = \frac{1}{ei!}$, and it can be shown that the π_n distributions are uniform. It is shown that for this children distribution, we have

$$y_n = \frac{n^{n-1}}{en!}.$$

By inserting y_n into Eq. 5.28, we get the following upper bound on the entropy of the conditioned model.

$$H_n \leq \log \frac{\binom{2n}{n}}{n+1} + (n-1)\log n - \log en!$$

Fig. 5.2 illustrates this upper bound using MATLAB to study its change with n. As expected, the entropy increases with n, and that this growth is almost linear for

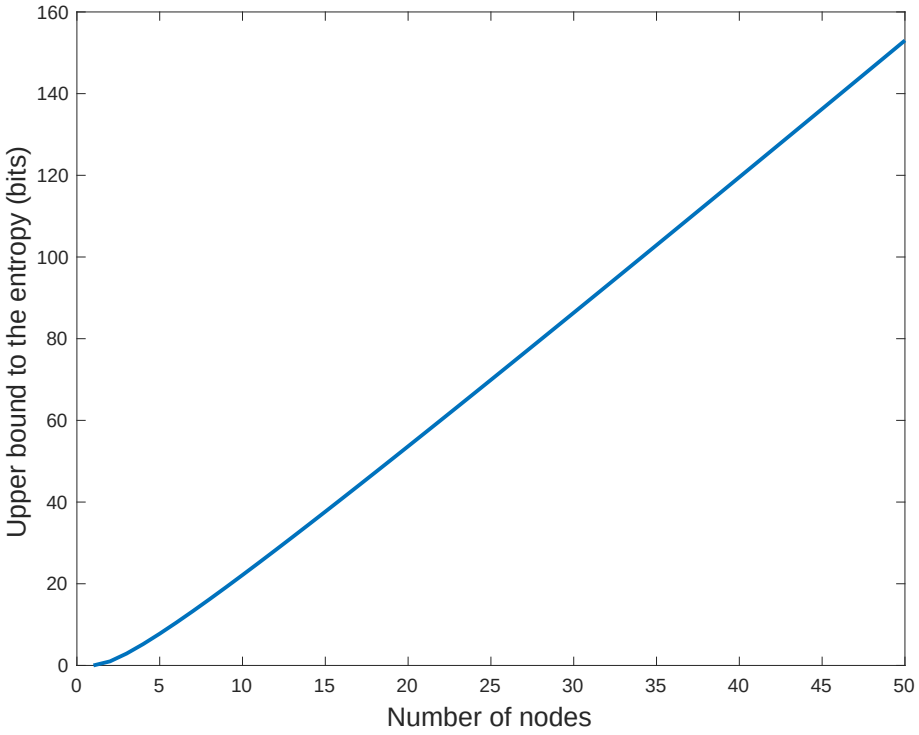


Figure 5.2. Upper bound to the entropy of the conditioned Galton-Watson model

large values of n. This can actually be checked by noting from section 5.2.1 that the asymptotic behaviour of the logarithm of Catalan numbers is $2n$, and the terms $(n-1)\log n$ and $\log n!$ cancel each other out as n increases. Therefore, for conditioned Galton-Watson trees with a uniform distribution, Eq. 5.28 is providing us with a tighter upper bound for small values of n, and tends towards the result obtained for 5.4 as n grows large.

Even though Eq. 5.28 provides us with a good upper bound to the entropy of the conditioned model, it heavily relies on y_n being computable from Eq. 5.24. Therefore, we will provide an alternative method for calculating an upper bound for the entropy of this model.

Consider the random variables $\{X_1, X_2, \ldots, X_n\}$ to be n i.i.d random variables sampled from the children distribution. For this sequence of random variables to be

able to represent a tree with n nodes, we need to have

$$\sum_{i=1}^{n} X_i = n - 1.$$

Therefore, the entropy that we are looking for can be formed using the following conditional entropy.

$$H_n = H(X_1, \ldots, X_n | \sum_{i=1}^{n} X_i = n - 1) \tag{5.29}$$

Calculating the entropy using Eq. 5.29 can prove to be challenging. Therefore, we will provide estimates to it by finding an upper bound. We can start by using a zero-order upper bound as below.

$$\begin{aligned} H(X_1, \ldots, X_n | \sum_{i=1}^{n} X_i = n - 1) &\leq H(X_1, \ldots, X_n) \\ &\leq \sum_{i=1}^{n} H(X_i) \\ &= nH_C \end{aligned} \tag{5.30}$$

Eq. 5.30 provides us with a simple upper bound. If we want to increase the accuracy of the upper bound, we can move up to higher order models. The equation below describes the first order model.

$$\begin{aligned} &H(X_1, \ldots, X_n | \sum_{i=1}^{n} X_i = n - 1) \\ &\leq H(X_1 | \sum_{i=1}^{n} X_i = n - 1) + \sum_{i=2}^{n} H(X_i | X_1, \sum_{i=1}^{n} X_i = n - 1) \\ &\leq H_{C,n-1} + (n-1)\mathbb{E}_{C,n-1}\left[H_{C,n-1-i}\right], \end{aligned} \tag{5.31}$$

where $H_{C,i}$ and $\mathbb{E}_{C,i}$ show the entropy of the children distribution conditioned on the number of children being limited to i and the expectation over this conditional distribution, respectively. The method used in Eq. 5.31 can provide us with a tighter

upper bound, but has higher computational complexity. We can also keep increasing the order in the same manner to get more accurate bounds at the cost of more computational complexity. By increasing the order we mean conditioning the entropy in Eq. 5.31 on more random variables from $\{X_i\}$.

5.2.4 The spanning tree model

Unlike graphs, the models for generating random trees are very limited. The variety observed in random graph models can not be seen in trees. Different distributions are fit to real-life networks in order to create mathematical models that can capture the properties of these networks. For instance, the Watts-Strogatz model [37] exhibits a high clustering coefficient, which is consistent with many real-life networks. However, random models that are able to exhibit the same properties of real-life tree data structures do not exist. Because of this reason, we were motivated to define a new random tree generation model, which is also based on practice. As a result, we introduce and study the spanning tree model in this section.

In practical applications, the trees we work with are often a spanning tree of a network. An example of this can be seen in network routing. Routing tables are usually used to store the shortest paths from any node in a network to any other one. The routing table for a node in a network is essentially representing a rooted tree, with the root being the origin node. Additionally, if the network forms a connected graph, this rooted tree is a spanning tree of the underlying network. Therefore, selecting a random spanning tree of the underlying network can be a practical model for generating random trees. The combination of random graph generation with random spanning tree selectors for creating random trees is a novel approach. One of the key points of this combination is that both fields have been studied extensively and powerful tools exist in both of them. We have already discussed the variety of different models that exist for random graph generation. In addition to that, there

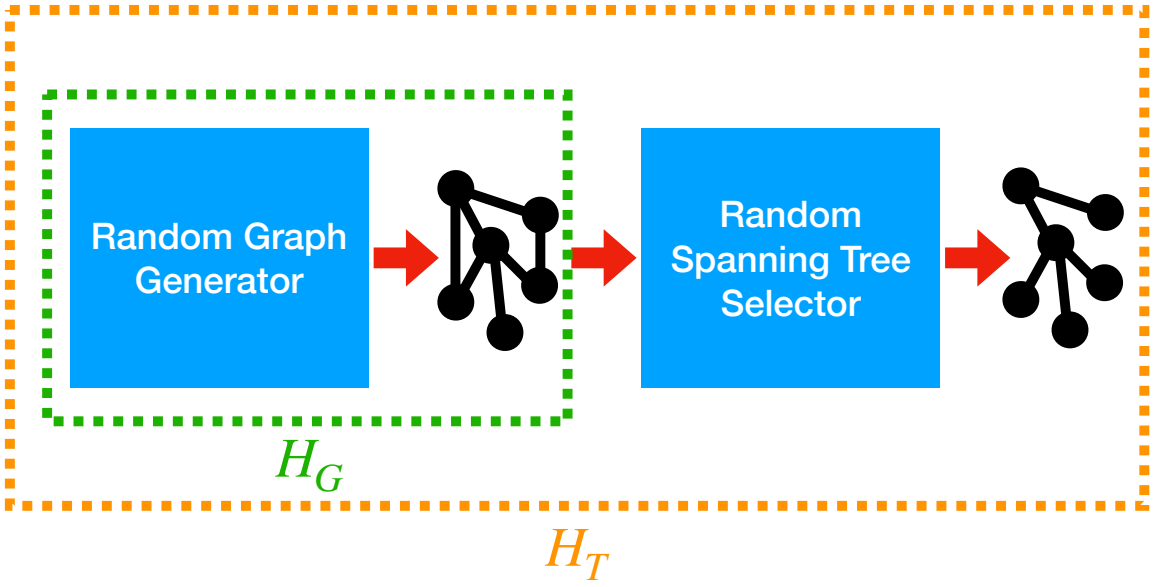


Figure 5.3. Steps of the spanning tree model

are powerful ways of randomly selecting a spanning tree of a single graph [133–135]. The introduced model is very flexible in the sense that any existing random graph generation model can be combined with any existing random spanning tree selector in order to fit the real-life scenario that we want to simulate.

Assume that we have a random graph source defined on G with the distribution p_G. The first step is to create a graph g, according to the distribution of G. g will then have a number of spanning trees (which can also be zero). We then choose one of the spanning trees of g as the generated tree. This can generally be done according to an arbitrary distribution, which can also be dependent on g. We use the term spanning tree model to refer to this method of generating a random tree. We use H_G to show the entropy of the random graph generator used in the model, and H_T to show the entropy of the trees generated using the spanning tree model. Fig. 5.3 shows the steps of the spanning tree model.

The goal in this section is to find H_T, assuming that H_G is known or can be easily calculated. We can write the following equation to find H_T.

$$H_T = H_G + H(T|G) - H(G|T) \tag{5.32}$$

If we assume that the output trees are chosen uniformly from the spanning trees of the graph, then we can write the following equation for calculating $H(T|G)$.

$$H(T|G) = \sum_{g \in G} p_G(g) H(T|G = g) \tag{5.33a}$$

$$= \sum_{g \in G} p_G(g) \log \tau(g), \tag{5.33b}$$

where the sum is taken over all the possible graphs that can be generated using the random graph generator and $\tau(g)$ shows the number of spanning trees of graph g. Notice that the summation needs to be taken over connected graphs of the source, as they need to have at least one spanning tree for $\log \tau(g)$ to be defined.

Eq. 5.32 and Eq. 5.33b show that to calculate the entropy of the spanning tree model, we need knowledge about the underlying distribution for the network topology, as well as the number of spanning trees that exists for each graph. Unfortunately, the number of spanning trees of a graph, often called its tree-number, does not have a closed form representation in terms of its number of nodes and edges. There are methods such as Kirchhoff's theorem [136, Theorem. 13.1] that provide us with a way for calculating the tree-number of graphs, but they require knowledge about the graph's adjacency information. The spanning tree entropy also depends on the model that is used to generate the underlying graph topology, which is not always known. Because of these limitations, we will consider a specific class of random graphs, those created using the famous Erdős–Rényi model [62].

The ER model was introduced in Section 2.5.1 in full detail, and its entropy was also calculated. As a reminder, each edge in an ER graph will be present with a

probability of p, independent of other edges. As the ER model does not guarantee connectivity, there will be some graphs that do not even have a spanning tree. Additionally, it is known that there is no closed-form formula to calculate the number of spanning trees of a graph given its number of nodes and edges. Because of these reasons, we will find an upper bound to the entropy of the spanning trees of ER graphs rather than its actual value.

We consider Erdős–Rényi graphs with n nodes, with parameter p. The entropy of the graphs created using this model was calculated in section 2.5.1 as below.

$$H_G = \binom{n}{2} h(p) \tag{5.34}$$

Additionally, we assume that once an ER graph is created, one of its spanning trees is chosen uniformly. If the graph does not have any spanning trees, then simply no tree is chosen. The next step is therefore to calculate an upper bound to $H(T|G)$ based on this. Note that as the spanning tree is chosen uniformly after the graph is created, we can write

$$H(T|G) = \mathbb{E}\left[\log \tau(g)\right]. \tag{5.35}$$

As logarithm is a concave function, we can apply Jensen's inequality [137] to Eq. 5.35 and obtain the following inequality.

$$H(T|G) \leq \log \mathbb{E}\left[\tau(g)\right]. \tag{5.36}$$

Now, we note that based on Cayley's formula, there are n^{n-2} possible trees on n nodes, each of which are a spanning tree of the underlying n-node graph. As the underlying graph is an ER graph, the probability of each of these trees being present in the graph is simply p^{n-1}. Considering the existence of each spanning tree to be independent of the others, we have the following upper bound on the expected number of spanning

trees of an ER graph.

$$\mathbb{E}\left[\tau(g)\right] \leq p^{n-1}n^{n-2} \tag{5.37}$$

By inserting 5.37 into 5.36, we get the following upper bound on $H(T|G)$.

$$\begin{aligned} H(T|G) &\leq \log p^{n-1}n^{n-2} \\ &= (n-1)\log p + (n-2)\log n \end{aligned} \tag{5.38}$$

We now move on to calculate the term $H(G|T)$ in Eq. 5.32. Notice that given a spanning tree of an ER graph with n nodes, we will know the status of $n-1$ edges out of the possible $\binom{n}{2}$ edges of the graph. Therefore, given a spanning tree of the graph, the remaining entropy is simply the entropy of the remaining $\binom{n}{2} - (n-1)$ edges of an ER graph. Consequently, we can write the following equation.

$$\begin{aligned} H(G|T) &= \sum_t p_T(t) H(G|T=t) \\ &= \sum_t p_T(t) \left(\binom{n}{2} - (n-1)\right) h(p) \\ &= \left(\binom{n}{2} - (n-1)\right) h(p) \sum_t p_T(t) \\ &= \left(\binom{n}{2} - (n-1)\right) h(p), \end{aligned} \tag{5.39}$$

where the sum is taken over all possible spanning trees on the n nodes of the graph, and p_T shows the probability distribution of the trees.

Ultimately, we insert the results from equations 5.34, 5.38, and 5.39 into Eq. 5.32 to get the total upper bound on the entropy of the spanning trees of the ER model. This will provide us with the following equation after simplification.

$$H_T \leq (n-1)\left(h(p) + \log(np)\right) - \log n \tag{5.40}$$

Eq. 5.40 gives us an upper bound on the entropy of trees created using the ER Spanning Tree model. Fig. 5.4 illustrates this entropy, and compares it with the entropy of the graph, and the maximum entropy for trees. The maximum entropy is calculated using the fact that a uniform distribution maximises the entropy, and there exist n^{n-2} possible labelled trees on n nodes. The simulation is run for ER graphs with 100 nodes, and the entropy is plotted as a function of the ER parameter p. For larger values of p, the estimated upper bound for the entropy is larger than the maximum entropy. However, Eq. 5.40 is providing us with a tighter upper bound when used for lower values of p. The exact boundary for which our upper bound is providing a better bound compared to the maximum possible entropy can be calculated by solving the following equation.

$$(n-2)\log n = (n-1)\left(h(p) + \log(np)\right) - \log n \tag{5.41}$$

The value of p that satisfies Eq. 5.41 is $p = 0.5$.

Additionally, we performed a simulation for the regime of giant components for ER graphs. According to [138, Ch. 11.5], an ER graph has a giant component when $p > 1/(n-1)$. A giant component is defined as a connected component, whose size grows with n. As such, this regime is of particular importance in this study. Therefore, we simulated the entropy upper bounds per node (divided by n) by setting p to the giant component regime threshold $(1/(n-1))$, and sweeping over n. Fig. 5.5 illustrates the results. As n grows large, our estimate is providing a much tighter upper bound on the entropy of the generated trees. By inserting $p = 1/(n-1)$ into Eq. 5.40, the entropy upper bound for the giant component regime threshold is calculated as $(n-1)\log n - (n-2)\log(n-2)$. Divided by n to get the entropy per node, the limit of entropy per node will tend to zero as n grows large. However, this is not the case with the maximum tree entropy, and our proposed upper bound is performing more

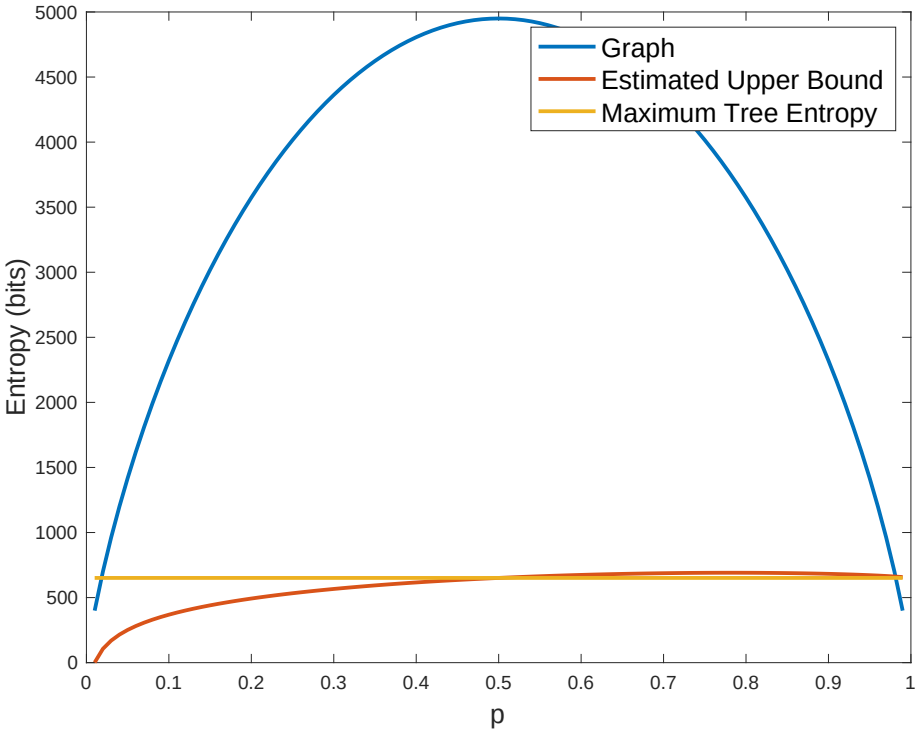


Figure 5.4. Entropy upper bound for ER spanning trees for graphs with 100 nodes as a function of the ER parameter p

accurately in this scenario.

5.3 Universal compression of tree structures

In this section, we will attempt to find universal compression algorithms for the models discussed in the previous section. Before doing so, we need to define what we mean by universality and optimality in the context of these methods.

Optimality: In the previous section, we calculated the entropies of different random tree models. It is shown by Shannon [29] that the entropy of a random variable provides a lower bound on the average code length that we can use to compress that random variable. Therefore, we are looking for compression algorithms whose average codeword length is close enough to the entropy of the random tree source at hand.

Universality: As seen in the previous section, each of the introduced models for generating random trees has specific parameters. For instance, Simply Gener-

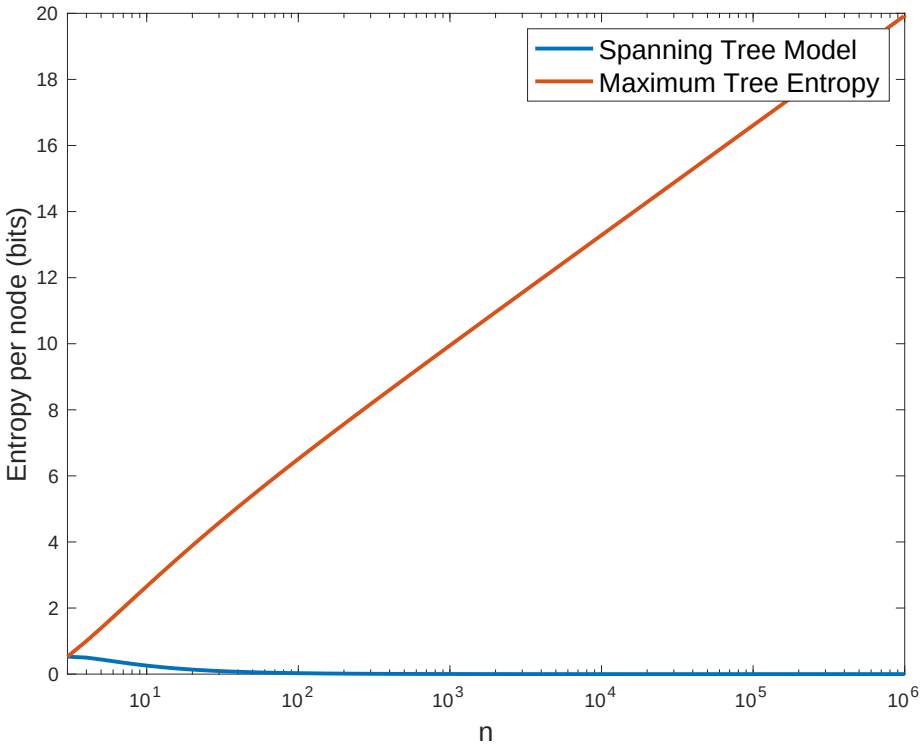


Figure 5.5. Entropy per node upper bound and maximum entropy per node for ER spanning trees as a function of node number n in the giant component regime threshold

ated Trees have their respective children distribution as their parameter. By looking for universal compression algorithms for specific families of random trees, we are essentially looking for compression algorithms that perform optimally regardless of the model parameter. For example, if we develop a universal compression algorithm for SGTs, we want it to be able to optimally compress any SGT, regardless of the respective children distribution.

The idea of all existing universal compression algorithms is that as the parameter of the distribution is unknown, the distribution needs to be somehow learned from the data. For instance, the renowned family of Lempel-Ziv (LZ) [139] compression algorithms, uses dictionaries to store and learn the most common patterns that can happen for the random variable at hand. However, this demands having a sequence of the random variable, so that the most common patterns can be learned. Whereas, in this chapter, we focus on compressing single large trees, rather than a sequence

of them. In this case, optimality translates into the average codeword length for all possible trees to be close enough to the entropy of the source. Based on this, we give the following metrics for the optimality of the compression algorithm for different tree sources.

Sources with a fixed number of nodes: If we use $E_{L,n}$ to show the expected codeword length for trees with n nodes and H_n to show the entropy of the tree source for trees with n nodes, our goal is for the compressor to satisfy

$$\lim_{n\to\infty} E_{L,n} = \lim_{n\to\infty} H_n, \tag{5.42}$$

to ensure that our compression algorithm is asymptotically optimal on single trees when n and consequently the need for compression grow.

Sources with no bound on their number of nodes: If we use $L(t)$ to show the codeword length of the compressed tree, our goal is for the compressor to satisfy the following condition.

$$\mathbb{E}[L(t)] = H_T + C, \tag{5.43}$$

where the expectation is taken over all possible trees in source T, and C is a constant with respect to $|T|$.

In order to achieve the conditions in Eq. 5.42 or Eq. 5.43 based on the model, and still have a universal compression algorithm, our main approach will be to decompose each single tree into a sequence of other random variables, and then apply existing universal compression algorithms to those sequences.

Generally speaking, there are numerous universal compression algorithms. The most famous of these algorithms are those designed by Lempel and Ziv in two papers published in 1977 and 1978, which are known as LZ77 [139] and LZ78 [140], respectively. These algorithms are proven to be asymptotically optimal for stationary and ergodic stochastic processes [57, Ch. 13]. Therefore, if a stochastic process

of trees can be considered a stationary and ergodic process, then Lempel-Ziv algorithms can simply be applied to it to get an optimal compression. Another powerful variation of this family of compression algorithms is the Lempel-Ziv-Welch (LZW) algorithm [141], which also satisfies the same conditions. In the following sections, instead of using the term universal compression algorithm, we simply mention LZ. It must be noted by the reader that LZ can simply be replaced by any version of LZ, as well as any other universal compression algorithm that is able to optimally compress stationary and ergodic stochastic processes.

We first introduce a coding algorithm for trees generated from a uniform source. We then continue to introduce a universal compression algorithm for Simply Generated Trees, and then move on to Erdős–Rényi random spanning trees.

5.3.1 Compressing uniform tree sources

In this section, we start by proposing two simple, yet effective, tree coding algorithms called *Pit-Climbing* (PC) and *Tunnel-Digging* (TD).

We will then move on to introduce TreeExplorer, which is a combination of PC and TD.

5.3.1.1 Pit-climbing algorithm

In this section, we introduce a novel tree structure coding algorithm that we call Pit-Climbing. We use this term because of the analogy between the proposed method, and a climber that has been trapped in a pit and wants to climb up.

Ternary pit-climbing algorithm (TPC): We start traversing the tree from the leftmost leaf. We log our tree traversal using three symbols: $\uparrow$, $\Uparrow$ and $\downarrow$. Anytime that we are at a leaf, we take the only possible path, which is upwards. If we take an upward path at any point from an edge, we consider that edge and the subtree below it as deleted (or filled-in) from the original tree so that we do not explore it

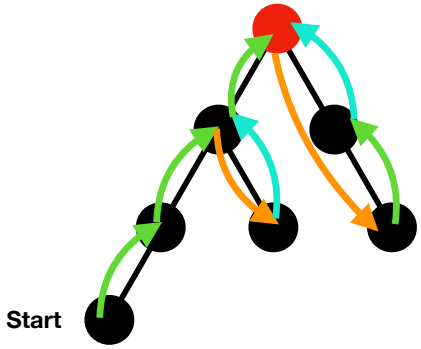


Figure 5.6. Running TPC on a sample tree

again. Additionally, we log this upward movement in our code. If we have moved to a node that we have never been to before, we log a $\uparrow$ in the code. Otherwise, if we move upwards to a node that we have seen before, we log it with a $\Uparrow$. When we reach a node that is not a leaf, we look at the leaves of the rooted subtree whose root is the node we are currently at. We then take the path downwards that falls into the leftmost leaf of that subtree. We log this entire fall with a single $\downarrow$. We continue exploring the tree and logging the code in the same manner until we reach the root of the tree and there is no other edge to fall into.

We will clarify TPC with the following example.

Example 5.3.1. Assume that we are given the rooted tree structure of Fig. 5.6, where the red node shows the root. The starting point of the algorithm is indicated, and the arrows show the path that PC takes. The orange, green, and blue arrows are used to show $\downarrow$, $\uparrow$, and $\Uparrow$, respectively.

We now move on to the following theorem which states that ternary pit-climbing uniquely codes a tree structure.

Theorem 5.3.1. *Ternary pit-climbing algorithm provides a uniquely decodable code for rooted tree structures.*

Proof. We prove this theorem using induction on the depth of the tree structure.

Firstly, it is clear that the code for a tree structure with one node (the root) is $\emptyset$. Then, assume that we know that the ternary pit-climbing code is uniquely decodable for all rooted tree structures with a depth of k or less. Now suppose that we have a rooted tree structure of depth $k+1$. The way that pit-climbing codes this tree is that it describes how to build each of the subtrees of the children of the root, and connects them together using some connector codes. Therefore, if we show that the code for each of these subtrees and the connectors are uniquely decodable, then we can conclude that the code for the whole tree is also uniquely decodable. We already know this for all of the subtrees rooted at the children of the root, as they have a depth of at most k. Additionally, we know that the code of these subtrees are connected together with a staple of $\uparrow\downarrow$ if for connecting the first two subtrees, and $\Uparrow\downarrow$ for connecting the rest. This connector code basically goes up one node (or creates it if it doesn't exist), and then falls down to the leftmost leaf of the next subtree to be explored. Therefore, the ultimate code will also be uniquely decodable. □

In source coding, usually a binary code is preferred over a ternary code, as most of our systems for storage and communication are binary-based. To transform TPC codes into binary, we look back at the definition of the symbols. We observe that we can never have consecutive $\downarrow$s. This is because whenever we fall, we fall down to a leaf, so we can never fall twice or more. We make use of this fact, and assign 0 to $\downarrow$, and 00 to $\Uparrow$. We also use 1 to represent $\uparrow$. We call this new binary code for rooted tree structures simply *pit-climbing* (PC). Even though this method of coding does not provide us with an instantaneous code, we claim that PC creates uniquely decodable codes. Theorem 5.3.2 proves this statement.

Theorem 5.3.2. *Codes generated by pit-climbing are uniquely decodable.*

Proof. We use induction on the depth of the tree. Firstly, the code for a tree with a single node is uniquely decodable ($\emptyset$). Next, assume that we know PC codes for

all trees with a depth of k or less are uniquely decodable. For a tree with a depth of $k+1$, we look at the subtrees of the children of the root. Based on the induction, we know that the PC code for all these subtrees are uniquely decodable. The PC code for the original tree is the concatenation of the PC codes of the subtrees, with a connector of $\uparrow\downarrow\equiv 10$ for the first two subtrees, and $\Uparrow\downarrow\equiv 000$ for all other subtrees. In case the root only has one child, the final code will be the code of the subtree rooted at the child, plus an additional $\uparrow\equiv 1$. Therefore, in all of the cases, the PC code of the tree with a depth of $k+1$ can be uniquely decoded. □

Furthermore, we would like to investigate the length of the codewords generated by PC.

Lemma 5.3.1. *For each $\downarrow$ in the TPC code, there is a unique $\Uparrow$ that comes after it and corresponds to that $\downarrow$. Conversely, for each $\Uparrow$, there is a unique $\downarrow$ that comes before it and corresponds to that $\Uparrow$.*

Proof. Any time we fall down from a node, we always come back up to it. This is because the whole tree needs to be traversed, and we always finish at the root. Therefore, for each $\downarrow$ from a node, there is a $\Uparrow$ later on for coming back up to that node. The same discussion can also be used for $\Uparrow$s. □

Theorem 5.3.3. *The PC codeword length for a rooted tree structure with n nodes and l leaves is $n+2l-3$ bits.*

Proof. Firstly, notice that each $\downarrow$ in the TPC code corresponds to a leaf, as we always fall into a leaf. Additionally, we fall into every leaf except for the one we start the algorithm from exactly once. Therefore, we have $l-1$ $\downarrow$s in the TPC code, which translates into $l-1$ bits in the PC code. Additionally, we climb up each edge of the tree exactly once. Therefore, the number of $\uparrow$s and $\Uparrow$s in the TPC code is equal to the number of edges, which is $n-1$. However, every $\Uparrow$ will translate into 2 bits in the PC code. The number of $\Uparrow$s is equal to the number of $\downarrow$s, as anytime we fall from

a node we will have to climb back up to it at some point. Therefore, the number of $\Uparrow$s is also $l-1$, and we will have $l-1$ additional bits when translating the TPC code into a PC code. Consequently, the total number of bits in the PC code will be $l-1+n-1+l-1=n+2l-3$. □

5.3.1.2 Tunnel-digging algorithm

Based on Theorem 5.3.3, the PC code length will increase with the number of leaves. However, the number of rooted trees with l leaves does not necessarily increase with l. Hence, there is no justified reason for having longer codes for trees with more leaves. As a result, another algorithm called *tunnel-digging* is developed to tackle this problem. The PC algorithm is based on traversing a tree along its edges, up and down. Whereas, in TD, the aim is to traverse the tree in a horizontal manner. The name *tunnel-digging* comes from seeing the traversal method of this algorithm as digging tunnels between nodes on the same depth.

Ternary tunnel-digging algorithm (TTD): We start with the leftmost child of the root, and start moving right to nodes with the same depth in the order of the nodes. For each node that we encounter, we log a $\leftarrow$ if it is a leaf, and a $\rightarrow$ otherwise. If at any point we have to move between two nodes that are not siblings, we use a $\Rightarrow$ to show the transition (digging a tunnel!). Additionally, if at any point there are no more nodes on the right to move to, we move to the leftmost node on the level below, and we mark this transition again with a $\Rightarrow$. We continue until all the leaves of the tree are logged in the code.

The following example illustrates running TTD on a sample rooted tree structure.

Example 5.3.2. Assume that we are given the rooted tree structure of Fig. 5.7, where the red node shows the root. The starting point of the algorithm is indicated, and the arrows show the path that TTD takes. The blue, green, and orange arrows are used to show $\leftarrow$, $\rightarrow$, and $\Rightarrow$, respectively.

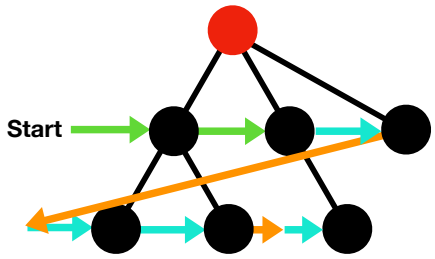


Figure 5.7. Running TTD on a sample tree

To transform the TTD code into a binary code which will be called tunnel-digging (TD), we again use the properties of the TTD symbols. We notice that we can never have two consecutive ⇒s, as there is always at least one node in between the dug tunnels. Therefore, we use 0 to represent ⇒ and 00 to represent →. Additionally, we use 1 to show ←. Notice that we use a shorter code for leaf nodes, as the number of leaf nodes is expected to be higher than the number of non-leaf nodes when TD is used to code the tree. The proof that the code is uniquely decodable can be done in the same manner as Theorem 5.3.2 by replacing ↓, ↑, and ⇑ with →, ←, and ⇒, respectively. The following Theorem calculates the code length of tunnel-digging.

Theorem 5.3.4. *The TD codeword length for a rooted tree structure with n nodes and l leaves is $3n - 2l - 3$ bits.*

Proof. The number of →s and ←s used in the code is exactly equal to the total number of nodes minus the root. However, we use two bits for each →, which shows non-leaf nodes. Therefore, the →s and ←s use $2(n-1)-l$ bits in total. Additionally, for every node that has at least one child (except for the root), we will have a ⇒ in the code. This would be equal to $n-1-l$ bits. Thus, we will have $3n-2l-3$ bits in total. □

5.3.1.3 TreeExplorer

To see in which scenarios TD performs better than PC we can compare their codeword lengths and write

$$3n - 2l - 3 < n + 2l - 3 \Rightarrow l > n/2. \tag{5.44}$$

Based on Eq. 5.44, PC works better when the number of leaves is less than $n/2$, and TD works better otherwise. They exhibit the same performance when the tree has exactly $n/2$ leaves. Based on this, we propose the following coding technique for rooted tree structures.

TreeExplorer: Firstly, the number of leaves of the rooted tree structure is counted (l). If $l < n/2$, the structure is coded with PC. The code is then prefixed with a 0 to specify that it has been coded using PC. Otherwise, the structure is coded using TD, and the code is prefixed with a 1.

It can easily be shown that the codes created using TreeExplorer are uniquely decodable. This is because the first bit of the code uniquely determines the coding method, and we have already proven that both PC and TD are uniquely decodable. The following theorem shows the upper bound for the average code length of TreeExplorer.

Theorem 5.3.5. *For any probability distribution on rooted tree structures with n nodes, the average code length of TreeExplorer is less than $2n - 2$.*

Proof. Assume that the probability of the number of leaves (l) being less than $n/2$ is q. If L is the length of the code produced using TreeExplorer, we can write

$$\begin{aligned}
\mathbb{E}[L] &= q \mathop{\mathbb{E}}_{l<n/2}[n + 2l - 2] + (1 - q) \mathop{\mathbb{E}}_{l\geq n/2}[3n - 2l - 2] \\
&< q(2n - 2) + (1 - q)(2n - 2) \\
&= 2n - 2.
\end{aligned}$$

□

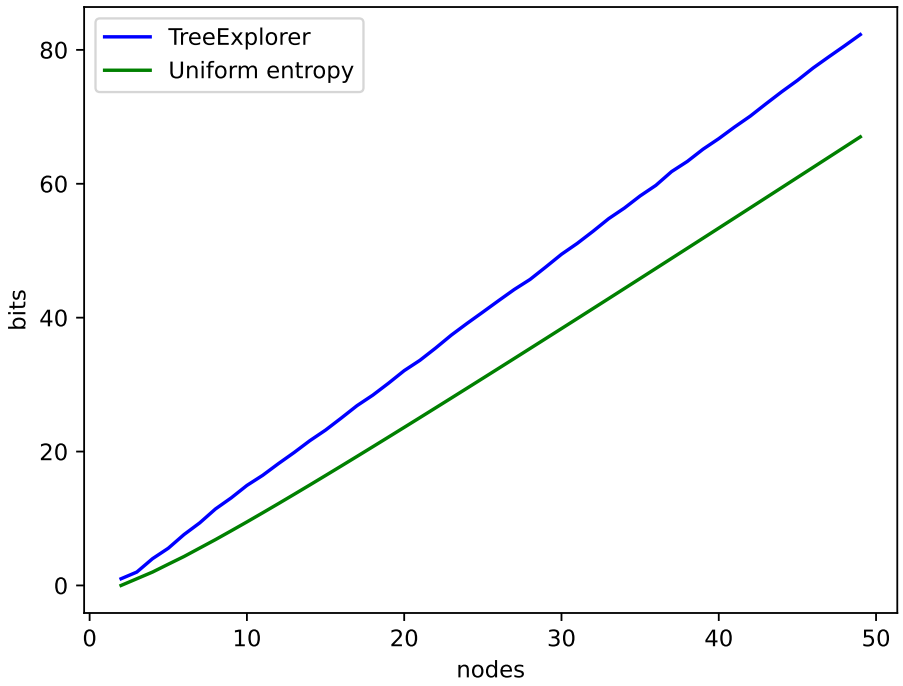
5.3.1.4 Comparison with entropy

We compare the average code length of TreeExplorer with the entropy of the uniform source, which was calculated in Section 5.2.1. The result is plotted in Fig. 5.8. Fig. 5.8a shows the average bit length, and Fig. 5.8b shows its average rate of change. The performance of TreeExplorer is observed to be very close to the entropy of the source, which is the optimal compression limit.

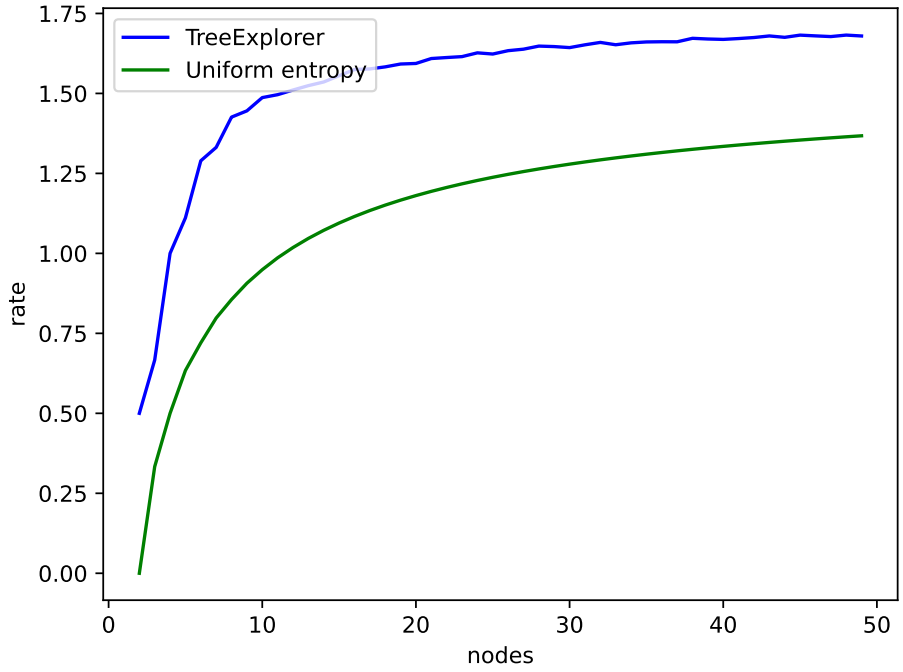
As the size of the trees becomes large, achieving zero redundancy with a compression algorithm becomes nearly impossible in practice. This is because of the fact that the increased complexity and storage requirements of larger sample spaces make it challenging to study and analyse each sample in order to find patterns for compression. This becomes more evident when taking into account the exponential growth of the sample space with n. TreeExplorer offers a practical compromise by being very easy to implement even for large trees while still being close to the theoretical limit of compression.

5.3.1.5 Comparison with adjacency list

The adjacency list representation is one of the most widely used methods for storing trees. This method uses $2n\lceil \log n \rceil$ bits to represent a tree. Fig. 5.9 compares the performance of adjacency list with TreeExplorer. This figure compares both the average codeword length, and the rate of change. Notice that for coding a labelled tree using TreeExplorer, an additional $n\lceil \log n \rceil$ bits are needed to list the node labels in the order of their appearance.

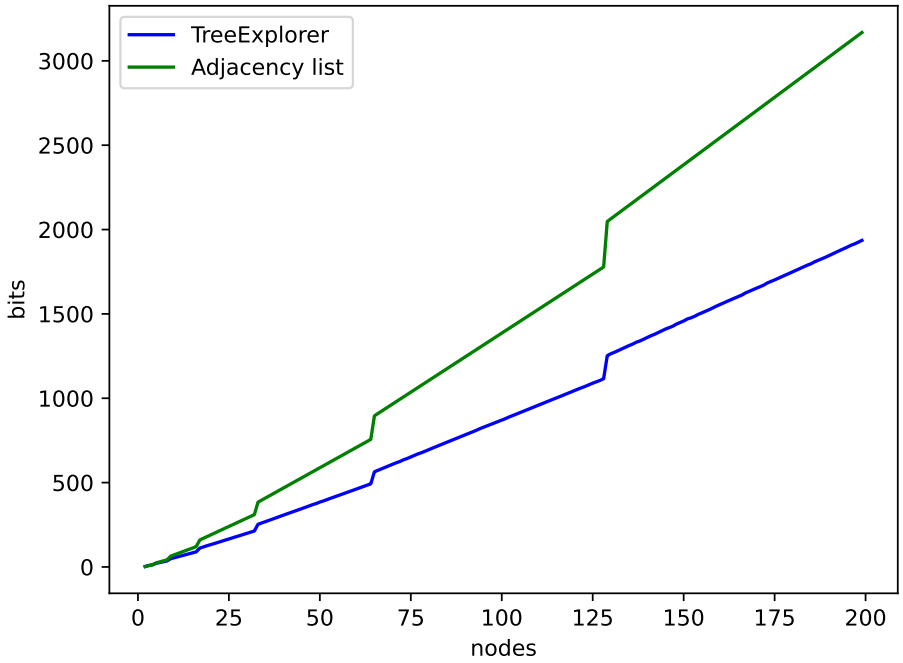


(a) Average length

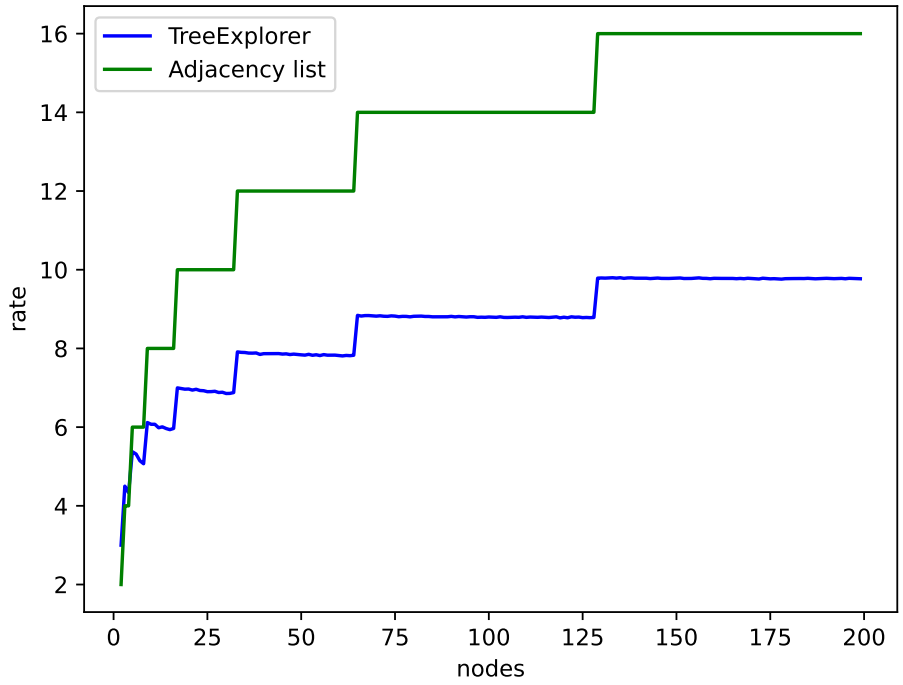


(b) Average rate of change

Figure 5.8. Comparing the performance of TreeExplorer on a uniform unlabelled unordered rooted tree source with its entropy



(a) Average length



(b) Average rate of change

Figure 5.9. Comparing the performance of TreeExplorer with adjacency list

5.3.1.6 The Newick format

The Newick format has been the standard for representing phylogenetic trees since its introduction back in 1986 [142]. In this method, trees are represented using parentheses and commas. This format starts from the root of the tree, and lists the children and subtrees of the root in a nested manner. We will not go into further details on how this method works. However, because of the similarities between this format and TreeExplorer, we compare the performance of these two methods.

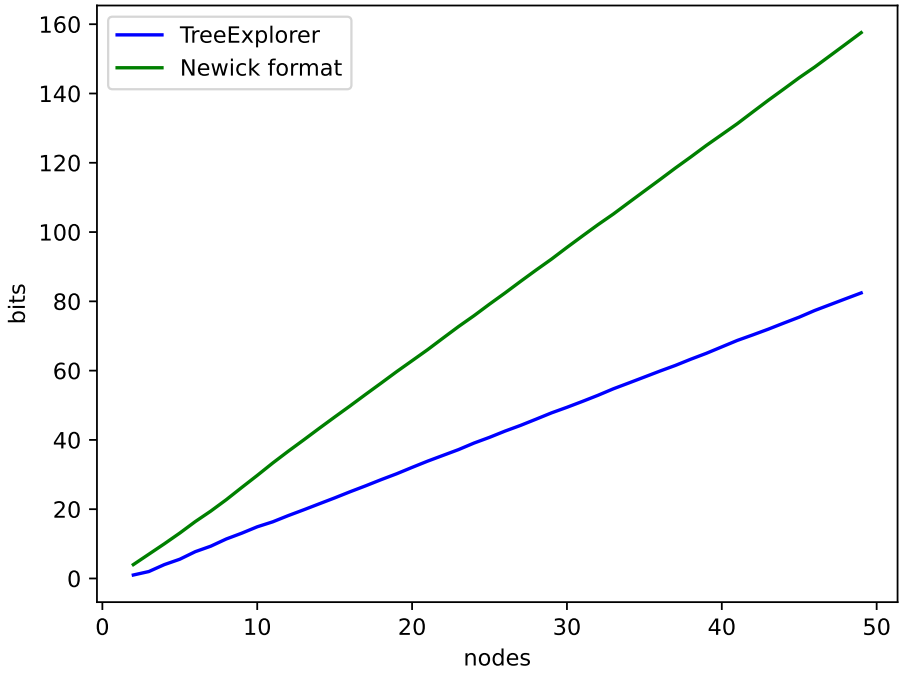
Our calculations show an additional $n+1$ bits and $3n-4l+1$ bits in the Newick format compared to when TreeExplorer uses TD and PC, respectively. Fig. 5.10 compares the average codeword lengths and rate of change of TreeExplorer and the Newick format for rooted trees of up to 50 nodes. The figure also confirms that TreeExplorer provides us with a shorter codeword length.

5.3.1.7 Other succinct tree coding methods

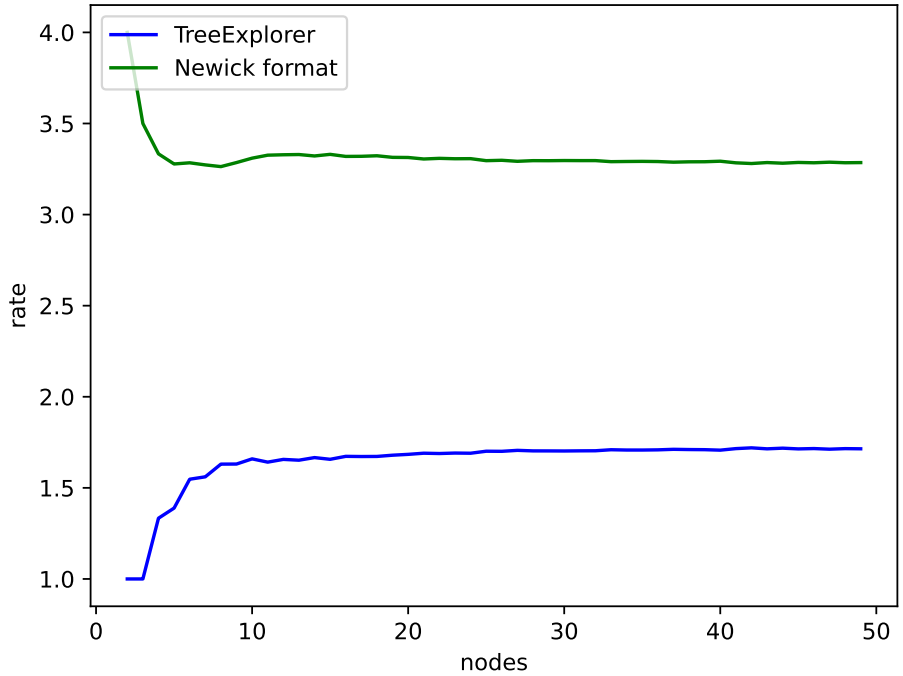
Two of the most well-known succinct representation methods for ordered trees are balanced parentheses (BP) [143], and depth-first unary degree sequence (DFUDS) [144]. These two algorithms are widely used as efficient tree coding methods. It is well known that the codeword length of both of these methods for an n node tree is $2n+O(n)$ bits [108]. Therefore, as TreeExplorer has a maximum length of $2n-2$ bits for an n node tree, it is performing better than both BP and DFUDS.

5.3.2 Universal compression of SGTs

In this section we look for a universal compression algorithm for SGTs. In accordance with what we discussed in the previous section, we will be looking for a way to decompose SGTs into a sequence of random variables that can be seen as a stationary and ergodic stochastic process. This will then allow us to simply run a universal compression algorithm on this sequence, and study its performance. We propose



(a) Average length



(b) Average rate of change

Figure 5.10. Comparing the performance of TreeExplorer with the Newick format

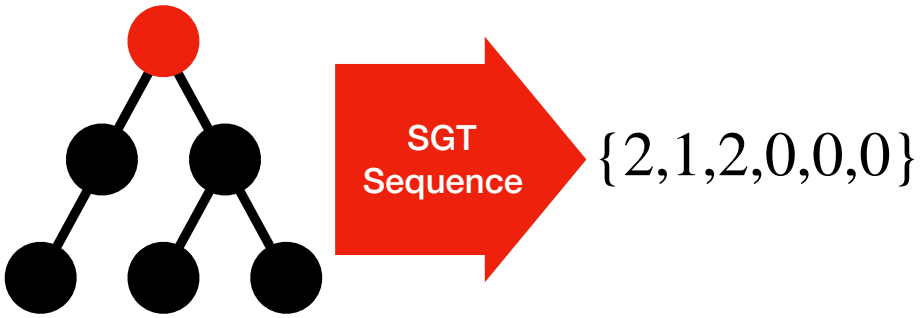


Figure 5.11. An example SGT and its corresponding SGT sequence

the following definition for an SGT sequence, which provides a method to create a sequence from a given SGT.

Definition 5.3.1 (SGT sequence)**.** For a given SGT, its unique SGT sequence is defined as follows. We start from the root, and add the number of its children as the first random variable in the sequence. We then move on to its leftmost child, and add its number of children to the sequence. We continue traversing the tree in a breadth-first manner [145, Ch. 22.2], and keep adding the number of children of each node that we see to the sequence. When we are finished exploring the tree and have added the number of children of all the nodes, the SGT sequence is done.

Fig. 5.11 shows an SGT alongside its SGT sequence. It can easily be shown that an SGT sequence creates a one-to-one mapping between the trees and the sequences, and the tree is fully recoverable given the SGT sequence. We use the notation $f_{SGT}(t)$ to show the SGT sequence of an SGT. Additionally, note that the tree traversal can generally be done in any way desirable. In this definition, we have used a Breadth-First Search (BFS). However, any other traversal algorithm such as Depth-First Search [145, Ch. 22.3] can also be used. This is of course as long as the same method is used when translating the sequence back into a tree.

Based on the definition of SGTs, the number of children for each node is chosen in an i.i.d manner from the same children distribution. Therefore, the SGT sequence of a tree is a sequence of n i.i.d random variables, with n being the number of children

of the tree. As i.i.d sequences are both stationary and ergodic, LZ can be used to compress SGT sequences.

First, let us consider a sequence of independently generated SGTs $\{t_1, t_2, \ldots, t_n\}$. If we create a new sequence $\{f_{SGT}(t_1), f_{SGT}(t_2), \ldots, f_{SGT}(t_n)\}$, the tree sequence can be fully recovered having the SGT sequence. This is because each SGT is uniquely coded using its respective SGT sequence, and the boundaries between consecutive SGTs is always known as the leafs are always marked by having zero children in an SGT. Therefore, the sequence $\{f_{SGT}(t_1), f_{SGT}(t_2), \ldots, f_{SGT}(t_n)\}$ is a sequence of i.i.d random variables, and can be compressed optimally using LZ as $n \to \infty$. Therefore, the proposed compression algorithm is both universal and optimal for a sequence of SGTs from the same family.

After having considered sequences of trees, we now move on to study the performance of the proposed compression algorithm on single SGTs. For a single tree t, the length of $f_{SGT}(t)$ will also be limited. Therefore, there will be an inevitable redundancy in the LZ code of $f_{SGT}(t)$ as LZ only guarantees optimality when the length of the sequence goes to infinity. We will try to quantify this redundancy in the remainder of this section. To this end, we will use the upper bounds on the redundancy of LZ algorithms, calculated by Savari [146]. We will focus on the LZW algorithm for now, but the analysis for other members of the LZ family is very similar. Let $R(t)$ show the redundancy per symbol when compressing $f_{SGT}(t)$ using LZW, $\mathscr{R}$ to show the average redundancy per symbol for all the possible trees using the SGT model at hand, and $I(t)$ to show the information in t as per Shannon's definition of information [29]. [147] provides an upper bound on the redundancy of a finite sequence compressed using LZW, which we use to get the following bound on the redundancy for one single tree t with n nodes (and therefore n symbols in its SGT sequence).

$$R(t) \leq \frac{I(t)}{n \ln n} \log\left(\frac{C \log e}{H_C}\right) + O\left(\frac{1}{\ln n}\right) \tag{5.45}$$

To find an upper bound on $\mathscr{R}$, we need to take an average of Eq. 5.45 over all possible trees in the SGT model. To this end, we first note that for a tree t with n nodes, we have $I(t) \leq \log n$. If we use K to show $\frac{1}{\ln 2} \log\left(\frac{C \log e}{H_C}\right)$, we can write the following upper bound on $R(t)$ based on Eq. 5.45.

$$R(t) \leq \frac{K}{n} + O\left(\frac{1}{\ln n}\right) \tag{5.46}$$

We now need to take the average of Eq. 5.46 over all possible number of nodes in the SGT model. For this purpose, we need to calculate the probability of a tree having exactly n nodes. This is challenging, and might not be possible. Therefore, we again find an upper bound to it. If we use p_0 to show $p_C(0)$, and use N as the random variable for the number of nodes in the trees generated by the SGT model, we can write the following equation.

$$p(N = n) \leq (1 - p_0)^{n-1} \tag{5.47}$$

The upper bound in Eq. 5.47 comes from the fact that in order to have n nodes, we can have n nodes to have at least one child. Additionally, we know that for $n = 1$, the probability is exactly p_0, and $\log C$ bits are used to code it in LZW. We also use the lower bound $1 - 1/x$ to $\ln x$ in order to make $O(1/\ln x)$ calculable. Having this in mind and combining equation 5.46 and 5.47 gives us the following upper bound on $\mathscr{R}$.

$$\mathscr{R} \leq p_0 \log C + \sum_{i=2}^{\infty} p(N = i) \left(\frac{K}{i} + O\left(\frac{1}{\ln i}\right)\right) \tag{5.48a}$$

$$\leq p_0 \log C + \sum_{i=2}^{\infty} (1 - p_0)^i \left(\frac{K}{i} + \frac{M}{1 - 1/i}\right) \tag{5.48b}$$

$$= K(p_0 - \log p_0 - 1) - M \log p_0 + \frac{M}{p_0} \tag{5.48c}$$

$$+ p_0(M + M \log p_0 + \log C) - 2M,$$

where M is a positive real number that comes from the definition of O.

Eq. 5.48c provides us with an upper bound on the average redundancy of using the proposed compression algorithm on a single SGT. This redundancy depends on a number of factors, such as p_0, H_C, and possible number of children in the SGT model. As this method only compresses single trees, we can not expect the redundancy to be zero, as zero redundancy in universal compression algorithms only happens asymptotically as the number of samples tends to infinity. However, the found upper bound gives an acceptable redundancy for many SGT models. Specifically, the redundancy matches the conditions of optimality defined by Eq. 5.43.

It was seen that the way the number of the children of the nodes is listed in this compression algorithm is based on a DFS traversal on the tree nodes. However, notice that as long as it is known to both the encoder and the decoder how the tree was traversed, it does not matter which tree traversal method is used. This is because the number of children of nodes are i.i.d. random variables, and therefore their order does not affect the optimality of the LZ algorithm. For instance, Breadth-First Search, TreeExplorer, or any other tree traversal algorithm that explores all the nodes of the tree could have also been used and we would still get the same results.

5.3.3 Universal compression of the Erdős–Rényi spanning tree model

In this section, we will examine a similar approach to the previous section in order to compress trees generated using the Erdős–Rényi (ER) Spanning Tree Model. Our first aim is to therefore see if we can represent each tree using a sequence of random variables that satisfy stationarity and ergodicity, while using the properties of the way these trees have been generated.

Based on their definition, ER spanning trees are labelled. Additionally, the number of nodes in the graph and the tree must be predetermined (n). We propose the following approach for coding ER spanning trees. We divide our coding process into two steps: extracting certain bits from the adjacency matrix of the tree, and then compressing the extracted sequence of bits. We first start by describing the bit extraction process.

Bit extraction: We start by looking at the adjacency matrix of the tree, starting with the row corresponding to the connections of node 1. This row consists of $n-1$ bits $(a_{1,2}, a_{1,3} \ldots, a_{1,n})$, where each bit represents the existence of a connection between node 1 and the other nodes in the tree. We take all these bits during the extraction process. After this, we know all the connections of node 1 in the tree. Let us show the number of these connections with random variables c_1. Each pair among these c_1 edges removes the possibility of having one other edge in the tree. For instance, if node 1 is connected to both nodes i and j, there can not be a connection between i and j in the tree. Therefore, having the connections of node 1 removes the need for including $\binom{c_1}{2}$ bits in the adjacency matrix of the tree, and we will know exactly which ones. After having described the connections of node 1, we go to the nodes to which node 1 is connected in the order of their labels. We continue by writing down the rows of the adjacency matrix corresponding to these rows, without the connections that have been covered before or whose state is known due to the described edge elimination process. This way, the second node requires less bits than the first due to the impossibility of this node being connected to other neighbours of node 1. We can carry on this way and explain the remaining connections of each node, until all the nodes in the tree are covered. Note that bit extraction can be applied to any simple graph, and not just ER graphs. We show the process of bit extraction from a tree t with $f(t)$.

After having extracted certain bits of the adjacency matrix using f, we simply

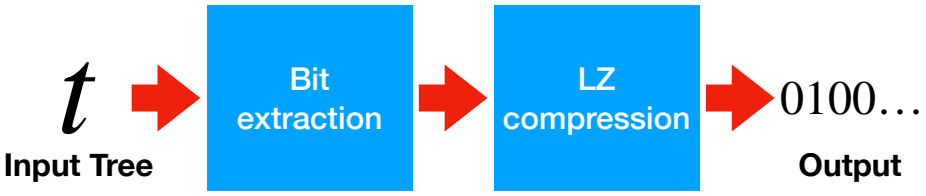


Figure 5.12. Proposed algorithm for compressing ER Spanning Trees

feed them into a universal compression algorithm, such as the Lempel-Ziv-Welch algorithm [141]. Fig. 5.12 summarizes our proposed compression technique.

We will now show that the redundancy of the proposed compression algorithm tends to zero as the trees grow large. We will perform the calculations for the case where LZ78 is used as the universal compression. However, the result is similar for other compression algorithms in the LZ family. To this end, we state and prove the following theorem.

Theorem 5.3.6. *The redundancy of the proposed compression algorithm for trees created using the ER spanning tree model tends to zero as the number of nodes of the tree n grow large. The algorithm is also universal in the sense that it does not depend on the value of the ER parameter p.*

Proof. It is shown in [148] that for a binary sequence of length l, the redundancy of LZ78, which we show with $\mathscr{R}$, satisfies the following inequality.

$$\mathscr{R} \leq \frac{\ln \ln l}{\ln l} + O\left(\frac{1}{\ln l}\right) \tag{5.49}$$

Therefore, if we use $L(t)$ to show the length of $f(t)$, we can use the following inequality to find the average redundancy of the proposed compression algorithm.

$$\mathbb{E}\left[\mathscr{R}\right] \leq \mathbb{E}\left[\frac{\ln \ln L(t)}{\ln L(t)}\right] + \mathbb{E}\left[O\left(\frac{1}{\ln L(t)}\right)\right] \tag{5.50}$$

As $\ln \ln x / \ln x$ is a concave function, we can use Jensen's inequality [137] for the

first term in Eq. 5.50 and write

$$\mathbb{E}\left[\frac{\ln\ln L(t)}{\ln L(t)}\right] \leq \frac{\ln\ln \mathbb{E}[L(t)]}{\ln \mathbb{E}[L(t)]}. \tag{5.51}$$

Based on Eq. 5.51, we need to calculate $\mathbb{E}[L(t)]$. Note that the bit extraction process induces an order on the nodes of tree based on the traversal order. Looking closely, this is simply a Breadth-First Search traversal of the tree, by choosing node 1 as the root. Let us show the number of connections left to be described for the ith node in this sequence using A_i. Firstly, we know that $A_1 = n - 1$. Going to each new node, the need for describing the connections to all the nodes that came before it and all their neighbours is removed. As in each step, we do not have any prior information about the connections to the remaining nodes, each bit can be considered an independent Bernoulli process just like in the original graph. Therefore, for $i > 1$ the expected value of unknown connections is $n - 1$, minus the $i - 1$ node that have come before and their expected number of edges, which is simply p times their number of expected connections. However, we must take into account that this way we are double counting all the prior nodes except for node 1, and this needs a correction term. Based on these reasons, we can write the following recursive equations for the expected values of A_is.

$$\begin{cases} \mathbb{E}[A_1] = n - 1 \\ \mathbb{E}[A_i] = n - i - p\sum_{j=1}^{i-1}\mathbb{E}[A_j] + (i-2)p, \quad i > 1 \end{cases} \tag{5.52}$$

Eq. 5.52 will result in the following recursive equation for $\mathbb{E}[A_i]$ for $i > 1$.

$$\begin{cases} \mathbb{E}[A_1] = n - 1 \\ \mathbb{E}[A_i] = (1-p)\mathbb{E}[A_{i-1}] - 1 + p, \quad i > 1 \end{cases} \tag{5.53}$$

Solving Eq. 5.53 gives us the following solution.

$$\mathbb{E}\left[A_i\right] = \frac{(p-1)^2 - ((n-2)p+1)\,(1-p)^i}{p(p-1)} \tag{5.54}$$

Eq. 5.54 gives us the following result for the expected number of bits to code.

$$\sum_{i=1}^{n} \mathbb{E}[A_i] = \frac{\left(np^2 - (1-p)^n - p\left(n(1-p)^n - 2(1-p)^n + 2\right) + 1\right)}{p^2} \tag{5.55}$$

If we use $h(n)$ to show the term calculated in Eq. 5.55, we will only need to code a maximum of $h(n)$ bits from the adjacency matrix of the tree on average. We can write the following equation by inserting Eq. 5.55 into Eq. 5.51.

$$\mathbb{E}\left[\frac{\ln\ln L(t)}{\ln L(t)}\right] \leq \frac{\ln\ln h(n)}{\ln h(n)} \tag{5.56}$$

For the $\mathbb{E}\left[O\left(\frac{1}{\ln L(t)}\right)\right]$ term in Eq. 5.50, we can simply replace it with a coefficient of $\mathbb{E}\left[\frac{\ln\ln L(t)}{\ln L(t)}\right]$ and the inequality will still hold. Therefore, we will have the following upper bound on the average redundancy of the compression algorithm.

$$\mathbb{E}[\mathscr{R}] \leq K\frac{\ln\ln h(n)}{\ln h(n)}, \tag{5.57}$$

where K is a constant. We have

$$\lim_{n\to\infty} h(n) = n. \tag{5.58}$$

Therefore, we can write

$$\lim_{n\to\infty} \mathbb{E}[\mathscr{R}] = \lim_{n\to\infty} K\frac{\ln\ln h(n)}{\ln h(n)} = 0. \tag{5.59}$$

Eq. 5.59 shows that the redundancy tends towards zero as n grows large. It tends towards zero regardless of the value of p and the way that the random spanning trees were chosen. Therefore, the proposed method is universal in the sense that it does

not depend on the ER parameter or the random tree selection process. □

An interesting observation is that the proposed compression algorithm can be further generalized by generalizing the bit extraction process. For instance, the process does not necessarily need to start from node 1 as mentioned in the bit extraction procedure. Additionally, other traversal methods such as a DFS traversal would have worked too. The reason for that is the fact that as long as we know the ordering on how the nodes have been traversed, we can reconstruct the respective parts of the adjacency matrix and recover the graph. This gives us the freedom to choose a traversal method that suits the needs of our application better. This can help us in designing a query-preserving tree coding algorithm. In other words, the tree traversal algorithm which determines the ordering of the nodes can be chosen in a way that is able to provide answers to queries of a particular scenario. If we look closely at the proof Theorem 5.3.6, the only importance of the bit extraction process is that it induces on order on the nodes, and then explores the rows of the adjacency matrix according to that order. Therefore, any traversal on the tree that induces an ordering on all the nodes of the tree can be used in the bit extraction process and we would still get the same results on the optimality of the algorithm. This includes both the traversal algorithm itself, and the choice for the node to start with. Some examples of other algorithms that can be used are any variations of the DFS (Preorder, Inorder, and Postorder) [145, Ch. 6], and the Best-First Search algorithm [149, p. 48], and even PC and TD.

5.4 Summary

In this chapter, we took a novel approach towards the entropy and compression of tree data structures. We started by looking at different random tree sources and analysing their complexity in terms of Shannon entropy. Uniform tree sources and Simply

Generated Trees were studied first as existing models of random tree generation. We then moved on to introduce a new random tree generation algorithm that we call the spanning tree model. It was discussed that this model can simulate many of the scenarios that happen in practice. After the entropy of the general model was formulated, we introduced a subcategory of this model whose underlying network is generated using the ER model. This made us able to quantify the entropy of the the source in terms of the model parameters. Ultimately, having the entropy of each of the studied models, we moved on to the compression domain. Universal compression algorithms were introduced for all of the studied models, and it was proven that the redundancy of these algorithms tends to zero as these trees grow large. Future directions of research can include considering other random graph generators for the spanning tree model, and finding a more general tree compression algorithm that goes beyond the ER model.

Chapter 6

Applications

The pivotal presence of networks in the world, alongside their immense scale and intricate complexity, is undeniable. Additionally, as they are essentially data structures, their storage, transmission, and processing is inevitable in any application they appear in. The complexity analysis and compression of graphs can be used in all these domains and has the potential of increasing the efficiency of existing methods or even introducing new ones.

In this chapter, we examine two possible applications for our research. These two applications are mere examples for the vast potential that measuring the complexity of graphical data structures and their compression holds. We will start with an application for tree compression. For this purpose, we focus on the problem of routing in Wireless Sensor Networks, and discuss the role of trees and tree compression in that context. We then shift our focus to temporal Graph Neural Networks, an area that has recently gained a lot of attention in the machine learning community. It will be discussed why an information-theoretic view on the temporal evolution of graphs can be utilized to improve existing algorithms in this domain, and how NECs can be adjusted to model such networks.

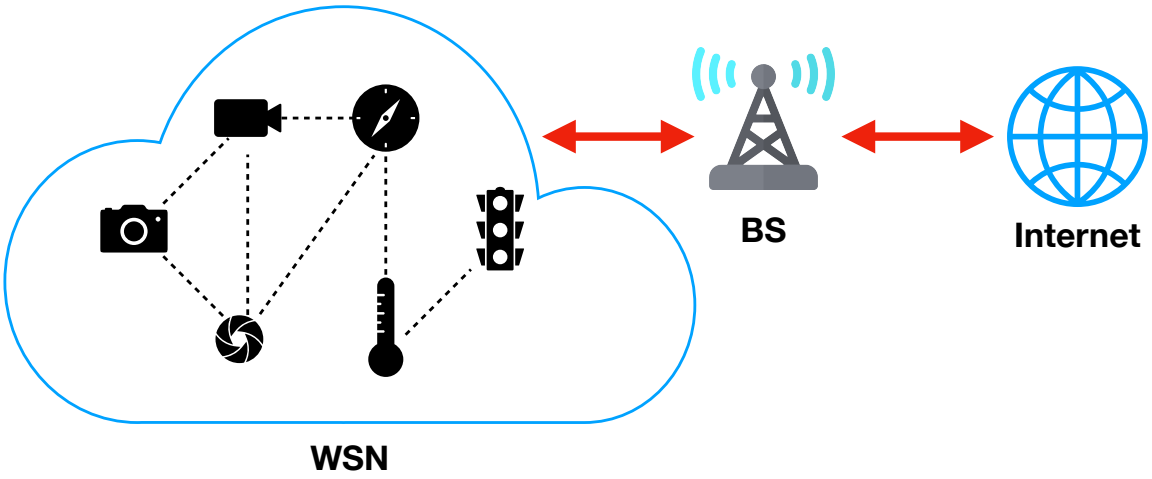


Figure 6.1. An example of a Wireless Sensor Network

6.1 Routing protocols in Wireless Sensor Networks

Wireless Sensor Networks (WSNs) are formed of a number of sensor nodes in an area, which gather information about the environment in which they are located. This information is then collected by a special node in the network, often known as the Base Station (BS) or Sink Node (SN). The BS is the gateway of communicating the information captured by the sensors to the outside world, via the internet. WSNs can vary in size anywhere from very small local networks, to networks with tens of thousands of sensor nodes in some practical scenarios. The sensors can be mobile or stationary. However, they are usually stationary or have very limited movement in practice. It is usually assumed that all the nodes in a WSN, other than the BS, have limited power supply. They are powered with a battery, installed at a location, and barely ever looked after. This is because constant maintenance of these sensors would be costly. Therefore, power consumption of the sensor nodes is one of the main challenges in WSNs. An example of a WSN can be seen in Fig. 6.1.

Routing protocols form an inseparable part of any network, especially WSNs as they demand the least computation and communication power as possible. As a result, designing routing protocols for WSNs is still an ongoing area of research. The

research conducted so far on developing routing protocols for WSNs is very vast. Some of the most notable ones are SPIN [150, 151], Directed Diffusion [152], and ACQUIRE [153]. However, the class of hierarchical routing protocols for WSNs seems to be attracting more attention in the recent years. This category of routing protocols is known to be very energy efficient and scalable [154]. In fact, some sources consider hierarchical routing protocols to be the most energy efficient routing protocols for WSNs [155]. LEACH (Low-Energy Adaptive Clustering Hierarchy) [156] is perhaps the most well-known and widely-used hierarchical routing protocol for WSNs. Examples of other hierarchical protocols include PEGASIS [157], TEEN [158], and HPAR [159].

In this section, we introduce LEAST, a Low-Energy, Adaptive, Scalable, and Tree-based routing protocol for WSNs. The routing paths in LEAST essentially form a rooted tree, with the BS acting as the root node. The structure of the tree is formed based on fairness in dividing the consumed communication power between nodes, while minimizing the distance between hops. The assumptions on the sensor network are described first. Then, LEAST is presented and the different steps of the protocol are described in detail. It will be shown how adjusting different parameters of LEAST can be used to tailor the protocol for different networks. The performance of LEAST is then analysed and compared with LEACH. Afterwards, simulation results of LEAST are exhibited, which show a significant improvement over existing hierarchical routing protocols. Ultimately, we will discuss how tree compression algorithms can be used in the context of the proposed protocol.

6.1.1 Preliminaries and network model

We assume the network to have n sensor nodes, and one Base Station. We also assume all the sensors to have enough transmission power to be able to communicate with any other node in the network if needed. We consider a TDMA/CDMA MAC

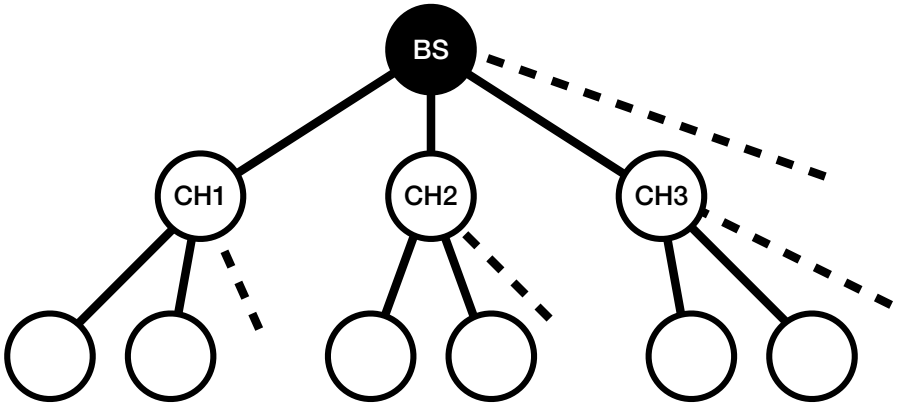


Figure 6.2. LEACH routing map

protocol, such as the one described in [160], to avoid collisions in the communication between nodes. Sensor nodes in the network can start with arbitrary energy levels, as energy levels are not considered in this version of LEAST. It is also assumed that most of the communication in the network will take place to and from the BS. If by any chance communication needs to take place between two sensor nodes in practical applications, they can do so by using the BS as an intermediary node.

For ease of deployment, LEAST is designed to be an extension of LEACH. This way, it will be easy to modify the systems that are currently based on LEACH to operate on LEAST. LEACH is divided into rounds, with each round having two phases: the setup phase and the steady-state phase. In the setup phase, the routing map is formed, and the messages are transmitted using these routes in the steady-state phase. In the setup phase, every node will elect itself as a Cluster Head (CH) with a probability of p_{CH}. The CHs will then be in charge of passing on the messages between the sensor nodes and the BS. Additionally, all the other nodes will need to choose a CH to be connected to. Non-CH nodes will choose the CH with the minimum distance to them as their parent. Fig. 6.2 illustrates how the routing tree looks like in LEACH.

6.1.2 Protocol description

In this section, we describe how LEAST works and discuss its differences with LEACH. Just like LEACH, LEAST is divided into rounds, and each round consists of a setup phase and a steady-state phase. The first round of LEAST will be exactly like that of LEACH, during which nodes will create a three-level routing map for the network. After the initial map is set up by the LEACH algorithm, the other rounds will have a different approach. Unlike LEACH, the routing map in LEAST can have more than three levels. We number the levels based on their distance from the BS: the BS will be level zero, the CHs will be on level one and so on. The steps below describe the operations that need to be done at the setup phase of each round after round one.

6.1.2.1 Selection of host nodes

At each round, the nodes on the first level of the network map (children of the BS) will need to be relocated. These nodes spend a lot of energy managing the packets received from their children, and performing tasks such as data fusion. Therefore, we relocate them to a lower position to keep the balance in energy consumption. For this purpose, a number of nodes are selected as Host Nodes (HN) in each round. The HNs will act as potential future parents for the first-level nodes. However, the actual relocation does not happen until later in the setup phase of the protocol.

Every node elects itself to become an HN for that round with a probability of p_{HN}. The election process will be exactly the same as electing the CHs in the LEACH protocol. Every node that has elected itself as a Host Node notifies the BS about it, which will in turn notify the first-level nodes about the list of HNs. In order to ensure that different nodes act as an HN for the same number of rounds, we use the same formula as LEACH [156, Eq. 3]. Therefore, in round r, a node can only decide to be an HN if it has not acted as an HN in the last np_{HN} rounds.

6.1.2.2 Selection of heir nodes

Before first-level nodes get relocated, we need to choose which nodes are going to replace them. For each first-level node, one or more of its children are going to be chosen as its heir(s). Each of these children are going to be chosen as an heir with a probability of p_H, independent from each other. At the end, if any of the first-level nodes did not have an heir, one of its children will be chosen as its heir in a uniform manner.

6.1.2.3 Relocation of first-level nodes

The final step of the setup phase is to relocate the first-level nodes. We remove first-level nodes from their current location, and connect each one to the HN with the minimum distance from it. Then, the heir nodes are moved up and attached to the BS. The other children of the former first-level nodes then get attached to the closest heir node of their former parent.

Fig. 6.3 summarizes the setup phase of LEAST, at the end of which the map of the network is formed. Fig. 6.4 shows a snapshot of the network map at a random round for a sample network. The tree-structured routing map is formed in a way that is acceptable in terms of node distance, while preserving the random nature of the algorithm to distribute energy consumption fairly among nodes.

6.1.3 Energy consumption analysis

In this section, we will perform a mathematical analysis of the energy consumption of LEAST, and compare it with that of LEACH. For this purpose, we assume the processing power consumed in the sensors to be negligible compared to that used for wireless communication. Additionally, we make use of the fact that energy required for wireless transmission between two nodes with a distance of d is proportional to the number of packets communicated, and to d^2. Therefore, we will try to estimate

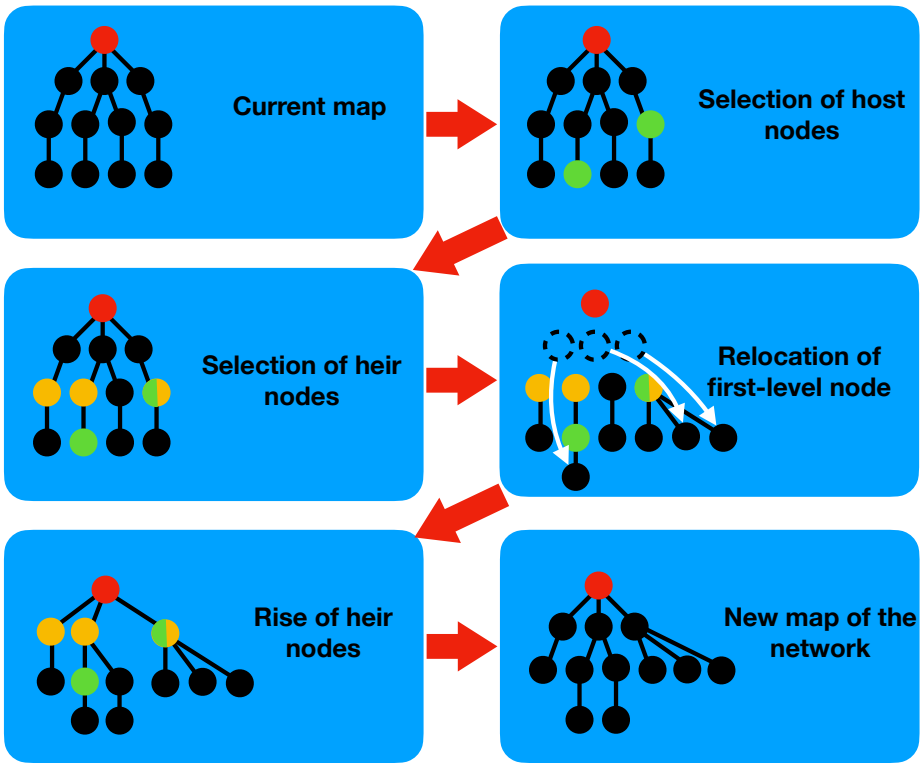


Figure 6.3. Setup phase of LEAST

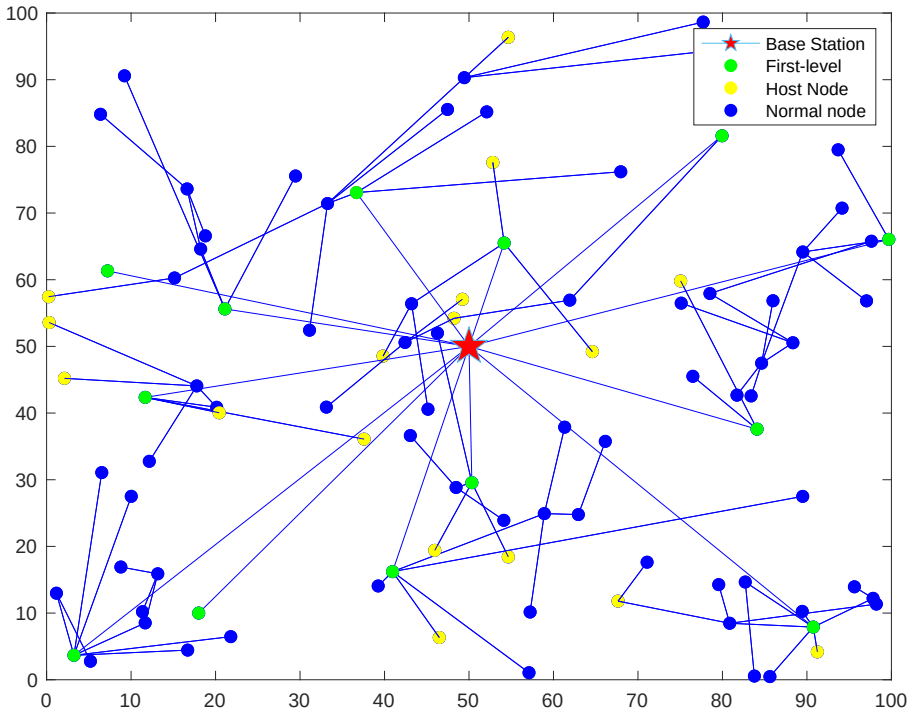


Figure 6.4. A snapshot of the network map for a sample network at a random around in the LEAST protocol

the number of messages and the distance over which each message needs to travel during the setup stage of LEAST. Additionally, for the purpose of this analysis, we assume p_H to be zero. We will discuss the effects of a nonzero p_H at the end of this section.

As a detailed analysis of the transmission energy requires knowledge about the location of the sensors, or the distribution that their locations follows, we use average values for distances in this analysis. We use $\bar{d}$ to show the average distance between pairs of nodes in the network. Additionally, we use $\bar{d_m}$ to show the average value for the distance between nodes and the furthest node from them.

The first point of communication between nodes in LEAST happens when HNs have to notify first-level nodes about themselves. The simplest, though not the most efficient, way to do so is for each HN to broadcast its status. This way, the (average) consumed power for the announcement of HNs will be proportional to $np_{HN}\bar{d_m}^2$. The other phase that consumes energy is the election of heir nodes. Once these nodes are selected, they need to notify the BS, which can be done in two hops through their corresponding first-level nodes. They also need to inform their sibling nodes. This can be done with a broadcast message, but the distance average distance for this can be considered to be $\bar{d}^2$ rather than $\bar{d_m}^2$. This is because the connections to a parent are inherently formed based on closeness of the nodes, so it is very unlikely to have sibling nodes whose distance is among the largest possible pair distances in the network. Considering p_H to be zero, every first-level node will only get one heir, which means that the number of first-level nodes will not change after the first round of the protocol. Therefore, we can consider the number of first-level nodes to be np_{CH} on average. This would mean an average power consumption of less than $3np_{CH}\bar{d}^2$ for this phase of the round. Consequently, communication power used in LEAST can be estimated using (6.1).

$$\text{Average power per round of LEAST } \propto n(p_{HN}\bar{d_m}^2 + 3p_{CH}\bar{d}^2) \tag{6.1}$$

For LEACH, each round consists of the broadcast announcement of CHs, followed by every node reaching out to its closest CH to attach to it. Using the same analysis as LEAST, we can use (6.2) to estimate the communication power usage of LEACH.

$$\text{Average power per round of LEACH } \propto n(p_{CH}\bar{d_m}^2 + (1 - p_{CH})\bar{d}^2) \tag{6.2}$$

We can now compare LEAST and LEACH using (6.1) and (6.2). The difference between these two estimates is $n((p_{CH} - p_{HN})\bar{d_m}^2 + (1 - 4p_{CH})\bar{d}^2)$. Assuming p_{CH} to be relatively small, and the values for p_{CH} and p_{HN} to be close (if not equal), LEACH consumes more energy compared to LEAST. This mainly comes from the fact that even though the network map is updated every round, not all nodes in LEAST need to be relocated. This saves a lot of energy in terms of the coordination that needs to be done between nodes to perform routing tasks. Additionally, note that we made some simplification assumptions in this analysis, which overestimates the power consumed in LEAST. Therefore, the actual difference between the power consumed by LEAST and LEACH is much more.

We now consider the effect of a nonzero value for p_H. There is always going to be at least one heir for every first-level node. Having a nonzero p_H gives rise to the possibility of having multiple heirs for a first-level node. p_H is mainly designed as a safety measure for preventing a first-level node from having too many children, and its value is usually quite small. This might change our estimate for the messages send from the heir nodes to announce their status. However, notice that even if at one round, a first-level nodes gets multiple heirs, this will not happen again in the next few rounds as the children get divided, and the value for p_H is small. Therefore, a nonzero value for p_H will have a negligible effect on our estimate for the consumed

power per round.

6.1.4 Simulation results

In this section, we analyse the simulation results of LEAST using MATLAB. The code used for simulating LEAST is a modified version of a code designed to simulate LEACH, which can be found at [161].

For the simulations, the network is created in a 100x100 grid. The BS is located at the center of the grid, $(50, 50)$. A number of 100 sensors are located randomly on the grid and their x and y coordinates are chosen uniformly. The LEAST protocol is then implemented as described in Section 6.1.2. The parameters for this simulation are chosen as follows.

$$p_{CH} = 0.1, \quad p_{HN} = 0.2, \quad p_{CH} = 0.1$$

The initial energy of the sensors was set to 0.1 Joules. Throughout the simulation, communication between sensors will reduce their energy, based on the transmission distance. A sensor node is dead if its energy level reaches zero.

We compare the performance of LEAST with LEACH. This is because of the fact that despite the numerous existing hierarchical routing protocols for WSNs, LEACH is still one of the most well-known and widely used ones. Additionally, most other protocols compare their result with LEACH, which means that comparing the performance of LEAST with LEACH will actually provide us with an image as to how LEAST will perform in comparison to other algorithms.

Fig. 6.5 illustrates the performance of LEAST and compares it with LEACH. The number of dead nodes in each round of the protocol for both LEAST and LEACH are shown. The first node in LEAST dies much later than LEACH. This shows a better energy performance for LEAST compared to LEACH. Another observation is

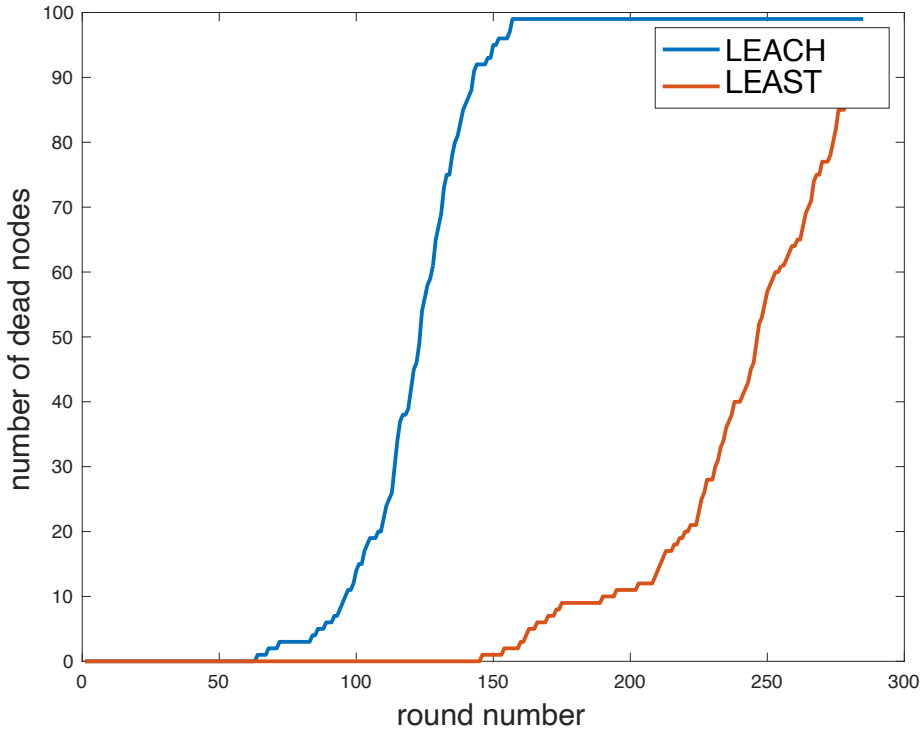


Figure 6.5. Comparison of LEAST with LEACH

that when the nodes do start to die, LEAST exhibits a slightly lower slope, which means that nodes die at a slower pace.

In order to compare LEAST with routing protocols other than LEACH, we refer to [162]. [162] introduces a multi-level hierarchical routing protocol for WSNs called BATR (Balanced Aggregation Tree Routing), and compares its performance with LEACH, direct transmission of data to the BS, and non balanced routing protocol using Prim's spanning tree algorithm as described in [162]. [162, Fig. 3] shows this comparison, by plotting the number of rounds versus the percentage of dead nodes in each round. Fig. 6.6 illustrates the same plot with the addition of the results obtained from LEAST. To obtain Fig. 6.6, we first extracted an estimate of the data points [162, Fig. 3], as the exact values were not given. We then normalised the scale of [162, Fig. 3] so that the results from our LEACH simulation match that of [162, Fig. 3]. The reason that we could not recreate the same simulation setup as [162] is that the initial energy of the sensor nodes in that simulation was not

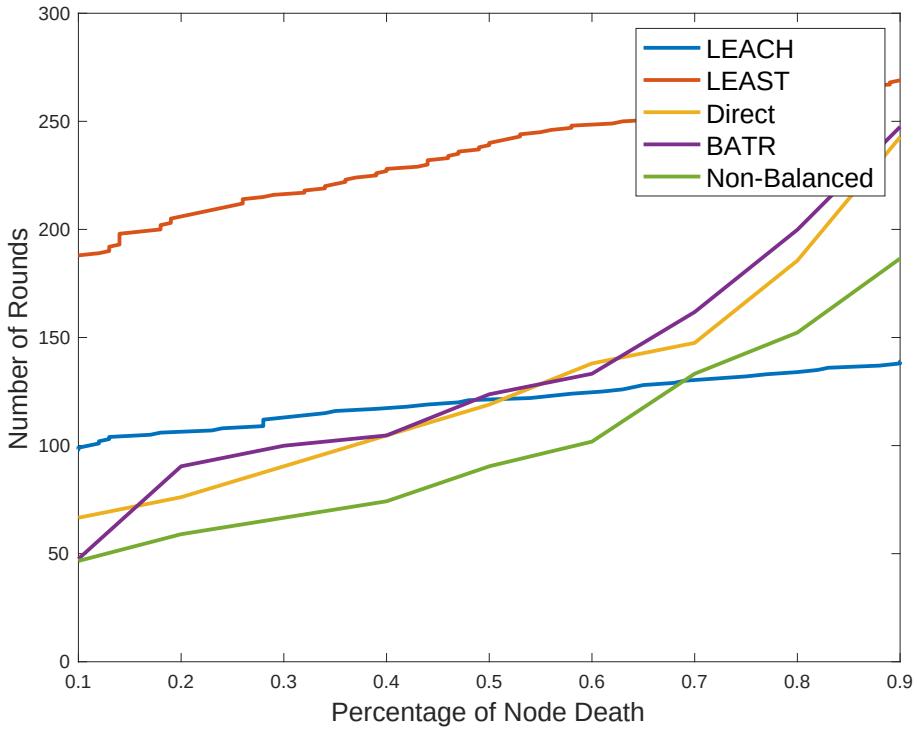


Figure 6.6. Comparison of LEAST with other protocols

given. LEAST is introducing a significant improvement over all the protocols studied in [162]. The curve for LEAST stands much higher than that of the other four protocols. This shows that all the nodes were alive for a much longer time in LEAST, and that LEAST significantly improves the overall lifetime of the network.

In Fig. 6.7, the total energy remaining in all the sensor nodes during each round is plotted for both LEACH and LEAST. Two remarkable points can be spotted in this figure. Firstly, the energy of the network finishes much later for the LEAST protocol compared to the LEACH protocol. Additionally, the energy drop is much more steady for LEAST compared to the sharp drop observed for LEACH. Overall, Fig. 6.5 and 6.7 show that LEAST is outperforming LEACH and other protocols in terms of overall energy consumption of the network.

Subsequently, we want to study the effect of the parameter p_{HN} on the performance of the algorithm. For this purpose, we used the same setup as the first simulation, and tested the lifetime of the network for values of p_{HN} between 0.01 and 0.9.

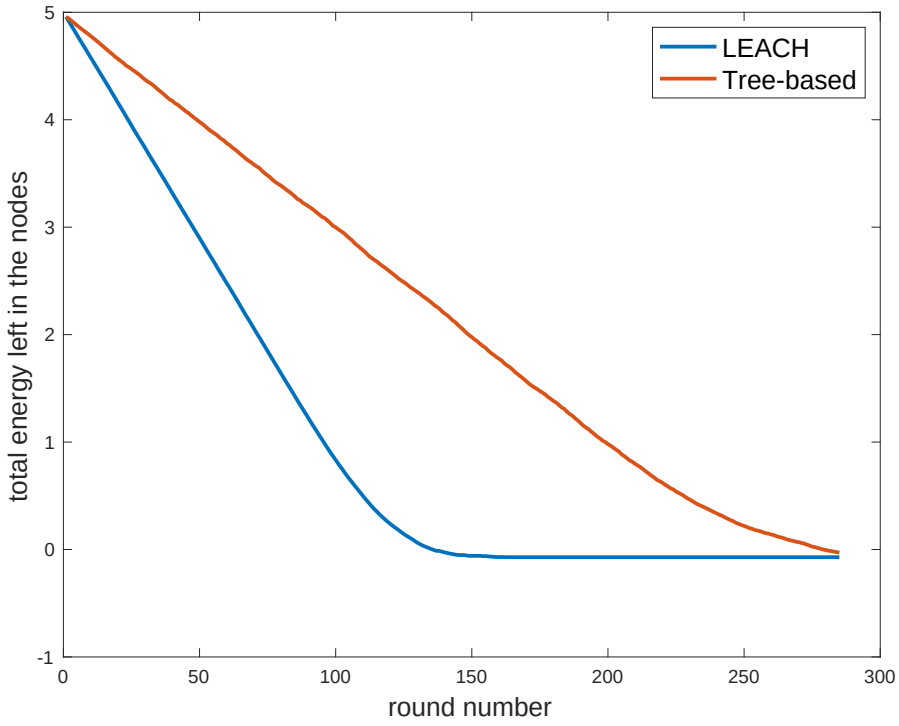


Figure 6.7. Comparison of LEAST with LEACH: total remaining energy per round

Additionally, we reduced the initial energy of nodes to 0.005 J so that the simulations run faster. Network lifetime is measured in terms of the number of rounds it takes for half of the nodes in the network to die, which we call half-life round. Fig. 6.8 shows the result of this simulation. This simulation suggests that lower values for p_{HN} will result in a better performance. However, notice that choosing low values for p_{HN} entails practical complications. This is because of the fact that the election process needs to be repeated if no node is chosen as a host node, which causes delays and an increase in communication overhead.

In order to prove the consistency of the results obtained in these simulations over different locations for sensors, we ran all of the simulations multiple times over different location distributions. We ran the simulation for 50 random location distributions instead of one. We averaged the results from all the 50 simulations to plot Fig. 6.9. The overall shape of the graph matches that of Fig. 6.5, and that the efficiency of LEAST is not dependent on the location of the sensor nodes.

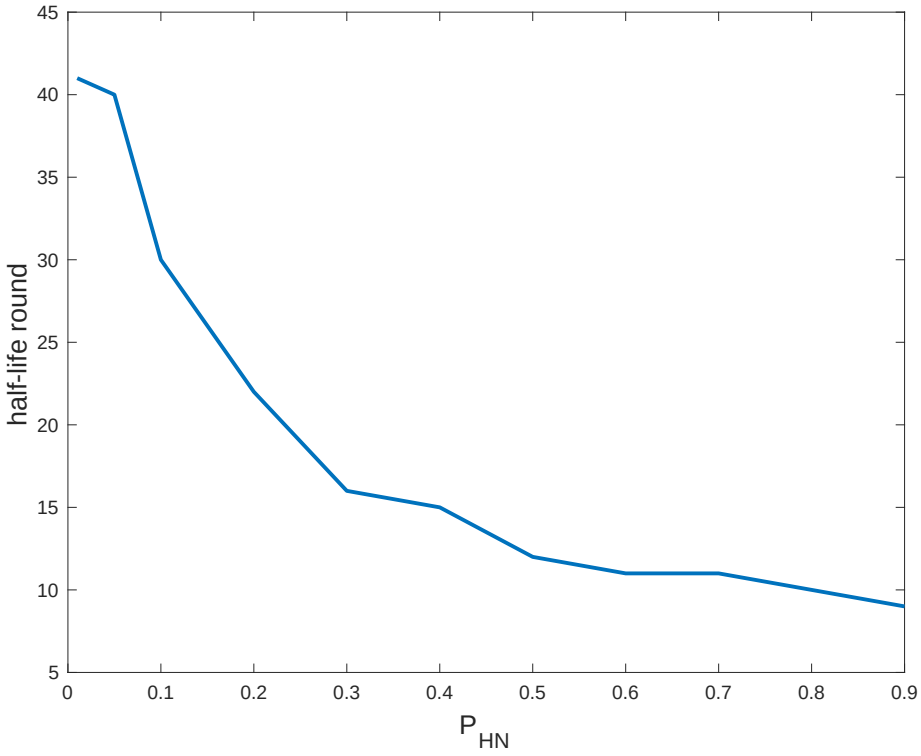


Figure 6.8. Effect of p_{HN} on the lifetime of the network

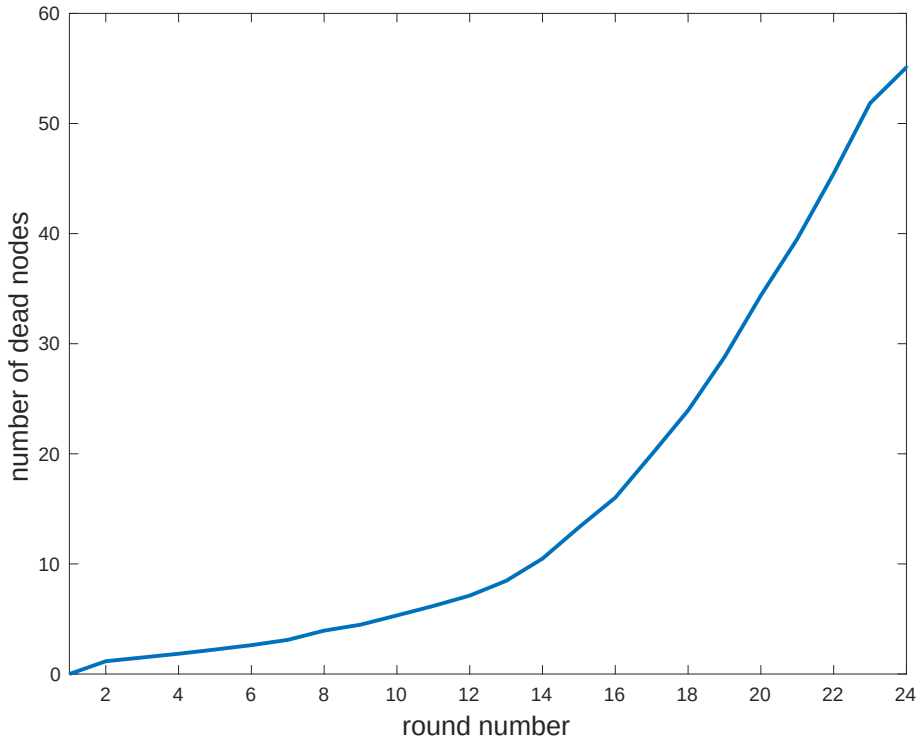


Figure 6.9. Simulation result for LEAST averaged over 50 random sensor node location distribution

6.1.5 Application of tree compression

In the previous section, LEAST was introduced as a tree-based routing protocol for WSNs, where the trees in the routing table can take any shape and form. This was unlike what is observed in other routing protocols for WSNs, such as LEACH, in which routing trees can only take on a specific format. It was also seen that this general shape for the routing tree already improves the overall lifetime of the network. However, this can be improved even further as the trees in the LEAST protocol are communicated are not compressed. As eloborated in previous Sections, compression results in a more efficient use of the available bandwidth and processing power.

As the routing table is now a general rooted tree, methods from Chapter 5 can be used to code and compress them. This will result in a much lower power and bandwidth consumption when this routing table needs to be communicated. Note that there are numerous scenarios during which the routing table is communicated between nodes. Firstly, during the steady-state phase of each round, sensor nodes might need to communicate with each other, and can therefore use the routing map to find the current path that exists between them and the destination node. Secondly, an external user might generate a query for a specific node, and therefore the BS needs to know the routing map to pass on the incoming message in the right direction. Therefore, there is a constant need for communicating such trees, and optimal compression methods for trees can further enhance resource utilization.

To see how much an optimal tree compression algorithm can reduce communication overhead, we use the example of the spanning tree model. The upper bound to the entropy of this method can be calculated using Eq. 5.40. Noting that the way routing tables are formed was one of our main motivations behind designing the spanning tree model, it is fair to assume that this model can simulate routing trees in WSNs well. Assuming that we are on the threshold of the giant component regime, Fig. 6.10 illustrates the ratio of the mean code length obtained using an optimal tree

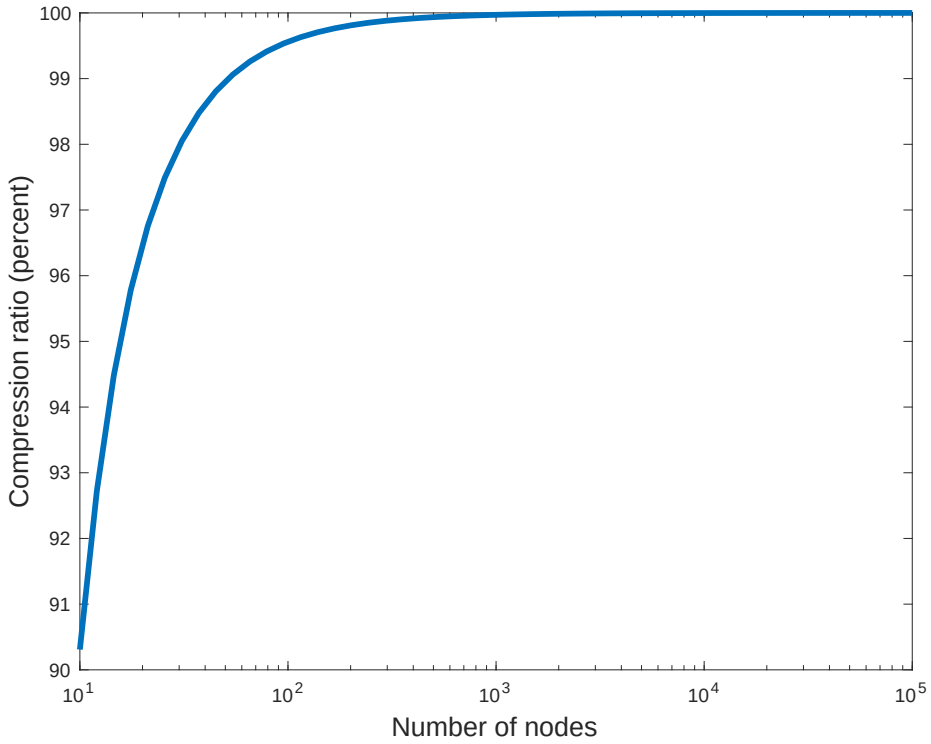


Figure 6.10. Compression ratio of the spanning tree model in the giant component regime boundary compared to adjacency list representation as a function of number of nodes in the WSN

compression algorithm, such as the one provided in Section 5.3.3, to the code length of storing the tree using the adjacency list representation.

Fig. 6.10 illustrates that as the WSNs grow large, an optimal tree compression algorithm can significantly benefit communication overhead and result in a more optimal use of the available bandwidth. This is just a mere example of how tree and graph compression can enhance WSN protocols. More work on this subject is out of the scope of this thesis, and is left to the interested reader.

6.2 Temporal Graph Neural Networks

The study of Graph Neural Networks (GNNs) is a rapidly growing area of machine learning and artificial intelligence. First introduced by [163] back in 2009, it has now become one of the trendiest areas in machine learning research. The importance of

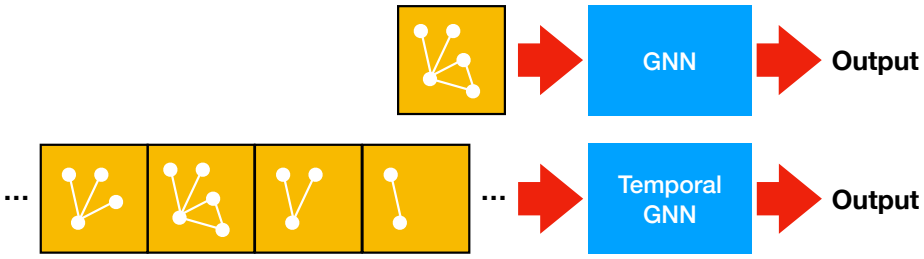


Figure 6.11. A comparison between GNNs and temporal GNNs

this field becomes more evident by observing the amount of graphical data structures around us. Social networks, communication networks, and biological networks are only a few examples of such graphs. Graphs can be used to represent the relationship present between data in many different fields of science, and therefore GNNs have been used to solve problems in areas such as modern recommender systems, computer vision, natural language processing (NLP), software mining, bioinformatics, and urban intelligence [164].

Additionally, as discussed in Chapter 4, most of these graphs are dynamic by nature. This means that we expect them to change and evolve over time. For instance, new nodes and links are always being formed or deleted in social media networks. Because of this reason, Temporal GNNs have recently attracted a lot of attention from the machine learning community. Temporal GNNs are used to study and learn from graphical data and their evolution through time, whereas standard GNNs learn from single graphs. A survey on existing methods on this topic has been recently published [165], and even newer methods have been introduced since then [36]. Fig. 6.11 illustrates the difference between GNNs and temporal GNNs.

Generally speaking, there is a lack of theoretical formulations to assess the performance of GNNs. Consequently, the temporal version of GNNs also suffers from the same lack of theoretical assessment in a more serious manner. Because of this, we propose an information-theoretic approach to analysing the performance of temporal graph learning algorithms. The information bottleneck principle is used as the core

idea, while the temporal graph evolutions are modeled using stochastic processes. Mutual Information Rate (MIR) is then used as the main metric for analysing the performance of algorithms that work on temporal graphs. Additionally, a new and more robust metric to replace MIR is introduced, and it is discussed how NECs can be used for analysing temporal GNNs.

6.2.1 The information bottleneck problem for temporal GNNs

The information-bottleneck problem [166] is often used as the information-theoretic formulation of Machine Learning. This method models the machine learning algorithm using an encoder-decoder pair, and formulates the problem as a trade-off between the compression ratio of the input and the accuracy of the output. We show the input to the algorithm with X, the desired output with Y, and the actual output of the algorithm with $\hat{Y}$. Additionally, we use Z to show the output of the encoder. This will be the output of the last layer if we are working with a neural network [167]. Fig. 6.12 shows these notations in the context of a learning algorithm.

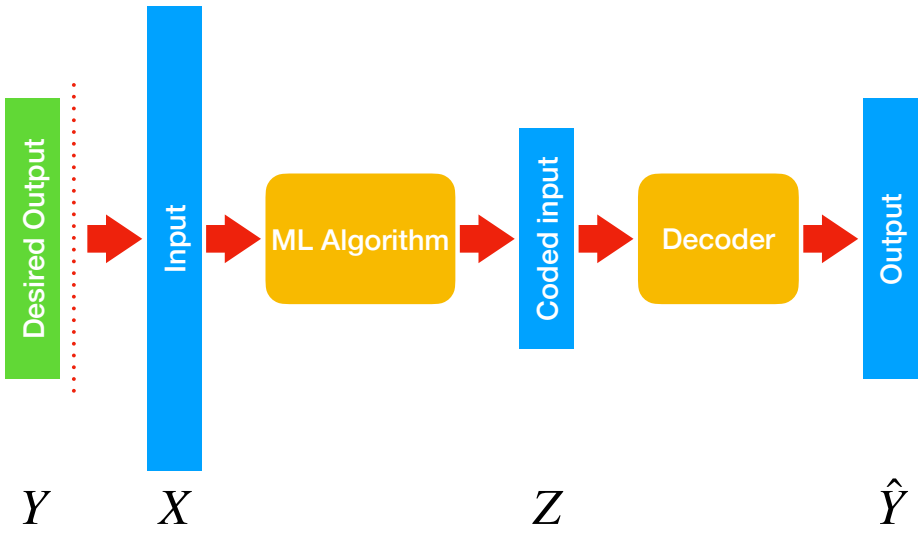


Figure 6.12. Model of an ML algorithm

Using the model of Fig. 6.12, the information bottleneck problem can be used to model the ML problem as below.

$$\begin{aligned} \inf \quad & I(X;Z) \\ \text{subject to} \quad & I(Y;Z) \geq \alpha, \end{aligned} \tag{6.3}$$

where I is used to show mutual information as per Shannon's definition [57, Ch. 2], and α is the parameter that sets the accuracy of the algorithm.

However, Eq. 6.3 can not be used in the context of dynamic graphs. This is because when dealing with temporal graphs, we should look at sequences of the input and output, rather than single ones. In this case, rather than assuming the variables in Eq. 6.3 to be single graphs, we will consider them to be stochastic processes. In other words we have

$$\begin{aligned} Y &= \{Y_1, Y_2, \ldots, Y_n\}, \\ X &= \{X_1, X_2, \ldots, X_n\}, \\ Z &= \{Z_1, Z_2, \ldots, Z_n\}, \\ \hat{Y} &= \{\hat{Y}_1, \hat{Y}_2, \ldots, \hat{Y}_n\}, \end{aligned} \tag{6.4}$$

where the following Markov chain governs each i, which is a time snapshot of the network.

$$Y_i \rightarrow X_i \rightarrow Z_i \rightarrow \hat{Y}_i \tag{6.5}$$

Eq. 6.3 tells us that in order to assess an ML algorithm we can look at the mutual information between X and Z, and Y and Z, respectively. However, in a dynamic graph setup, we have n time snapshots of these random variables. The natural question that arises in these scenario is the value of i to look at. In this case, we will use the concept of MIR [168].

When working with stochastic processes, we can look at the mutual information between pairs of corresponding variables, or $I(X_1, \ldots, X_n; Y_1, \ldots, Y_n)$. However, the

value of this can theoretically be unlimited as the process grows large. Therefore, we rely on the definition of MIR, which measures average mutual information per unit of time defined as below.

Definition 6.2.1 (Mutual Information Rate (MIR))**.** The mutual information rate between two stochastic processes X and Y is defined as below.

$$\text{MIR} = \lim_{n\to\infty} \frac{1}{n} I\left(X_1, X_2, \ldots, X_n; Y_1, Y_2, \ldots, Y_n\right)$$

Having defined the variables in Fig. 6.12 and using the notions from the information bottleneck principle, we propose the following metrics to be used for the assessment of temporal graph learning algorithms.

- **Accuracy:** $\text{MIR}(Y; Z)$
- **Compression:** $\text{MIR}(X; Z)$
- **Overall score:** $\text{MIR}(Y; Z)/\text{MIR}(X; Z)$

Note that there exist numerous methods for estimating the MIR of a pair of stochastic processes. For instance, the mutual information estimation setup proposed in [166][p. 17] can be used. However, because of the existence of $2n$ random variables in the definition of MIR, especially as n grows large, we propose an alternative metric for measuring the rate of mutual information.

In the remainder of this section, we will use the following notation for better readability when referring to subsets of a stochastic process.

$$X_{i:j} \equiv \{X_i, X_{i+1}, \ldots, X_j\} \tag{6.6}$$

6.2.2 Alternative Mutual Information Rate (AMIR)

The current definition for MIR makes its applications to stochastic processes very limited, as it involves the mutual information of the entire process as it grows large. This notion of MIR might not be feasible to study for real-world processes, as estimating the mutual information of such big stochastic processes might not be accurate. Because of this, we propose the definition of a novel relative metric for the mutual information rate between processes. This new metric is called alternative mutual information rate, or AMIR for short. Even though a similar metric exists for entropy rate of stochastic processes, the absence of this metric for analysing mutual information can be felt. This novel metric analyses the mutual information of single random variables, given the history of the sequence. We show that this novel metric can be estimated more accurately, especially if the given stochastic processes are Markov chains.

Firstly, note that the definition of MIR as stated in Definition 6.2.1 is very similar to that of entropy rate, which is stated in Definition 4.2.1. Additionally, an alternative definition for entropy rate was provided in Lemma 4.3.1, and was proven to be equal to the entropy rate defined in Definition 4.2.1. This new relative version of the entropy rate had the benefit of being easier to work with and use in our estimations and proofs. Based on the similarities between the concepts of entropy rate and MIR, we search for a relative format for MIR as well. This conditional concept of MIR can help us in many applications that involve the correlation of two stochastic processes, where we want to study the limit of the behaviour of single random variables in a conditional manner. Based on this, we provide the following definition for this alternative version of MIR.

Definition 6.2.2 (Alternative Mutual Information Rate (AMIR))**.** The Alternative Mutual Information Rate (AMIR) between two stochastic processes $\boldsymbol{X}$ and $\boldsymbol{Y}$ is defined

as

$$\mathrm{AMIR}(X;Y) = \lim_{n\to\infty} I(X_n; Y_n | X_{1:n-1}, Y_{1:n-1}), \tag{6.7}$$

when it exists.

In the remainder of this section, we will look at the properties of AMIR, its existence, and its relationship with MIR. We first start with the following theorem.

Theorem 6.2.1. *For two stationary processes X and Y, AMIR exists.*

Proof. It is well known that for two stationary processes X and Y, $\mathrm{MIR}(X;Y)$ exists [168]. For AMIR, we can write

$$\begin{aligned}
\mathrm{AMIR}(X;Y) &= \lim_{n\to\infty} I(X_n; Y_n | X_{1:n-1}, Y_{1:n-1}) \\
&= \lim_{n\to\infty} H(X_n | X_{1:n-1}, Y_{1:n-1}) \\
&+ H(Y_n | X_{1:n-1}, Y_{1:n-1}) \\
&- H(X_n, Y_n | X_{1:n-1}, Y_{1:n-1}).
\end{aligned} \tag{6.8}$$

Firstly, note that based on the stationarity of X, Y, and (X,Y), we have the following equation from Eq. 4.27.

$$\lim_{n\to\infty} H(X_n, Y_n | X_{1:n-1}, Y_{1:n-1}) = \lim_{n\to\infty} \frac{1}{n} H(X_{1:n}, Y_{1:n}) \tag{6.9}$$

Eq. 6.9 shows that $H(X_n, Y_n | X_{1:n-1}, Y_{1:n-1})$ tends towards the entropy rate of (X,Y), and has a limit. Therefore, in order to show that the limit for AMIR exists, we only need to show that the other two limits in Eq. 6.8 exist. For $H(X_n | X_{1:n-1}, Y_{1:n-1})$ we have

$$\begin{aligned}
H(X_n | X_{1:n-1}, Y_{1:n-1}) &\leq H(X_n | X_{2:n-1}, Y_{2:n-1}) \\
&= H(X_{n-1} | X_{1:n-2}, Y_{1:n-2}),
\end{aligned} \tag{6.10}$$

where the first line comes from the fact that conditioning reduces entropy, and the

second line can be concluded from the stationarity of both processes. Therefore, $H(X_n|X_{1:n-1}, Y_{1:n-1})$ forms a non-increasing sequence of positive numbers, which has a limit. A similar approach can be used to prove that $H(Y_n|X_{1:n-1}, Y_{1:n-1})$ also converges. Therefore, AMIR also has a limit and is well-defined for stationary processes. □

Additionally, we can state the following corollary for Theorem 6.2.1.

Corollary 6.2.1.1. *For stationary processes X and Y we always have*

$$AMIR(X;Y) \leq MIR(X;Y). \tag{6.11}$$

Proof. We can use Eq. 6.8 and write the following chain of equations.

$$\begin{aligned}
\mathrm{AMIR}(X;Y) &= \lim_{n\to\infty} H(X_n|X_{1:n-1}, Y_{1:n-1}) + H(Y_n|X_{1:n-1}, Y_{1:n-1}) \\
&\quad - H(X_n, Y_n|X_{1:n-1}, Y_{1:n-1}) && (6.12)\\
&= \lim_{n\to\infty} H(X_n|X_{1:n-1}) - I(X_n; Y_{1:n-1}) \\
&\quad + H(Y_n|Y_{1:n-1}; X_{1:n-1}) - I(Y_n; X_{1:n-1}) \\
&\quad - H(X_n, Y_n|X_{1:n-1}, Y_{1:n-1}) && (6.13)\\
&= H(X) + H(Y) - H(X,Y) - I(X_n; Y_{1:n-1}) - I(Y_n; X_{1:n-1}) && (6.14)\\
&= \mathrm{MIR}(X;Y) - I(X_n; Y_{1:n-1}) - I(Y_n; X_{1:n-1}) && (6.15)
\end{aligned}$$

Based on the fact that mutual information is non-negative, we can easily conclude the claim of the corollary based on Eq. 6.15. □

Additionally, the conditions under which MIR and AMIR are equal are stated in the following corollary.

Corollary 6.2.1.2. *For stationary processes X and Y, MIR and AMIR are equal if and only if the following conditions are met.*

1. *Given $X_{1:n-1}$, X_n and $Y_{1:n-1}$ are independent.*

2. Given $Y_{1:n-1}$, Y_n and $X_{1:n-1}$ are independent.

Proof. As $I(X_n; Y_{1:n-1}|X_{1:n-1})$ and $I(Y_n; X_{1:n-1}|Y_{1:n-1})$ are non-negative, the only way that the values of MIR and AMIR can be equal according to Eq. 6.15 is if these two values are zero. For $I(X_n; Y_{1:n-1}|X_{1:n-1})$ we can write

$$\begin{aligned} & I(X_n; Y_{1:n-1}|X_{1:n-1}) = 0 \\ & \Rightarrow H(X_n|X_{1:n-1}) - H(X_n|X_{1:n-1}, Y_{1:n-1}) = 0 \\ & \Rightarrow H(X_n|X_{1:n-1}) = H(X_n|X_{1:n-1}, Y_{1:n-1}), \end{aligned} \tag{6.16}$$

which translates into X_n and $Y_{1:n-1}$ being independent given $X_{1:n-1}$. The other condition can be obtained by setting $I(Y_n; X_{n-1}|Y_{n-1})$ to zero. □

In addition to the processes being stationary, the conditions in Corollary 6.2.1.2 need to be met in order for MIR and AMIR to be equal. This is unlike the equality of entropy rate and relative entropy rate, where the only condition was the stationarity of the process. We will now study the conditions under which MIR and AMIR are equal. In order to state this, we first provide a definition.

Definition 6.2.3 (Causal stochastic processes)**.** A pair of stochastic processes X and Y are called causal if for any given i, $X_{1:i-1}$ and $Y_{1:i-1}$ and all of their subsets are independent from X_i and Y_i.

We will now provide the condition for MIR and AMIR to be equal, which is based on Definition 6.2.3.

Theorem 6.2.2. *For a pair of causal stochastic processes X and Y, the values of MIR and AMIR are equal.*

Proof. We need to prove that the conditions of Corollary 6.2.1.2 holds for X and Y. In order to show the independence of X_n and $Y_{1:n-1}$ given $X_{1:n-1}$, we use their joint

distribution.

$$\begin{aligned} p(X_n, Y_{1:n-1}|X_{1:n-1}) &= p(Y_{1:n-1}|X_n)p(X_n|X_{1:n-1}) \\ &= p(Y_{1:n-1}|X_{1:n-1})p(X_n|X_{1:n-1}), \end{aligned} \tag{6.17}$$

where the last line comes from the fact that as X and Y are causal, $Y_{1:n-1}$ does not depend on X_n. Eq. 6.17 shows that X_n and $Y_{1:n-1}$ are independent given $X_{1:n-1}$. The independence of Y_n and $X_{1:n-1}$ given $Y_{1:n-1}$ can be proven in a similar manner. Therefore, it is concluded that MIR and AMIR are equal for casual processes. □

Based on Theorem 6.2.2, the condition for MIR and AMIR to be equal is not too strict. The condition of causality can be assumed for most pairs of processes that are considered to be happening at the same time (or any other space that the process is defined on). Additionally, note that this condition is just a sufficient condition, and not a necessary one. The assumption of causality is stronger than what is actually needed for MIR and AMIR to be equal. According to Corollary 6.2.1.2, for every i, we only need the $i+1$th sample to not have any effect on the previous i samples. This means that even if $i+2$nd sample and onwards affect X_i and Y_i, the conditions of Corollary 6.2.2 still hold.

6.2.3 Benefits of AMIR

In this section, we will briefly mention and discuss two advantages of AMIR over MIR.

6.2.3.1 Rate of convergence

Existing methods for estimating mutual information of random variables rely on samples from the distribution of those random variables [169, 170]. Therefore, as the space of possible data points grows, more sample are needed to accurately estimate

this metric. Looking at the definition of MIR and AMIR, the space of both metrics includes $(|X||Y|)^n$ possibilities. However, there are usually assumptions in place about the memory of the system, which dictates how many samples in the future will be affected by the current sample. Based on this, there usually exists a constant C for which we can assume

$$I(X_n; Y_n|X_{1:n-1}, Y_{1:n-1}) = I(X_n; Y_n|X_{n-C:n-1}, Y_{n-C:n-1}). \tag{6.18}$$

This assumption will reduce the size of the sample space to $(|X||Y|)^{C+1}$, which lowers the number of samples needed to estimate AMIR and increase the convergence rate.

6.2.3.2 Number of sample sequences needed

It must be noted that in order to estimate MIR, we need samples of (X_n, Y_n), which means samples of both sequences. Additionally, the number of required samples for accurate estimation will increase exponentially with n. However, with AMIR, one sequence for (X, Y) will suffice if the memory of the system is assumed to be limited. In other words, with every new sample for the sequence, we will have a new sample for estimating $I(X_n, Y_n|X_{n-C:n-1}, Y_{n-C:n-1})$, which removes the need for having numerous sequences.

6.2.4 AMIR of Markov chains

In this section, we will consider the Mutual Information Rate of stationary Markov chains. As a special case of stochastic processes, Markov chains have applications in many different areas.

Consider X and Y to be stationary Markov chains. Let μ_X, μ_Y, and μ_{XY} be the stationary distributions of X, Y, and (X, Y), respectively. Additionally, let P, Q, and R be the transition probability matrices for X, Y, and (X, Y), respectively. To

calculate the MIR of X and Y, we can write

$$\begin{aligned}\mathrm{MIR}(X;Y) &= H(X) + H(Y) - H(X,Y) \\ &= -\sum_{ij} \mu_{X,i} P_{ij} \log P_{ij} \\ &- \sum_{ij} \mu_{Y,i} Q_{ij} \log Q_{ij} \\ &+ \sum_{ij} \mu_{XY,i} R_{ij} \log R_{ij}.\end{aligned} \tag{6.19}$$

Note that the entropy rate of stationary Markov chains can be found using [57, Thm. 4.2.4].

Additionally, the condition of causality holds for most of the processes that are simulated using Markov chains. For instance, Markov chains have recently been used for temporal alignment of medical signals in dementia [171]. In this study, it is safe to assume that future signals will not affect the past, and therefore the assumption of causality holds. Consequently, it is safe to state that for most of the Markov chains we work with, MIR and AMIR will be equal, but it is always necessary to check the condition of causality according to Theorem 6.2.2.

Even though both MIR and AMIR can be calculated using Eq. 6.19, we do not necessarily always have all the parameters of the Markov chain, and therefore can not use Eq. 6.19 directly. Therefore, we need to rely on estimation. As less parameters are involved in the definition of AMIR, it can prove to be a more robust and reliable estimator compared to MIR. Observe that for MIR, we need to estimate $I(X_1, \ldots, X_n; Y_1, \ldots, Y_n)$, whereas the value to estimate for AMIR in Markov chains is simply $I(X_n; Y_n | X_{n-1}, X_{n-1})$.

6.2.5 Simulation results

In this section, we will perform simulations on a number of stochastic processes to compare the performance of MIR and AMIR as metrics of mutual information to study stochastic processes.

6.2.6 Gaussian processes

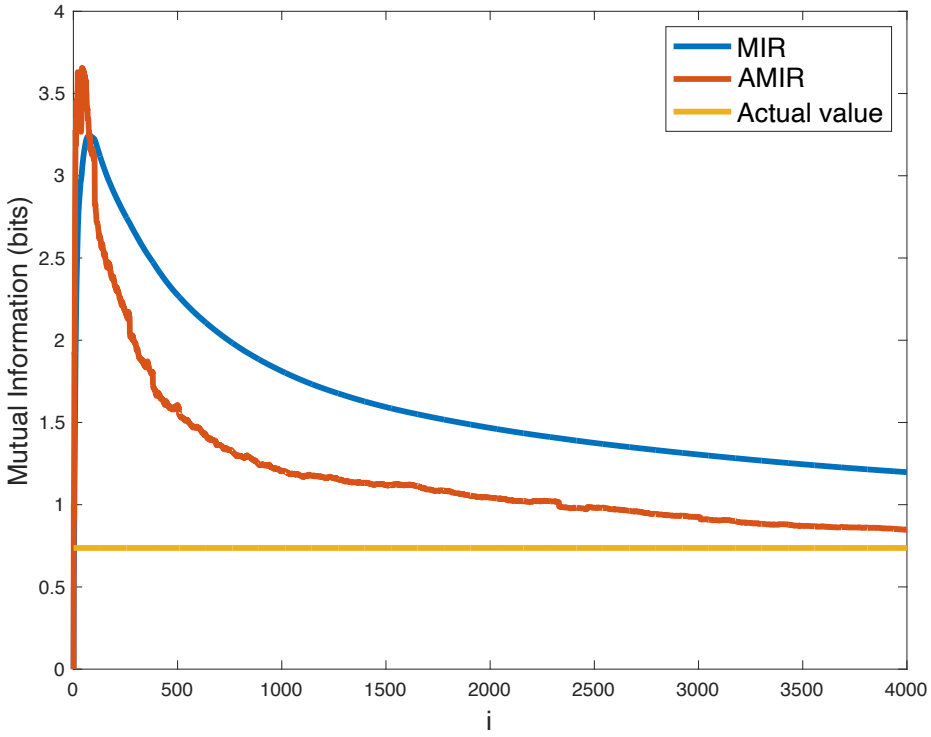
In this section, we simulate MIR and AMIR for two correlated Gaussian processes. For this purpose, we chose the following processes.

- $X_i \sim \mathcal{N}(0,\, 1)$
- $Y_i \sim 0.8X_i + 0.6\mathcal{N}(0,\, 1)$

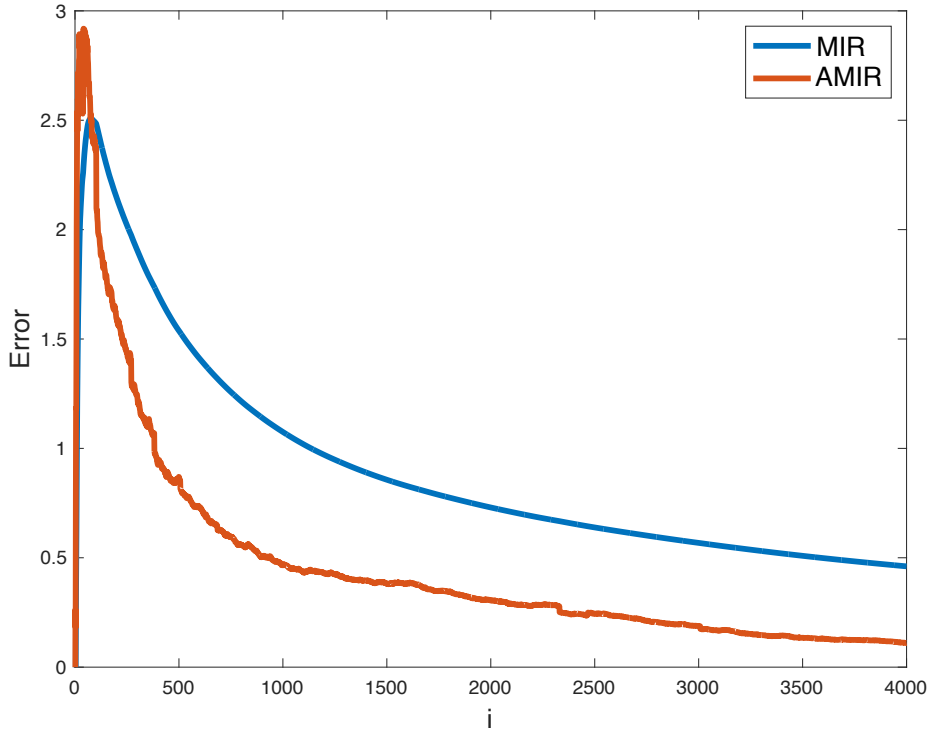
It can be checked that as the variables in these processes are i.i.d, they satisfy the conditions of Corollary 6.2.1.2, and MIR and AMIR are equal for these two processes. Additionally, both MIR and AMIR are simply equal to the mutual information of a pair of these random variables, which can be calculated as below.

$$\begin{aligned} I(X_i, Y_i) &= H(Y_i) - H(Y_i|X_i) \\ &= H(\mathcal{N}(0,\, 1)) - H(\mathcal{N}(0,\, 0.36)) \\ &= 0.5 \log \frac{1}{0.36} \end{aligned} \tag{6.20}$$

Fig. 6.13 shows the result of this simulation for estimating MIR and AMIR. Fig. 6.13a shows the estimate for MIR and AMIR as the sequence grows large, alongside the actual calculated value for the mutual information. Fig. 6.13b shows the error in the estimate as the difference between the estimated values for MIR and AMIR with the actual value. Even though AMIR shows more fluctuations, it is faster in converging to the theoretical limit. This is in accordance with our predictions about the behaviour of AMIR.



(a) The estimations for MIR and AMIR



(b) Error in the convergence of MIR and AMIR

Figure 6.13. Estimating MIR and AMIR

6.2.7 Hidden Markov Model

In this section, we created and simulated a two state Hidden Markov Model (HMM) (X, Y) with the following properties.

$$\begin{cases} \mathbf{Z}_X = \begin{bmatrix} 0.8 & 0.2 \\ 0.4 & 0.6 \end{bmatrix} \\ p(Y_i|X_i) = \begin{bmatrix} 0.7 & 0.3 \\ 0.3 & 0.7 \end{bmatrix} \end{cases} \tag{6.21}$$

Firstly, it must be noted that this pair of stationary processes does not satisfy the conditions of Corollary 6.2.1.2. This is because Y_i is not independent from X_{i-1}. Therefore, we expect MIR and AMIR to be different for this pair of processes. Based on the parameters given in Eq. 6.21, the values for MIR and AMIR can be calculated, and have the following values.

$$\begin{cases} \text{MIR}(X;Y) = 0.1035 \text{ bits} \\ \text{AMIR}(X;Y) = 0.0892 \text{ bits} \end{cases} \tag{6.22}$$

We then ran the simulation for one random instance of the HMM with a length of 4000, in order to estimate the value for AMIR using its definition. The results of this simulation are illustrated in Fig. 6.14. The first thing that can be observed is the fact that the estimated AMIR is clearly tending towards its theoretical limit, as proven in Theorem 6.2.1. Secondly, as stated in Corollary 6.2.1.1, the value for AMIR is less than MIR. Ultimately, this result is showcasing one of the main advantages of AMIR, which is that it can be estimated from having a single sequence from both processes. However, to estimate MIR from observed data, we are in need of numerous instances of both sequences.

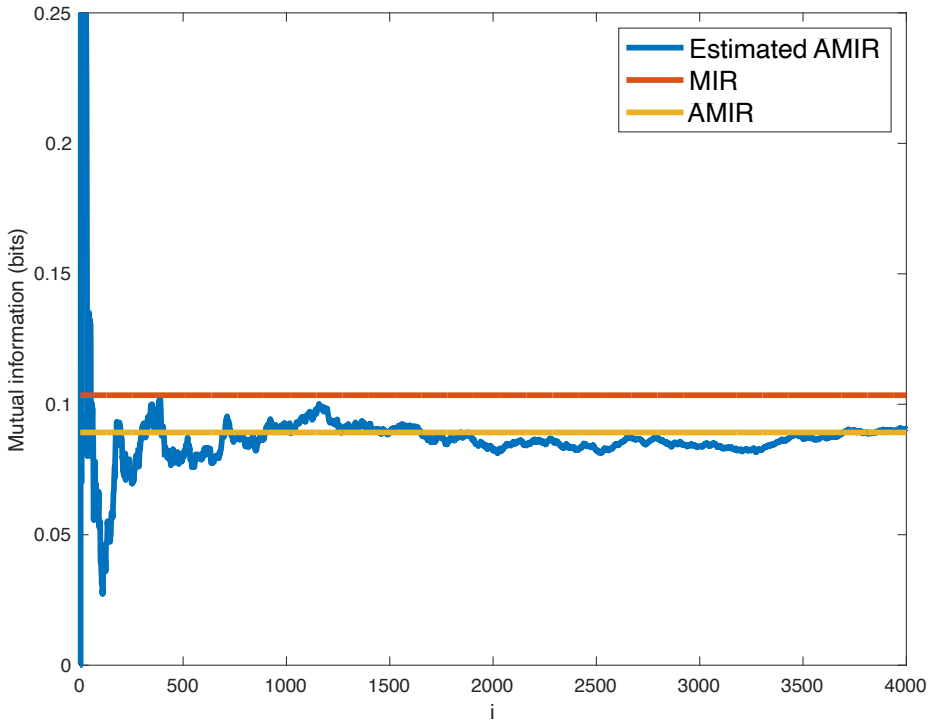


Figure 6.14. Estimating AMIR of an HMM

6.2.8 Application of NEC to temporal GNNs

Due to the similarities between the concepts of MIR and entropy rate, NECs can be used to simulate the sequence of input graphs, and the information-theoretic analysis done in Chapter 4 can be employed to study temporal GNNs. Using Network Evolution Chains to model the input sequence of graphs will impose a Markov chain assumption on the input, which will result in the following simplification for MIR.

$$\text{MIR} = \lim_{n\to\infty} I\left(X_n; Y_n | X_{n-1}, Y_{n-1}\right) \tag{6.23}$$

Additionally, there are parallels to all the concepts discussed in Chapter 4 in terms of MIR. For instance, an equivalent of the AEP can be stated for MIR in the following format.

$$\frac{1}{n}\log\frac{p(X_{1:n}, Y_{1:n})}{p(X_{1:n})p(Y_{1:n})} \to \text{MIR}(X;Y) \quad \text{with probability 1} \tag{6.24}$$

The proof for Eq. 6.24 can be done in a similar way to the original AEP, which can be found in [57, Thm. 16.8.1].

If the input graphs to a temporal GNN are modeled using an NEC, then it satisfies the conditions for the AEP for MIR. Additionally, as the relative version of MIR, AMIR can be used to calculate the MIR of the process, just like the approach done in Eq. 4.6. This then gives rise to the extension of all the concepts discussed in Chapter 4 in the context of temporal GNNs and MIR. This goes beyond the scope of this thesis, and is left to the interested reader.

6.3 Summary

This chapter explored some of the potential applications of the research done in this thesis. For this purpose, two main application scenarios were chosen: routing in WSNs, and analyzing the performance of temporal GNNs.

For the first application, we started by showcasing the benefits of hierarchical routing protocols for WSNs. Subsequently, we introduced a novel routing protocol for WSNs, which we named LEAST. The advantage of LEAST over existing hierarchical routing protocols was that its routing map has the capability of taking on the form of any rooted tree. This feature of LEAST proved to be highly beneficial in terms of energy consumption using simulations. Ultimately, we argued why tree compression algorithms can enhance LEAST even further, contributing to a more efficient use of resources in WSNs.

We then moved on to the second application domain, which was the information-theoretic analysis of temporal GNNs. After emphasizing on the importance and relevance of GNNs, we argued why there has recently been a lot of research on the temporal analysis of such networks. Because of the lack of a theoretical framework for analysing temporal GNNs, we introduced an information-theoretic methodology

to assess the performance of temporal GNNs. Finally, we argued why simulating the temporal evolution of graphs using NECs can further improve the benefits of the introduced model, and expand its applications.

Chapter 7

Conclusions and Future Work

7.1 Thesis summary

In this thesis, we provided an information-theoretic framework for the analysis of graphical data structures. We believe the existence of such an approach is very much needed, as not only can graphs be observed all around us in the form of different networks, but they are also growing faster than ever. It was noted that the inherent complexity of graphs also necessitates the need for a more formal analysis of this complexity.

The significance of integrating the two fields of graph theory and information theory becomes evident when examining the specific demands within graph theory and the valuable tools provided by information theory. The complex nature of graphs and networks requires a more formal analysis, which can lead to a better understanding of the behaviour of networks, and improving existing algorithms in network science. Concurrently, information theory's ability to study the complexity of random variables has stood the test of time for over seven decades. Consequently, the main aim of this thesis was to study graphs and networks through the lens of information theory, an approach that has been relatively underexplored but has proven highly advantageous

throughout the course of this study.

In **Chapter 2**, we used a classical approach, and analysed graphs using Shannon's entropy and classical source coding methods. Random sources were defined for graphs, and the complexity of these sources was studied through their entropy. As the random graph model is not always known, we first provided a method to estimate the entropy of graphical data structures using observations. We then studied and measured the entropy of some the most widely-used random models for graphs, such as the ER and SBM models. Ultimately, ways for near-optimal compression of these graphs were also introduced.

Chapter 3 studied the complexity of graphs using another important complexity measure in information theory, Kolmogorov complexity. Kolmogorov complexity was introduced as a measure of complexity for single objects and random variables, and it was shown that it can provide a robust and reliable measure of complexity for static graphs. Acknowledging the incomputability of K-complexity, we presented a method for estimating an upper bound to the K-complexity of graphs. It was also discussed that the introduced method, called Kolmogorov graph covering, can also be used for coding graphs given enough computation power. The chapter was then concluded by providing simulation results of Kolmogorov graph covering on a number of graphical data structures, and benchmarking it with some existing methods for measuring the K-complexity of graphs.

Network evolution models provide a mathematical modelling of network evolution, and are therefore very important in studying and simulating dynamic networks. In **Chapter 4**, we studied the complexity of these models using information theory. The main information-theoretic tools that we used for this purpose were entropy rate, and the AEP. It was discussed that despite the abundance of models for simulating network evolution, the AEP can not be directly applied to these models. As a result, we introduced the NEC model, a model capable of simulating network dynamics

and being analysable using information theory. The entropy rate of this model was studied, and it was discussed how the parameters of this model can be adjusted to the network that we are aiming to simulate. Furthermore, extensions of this model were introduced to allow the analysis of functions of networks, as well as simulating their dynamics in the continuous time domain. The chapter was concluded by providing simulation results for the proposed model.

In **Chapter 5**, we narrowed down our focus on an important sub-family of graphs: trees. The widespread use of trees, alongside their unique characteristics, demanded a more specialised study of the information-theoretic properties of these important data structures. As a result, we studied random tree sources, their entropy, and compression. We started by analysing the entropy of some existing random tree sources, as well as introducing a novel one called the spanning tree model. We then moved on to provide near-optimal compression algorithms for the studied sources, whose optimality was proven both mathematically and illustrated using simulation results.

Ultimately, we identified and discussed the potential applications of our research in **Chapter 6**. We focused on two major applications. Firstly, we argued the importance of efficient routing in WSNs, and proposed a tree based routing protocol. It was shown that the proposed algorithm, called LEAST, is not only able to improve the performance of the existing algorithms by itself, but can also gain additional benefits from tree compression methods. As the second application, we tested the concepts of NECs on analysing temporal GNNs. It was discussed how MIR can be used to gain a better understanding into the performance of these systems, and how it can be used to extend our results for NECs to temporal GNNs.

7.2 Future works

This thesis provided the groundwork for an information-theoretic framework for the analysis of graphs. However, we believe that this is still the beginning of this work, and there still exists uncharted territories in this field that need to be studied. In this section, we will suggest some of the directions in which one can build up on the research undertaken in this thesis.

7.2.1 Universal compression of graphs

Finding a universal compression algorithm for graphical data structures was stated as one of the main motivations behind the framework presented in this thesis. While comprehensive methods for compressing tree structures were introduced in Chapter 5, it is crucial to recognise that many networks in our environment do not conform to a tree-like structure but still stand to benefit significantly from compression. As previously discussed, the potential implications of optimal compression alone are profound, particularly in the context of the technology industry. Both data storage and communication sectors rank among the highest CO2 emitting industries in today's world. Data storage alone accounts for 2% of global carbon emissions. Considering the exponential growth in complexity of the networks that surround us, these optimal graph compression algorithms possess the potential to exert a significantly positive environmental impact.

Even though universal compression algorithms such as LZ have been around for years for one-dimensional data such as text, there is still no universal compression algorithm designed for graphs. A universal graph compression algorithm must guarantee an optimal compression ratio with respect to information-theoretic metrics, regardless of the underlying distribution of the random graph source. Studies such as [172] have already started tackling this problem, and are a step in the right direc-

tion.

There is still a long way to go to be able to propose a one-size-fits-all graph compression algorithm, owing to the complex structures and intricacies that can be present in graphs. However, we believe that using information theory to analyse the complexity of graphs is a necessary step to achieve that goal. Therefore, we hope that the research results provided in this thesis can be used as a basis for studying universal graph compression algorithms.

7.2.2 Improving the performance of Kolmogorov graph covering algorithm

Kolmogorov graph covering was introduced in Chapter 3 as an effective method to estimate an upper bound to the Kolmogorov complexity of graphs. Additionally, it was discussed that this method can also be used as an effective graph compression algorithm. However, the main issue with employing Kolmogorov graph covering as a graph compression algorithm is its high computational complexity.

Finding more efficient ways of implementing the Kolmogorov graph covering algorithm can prove to be very beneficial. This is not an easy task and requires tools from both graph theory and computer science. Consequently, we believe that this can be an impactful way of continuing our research.

7.2.3 Enhancing NECs

NEC was introduced in Chapter 4 as a novel model to simulate network dynamics. It was discussed that the transitions in NECs are designed to be as general as possible and be able to simulate most possible scenarios of network evolution. However, they have a limitation that is inherent in the fact that all possible transitions are node-based. In other words, at each step, only nodes can be added or deleted, and not

edges. Even though it was explained in Section 4.2 how edge-wise operations can be implemented for NECs, it still makes sense to introduce a set of edge-based operations so that the model is able to capture network dynamics better. However, it must be noted that these edge-wise operations must be incorporated in a way that still allows for the AEP to be applied to the model. In light of this, we believe that a generalization of NECs could allow for a better simulation and analysis of evolving networks.

7.2.4 Applications of the proposed research

Some of the potential applications of this research were discussed in Chapter 6. We would like to emphasise that the applications discussed in Chapter 6 are just two examples of the numerous applications of the proposed field of research. Other examples include network tomography, database management, and even interdisciplinary fields such as the study of molecular networks or metabolic pathways. Therefore, one of the impactful ways of continuing this research is by applying the results to different domains.

Furthermore, the applications explored in Chapter 6 could benefit from additional enhancements. The analysis presented in that chapter was introductory, primarily due to space constraints within the thesis. Consequently, we believe that there is ample opportunity for the interested reader to delve deeper into the applications of the findings of this thesis in the context of routing protocols for WSNs and the analysis of temporal GNNs.

7.3 Conclusions

This thesis provided a preliminary analysis on the topology of networks, viewed through the lens of information theory. For this purpose, networks were consid-

ered in the context of different scenarios they can appear in, resulting in the analysis of networks using three different mathematical models. Random graph generators were used to study random networks, Kolmogorov complexity was used to measure the intricacies of constant networks, and network evolution models were considered for modelling the behaviour of evolving networks. These analyses provided us with new insights into the information-theoretic properties of the topology of the networks around us, and can be used to better understand and model said networks. This thesis also studied trees, as a subset of graphs, in a more detailed manner. Because of the unique applications of trees and their presence and importance in different fields, the study of their complexity and compression can impact a wide range of fields. The thesis was ended by an overview on some of the potential applications of this research topic in engineering.

It is our firm conviction that the subject studied throughout this thesis has the potential of having a significant impact on the world of communications engineering, and any field in which networks play a significant role. The novel approach of this thesis, which lies in the intersection of graph theory and information theory, can act as a cornerstone for transforming this analysis to its own branch of science in the near future, a branch that I would like to call **information-theoretic network science**.

Bibliography

[1] S. L. Hakimi, "Optimum distribution of switching centers in a communication network and some related graph theoretic problems," *Operations research*, vol. 13, no. 3, pp. 462–475, 1965.

[2] M. E. Newman, D. J. Watts, and S. H. Strogatz, "Random graph models of social networks," *Proceedings of the national academy of sciences*, vol. 99, no. suppl 1, pp. 2566–2572, 2002.

[3] S. Karra, D. O'Malley, J. Hyman, H. S. Viswanathan, and G. Srinivasan, "Modeling flow and transport in fracture networks using graphs," *Physical Review E*, vol. 97, no. 3, p. 033304, 2018.

[4] T. Aittokallio and B. Schwikowski, "Graph-based methods for analysing networks in cell biology," *Briefings in bioinformatics*, vol. 7, no. 3, pp. 243–255, 2006.

[5] T. V. T. Hong and P. Do, "Developing a graph-based system for storing, exploiting and visualizing text stream," in *Proceedings of the 2nd international conference on machine learning and soft computing*, pp. 82–86, 2018.

[6] O. Kanhere, H. Poddar, Y. Xing, D. Shakya, S. Ju, and T. S. Rappaport, "A power efficiency metric for comparing energy consumption in future wireless networks in the millimeter-wave and terahertz bands," *IEEE Wireless Communications*, vol. 29, no. 6, pp. 56–63, 2022.

[7] J. R. Platt, "Influence of neighbor bonds on additive bond properties in paraffins," *The Journal of Chemical Physics*, vol. 15, no. 6, pp. 419–420, 1947.

[8] N. Rashevsky, "Life, information theory, and topology," *The bulletin of mathematical biophysics*, vol. 17, no. 3, pp. 229–235, 1955.

[9] A. Mowshowitz, "Entropy and the complexity of graphs: I. An index of the relative complexity of a graph," *The bulletin of mathematical biophysics*, vol. 30, no. 1, pp. 175–204, 1968.

[10] A. Mowshowitz, "Entropy and the complexity of graphs: II. The information content of digraphs and infinite graphs," *The Bulletin of mathematical biophysics*, vol. 30, no. 2, pp. 225–240, 1968.

[11] A. Mowshowitz, "Entropy and the complexity of graphs: III. Graphs with prescribed information content," *The bulletin of mathematical biophysics*, vol. 30, no. 3, pp. 387–414, 1968.

[12] A. Mowshowitz, "Entropy and the complexity of graphs: IV. Entropy measures and graphical structure," *The bulletin of mathematical biophysics*, vol. 30, no. 4, pp. 533–546, 1968.

[13] A. Mowshowitz and M. Dehmer, "Entropy and the complexity of graphs revisited," *Entropy*, vol. 14, no. 3, pp. 559–570, 2012.

[14] A. N. Kolmogorov, "Three approaches to the quantitative definition of information," *Problems of information transmission*, vol. 1, no. 1, pp. 1–7, 1965.

[15] M. Li, P. Vitányi, *et al.*, *An introduction to Kolmogorov complexity and its applications*, vol. 3. Springer, 2008.

[16] P. Vitányi, "How incomputable is Kolmogorov complexity?," *Entropy*, vol. 22, no. 4, p. 408, 2020.

[17] L. Staiger, "A tight upper bound on Kolmogorov complexity and uniformly optimal prediction," *Theory of Computing systems*, vol. 31, no. 3, pp. 215–229, 1998.

[18] M. Morzy, T. Kajdanowicz, and P. Kazienko, "On measuring the complexity of networks: Kolmogorov complexity versus entropy," *Complexity*, vol. 2017, 2017.

[19] H. Zenil, N. A. Kiani, and J. Tegnér, "Low-algorithmic-complexity entropy-deceiving graphs," *Physical Review E*, vol. 96, no. 1, p. 012308, 2017.

[20] G. M. Constantine, "Graph complexity and the Laplacian matrix in blocked experiments," *Linear and Multilinear Algebra*, vol. 28, no. 1-2, pp. 49–56, 1990.

[21] D. Bonchev, "Complexity analysis of yeast proteome network," *Chemistry & biodiversity*, vol. 1, no. 2, pp. 312–326, 2004.

[22] S. Bertz and T. Sommer, "Rigorous mathematical approaches to strategic bonds and synthetic analysis based on conceptually simple new complexity indices," *Chemical Communications*, no. 24, pp. 2409–2410, 1997.

[23] P. Pudlák, V. Rödl, and P. Savickỳ, "Graph complexity," *Acta Informatica*, vol. 25, no. 5, pp. 515–535, 1988.

[24] M. Dehmer and A. Mowshowitz, "Generalized graph entropies," *Complexity*, vol. 17, no. 2, pp. 45–50, 2011.

[25] A. Rényi, "On measures of entropy and information," in *Proceedings of the Fourth Berkeley Symposium on Mathematical Statistics and Probability, Volume 1: Contributions to the Theory of Statistics*, pp. 547–561, University of California Press, 1961.

[26] M. W. Akhtar, S. A. Hassan, R. Ghaffar, H. Jung, S. Garg, and M. S. Hossain, "The shift to 6G communications: vision and requirements," *Human-centric Computing and Information Sciences*, vol. 10, no. 1, pp. 1–27, 2020.

[27] Y. Lim, U. Kang, and C. Faloutsos, "Slashburn: Graph compression and mining beyond caveman communities," *IEEE Transactions on Knowledge and Data Engineering*, vol. 26, no. 12, pp. 3077–3089, 2014.

[28] R. A. Rossi and R. Zhou, "Graphzip: a clique-based sparse graph compression method," *Journal of Big Data*, vol. 5, no. 1, pp. 1–14, 2018.

[29] C. E. Shannon, "A mathematical theory of communication," *The Bell system technical journal*, vol. 27, no. 3, pp. 379–423, 1948.

[30] B. Donnet and T. Friedman, "Internet topology discovery: a survey," *IEEE Communications Surveys & Tutorials*, vol. 9, no. 4, pp. 56–69, 2007.

[31] C. Press, "Ccie professional development routing tcp/ip volume i," 2006.

[32] A. S. Tanenbaum, *Computer networks*. Pearson Education India, 2003.

[33] Z. Manap, B. M. Ali, C. K. Ng, N. K. Noordin, and A. Sali, "A review on hierarchical routing protocols for wireless sensor networks," *Wireless personal communications*, vol. 72, no. 2, pp. 1077–1104, 2013.

[34] T. Behera, U. Samal, and S. Mohapatra, "Routing protocols," in *Computational Intelligence in Sensor Networks*, pp. 79–99, Springer, 2019.

[35] Z. Wu, S. Pan, F. Chen, G. Long, C. Zhang, and S. Y. Philip, "A comprehensive survey on graph neural networks," *IEEE transactions on neural networks and learning systems*, vol. 32, no. 1, pp. 4–24, 2020.

[36] E. Rossi, B. Chamberlain, F. Frasca, D. Eynard, F. Monti, and M. M. Bronstein, "Temporal graph networks for deep learning on dynamic graphs," *CoRR*, vol. abs/2006.10637, 2020.

[37] D. J. Watts and S. H. Strogatz, "Collective dynamics of 'small-world' networks," *nature*, vol. 393, no. 6684, pp. 440–442, 1998.

[38] M. Dehmer and A. Mowshowitz, "A history of graph entropy measures," *Information Sciences*, vol. 181, no. 1, pp. 57–78, 2011.

[39] E. Trucco, "A note on the information content of graphs," *The bulletin of mathematical biophysics*, vol. 18, pp. 129–135, 1956.

[40] T. P. Peixoto, "Entropy of stochastic blockmodel ensembles," *Physical Review E*, vol. 85, no. 5, p. 056122, 2012.

[41] Y. Choi and W. Szpankowski, "Compression of graphical structures: Fundamental limits, algorithms, and experiments," *IEEE Transactions on Information Theory*, vol. 58, no. 2, pp. 620–638, 2012.

[42] M.-A. Badiu and J. P. Coon, "Structural complexity of one-dimensional random geometric graphs," *IEEE Transactions on Information Theory*, 2022.

[43] J. Han, T. Guo, Q. Zhou, W. Han, B. Bai, and G. Zhang, "Structural entropy of the stochastic block models," *Entropy*, vol. 24, no. 1, p. 81, 2022.

[44] G. Bianconi, "Entropy of network ensembles," *Physical Review E*, vol. 79, no. 3, p. 036114, 2009.

[45] G. Bianconi, A. C. Coolen, and C. J. P. Vicente, "Entropies of complex networks with hierarchically constrained topologies," *Physical Review E*, vol. 78, no. 1, p. 016114, 2008.

[46] C. Rovelli and F. Vidotto, "Single particle in quantum gravity and braunstein-ghosh-severini entropy of a spin network," *Physical Review D*, vol. 81, no. 4, p. 044038, 2010.

[47] L. Bogacz, Z. Burda, and B. Wacław, "Homogeneous complex networks," *Physica A: Statistical Mechanics and its Applications*, vol. 366, pp. 587–607, 2006.

[48] K. Anand and G. Bianconi, "Entropy measures for networks: Toward an information theory of complex topologies," *Physical Review E*, vol. 80, no. 4, p. 045102, 2009.

[49] M. Besta and T. Hoefler, "Survey and taxonomy of lossless graph compression and space-efficient graph representations," *CoRR*, vol. abs/1806.01799, 06 2018.

[50] P. Boldi and S. Vigna, "The webgraph framework I: compression techniques," in *Proceedings of the 13th international conference on World Wide Web*, pp. 595–602, 2004.

[51] A. Apostolico and G. Drovandi, "Graph compression by BFS," *Algorithms*, vol. 2, no. 3, pp. 1031–1044, 2009.

[52] S. Maneth and F. Peternek, "Grammar-based graph compression," *Information Systems*, vol. 76, pp. 19–45, 2018.

[53] N. Larsson and A. Moffat, "Offline dictionary-based compression," in *Proceedings DCC'99 Data Compression Conference (Cat. No. PR00096)*, pp. 296–305, 1999.

[54] W. Fan, J. Li, X. Wang, and Y. Wu, "Query preserving graph compression," in *Proceedings of the 2012 ACM SIGMOD international conference on management of data*, pp. 157–168, 2012.

[55] T. Feder and R. Motwani, "Clique partitions, graph compression and speeding-up algorithms," *Journal of Computer and System Sciences*, vol. 51, no. 2, pp. 261–272, 1995.

[56] N. Deo and B. Litow, "A structural approach to graph compression," in *Proc. of the 23th MFCS Workshop on Communications*, pp. 91–101, Citeseer, 1998.

[57] T. M. Cover and J. A. Thomas, *Elements of Information Theory (Wiley Series in Telecommunications and Signal Processing)*. USA: Wiley-Interscience, 2006.

[58] C. E. Shannon, "Prediction and entropy of printed english," *Bell system technical journal*, vol. 30, no. 1, pp. 50–64, 1951.

[59] E. N. Gilbert, "Random graphs," *Ann. Math. Statist.*, vol. 30, pp. 1141–1144, 12 1959.

[60] I. Csiszár, "I-divergence geometry of probability distributions and minimization problems," *The annals of probability*, pp. 146–158, 1975.

[61] A.-L. Barabási and R. Albert, "Emergence of scaling in random networks," *science*, vol. 286, no. 5439, pp. 509–512, 1999.

[62] E. N. Gilbert, "Random graphs," *The Annals of Mathematical Statistics*, vol. 30, no. 4, pp. 1141–1144, 1959.

[63] R. Kumar, P. Raghavan, S. Rajagopalan, D. Sivakumar, A. Tomkins, and E. Upfal, "Stochastic models for the web graph," in *Proceedings 41st Annual Symposium on Foundations of Computer Science*, pp. 57–65, IEEE, 2000.

[64] N. Pržulj, D. G. Corneil, and I. Jurisica, "Modeling interactome: scale-free or geometric?," *Bioinformatics*, vol. 20, pp. 3508–3515, 07 2004.

[65] Delarue, François, "Mean field games: A toy model on an Erdös-Renyi graph.," *ESAIM: Procs*, vol. 60, pp. 1–26, 2017.

[66] T. Cover, "Enumerative source encoding," *IEEE Transactions on Information Theory*, vol. 19, no. 1, pp. 73–77, 1973.

[67] D. A. Huffman, "A method for the construction of minimum-redundancy codes," *Proceedings of the IRE*, vol. 40, no. 9, pp. 1098–1101, 1952.

[68] D. Berend and A. Kontorovich, "A sharp estimate of the binomial mean absolute deviation with applications," *Statistics & Probability Letters*, vol. 83, no. 4, pp. 1254–1259, 2013.

[69] C. Knessl, "Integral representations and asymptotic expansions for Shannon and Renyi entropies," *Applied mathematics letters*, vol. 11, no. 2, pp. 69–74, 1998.

[70] P. W. Holland, K. B. Laskey, and S. Leinhardt, "Stochastic blockmodels: First steps," *Social networks*, vol. 5, no. 2, pp. 109–137, 1983.

[71] E. Abbe, "Graph compression: The effect of clusters," in *2016 54th Annual Allerton Conference on Communication, Control, and Computing (Allerton)*, pp. 1–8, IEEE, 2016.

[72] E. Abbe, A. S. Bandeira, and G. Hall, "Exact recovery in the stochastic block model," *IEEE Transactions on information theory*, vol. 62, no. 1, pp. 471–487, 2015.

[73] L. Bienvenu and W. Merkle, "Reconciling data compression and Kolmogorov complexity," in *International Colloquium on Automata, Languages, and Programming*, pp. 643–654, Springer, 2007.

[74] C. Faloutsos and V. Megalooikonomou, "On data mining, compression, and Kolmogorov complexity," *Data mining and knowledge discovery*, vol. 15, no. 1, pp. 3–20, 2007.

[75] D. Bonchev, "Kolmogorov's information, Shannon's entropy, and topological complexity of molecules," *Bulgarian chemical communications*, vol. 28, no. 3-4, pp. 567–582, 1995.

[76] H. Zenil, N. A. Kiani, and J. Tegnér, "Methods of information theory and algorithmic complexity for network biology," in *Seminars in cell & developmental biology*, vol. 51, pp. 32–43, Elsevier, 2016.

[77] S. Maneth and F. Peternek, "A survey on methods and systems for graph compression," *arXiv preprint arXiv:1504.00616*, 2015.

[78] M. Besta and T. Hoefler, "Survey and taxonomy of lossless graph compression and space-efficient graph representations," *arXiv preprint arXiv:1806.01799*, 2018.

[79] A. M. Turing *et al.*, "On computable numbers, with an application to the entscheidungsproblem," *J. of Math*, vol. 58, no. 345-363, p. 5, 1936.

[80] M. J. Golin and Y. C. Leung, "Unhooking circulant graphs: A combinatorial method for counting spanning trees and other parameters," in *International Workshop on Graph-Theoretic Concepts in Computer Science*, pp. 296–307, Springer, 2004.

[81] H. Buhrman, M. Li, J. Tromp, and P. Vitányi, "Kolmogorov random graphs and the incompressibility method," *SIAM Journal on Computing*, vol. 29, no. 2, pp. 590–599, 1999.

[82] W. W. Zachary, "An information flow model for conflict and fission in small groups," *Journal of anthropological research*, vol. 33, no. 4, pp. 452–473, 1977.

[83] S. N. Dorogovtsev and J. F. Mendes, "Evolution of networks," *Advances in physics*, vol. 51, no. 4, pp. 1079–1187, 2002.

[84] M. Bell, S. Perera, M. Piraveenan, M. Bliemer, T. Latty, and C. Reid, "Network growth models: A behavioural basis for attachment proportional to fitness," *Scientific reports*, vol. 7, no. 1, p. 42431, 2017.

[85] L. A. Bunimovich, D. Smith, and B. Z. Webb, "Specialization models of network growth," *Journal of Complex Networks*, vol. 7, no. 3, pp. 375–392, 2019.

[86] D. L. Banks and K. M. Carley, "Models for network evolution," *The Journal of Mathematical Sociology*, vol. 21, no. 1-2, pp. 173–196, 1996.

[87] R. Toivonen, L. Kovanen, M. Kivelä, J.-P. Onnela, J. Saramäki, and K. Kaski, "A comparative study of social network models: Network evolution models and nodal attribute models," *Social Networks*, vol. 31, no. 4, pp. 240–254, 2009.

[88] M. Boguná and R. Pastor-Satorras, "Epidemic spreading in correlated complex networks," *Physical Review E*, vol. 66, no. 4, p. 047104, 2002.

[89] R. Lambiotte, V. D. Blondel, C. De Kerchove, E. Huens, C. Prieur, Z. Smoreda, and P. Van Dooren, "Geographical dispersal of mobile communication networks," *Physica A: Statistical Mechanics and its Applications*, vol. 387, no. 21, pp. 5317–5325, 2008.

[90] R. G. Clegg, R. Landa, H. Haddadi, and M. Rio, "Measuring the likelihood of models for network evolution," in *IEEE INFOCOM Workshops 2009*, pp. 1–6, IEEE, 2009.

[91] R. Fani and M. Fondi, "Origin and evolution of metabolic pathways," *Physics of Life Reviews*, vol. 6, no. 1, pp. 23–52, 2009.

[92] K. Zhao, A. Halu, S. Severini, and G. Bianconi, "Entropy rate of nonequilibrium growing networks," *Physical Review E*, vol. 84, no. 6, p. 066113, 2011.

[93] A. G. Pakes, "Some conditions for ergodicity and recurrence of Markov chains," *Operations Research*, vol. 17, no. 6, pp. 1058–1061, 1969.

[94] R. Albert and A.-L. Barabási, "Statistical mechanics of complex networks," *Reviews of modern physics*, vol. 74, no. 1, p. 47, 2002.

[95] E. M. Jin, M. Girvan, and M. E. Newman, "Structure of growing social networks," *Physical review E*, vol. 64, no. 4, p. 046132, 2001.

[96] G. D. Birkhoff, "Proof of the ergodic theorem," *Proceedings of the National Academy of Sciences*, vol. 17, no. 12, pp. 656–660, 1931.

[97] L. Rabiner and B. Juang, "An introduction to hidden Markov models," *IEEE ASSP Magazine*, vol. 3, no. 1, pp. 4–16, 1986.

[98] L. R. Rabiner, "A tutorial on hidden Markov models and selected applications in speech recognition," *Proceedings of the IEEE*, vol. 77, no. 2, pp. 257–286, 1989.

[99] S. M. Ross, J. J. Kelly, R. J. Sullivan, W. J. Perry, D. Mercer, R. M. Davis, T. D. Washburn, E. V. Sager, J. B. Boyce, and V. L. Bristow, *Stochastic processes*, vol. 2. Wiley New York, 1996.

[100] S. Karlin and J. L. McGregor, "The differential equations of birth-and-death processes, and the stieltjes moment problem," *Transactions of the American Mathematical Society*, vol. 85, no. 2, pp. 489–546, 1957.

[101] S. C. Ergen, "Zigbee/ieee 802.15. 4 summary," *UC Berkeley, September*, vol. 10, no. 17, p. 11, 2004.

[102] R. P. Draves, C. King, S. Venkatachary, and B. D. Zill, "Constructing optimal IP routing tables," in *IEEE INFOCOM'99. Conference on Computer Communications. Proceedings. Eighteenth Annual Joint Conference of the IEEE Computer*

and Communications Societies. The Future is Now (Cat. No. 99CH36320), vol. 1, pp. 88–97, IEEE, 1999.

[103] J. Felsenstein, *Inferring phylogenies*, vol. 2. Sinauer associates Sunderland, MA, 2004.

[104] M. A. Marcinkiewicz, "Building a large annotated corpus of English: The Penn Treebank," *Using Large Corpora*, vol. 273, 1994.

[105] J. Barnes and P. Hut, "A hierarchical O (N log N) force-calculation algorithm," *nature*, vol. 324, no. 6096, pp. 446–449, 1986.

[106] J. C. Kieffer, E.-H. Yang, and W. Szpankowski, "Structural complexity of random binary trees," in *2009 IEEE International Symposium on Information Theory*, pp. 635–639, IEEE, 2009.

[107] Z. Golebiewski, A. Magner, and W. Szpankowski, "Entropy of some general plane trees," in *2017 IEEE International Symposium on Information Theory (ISIT)*, pp. 301–305, IEEE, 2017.

[108] J. Jansson, K. Sadakane, and W.-K. Sung, "Ultra-succinct representation of ordered trees with applications," *Journal of Computer and System Sciences*, vol. 78, no. 2, pp. 619–631, 2012.

[109] H. Prüfer, "Neuer beweis eines satzes über permutationen," *Arch. Math. Phys*, vol. 27, no. 1918, pp. 742–744, 1918.

[110] E. Mäkinen, "Generating random binary trees—a survey," *Information Sciences*, vol. 115, no. 1-4, pp. 123–136, 1999.

[111] M. Drmota, *Random trees: an interplay between combinatorics and probability*. Springer Science & Business Media, 2009.

[112] R. D. Mauldin, W. D. Sudderth, and S. C. Williams, "Polya trees and random distributions," *The Annals of Statistics*, pp. 1203–1221, 1992.

[113] H. W. Watson and F. Galton, "On the probability of the extinction of families," *The Journal of the Anthropological Institute of Great Britain and Ireland*, vol. 4, pp. 138–144, 1875.

[114] J. W. Moon, "The distance between nodes in recursive trees," *London Mathematical Society Lecture Notes*, vol. 13, pp. 125–132, 1974.

[115] D. Najock and C. Heyde, "On the number of terminal vertices in certain random trees with an application to stemma construction in philology," *Journal of Applied Probability*, vol. 19, no. 3, pp. 675–680, 1982.

[116] J. L. Gastwirth and P. Bhattacharya, "Two probability models of pyramid or chain letter schemes demonstrating that their promotional claims are unreliable," *Operations Research*, vol. 32, no. 3, pp. 527–536, 1984.

[117] S. Janson, "Simply generated trees, conditioned Galton-Watson trees, random allocations and condensation," *Discrete Mathematics & Theoretical Computer Science*, no. Proceedings, 2012.

[118] L. Devroye, "Branching processes and their applications in the analysis of tree structures and tree algorithms," in *Probabilistic methods for algorithmic discrete mathematics*, pp. 249–314, Springer, 1998.

[119] J. Katajainen and E. Mäkinen, "Tree compression and optimization with applications," *International Journal of Foundations of Computer Science*, vol. 1, no. 04, pp. 425–447, 1990.

[120] S. Caminiti, I. Finocchi, and R. Petreschi, "On coding labeled trees," *Theoretical computer science*, vol. 382, no. 2, pp. 97–108, 2007.

[121] P. Bille, I. L. Gørtz, G. M. Landau, and O. Weimann, "Tree compression with top trees," *Information and Computation*, vol. 243, pp. 166–177, 2015.

[122] M. Lohrey, S. Maneth, and R. Mennicke, "XML tree structure compression using RePair," *Information Systems*, vol. 38, no. 8, pp. 1150–1167, 2013.

[123] M. Hosseini, "A survey of data compression algorithms and their applications," *Network Systems Laboratory, School of Computing Science, Simon Fraser University, BC, Canada*, 2012.

[124] T. Bell, I. H. Witten, and J. G. Cleary, "Modeling for text compression," *ACM Computing Surveys (CSUR)*, vol. 21, no. 4, pp. 557–591, 1989.

[125] M. A. Rahman and M. Hamada, "Lossless image compression techniques: A state-of-the-art survey," *Symmetry*, vol. 11, no. 10, p. 1274, 2019.

[126] OEIS Foundation Inc., "The On-Line Encyclopedia of Integer Sequences," 2022. Published electronically at `http://oeis.org`.

[127] G. Polya and R. C. Read, *Combinatorial enumeration of groups, graphs, and chemical compounds*. Springer Science & Business Media, 2012.

[128] T. Koshy, *Catalan numbers with applications*. Oxford University Press, 2008.

[129] R. Sedgewick and P. Flajolet, *An introduction to the analysis of algorithms*. Addison-Wesley Longman Publishing Co., Inc., 1996.

[130] M. L. Fredman, R. Sedgewick, D. D. Sleator, and R. E. Tarjan, "The pairing heap: A new form of self-adjusting heap," *Algorithmica*, vol. 1, no. 1-4, pp. 111–129, 1986.

[131] A. Cayley, *A theorem on trees*, vol. 13 of *Cambridge Library Collection - Mathematics*, p. 26–28. Cambridge University Press, 2009.

[132] A. Meir and J. W. Moon, "On the altitude of nodes in random trees," *Canadian journal of Mathematics*, vol. 30, no. 5, pp. 997–1015, 1978.

[133] A. Z. Broder, "Generating random spanning trees," in *FOCS*, vol. 89, pp. 442–447, 1989.

[134] J. A. Kelner and A. Madry, "Faster generation of random spanning trees," in *2009 50th Annual IEEE Symposium on Foundations of Computer Science*, pp. 13–21, IEEE, 2009.

[135] A. Madry, D. Straszak, and J. Tarnawski, "Fast generation of random spanning trees and the effective resistance metric," in *Proceedings of the twenty-sixth annual ACM-SIAM symposium on Discrete algorithms*, pp. 2019–2036, SIAM, 2014.

[136] C. Moore and S. Mertens, *The nature of computation*. OUP Oxford, 2011.

[137] J. L. W. V. Jensen, "Sur les fonctions convexes et les inégalités entre les valeurs moyennes," *Acta mathematica*, vol. 30, no. 1, pp. 175–193, 1906.

[138] M. Newman, *Networks*. Oxford university press, 2018.

[139] J. Ziv and A. Lempel, "A universal algorithm for sequential data compression," *IEEE Transactions on information theory*, vol. 23, no. 3, pp. 337–343, 1977.

[140] J. Ziv and A. Lempel, "Compression of individual sequences via variable-rate coding," *IEEE transactions on Information Theory*, vol. 24, no. 5, pp. 530–536, 1978.

[141] T. A. Welch, "A technique for high-performance data compression," *Computer*, vol. 17, no. 06, pp. 8–19, 1984.

[142] G. Cardona, F. Rosselló, and G. Valiente, "Extended Newick: it is time for a standard representation of phylogenetic networks," *BMC bioinformatics*, vol. 9, no. 1, pp. 1–8, 2008.

[143] J. I. Munro and V. Raman, "Succinct representation of balanced parentheses and static trees," *SIAM Journal on Computing*, vol. 31, no. 3, pp. 762–776, 2001.

[144] D. Benoit, E. D. Demaine, J. I. Munro, R. Raman, V. Raman, and S. S. Rao, "Representing trees of higher degree," *Algorithmica*, vol. 43, pp. 275–292, 2005.

[145] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein, *Introduction to algorithms*. MIT press, 2022.

[146] S. A. Savari, "Redundancy of the Lempel-Ziv incremental parsing rule," *IEEE Transactions on Information Theory*, vol. 43, no. 1, pp. 9–21, 1997.

[147] S. A. Savari, "Redundancy of the Lempel-Ziv-Welch code," in *Proceedings DCC'97. Data Compression Conference*, pp. 191–200, IEEE, 1997.

[148] E. Plotnik, M. J. Weinberger, and J. Ziv, "Upper bounds on the probability of sequences emitted by finite-state sources and on the redundancy of the Lempel-Ziv algorithm," *IEEE transactions on information theory*, vol. 38, no. 1, pp. 66–72, 1992.

[149] J. Pearl, *Heuristics: intelligent search strategies for computer problem solving*. Addison-Wesley Longman Publishing Co., Inc., 1984.

[150] W. R. Heinzelman, J. Kulik, and H. Balakrishnan, "Adaptive protocols for information dissemination in wireless sensor networks," in *Proceedings of the 5th Annual ACM/IEEE International Conference on Mobile Computing and Networking*, pp. 174–185, 1999.

[151] J. Kulik, W. Heinzelman, and H. Balakrishnan, "Negotiation-based protocols for disseminating information in wireless sensor networks," *Wireless Networks*, vol. 8, no. 2, pp. 169–185, 2002.

[152] C. Intanagonwiwat, R. Govindan, and D. Estrin, "Directed diffusion: A scalable and robust communication paradigm for sensor networks," in *Proceedings of the 6th Annual International Conference on Mobile Computing and Networking*, pp. 56–67, 2000.

[153] N. Sadagopan, B. Krishnamachari, and A. Helmy, "The acquire mechanism for efficient querying in sensor networks," in *Proceedings of the First IEEE International Workshop on Sensor Network Protocols and Applications, 2003.*, pp. 149–155, IEEE, 2003.

[154] J. N. Al-Karaki and A. E. Kamal, "Routing techniques in wireless sensor networks: a survey," *IEEE Wireless Communications*, vol. 11, no. 6, pp. 6–28, 2004.

[155] B. Bhushan and G. Sahoo, "Routing protocols in wireless sensor networks," in *Computational Intelligence in Sensor Networks*, pp. 215–248, Springer, 2019.

[156] W. B. Heinzelman, A. P. Chandrakasan, and H. Balakrishnan, "An application-specific protocol architecture for wireless microsensor networks," *IEEE Transactions on Wireless Communications*, vol. 1, no. 4, pp. 660–670, 2002.

[157] S. Lindsey and C. S. Raghavendra, "PEGASIS: Power-efficient gathering in sensor information systems," in *Proceedings, IEEE Aerospace Conference*, vol. 3, pp. 3–3, IEEE, 2002.

[158] A. Manjeshwar and D. P. Agrawal, "TEEN: a routing protocol for enhanced efficiency in wireless sensor networks," in *Proceedings of the 1st International*

Workshop on Parallel and Distributed Computing Issues in Wireless Networks and Mobile Computing, vol. 1, p. 189, 2001.

[159] Q. Li, J. Aslam, and D. Rus, "Hierarchical power-aware routing in sensor networks," in *Proceedings of the DIMACS Workshop on Pervasive Networking*, vol. 10, Citeseer, 2001.

[160] L. Ma, H. Leung, and D. Li, "Hybrid TDMA/CDMA MAC protocol for wireless sensor networks," *Journal of Networks*, vol. 9, no. 10, p. 2665, 2014.

[161] M. H. Homaei, "Low Energy Adaptive Clustering Hierarchy protocol (LEACH)." `https://www.mathworks.com/matlabcentral/fileexchange/44073-low-energy-adaptive-clustering-hierarchy-protocol-leach`, 2022. [Online; accessed September 09, 2022].

[162] H.-s. Kim and K.-j. Han, "A power efficient routing protocol based on balanced tree in wireless sensor networks," in *First International Conference on Distributed Frameworks for Multimedia Applications*, pp. 138–143, IEEE, 2005.

[163] F. Scarselli, M. Gori, A. C. Tsoi, M. Hagenbuchner, and G. Monfardini, "The graph neural network model," *IEEE transactions on neural networks*, vol. 20, no. 1, pp. 61–80, 2008.

[164] L. Wu, P. Cui, J. Pei, and L. Zhao, *Graph Neural Networks: Foundations, Frontiers, and Applications*. Singapore: Springer Singapore, 2022.

[165] S. M. Kazemi, R. Goel, K. Jain, I. Kobyzev, A. Sethi, P. Forsyth, and P. Poupart, "Representation learning for dynamic graphs: A survey," *The Journal of Machine Learning Research*, vol. 21, no. 1, pp. 2648–2720, 2020.

[166] Z. Goldfeld and Y. Polyanskiy, "The information bottleneck problem and its applications in machine learning," *IEEE Journal on Selected Areas in Information Theory*, vol. 1, no. 1, pp. 19–38, 2020.

[167] R. Shwartz-Ziv and N. Tishby, "Opening the black box of deep neural networks via information," *arXiv preprint arXiv:1703.00810*, 2017.

[168] R. Gray and J. Kieffer, "Mutual information rate, distortion, and quantization in metric spaces," *IEEE Transactions on Information Theory*, vol. 26, no. 4, pp. 412–422, 1980.

[169] A. Kraskov, H. Stögbauer, and P. Grassberger, "Estimating mutual information," *Physical review E*, vol. 69, no. 6, p. 066138, 2004.

[170] N. Carrara and J. Ernst, "On the estimation of mutual information," in *Proceedings*, vol. 33, p. 31, MDPI AG, 2020.

[171] L. M. Costa, J. Colaço, A. M. Carvalho, S. Vinga, and A. S. Teixeira, "Using Markov chains and temporal alignment to identify clinical patterns in Dementia," *Journal of Biomedical Informatics*, vol. 140, p. 104328, 2023.

[172] P. Delgosha and V. Anantharam, "Universal lossless compression of graphical data," *IEEE Transactions on Information Theory*, vol. 66, no. 11, pp. 6962–6976, 2020.