GENERATING NETWORKS BY LEARNING HYPEREDGE REPLACEMENT GRAMMARS

A Dissertation

Submitted to the Graduate School
of the University of Notre Dame
in Partial Fulfillment of the Requirements
for the Degree of

Doctor of Philosophy

by

Salvador Aguiñaga

Tim Weninger, Director

Graduate Program in Computer Science and Engineering
Notre Dame, Indiana
July 2018

GENERATING NETWORKS BY LEARNING HYPEREDGE REPLACEMENT GRAMMARS

Abstract

by

Salvador Aguiñaga

Network modeling is critical and central to the study of complex systems. Modeling enables researchers to examine emergent behavior and related phenomena from the milieu of interacting patterns at the local level. These complex systems are diverse, ranging from the global economy, neuroscience, protein folding molecular interactions, to the Internet. Evaluating network models on their ability to automatically learn the underlying features is integral to algorithm development in many areas of computational science.

Here we describe methods and develop algorithms that extend and evaluate hyperedge replacement grammars, a formalism in formal language theory. We detail extensions for model-inference on real-world networks and graph generation. Discovering patterns involved in system behavior to build models for real-world systems that preserve many of the network properties during the generation step is the central focus of this work. Growing similar structures at various scale is also crucial to the evolution of the scientific tools required in today's information landscape. Experimental results demonstrate that hyperedge replacement grammars offer a new way to learn network features that facilitate compelling graphical structure generation that advances network science in areas of modeling and network analysis.

To my brilliant and beautiful wife, Kristin and to my darling children, Izzy, Adri, and Vivi.

CONTENTS

FIGURES

TABLES

ACKNOWLEDGMENTS

First, I am indebted to my advisor, Tim Weninger, one of the most positive, generous, and extraordinary individuals I have ever met. I consider myself lucky for the opportunity to work with him and all he has taught me. I am grateful for his support and generous advice for the last four years. The work presented in this thesis would not have been possible without him.

Special thanks go out to Joyce Yeats, Ginny Watterson, M. D. McNally, and Diane Wordinger in the Department of Computer Science and Engineering. Joyce has always guided me and helped navigate all aspects of the graduate program from day one. Their help and dedication to graduate students' success are invaluable. I would like to thank iCENSA. My interaction with the students and professors in iCENSA has had an impact on how I see the world of research and science. I would like to extend another especial thanks to Ms. Jasmin Botello for making iCENSA a great place to work. iCENSA would not be the great place it is without her. Her enthusiasm and commitment to both, professors and students, makes us feel that we belong and are part of an exceptional community of researchers.

I would like to thank every one of my collaborators. They are a fantastic group of talented individuals. I would like to acknowledge Nitesh Chawla and Aastha Nigam, Corey Pennycuff, Cindy Wang who is now a graduate student at CMU. To David Chiang, I would like to extend my thanks for his guidance, and for challenging me and offering insights that have shaped my work. Before working with Tim, Chris Poellabauer supported me and worked with me on a pair of exciting projects. I am grateful to have gotten to know him and his students. Chris made feel at home

my first two years of graduate school, and I am thankful for his enthusiasm and guidance during my most challenging years of graduate school. I would like to thank and acknowledge my thesis committee for their support and advice: Professor James Evans, Professor Nitesh Chawla, and Professor David Chiang.

Next, I would like to acknowledge the entire Weninger Lab especially my lab mates: Maria Glenski, Corey Pennycuff, and Baoxu (Dash) Shi. I am especially thankful to Dash for engaging with me on many exciting conversations and for allowing me and my family to be part of a special day: Jessie and Dash's wedding day. The DSG-ND group is an incredible collection of individuals; it has been my honor to know them and worked with them.

Lastly and most importantly, I would like to thank my love, my best friend, and the smartest person I know, my beautiful wife, Kristin. Her belief in me, her support, advice, and her endurance for putting up with me especially during the most trying days shows me how fortunate I am. All that I have accomplished during my tenure as a graduate student I owe it all to her and the love of my beautiful children: Isabella, Adrian, and Violetta. Their zest, curiosity, love, and affection kept me going. I am indeed a lucky man.

CHAPTER 1

INTRODUCTION

Systems, natural or artificial, can be characterized by their function or their purpose. A system consists of a set of entities and the interactions within. These interactions, also referred to as relationships, give rise to global behavior of the system that is characterized as either simple or complex. Complex systems can differ from simple systems by way of lattices or random graphs from their intrinsic and non-trivial topological features and patterns of connections found in real-world networks. Mathematics aims to represents these systems through a powerful abstraction called a graph or a network. Network science applies to problems across fields as diverse as medicine, healthcare, politics, economics, and social science.

Representing complex systems as a graph, or as a network, allows us to draw on methods and theories from mathematics (graph theory), physics (statistical mechanics), computer science (data mining), and sociology (social structure). This combination of methods leads to the development of predictive models that shed light into physical, biological, and social phenomena [119]. Model development allows us to understand the many features at the heart of real-world graphs. For example, a significant property of many graphs is community structure. This property examines how tightly connected entities are (or not) in a graph. Combining statistical mechanics, sociometry, and graph theory allows for methods to explore where nodes fall in relation to a community's boundary. Evaluating community detection methods on a real-world graph often requires the same evaluation on computer-generated graphs [38, 74]. Constructing computer-generated or synthetic graphs to mimic and

preserve one or more of these network properties is challenging. Automatically generating graphs using novel computational approaches is one way to address this. The rate at which people and machines create and consume information pushes the boundaries of our technology to be able to gain insight and new knowledge. The new digital age begs for innovative tools to help us analyze the deluge of data in order to glean new knowledge and insights [102].

Capturing and utilizing certain properties in a graph of interest is critical to the development or generation of new graphs. One method is to incorporate certain network features into the graph generation step. For example, if the number of triangles in a reference graph is important, then we want to maintain that network feature in our synthetic graphs; Exponential Random Graphs Model (ERGM) is a useful way to achieve this. Other models excel at generating similar graphs with a degree centrality consistent with properties observed in the original (or reference) graph [48, 75]. This thesis focuses on computational methods that examine graph micro-structure and leverage local connection patterns to infer a model of system structure. The graph model's usefulness includes growing synthetic graphs, graph classification, and generating network architectures. By generating realistic synthetic graphs we can create new networks whose properties are similar to those in the reference or training graph.

1.1 Graph Mining

For the purposes of this thesis, graph mining technologies can be divided into two classes: (1) subgraph mining algorithms and (2) graph generating models.

Subgraph Mining. Rooted in data mining and knowledge discovery, subgraph mining methods are efficient and scalable algorithms for traditional frequent itemset mining on graphs [42, 52]. Frequent graph patterns are subgraphs that are found

from a single large graph or a collection of many smaller graphs. A subgraph is deemed to be frequent if it appears more than some user-specified support threshold. Being descriptive models, frequent subgraphs are useful in characterizing graphs and can be used for clustering, classification or other discriminative tasks. Unfortunately, these methods have a so-called "combinatorial explosion" problem [114] wherein the search space grows exponentially with the pattern size. This causes computational headaches, and can also return a massive result set that hinders real world applicability. Recent work that heuristically mines graphs for important or representative subgraphs have been developed in response, but are still limited by their choice of heuristic [77, 92, 110, 120]. Alternatively, researchers characterize a network by counting small subgraphs called graphlets, and therefore forfeit any chance of finding larger, more interesting structures[5, 81, 98]. Overcoming these limitations will require a principled approach that discovers the structures within graphs and is the first research objective of the proposed work.

Graph Generators. Graph generators, like frequent subgraph mining, also find distinguishing characteristics of networks, but go one step further by generating new graphs that "look like" the original graph(s). What a graph looks like includes local graph properties like the counts of frequent subgraphs, but can also include global graph properties like the degree distribution, clustering coefficient, diameter, and assortativity among many others. Early graph generators, like the random graph of Erdős and Reyni [30], the small world network of Watts and Strogatz [119], or the scale free graph of Albert and Barabási [8], did not learn a model from a graph directly, but rather had parameters that could be tuned to generate graphs with certain desirable properties. Recent work in exponential random graphs [100], Kronecker graphs [17, 75], Chung-Lu graphs [22], and their many derivatives [60, 85, 86, 95] create a model from some example graph in order to generate a new graph that has

many of the same *global* properties as the original graph.

Despite their conceptual similarity, subgraph mining algorithms and graph generators have little in common algorithmically. Simply put, they solve different problems. Although Kronecker and Chung-Lu graph generators learn their model from an exemplar graph, only the exponential random graph (ERG) model actually learns a model based on the specific patterns found in the graph. Unfortunately, the ERG model must pre-define all possible patterns, and the complexity of the ERG model is exponential in the number and size of the pre-defined patterns. The standard ERG model is not good at generating new graphs; however the learned model can be informative about the nature of the underlying graph, albeit through the lens of only a handful of small structures, *e.g.*, edges, triangles, 4-cliques [40]. The proposed work will bridge the gap between subgraph mining and graph generation to create a new suite of models and tools that can not only create informative models of real world data, but also generate, extrapolate, and infer new graphs in a precise, principled way.

1.2 Formal Language Theory and Graph Theory

The ideas in this thesis originate from a newfound relationship between graph theory and formal language theory. The relationship between graph theory and formal language theory allows for a *Hyperedge Replacement Grammar* (HRG) to be extracted from any graph without loss of information. Like a context free grammar, but for graphs, the extracted HRG contains the precise building blocks of the network as well as the instructions by which these building blocks ought to be pieced together. Because of the principled way it is constructed, the HRG can even be used to regenerate an isomorphic copy of the original graph. Although the isomorphic guarantees are exciting and important, this thesis will focus instead on finding meaning in the building blocks and their instructions [4].

Just as a context free string grammar generates a string, an HRG can generate a graph by repeatedly choosing a nonterminal A and rewriting it using a production rule $A \rightarrow R$. The replacement hypergraph fragment R can itself have other nonterminal hyperedges, so this process is repeated until there are no more nonterminals in the graph.

HRGs have been studied some time in discrete mathematics and graph theory literature. HRGs are conventionally used to generate graphs with very specific structures, *e.g.*, rings, trees, stars. A drawback of many current applications of HRGs is that their production rules must be manually defined. For example, the production rules that generate a ring-graph are distinct from those that generate a tree, and defining even simple grammars by hand is difficult or impossible. Very recently, Kemp and Tenenbaum developed an inference algorithm that learned probabilities of the HRG's production rules from real world graphs, but they still relied on a handful of rather basic hand-drawn production rules (of a related formalism called vertex replacement grammar) to which probabilities were learned [58]. Kukluk, Holder and Cook were able to define a grammar from frequent subgraphs [24, 47, 64–66], but their methods have a coarse resolution because *frequent* subgraphs only account for a small portion of the overall graph topology.

In this thesis, I investigate a relationship between graph theory and formal language theory that allows for a Hypererdge Replacement Grammar (HRG) to be extracted from a graph. I further show that the HRG can be used to represent the building blocks of these graphs and go on to use this to generate synthetic graphs that remain similar to the original graph.

This work builds on earlier efforts by my collaborators, Tim Weninger and David Chiang, who pieced together the utility of HRG for graphs other than abstract meaning representations (AMR) [20].

TABLE 1.1

THESIS OVERVIEW

Chapter	**Title**	**Research Problem**
3	HRG model inference and graph generation	Can an HRG capture graph features to grow graphs with properties matching the input or reference graph?
4	Experimental evaluation of HRG graph generation	Do HRGs actually generate realistic graphs?
5	Model Robustness	How well does the model captures essential features in the real world network?
6	Tree decomposition	What controls or biases the production rules?

1.3 Contribution

This thesis is organized around the following themes: (I) HRG for graph generation [4], (II) Experimental evaluation of graph generation, (III) Measures of model resilience [2], and (IV) Understanding model bias resulting from tree decomposition algorithms. A summary of the key challenges for each theme is detailed in Table 1.1.

1.4 Impact

By marrying the fields of graph theory and formal language theory, lessons from the previous 50 years of study in formal language theory, grammars, and much of theoretical computer science can now be applied to graph mining and network science! This thesis takes the steps towards reconciling these disparate fields by asking incisive questions about the extraction, inference, and analysis of network patterns in a mathematically elegant and principled way.

The introduction of the HRG graph model has triggered interest in areas of neural network architecture designs for deep learning. HRG graph model is now taught in undergraduate and graduate Network Science course in the Department of Computer Science and Engineering.

CHAPTER 2

BACKGROUND

2.1 Graphs and Hypergraphs

In discrete mathematics graphs are powerful and versatile abstractions. A graph is usually described as sets of vertices, V, and the interactions between vertices are often represented as edges that connect at most two vertices, E.

A hypergraph H is a generalization of a classical *graph*. A hypergraph consists of a collections of finite sets containing vertices. These sets are called *hyperedges* such that edges may connect any number of vertices.

2.1.1 Graph Properties

The semantics of graphs gives rise to sets of properties, features, or attributes that intrinsically characterize the graph. Graph properties take on the role of measuring the graph. For instance, certain graph properties describe size (the number of edges) and order (the number of nodes or vertices). Other measures help describe the network in greater detail and, in some instances, infer function. For example, the degree is a network property that helps understand a vertex's connectivity to other nodes [10, 33]. The number of edges incident on any given node helps understand each node's importance or influence in the graph.

Other graph properties have been described in scientific literature as far back as the 1930's with Jacob L. Moreno's seminal work in sociometry [83] to the more recent work describing scale-free power law distribution as a common property observed in large networks [8]. For more on the historical unfolding of social network

analysis metrics see the survey's by Wasserman and Faust [117], Freeman [34], and Newman [90].

In this dissertation we employ several graph metrics to analyze graphs and hypergraphs. Although many properties have been discovered and detailed in related literature, we focus on three of the principal properties from which most others can be derived.

- **Degree Distribution.** The degree distribution of a graph is the distribution of the number of edges connecting to a particular vertex. Barabási and Albert initially discovered that the degree distribution of many real world graphs follows a power law distribution such that the number of nodes $N_d \propto d^{-\gamma}$ where $\gamma > 0$ and γ is typically between 2 and 3 [8].

- **Eigenvector Centrality.** The principal eigenvector is often associated with the centrality or "value" of each vertex in the network, where high values indicate an important or central vertex and lower values indicate the opposite. A skewed distribution points to a relatively few "celebrity" vertices and many common nodes. The principal eigenvector value for each vertex is also closely associated with the PageRank and degree value for each node.

- **Hop Plot.** The hop plot of a graph shows the number of vertex-pairs that are reachable within x hops. The hop plot, therefore, is another way to view how quickly a vertex's neighborhood grows as the number of hops increases. As in related work [72] we generate a hop plot by picking 50 random nodes and perform a complete breadth first traversal over each graph.

  The aforementioned network properties primarily focus on statistics of the global network. However, there is mounting evidence arguing that graphlet comparisons are the most complete way to measure the similarity between two graphs [98, 115]. The graphlet distribution succinctly describes the number of small, local substructures that compose the overall graph and therefore more completely represents the details of what a graph "looks like." It is possible for two very dissimilar graphs to have the same degree distributions, hop plots, etc., but it remains difficult for two dissimilar graphs to fool a comparison with the graphlet distribution.

- **Graphlet Correlation Distance** Recent work from systems biology has identified a new metric called the Graphlet Correlation Distance (GCD). The GCD computes the distance between two graphlet correlation matrices – one matrix for each graph [122]. It measures the frequency of the various graphlets present in each graph, *i.e.*, the number of edges, wedges, triangles, squares,

4-cliques, etc., and compares the graphlet frequencies between two graphs. Because the GCD is a distance metric, lower values are better. The GCD can range from $[0, +\infty]$, where the GCD is 0 if the two graphs are isomorphic.

2.2 Graph Models

Graph models are powerful mathematical abstractions frequently and widely used in the natural sciences, engineering, and in the social sciences. A core function of these models is to represent the essential properties of systems or system behaviour at all scales. Moreover, these models consist of a variety of abstract structures that can be classified into state variables (or random variables), which, when linked together, can describe a range of network information from simple to complex network structures.

Swiss mathematician Leonhard Euler is credited as one of the first mathematicians to work on problems using graphical abstractions [31, 89] as far back as the early 1700s. Two hundred years later, the social sciences invested in the study of social networks. By the late 1950s and early sixties, interest in models that generate networks as a mode to better understand them was gaining attention in academia and industry. Currently, significant advancement in the study of network structure using models for generating random graphs was pioneered in by Edgar Gilber, Paul Erdos and Alfred Renyi [29, 37]. Two decades ago Watts, Strogatz, Barabasi, and Albert introduced models of *Small World* networks [119] and *Preferential Attachment* [6] a seminal set of works. With the arrival of the Internet and development of the Web, applied network and graph theory spark feverish interest from both, academia and industry.

2.2.1 Generative Graph Models

Generative models that underlie the ever-growing Web graph have received a great deal attention for some time now. This graph's nodes and edges exhibit power law

distributions based on empirical studies [32]. Network scientists have found that the *preferential attachment* model, which generates a graph by attaching new nodes to popular existing nodes, approximates the empirical growth pattern. Based on this model, and subsequent refinements, network engineers are able to understand the large scale behavior of the Web.

In this dissertation we focus on several different generative graph models: HRG, Kronecker [75], Chung-Lu [23], Block Two-level Erdos Reyni (BTER) [104], and exponential random graph (ERGM) [100] models. Other models, such as the Erdős-Rényi random graph model, the Watts-Strogatz small world model, the preferential attachment generator, etc. are only tangentially relevant to this thesis because they do not learn a model from an empirical graph.

Kronecker graphs operate by learning an initiator matrix and then performing a recursive multiplication of that initiator matrix in order to create an adjacency matrix of the approximate graph. In our case, we use KronFit [75] with default parameters to learn a 2×2 initiator matrix and then use the recursive Kronecker product to generate the graph. Unfortunately, the Kronecker product only creates graphs where the number of nodes is a power of 2, *i.e.*, 2^x, where we chose $x = 15$, $x = 12$, $x = 13$, and $x = 18$ for Enron, ArXiv, Routers and DBLP graphs respectively to match the number of nodes as closely as possible.

The Chung-Lu Graph Model (CL) takes, as input, a degree distribution and generates a new graph of the similar degree distribution and size [21].

Exponential Random Graph Models (ERGMs) are a class of probabilistic models used to directly describe several structural features of a graph [100]. We used default parameters in R's ERGM package [49] to generate graph models for comparison. Exponential random graphs models belong to a class of statistical models, also known as p^* models. They have been used extensively to model social behavior in humans and animals. More recently, ERGMs have been used to model complex neurological

interactions of the brain [16, 41, 106]. Goldenberg *et al.* survey statistical models and discuss how ERGMs are an extension of the Erdos-Renyi-Gilbert model to account for popularity, expansiveness and network effects due to reciprocation [35, 40]. However, in addition to the problem of model degeneracy, ERGMs do not scale well to large graphs, therefore ERGM results are omitted from this thesis.

2.3 Hyperedge Replacement Grammars

This dissertation presents a new graph generation methodology based on the formalism of Hyperedge Replacement Grammars (HRGs). HRGs are a graphical counterpart to context free string grammars used in compilers and natural language processing [26]. Like in a context free string grammar, an HRG contains a set of production rules $\mathcal{P}$, each of which contains a left hand side (LHS) A and a right hand side (RHS) R. In context free string grammars, the LHS must be a nonterminal character, which can be replaced by some set of nonterminal or terminal characters on the RHS of the rule. In HRGs, nonterminals are graph-cliques and a RHS can be any graph (or hypergraph) fragment.

HRGs have been studied for some time in discrete mathematics and graph theory literature. They are conventionally used to generate graphs with very specific structures, *e.g.*, rings, trees, stars. A drawback of many current applications of HRGs is that their production rules must be manually defined. For example, the production rules that generate a ring-graph are distinct from those that generate a tree, and defining even simple grammars by hand is difficult to impossible. Very recently, Kemp and Tenenbaum developed an inference algorithm that learned probabilities of the production rules from real world graphs, but they still relied on a handful of rather basic hand-drawn production rules (of a related formalism called vertex replacement grammar) to which probabilities were learned [58]. Kukluk, Holder and Cook were able to define a grammar from frequent subgraphs [24, 47, 64–66], but

their methods have a coarse resolution because *frequent* subgraphs only account for a small portion of the overall graph topology.

CHAPTER 3

LEARNING HYPEREDGE REPLACEMENT GRAMMARS

Teasing out signatures of interactions buried in overwhelming volumes of information is one of the most basic challenges in scientific research. Understanding how information is organized and how it evolves can help us discover its fundamental underlying properties. Researchers do this when they investigate the relationships between diseases, cell functions, chemicals, or particles, and we all learn new concepts and solve problems by understanding the relationships between the various entities present in our everyday lives. These entities can be represented as networks, or graphs, in which local behaviors can be understood, but whose global view is highly complex.

Discovering and analyzing network patterns to extract useful and interesting patterns (building blocks) is critical to the advancement of many scientific fields. Indeed the most pivotal moments in the development of a scientific field are centered on discoveries about the structure of some phenomena [63]. For example, biologists have agreed that tree structures are useful when organizing the evolutionary history of life [25, 55], and sociologists find that triadic closure underlies community development [28, 43]. In other instances, the structural organization of the entities may resemble a ring, a clique, a star, a constellation, or any number of complex configurations.

Unfortunately, current graph mining research deals with small pre-defined patterns [58, 79] or frequently reoccurring patterns [47, 54, 64, 66], even though interesting and useful information may be hidden in unknown and non-frequent patterns.

Principled strategies for extracting these complex patterns are needed to discover the precise mechanisms that govern network structure and growth. In-depth examination of this mechanism leads to a better understanding of graph patterns involved in structural, topological, and functional properties of complex systems. This is precisely the focus of the present work: to develop and evaluate techniques that learn the building blocks of real-world systems that, in aggregate, succinctly describe the observed interactions expressed in a network.

These networks exhibit a long and varied list of global properties, including heavy-tailed degree distributions [109], and interesting community structures [104] to name a few. Recent work has found that these global properties are products of a graph's local properties [98, 116]. In the present work, our goal is to learn the local structures that, in aggregate, help describe the interactions observed in the network and generalize to applications across a variety of fields like computer vision, computational biology, and graph compression.

The key insight for this task is that a network's *tree decomposition* encodes robust and precise information about the network. A *hyperedge replacement grammar* (HRG), extracted from the tree decomposition, contains graphical rewriting rules that can match and replace graph fragments similar to how a context-free grammar (CFG) rewrites characters in a string. These graph fragments represent a succinct, yet complete description of the building blocks of the network, and the rewriting rules of the HRG describe the instructions on how the graph is pieced together.

Unlike previous models that manually define the space of possible structures [57] or define the grammar by extracting frequent subgraphs [65, 67], our framework can automatically discover the necessary forms and use them to recreate the original graph *exactly* as well as infer generalizations of the original network. Our approach can handle any type of graph and does not make any assumption about the topology of the data.

The HRG framework is divided into two steps: 1) graph model learning and 2) graph generation. After reviewing some of the theoretical foundations of tree decompositions and HRGs, we show how to extract an HRG from a graph. These graph rewriting rules can be applied randomly to generate larger and larger graphs. However, scientists typically have a specific size in mind, so we introduce a fixed-size graph generation algorithm that will apply HRG rules to generate a realistic graph of a user-specified size.

The extraction method was conceived by Prof. David Chiang and the approximate tree decomposition was was contributed by myself and Prof. Weninger. The code was implemented by Prof. Chiang, Prof. Weninger, and myself.

3.1 Preliminaries

The new work in this thesis begins where previous work [4, 20, 69, 99] left off. However, before we begin, some background knowledge is crucial to understand the key insights of our main contributions.

We begin with an arbitrary input *hypergraph* $H = (V, E)$, where V is a finite set of vertices and $E \subseteq V^+$ is a set of *hyperedges*. A hyperedge $e \in E$ can connect one or more ordered vertices and is written $e = (v_1, v_2, \ldots, v_k)$. Common *graphs* (*e.g.*, social networks, Web graphs, information networks) are a particular case of hypergraphs where each edge connects exactly two vertices. For convenience, all of the graphs in this thesis will be *simple*, *connected* and *undirected*, although these restrictions are not vital. In the remainder of this section, we refer mainly to previous developments in tree decompositions and their relationship to hyperedge replacement grammars in order to support the claims made in sections 3 and 4.

3.1.1 Tree Decomposition

In graph theory, all graphs can be decomposed (though not uniquely) into a *tree decomposition* [61]. A tree decomposition of any graph (or any hypergraph) is a tree, each of whose nodes is labeled with nodes and edges from the original graph, such that *vertex cover*, *edge cover* and the *running intersection* properties hold [99], and the "width" of the optimal tree decomposition measures how tree-like a graph is. The reason for the wide interest in finding the tree decomposition of a graph is because many computationally difficult problems can be solved efficiently when the data is constrained to be a tree. For an expanded introduction, we refer the reader to Chapters 9 and 10 of Koller and Friedman's textbook [61].

Within data mining and machine learning, tree decompositions are best known for their role in exact inference in probabilistic graphical models, constraint satisfaction, and query optimization. Unfortunately, finding the optimal, *i.e.*, the minimal-width, tree decomposition is NP-Complete [7]. However many reasonable approximations exist for general graphs [14, 112] and the discovery of better algorithms is an active area of research in discrete mathematics [1, 76, 82].

Definition 3.1.1. *A* tree decomposition *of a graph $H = (V, E)$ is a tree TD, each of whose nodes η is labeled with a $V_\eta \subseteq V$ and $E_\eta \subseteq E$, such that the following properties hold:*

1. *Vertex Cover: For each $v \in V$, there is a vertex $\eta \in TD$ such that $v \in V_\eta$.*
2. *Edge Cover: For each hyperedge $e_i = \{v_1, \ldots, v_k\} \in E$ there is exactly one node $\eta \in TD$ such that $e \in E_\eta$. Moreover, $v_1, \ldots, v_k \in V_\eta$.*
3. *Running Intersection: For each $v \in V$, the set $\{\eta \in TD \mid v \in V_\eta\}$ is connected.*

Definition 3.1.2. *The* width *of a tree decomposition is* $\max(|V_\eta - 1|)$*, and the* treewidth *of a graph H is the minimal width of any tree decomposition of H.*

Unfortunately, finding the optimal elimination ordering and corresponding minimal-width tree decomposition is NP-Complete [7]. Fortunately, many reasonable approxi-

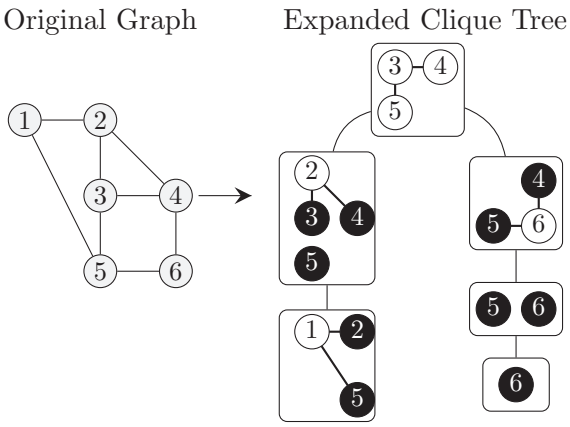

Figure 3.1. A graph and one possible minimal-width tree decomposition for it. Ghosted edges are not part of E_η; they are shown only for explanatory purposes.

mations exist for general graphs: in this chapter, we employ the commonly used maximum cardinality search (MCS) heuristic introduced by Tarjan and Yannikakis [113] to compute a tree decomposition with a reasonably-low, but not necessarily minimal, width.

Simply put, a tree decomposition of any graph (or any hypergraph) is a tree. Each of whose nodes we label with nodes and edges from the original graph, such that *vertex cover*, *edge cover* and the *running intersection* properties hold, and the "width" of the tree decomposition measures how tree-like the graph is. The reason for the interest in finding the tree decomposition of a graph is because many computationally difficult problems can be solved efficiently when the data is constrained to be a tree.

Figure 3.1 shows a graph and its minimal-width tree decomposition (showing V_η for each node η). We label nodes with lowercase Latin letters. We will refer back to this graph and tree decomposition as a running example throughout this chapter.

3.1.2 Hyperedge Replacement Grammar

The key insight for this task is that a network's tree decomposition encodes robust and precise information about the network. An HRG, extracted from the clique-tree, contains graphical rewriting rules that can match and replace graph fragments similar to how a context-free Grammar (CFG) rewrites characters in a string. These graph fragments represent a succinct, yet complete description of the building blocks of the network, and the rewriting rules of the HRG describe the instructions on how the graph is pieced together. For a thorough examination of HRGs, we refer the reader to the survey by Drewes *et al.* [27].

Definition 3.1.3. *A* hyperedge replacement grammar *is a tuple* $G = \langle N, T, S, \mathcal{P} \rangle$, *where*

1. N *is a finite set of nonterminal symbols. Each nonterminal* A *has a nonnegative integer* rank, *which we write* $|A|$.
2. T *is a finite set of terminal symbols.*
3. $S \in N$ *is a distinguished starting nonterminal, and* $|S| = 0$.
4. $\mathcal{P}$ *is a finite set of production rules* $A \to R$, *where*
   - A, *the left hand side (LHS), is a nonterminal symbol.*
   - R, *the right hand side (RHS), is a hypergraph whose edges are labeled by symbols from* $T \cup N$. *If an edge* e *is labeled by a nonterminal* B, *we must have* $|e| = |B|$.
   - *Exactly* $|A|$ *vertices of* R *are designated* external vertices *and numbered* $1, \ldots, |A|$. *The other vertices in* R *are called* internal *vertices.*

When drawing HRG rules, we draw the LHS A as a hyperedge labeled A with arity $|A|$. We draw the RHS as a hypergraph, with external vertices drawn as solid black circles and the internal vertices as open white circles.

If an HRG rule has no nonterminal symbols in its RHS, we call it a *terminal rule.* If an HRG rule has exactly one nonterminal symbol in its RHS, we call it a *unary rule.*

Definition 3.1.4. *Let G be an HRG and $P = (A \to R)$ be a production rule of G. We define the relation $H' \Rightarrow H^*$ (H^* is derived in one step from H') as follows. H' must have a hyperedge e labeled A; let $v_1, \ldots, v_k$ be the vertices it connects. Let $u_1, \ldots, u_k$ be the external vertices of R. Then H^* is the graph formed by removing e from H', making an isomorphic copy of R, and identifying v_i with the copies of u_i for each $i = 1, \ldots, k$.*

Let $\Rightarrow^*$ be the reflexive, transitive closure of $\Rightarrow$. Then we say that G generates a graph H if there is a production $S \to R$ and $R \Rightarrow^* H$ and H has no edges labeled with nonterminal symbols.

In other words, a derivation starts with the symbol S, and we repeatedly choose a nonterminal A and rewrite it using a production $A \to R$. The replacement hypergraph fragments R can itself have other nonterminal hyperedges, so this process repeats until there are no more nonterminal hyperedges. The following sections illustrate these definitions more clearly.

3.2 Learning HRGs

The first step in learning an HRG from a graph is to compute a tree decomposition from the original graph. Then, this clique-tree directly induces an HRG, which we demonstrate in this section.

3.2.1 Binarization

Just as context-free string grammars are more convenient to parse if put into Chomsky normal form (CNF), we also assume, without loss of generality, that our HRG also follows CNF. This means that each rule's right-hand side has at most two nonterminals. By the HRG induction methods presented later in this section, each tree node η yields an HRG rule, and the number of children of η determines the

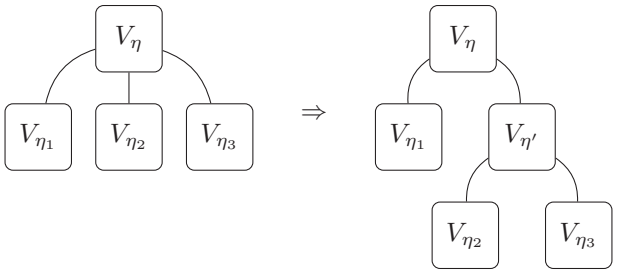

Figure 3.2. Binarization of a bag in a tree decomposition.

number of nonterminals on the right-hand side of the resulting rule. Thus, it suffices for the tree decomposition to have a branching factor of at most two. Although the branching factor of a tree decomposition may be greater than two, it is always easy to binarize it.

There is more than one way to do this; we use the following scheme. Let η be a tree decomposition node with children $\eta_1, \ldots, \eta_d$, where $d > 2$ (here d corresponds to the number of children for a given parent node). These are labeled with bags $V_\eta, V_{\eta_1}, \ldots, V_{\eta_d}$, respectively. Make a copy of η; call it η', and let $V_{\eta'} = V_\eta$. Let the children of η be η_1 and η', and let the children of η' be $\eta_2, \ldots, \eta_r$. See Fig. 3.2 for an example. Then, if η' has more than two children, apply this procedure recursively to η'.

It is easy to see that this procedure terminates and results in a tree decomposition whose nodes are at most binary-branching and still has the vertex cover, edge cover, and running intersection properties for H.

3.2.2 Tree Decomposition Pruning

Later we will introduce a dynamic programming algorithm for constructing graphs that require every leaf node of the tree decomposition to have at least one internal vertex. Tree decomposition algorithms, such as the MCS algorithm used in this chapter, do no guarantee these conditions. Fortunately, we can just remove these leaf

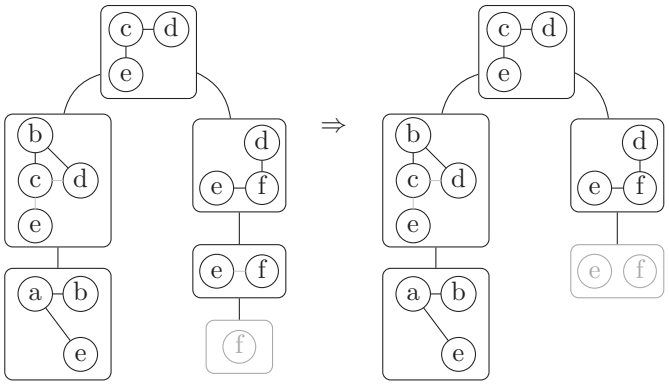

Figure 3.3. Pruning a tree decomposition to remove leaf nodes without internal vertices. Ghosted tree nodes show nodes that are pruned.

nodes from the tree.

The bottom-right tree decomposition node in Fig. 3.1 is such an example because f is an external vertex; that is, f exists in its parent. Because no internal vertices exist in this leaf node, it is removed from the tree. The node with vertices e and f is now a leaf, as illustrated in the left side of Fig. 3.3. Vertices e and f in the new leaf node are still both external vertices, so this tree node must also be removed creating a final tree decomposition illustrated in the right side of Fig. 3.3.

3.2.3 Tree Decompositions and HRGs

Here we show how to extract an HRG from the tree decomposition. Let η be an interior node of the tree TD, let η' be its parent, and let $\eta_1, \ldots, \eta_m$ be its children. Node η corresponds to an HRG production rule $A \to R$ as follows. First, $|A| = |V_{\eta'} \cap V_\eta|$. Then, R is formed by:

- Adding an isomorphic copy of the vertices in V_η and the edges in E_η
- Marking the (copies of) vertices in $V_{\eta'} \cap V_\eta$ as external vertices
- Adding, for each η_i, a nonterminal hyperedge connecting the (copies of) vertices in $V_\eta \cap V_{\eta_i}$.

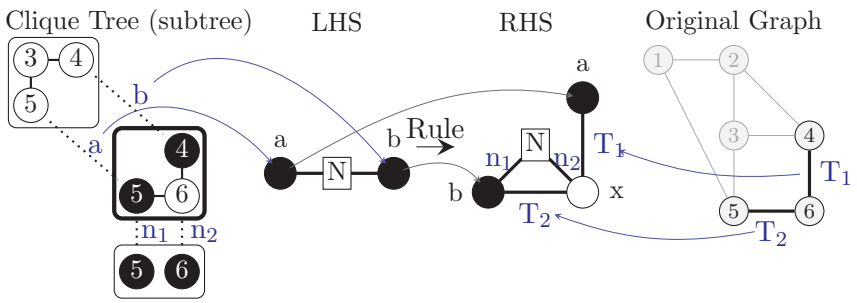

Figure 3.4. Example of hyperedge replacement grammar rule creation from an interior vertex of the tree decomposition. Note that lowercase letters inside vertices are for explanatory purposes only; only the numeric labels outside external vertices are actually part of the rule.

Figure 3.4 shows an example of the creation of an HRG rule. In this example, we focus on the middle clique-tree node $V_\eta = \{\mathrm{d}, \mathrm{e}, \mathrm{f}\}$, outlined in bold. We choose nonterminal symbol N for the LHS, which must have rank 2 because η has 2 vertices in common with its parent. The RHS is a graph whose vertices are (copies of) $V_\eta = \{\mathrm{d}, \mathrm{e}, \mathrm{f}\}$. Vertices d and e are marked external (and numbered 1 and 2, arbitrarily) because they also appear in the parent node. The terminal edges are $E_\eta = \{(\mathrm{d}, \mathrm{f}), (\mathrm{e}, \mathrm{f})\}$. There is only one child of η, and the nodes they have in common are e and f, so there is one nonterminal hyperedge connecting e and f. Next we deal with the special cases of the root and leaves.

3.2.3.1 Root Node

If η is the root node, then it does not have any parent cliques, but may still have one or more children. Because η has no parent, the corresponding rule has a LHS with rank 0 and a RHS with no external vertices. In this case, we use the start nonterminal S as the LHS, as shown in Figure 3.5.

The RHS is computed in the same way as the interior node case. For the example in Fig. 3.5, the RHS has vertices that are copies of c, d, and e. In addition, the

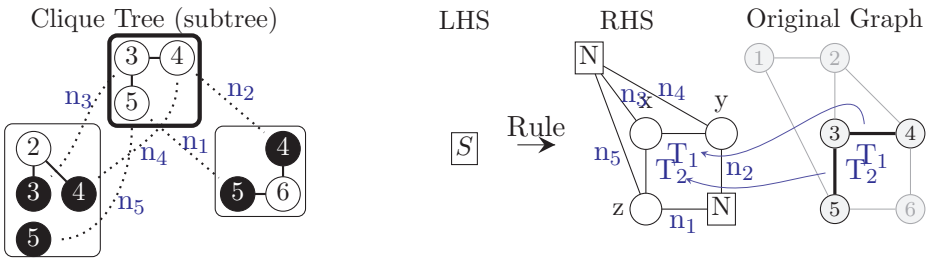

Figure 3.5. Example of hyperedge replacement grammar rule creation from the root node of the tree decomposition.

RHS has two terminal hyperedges, $E_\eta = \{(\mathrm{c},\mathrm{d}),(\mathrm{c},\mathrm{e})\}$. The root node has two children, so there are two nonterminal hyperedges on the RHS. The right child has two vertices in common with η, namely, d and e; so the corresponding vertices in the RHS are attached by a 2-ary nonterminal hyperedge. The left child has three vertices in common with η, namely, c, d, and e, so the corresponding vertices in the RHS are attached by a 3-ary nonterminal hyperedge.

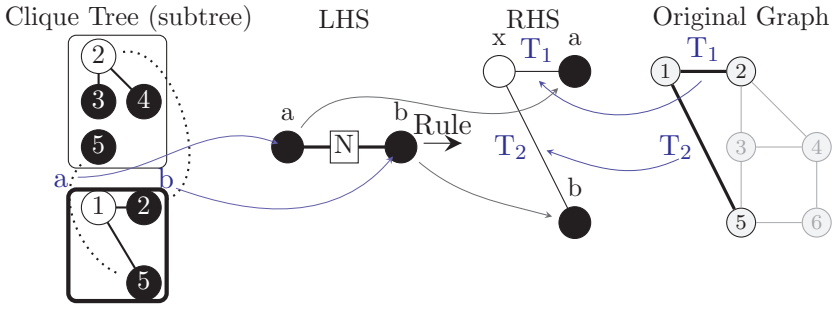

Figure 3.6. Example of hyperedge replacement grammar rule creation from a leaf vertex of the tree decomposition.

3.2.3.2 Leaf Node

If η is a leaf node, then the LHS is calculated the same as in the interior node case. Again we return to the running example in Fig. 3.6 (on the next page). Here, we focus on the leaf node $\{\mathrm{a}, \mathrm{b}, \mathrm{e}\}$, outlined in bold. The LHS has rank 2, because η has two vertices in common with its parent.

The RHS is computed in the same way as the interior node case, except no new nonterminal hyperedges are added to the RHS. The vertices of the RHS are (copies of) the nodes in η, namely, a, b, and e. Vertices b and e are external because they also appear in the parent clique. This RHS has two terminal hyperedges, $E_\eta = \{(\mathrm{a}, \mathrm{b}), (\mathrm{a}, \mathrm{e})\}$. Because the leaf clique has no children, it cannot produce any nonterminal hyperedges on the RHS; therefore this rule is a terminal rule.

3.2.4 Top-Down HRG Rule Induction

We induce production rules from the tree decomposition by applying the above extraction method top down. Because trees are acyclic, the traversal order does not matter, yet there are some interesting observations we can make about traversals of moderately sized graphs. First, exactly one HRG rule will have the special starting nonterminal S on its LHS; no mention of S will ever appear in any RHS. Similarly, the number of terminal rules is equal to the number of leaf nodes in the tree decomposition.

Larger graphs will typically produce larger tree decompositions, especially sparse graphs because they are more likely to have a greater number of small maximal cliques. These larger tree decompositions will produce a large number of HRG rules, one for each node in the tree decomposition. Although it is possible to keep track of each rule and its traversal order, we find, and will later show in the experiments section, that the same rules often repeat many times.

Figure 3.7 shows the 4 rules that are induced from the tree decomposition illus-

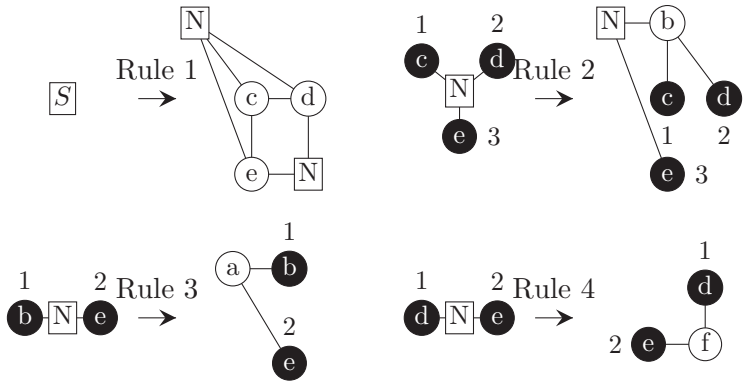

Figure 3.7. Complete set of production rules extracted from the example tree decomposition. Note that lowercase letters inside vertices are for explanatory purposes only; only the numeric labels outside external vertices are actually part of the rule.

trated in Fig. 3.1 and used in the running example throughout this section.

3.2.5 Complexity Analysis

The HRG rule induction steps described in this section can be broken into two steps: (i) creating a tree decomposition and (ii) the HRG rule extraction process.

Unfortunately, finding a tree decomposition with minimal width *i.e.*, the treewidth tw, is NP-Complete. Let n and m be the number of vertices and edges respectively in H. Tarjan and Yannikakis' Maximum Cardinality Search (MCS) algorithm finds a usable tree decomposition [111] in linear time $O(n + m)$, but is not guaranteed to be minimal.

The running time of the HRG rule extraction process is determined exclusively by the size of the tree decomposition as well as the number of vertices in each tree node. From Defn. 3.1.1 we have that the number of nodes in the tree decomposition is m. When minimal, the number of vertices in an the largest tree node $\max(|\eta_i|)$ (minus 1) is defined as the treewidth tw, however, tree decompositions generated by MCS have $\max(|\eta_i|)$ bounded by the maximum degree of H, denoted as Δ [36].

Therefore, given an elimination ordering from MCS, the computational complexity of the extraction process is in $O(m \cdot \Delta)$.

3.3 Graph Generation

In this section we show how to use the HRG extracted from the original graph H (as described in the previous section) to generate a new graph H^*. Ideally, H^* will be similar to, or have features that are similar to the original graph H. We present two generation algorithms. The first generation algorithm is *exact generation*, which, as the name implies, creates an isomorphic copy of the original graph $H^* \equiv H$. The second generation algorithm is a fast *stochastic generation* technique that generates random graphs with similar characteristics to the original graph. Each generation algorithm starts with H' containing only the starting nonterminal S.

3.3.1 Exact Generation

Exact generation operates by reversing the HRG extraction process. In order to do this, we must make sure to store an ordering of the tentacles in the hyperedges, as well as the complete tree decomposition TD (or at least the order that the rules were created). The first HRG rule considered is always the rule with the nonterminal labelled S as the LHS. This is because the tree traversal starts at the root, and because the root is the only case that results in S on the LHS.

The previous section defined an HRG G that is constructed from a tree decomposition TD of some given hypergraph H, and Defn. 3.1.4 defines the application of a production rule $(A \rightarrow R)$ that transforms some hypergraph H' into a new hypergraph H^*. By applying the rules created from the tree decomposition in order, we will create an H^* that is isomorphic to the original hypergraph H.

In the remainder of this section, we provide a more intuitive look at the exact generation property of the HRG by recreating the graph decomposed in the running

example.

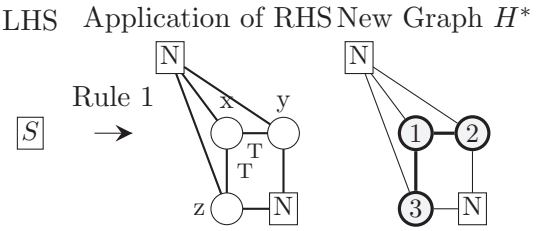

Figure 3.8. Application of Rule 1 to replace the starting nonterminal S with the RHS to create a new graph H^*.

Using the running example from the previous section, the application of Rule 1 illustrated in Fig. 3.8 shows how we transform the starting nonterminal into a new hypergraph, H^*. This hypergraph now has two nonterminal hyperedges corresponding to the two children that the root clique had in Fig. 3.1. The next step is to replace H' with H^* and then pick a nonterminal corresponding to the leftmost unvisited node of the tree decomposition.

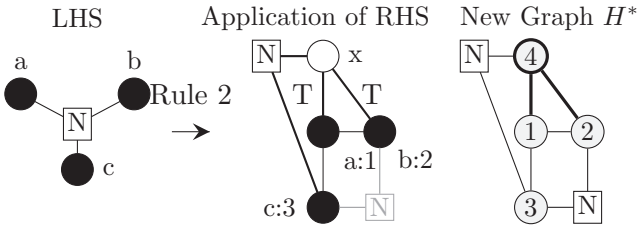

Figure 3.9. Application of Rule 2 to replace a size-3 nonterminal in H' with the RHS to create a new graph H^*.

We proceed down the left hand side of the tree decomposition, applying Rule 2 to H' as shown in Fig. 3.9. The LHS of Rule 2 matches the 3-ary hyperedge and replaces it with the RHS, which introduces a new internal vertex, two new terminal edges and a new nonterminal hyperedge. Again we set H' to be H^* and continue to the leftmost leaf in the example tree decomposition.

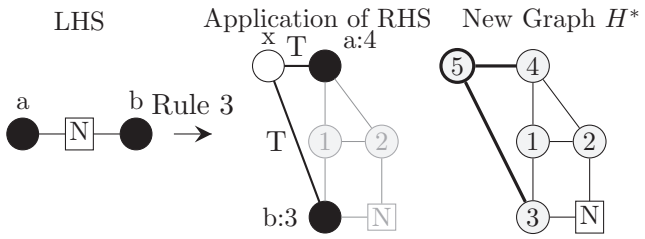

Figure 3.10. Application of Rule 3 to replace a size-2 nonterminal in H' with the RHS to create a new graph H^*.

The leftmost leaf in Fig. 3.1 corresponds to the application of Rule 3; it is the next to be applied to the new nonterminal in H^* and replaced by the RHS as illustrated in Figure 3.10. The LHS of Rule 3 matches the 2-ary hyperedge shown and replaces it with the RHS, which creates a new internal vertex along with two terminal edges. Because Rule 3 comes from a leaf node, it is a terminal rule and therefore does not add any nonterminal hyperedges. This concludes the left subtree traversal from Fig. 3.1.

Continuing the example, the right subtree in the tree decomposition illustrated in Fig. 3.1 has three further applications of the rules in $\mathcal{P}$. As illustrated in Fig. 3.11, Rule 4 adds the final vertex, two terminal edges and one nonterminal hyperedge to H^*.

After all 4 rules are applied in order, we are guaranteed that H and H^* are

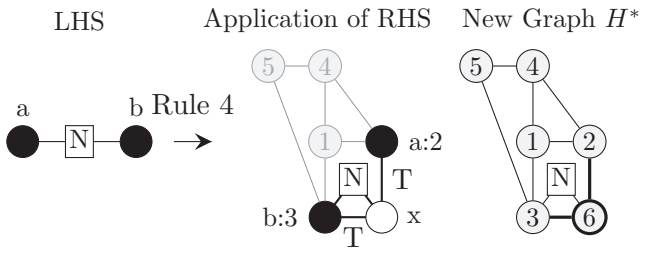

Figure 3.11. Application of Rule 4 to create an H^* that is isomorphic to the original graph H.

isomorphic.

3.3.2 Stochastic Generation

There are many cases in which we prefer to create very large graphs in an efficient manner that still exhibit the local and global properties of some given example graph *without storing the large tree decomposition* as required in exact graph generation. Here we describe a simple stochastic hypergraph generator that applies rules from the extracted HRG in order to efficiently create graphs of arbitrary size.

In larger HRGs we usually find many $A \to R$ production rules that are identical. We can merge these duplicates by matching rule-signatures in a dictionary, and keep a count of the number of times that each distinct rule has been seen. For example, if there were some additional Rule 5 in Fig. 3.7 that was identical to, say, Rule 3, then we would simply note that we saw Rule 3 two times.

To generate random graphs from a probabilistic HRG, we start with the special starting nonterminal $H' = S$. From this point, H^* can be generated as follows: (1) Pick any nonterminal A in H'; (2) Find the set of rules $(A \to R)$ associated with LHS A; (3) Randomly choose one of these rules with probability proportional to its count; (4) replace A in H' with R to create H^*; (5) Replace H' with H^* and repeat until there are no more nonterminal edges.

However, we find that although the sampled graphs have the same mean size as the original graph, the variance is much too high to be useful. So we want to sample only graphs whose size is the same as the original graph's, or some other user-specified size. Naively, we can do this using rejection sampling: sample a graph, and if the size is not right, reject the sample and try again. However, this would be quite slow. Our implementation uses a dynamic programming approach to do this exactly while using quadratic time and linear space, or approximately while using linear time and space. We omit the details of this algorithm here, but the source code is available online at `https://github.com/nddsg/HRG/`.

3.3.3 Fixed-Size Generation

A problem we find with the stochastic generation procedure is that, although the generated graphs have the same median size as the original graph, the variance is much too high to be useful. So we want to sample only graphs whose size is the same as the original graph's, or some other user-specified size. Naively, we can do this using rejection sampling: sample a graph, and if the size is not right, reject the sample and try again. However, this would be quite slow. Our implementation uses a dynamic programming approach to sample a graph with specified size, while using quadratic time and linear space, or approximately while using linear time and space.

More formally, the learned PHRG defines a probability distribution over graphs, $P(H^*)$. But rather than sampling from $P(H^*)$, we want to sample from $P(H^* \mid |H^*| = n)$, where n is the desired size.

Here, the stochastic generation sampling procedure is modified to rule out all graphs of the wrong size, as follows. Define a *sized* nonterminal $X^{(\ell)}$ to be a nonterminal X together with a size $\ell > 0$. If n is the desired final size, we start with $S^{(n)}$, and repeatedly:

1. Choose an arbitrary edge labeled with a sized nonterminal (call it $X^{(\ell)}$).

2. Choose a rule from among all rules with LHS X.
3. Choose sizes for all the nonterminals in the rule's RHS such that the total size of the RHS is ℓ.
4. Choose an ordering of the external vertices of the rule's RHS, with uniform probability.
5. Apply the rule.

A complication arises when choosing the rule and the RHS nonterminal sizes (steps 2 and 3) because the weights of these choices no longer form a probability distribution. Removing graphs with the wrong size causes the probability distribution over graphs to sum to less than one, and it must be renormalized [87]. To do this, we precompute a table of *inside probabilities* $\alpha[X, \ell]$ for $\ell = 1, \ldots, n$, each of which is the total weight of derivations starting with X and yielding a (sub)graph of size exactly ℓ. These are computed using dynamic programming, as shown in Algorithm 1.

If $X \to R$ is a HRG rule, define size(R) to be the increase in the size of a graph upon application of rule ($X \to R$). If size is measured in vertices, then size(R) is the number of *internal* vertices in R.

Rules that are unary and have zero size require some special care because they do not change the size of the graph. If there is a unary size-zero rule $X \to Y$, we need to ensure that $\alpha[Y, \ell]$ is computed before $\alpha[X, \ell]$, or else the latter will be incorrect. Thus, in Algorithm 1, we start by forming a weighted directed graph U whose nodes are all the nonterminals in N, and for every unary rule $X \xrightarrow{p} Y$, there is an edge from X to Y with weight p. We perform a topological sort on U, and the loop over nonterminals $X \in N$ is done in reverse topological order.

However, if U has a cycle, then no such ordering exists. The cycle could apply an unbounded number of times, and we need to sum over all possibilities. Algorithm 2 handles this more general case [108]. We precompute the strongly connected components of U, for example, using Tarjan's algorithm, and for each component C, we form the weighted adjacency matrix of C; call this U_C. The matrix

```
compute digraph U of unary size-zero rules;
topologically sort U;
assert (U is acyclic);
for ℓ ← 1, ..., n do
    for X ∈ N in reverse topological order do
        for rules X --p--> R do
            ℓ' = ℓ − size(R);
            if R has no nonterminals and ℓ' = 0 then
                α[X, ℓ] += p;
            end
            else if R has nonterminal Y then
                α[X, ℓ] += p × α[Y, ℓ'];
            end
            else if R has nonterminals Y and Z then
                for k ← 1, ..., ℓ' − 1 do
                    α[X, ℓ] += p × α[Y, k] × α[Z, ℓ' − k];
                end
            end
        end
    end
end
```

Algorithm 1: Compute inside probabilities (no cycles of size-zero unary rules)

$U_C^* = \sum_{i=0}^{\infty} U_C^i = (I - U_C)^{-1}$ gives the total weight of all chains of unary rules within C. So, after computing all the $\alpha[X, \ell]$ for $X \in C$, we apply the unary rules by treating the $\alpha[X, \ell]$ (for $X \in C$) as a vector and left-multiplying it by U_C^*. Some tricks are needed for numerical stability; for details, please see the released source code at `https://github.com/nddsg/PHRG/`.

In principle, a similar problem could arise with binary rules. Consider a rule $X \to R$ where R is zero-size and has two nonterminals, Y and Z. If $\alpha[Y, 0] > 0$, then $\alpha[X, \ell]$ is defined in terms of $\alpha[Y, \ell]$, which could lead to a circularity. Fortunately, we can avoid such situations easily. Recall that after tree decomposition pruning (Sec. 3.2.2), every leaf of the tree decomposition has at least one internal vertex. In terms of HRG rules, this means that if R has no nonterminals, then $\text{size}(R) > 0$. Therefore, we have $\alpha[X, 0] = 0$ for all X, and no problem arises.

```
compute weighted digraph U of unary size-zero rules;
find strongly connected components (scc's) of U;
compute U_C^* for each scc C;
for ℓ ← 1, ..., n do
    for scc's C in reverse topological order do
        for X ∈ C do
            for rules X --p--> R do
                ℓ' = ℓ − size(R);
                if R has no nonterminals and ℓ' = 0 then
                    α[X, ℓ] += p;
                end
                else if R has nonterminal Y and ℓ' < ℓ then
                    α[X, ℓ] += p × α[Y, ℓ'];
                end
                else if R has nonterminals Y and Z then
                    for k ← 1, ..., ℓ' − 1 do
                        α[X, ℓ] += p × α[Y, k] × α[Z, ℓ' − k];
                    end
                end
            end
        end
        for X ∈ C do
            α[X, ℓ] = Σ_{Y∈C} [U_C^*]_{XY} × α[Y, ℓ];
        end
    end
end
```

Algorithm 2: Compute inside probabilities (general)

Once we have computed α, we can easily sample a graph of size n using Algorithm 3. Initially, we start with the sized start nonterminal $S^{(n)}$. Then, we repeatedly choose an edge labeled with a sized nonterminal $X^{(\ell)}$, use the table α of inside probabilities to recompute the weight of all the rewriting choices quickly, sample one of them, and apply it.

3.3.4 Pruning Inside Probabilities

The slowest step in the above method is the precomputation of inside probabilities (Alg. 2), which is quadratic in the number of vertices. To speed up this step up, we

$H \leftarrow S^{(n)}$;
while H *contains a nonterminal* $X^{(\ell)}$ **do**
  for *all rules* $X \xrightarrow{p} R$ **do**
    $\ell' = \ell - \text{size}(R)$;
    if R *has no nonterminals and* $\ell' = 0$ **then**
      $\text{weight}[R] = p$;
    end
    else if R *has nonterminal* Y **then**
      $R' = R\{Y \mapsto Y^{(\ell')}\}$;
      $\text{weight}[R'] = p \times \alpha[Y, \ell']$;
    end
    else if R *has nonterminals* Y *and* Z **then**
      for $k \leftarrow 1, \ldots, \ell' - 1$ **do**
        $R' = R\{Y \mapsto Y^{(k)}, Z \mapsto Z^{(\ell'-k)}\}$;
        $\text{weight}[R'] = p \times \alpha[Y, k] \times \alpha[Z, \ell' - k]$;
      end
    end
  end
  let $P(R) = \text{weight}[R] / \sum_{R'} \text{weight}[R']$;
  sample sized RHS R from $P(R)$;
  choose ordering of the external vertices of R;
  $H \leftarrow H\{X^{(\ell)} \mapsto R\}$;
end

Algorithm 3: Generate a graph with n nodes

observe that randomly generated graphs tend to be highly unbalanced in the sense that if a rule has two nonterminal symbols, one is usually much larger than the other (see Figure 3.12). This is related to the fact, familiar with the study of algorithms, that random binary search trees tend to be highly unbalanced [103].

Therefore, we can modify Algorithm 2 to consider only splits where at most (say) 1000 nodes go to one nonterminal and the rest of the nodes go the other. This makes the algorithm asymptotically linear.

3.4 Summary

In this chapter we have shown how to use tree decompositions (also known as junction trees, clique tree, intersection trees) constructed from a simple, general graph

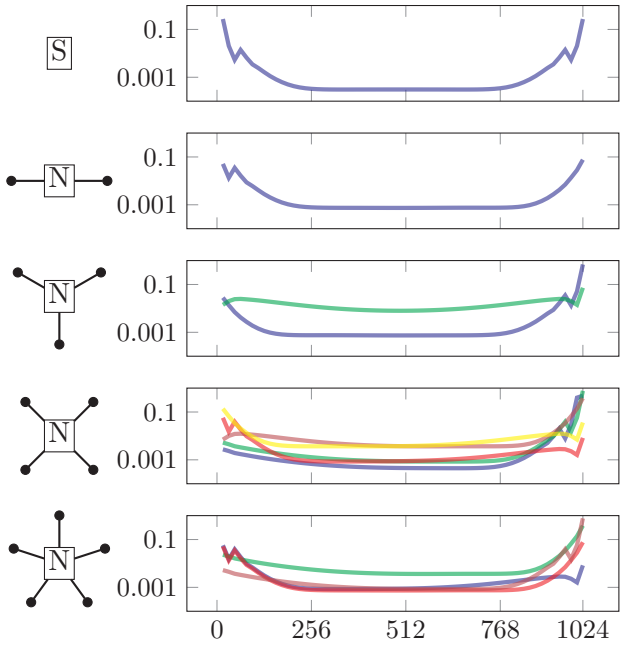

Figure 3.12. When an HRG rule has two nonterminal symbols, one is overwhelmingly likely to be much larger than the other. This plot shows, for various grammar rules (one LHS per row, one RHS per colored line), the probability (log scale) of apportioning 1024 nodes between two nonterminal symbols. This plot is best viewed in color.

to learn a hyperedge replacement grammar (HRG) for the original graph. We have shown that the extracted HRG can be used to reconstruct a new graph that is isomorphic to the original graph if the tree decomposition is traversed during reconstruction. More practically, we show that a stochastic application of the grammar rules creates new graphs. We also present a fixeds-size graph generator that generates graphs of exact size.

Perhaps the most important finding that comes from this work is the ability to interrogate the generation of substructures and subgraphs within the grammar rules that combine to create a holistic graph. Forward applications of the technology described in this work may allow us to identify novel patterns analogous to the previously discovered triadic closure and bridge patterns found in real world social

networks. Thus, an investigation in to the nature of the extracted rules and their meaning (if any) is a top priority for future work.

The next step is evaluate the HRG graph model. We do this by comparing generated graphs against their empirical counterparts in the next chapter.

CHAPTER 4

EVALUATING GRAPH GENERATORS

In this chapter, we show that HRGs contain rules that succinctly represent the global and local structure of the original graph. Showing this involves comparing HRG models against some of the state-of-the-art graph generators. We consider the properties that characterize some real-world networks and examine the distribution of graphs generated using Kronecker Graphs, the Exponential Random Graph, Chung-Lu Graphs, and the graphs produced by HRG.

Like HRGs, the Kronecker and Exponential Random Graph Models learn parameters that can be used to approximately recreate the original graph H or a graph of some other size such that the probabilistically generated graph holds many of the same properties as the original graph. The Chung-Lu graph model relies on node degree sequences to yield graphs that maintain this distribution.

There are three distinct algorithms presented in the previous chapter: exact (*i.e.*, isomorphic) generation, stochastic generation, and fixed-size generation. The exact generator will generate an isomorphic copy of the original graph; so evaluating the fit against the original graph is moot. In the stochastic generation, we only pick generated graphs that have the same size as the original graph, which is a slow and cumbersome process. However, the resulting graphs will be very very similar to the fixed-size generation except the fixed-size generator does not require the guess-and-check method of the stochastic generator. Thus, in all experiments in this chapter and the next, we will only use the fixed-size HRG generation method.

The evaluation metrics, methodology, and result plots were created by myself with input from Prof. Weninger.

4.1 Real-world Datasets

To get a holistic and varied view of the strengths and weaknesses of HRGs in comparison to the other leading graph generation models, we consider real-world networks that exhibit properties that are both common to many networks across different fields, but also have certain distinctive properties.

TABLE 4.1

EXPERIMENTAL DATASET

Dataset Name	**Nodes**	**Edges**
Karate Club	34	78
Proteins (metabolic)	1,870	2,277
arXiv GR-QC	5,242	14,496
Internet Routers	6,474	13,895
Enron Emails	36,692	183,831
DBLP	317,080	1,049,866
Amazon	400,727	2,349,869
Flickr	105,938	2,316,948

The real-world networks considered in this chapter are described in Table 4.1. The networks vary in their number of vertices and edges as indicated, but also vary

in clustering coefficient, diameter, degree distribution and many other graph properties. Specifically, Karate Club graph is a network of interactions between members of a karate club; the Protein network is a protein-protein interaction network of *S. cerevisiae* yeast; the Enron graph is the email correspondence graph of the now defunct Enron corporation; the arXiv GR-QC graph is the co-authorship graph extracted from the General Relativity and Quantum Cosmology section of arXiv; the Internet router graph is created from traffic flows through Internet peers; DBLP is the co-authorship graph from the DBLP dataset; Amazon is the co-purchasing network from March 12, 2003; and, finally, Flickr is a network created from photos taken at the same location.

In the following experiments, we use the larger networks (arXiv, Routers, Enron, DBLP, Amazon, Flickr) for network generation and the smaller networks (Karate, Protein) for a special graph extrapolation task. Datasets were downloaded from the SNAP and KONECT dataset repositories.

4.2 Methodology

We compare several different graph properties from the four classes of graph generators (fixed-size HRG, Kronecker, Chung-Lu and exponential random graph (ERGM) models) to the original graph H. Other models, such as the Erdős-Rényi random graph model, the Watts-Strogatz small world model, the Barabási-Albert generator, etc. are not compared here because Kronecker, Chung-Lu and ERGM have been shown to outperform these earlier models when matching network properties in empirical networks.

Kronecker graphs operate by learning an initiator matrix and then performing a recursive multiplication of that initiator matrix to create an adjacency matrix of the approximate graph. In our case, we use KronFit [75] with default parameters to learn a 2×2 initiator matrix and then use the recursive Kronecker product to generate the

graph. Unfortunately, the Kronecker product only creates graphs where the number of nodes is a power of 2, *i.e.*, 2^x, where we chose $x = 15$, $x = 12$, $x = 13$, and $x = 18$ for Enron, ArXiv, Routers and DBLP graphs respectively to match the number of nodes as close as possible.

The Chung-Lu Graph Model takes, as input, a degree distribution and generates a new graph of the similar degree distribution and size [23].

Exponential Random Graph Models are a class of probabilistic models. Their usefulness lies in that they directly describe several structural features of a graph [100]. We used default parameters in R's ERGM package [50] to generate graph models for comparison. In addition to the problem of model degeneracy, ERGMs do not scale well to large graphs. As a result, DBLP, Enron, Amazon, and Flickr could not be modelled due to their size, and the arXiv graph always resulted in a degenerate model. Therefore ERGM results are omitted from this section.

The main strength of HRG is to learn the patterns and rules that generate a large graph from only a few small subgraph-samples of the original graph. So, in all experiments, we make k random samples of size s node-induced subgraphs by a breadth first traversal starting from a random node in the graph [70]. By default we set $k = 4$ and $s = 500$ empirically. We then compute tree decompositions from the k samples, learn HRGs $G_1, G_2, \ldots, G_k$, and combine them to create a single grammar $G = \bigcup_i G_i$.

Unless otherwise noted, we generate 20 graphs each for the HRG, Chung-Lu, and Kronecker models and plot the mean values in the results section. We did compute the confidence intervals for each of the models but omitted them from the graphs for clarity. In general, the confidence intervals were small for HRG, Kronecker, and Chung-Lu.

4.2.1 Graph Generation Results

Here we compare and contrast the results of approximate graphs generated from the HRG, Kronecker, and Chung-Lu models. Before presenting each result, we briefly introduce the graph properties that we used to compare the similarity between the real networks and their approximate counterparts. Although many properties have been discovered and detailed in related literature, we focus on five of the principal properties from which most others can be derived.

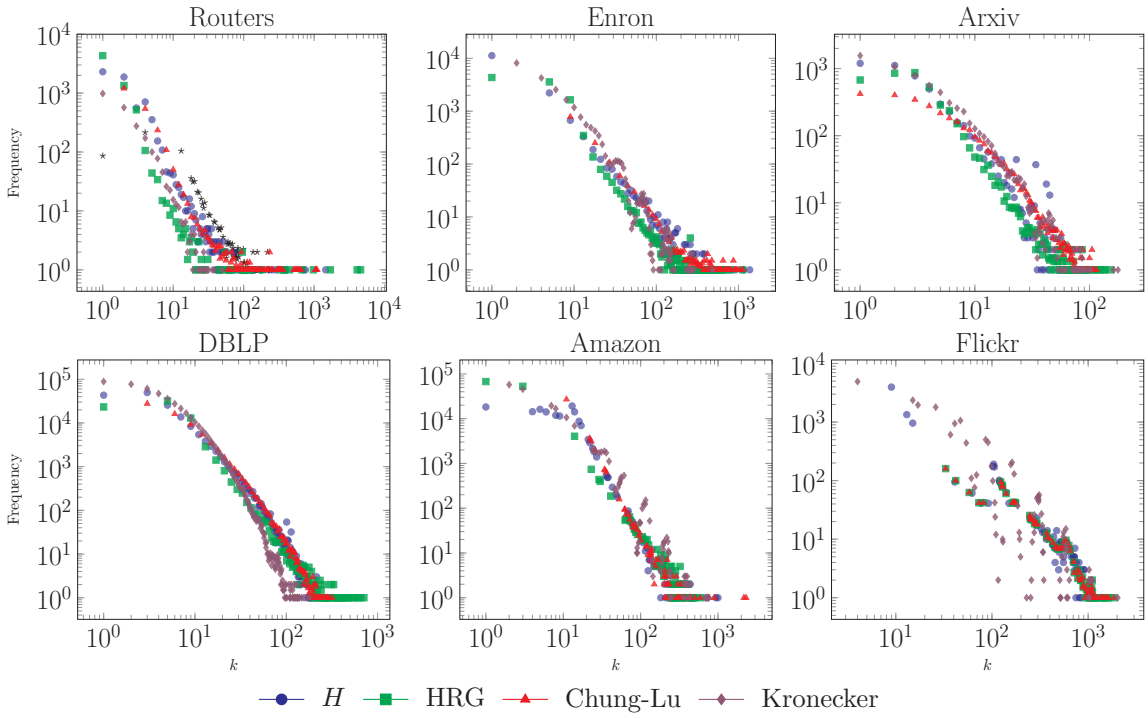

Figure 4.1. Degree Distribution. Dataset graphs exhibit a power law degree distribution that is well captured by existing graph generators as well as HRG.

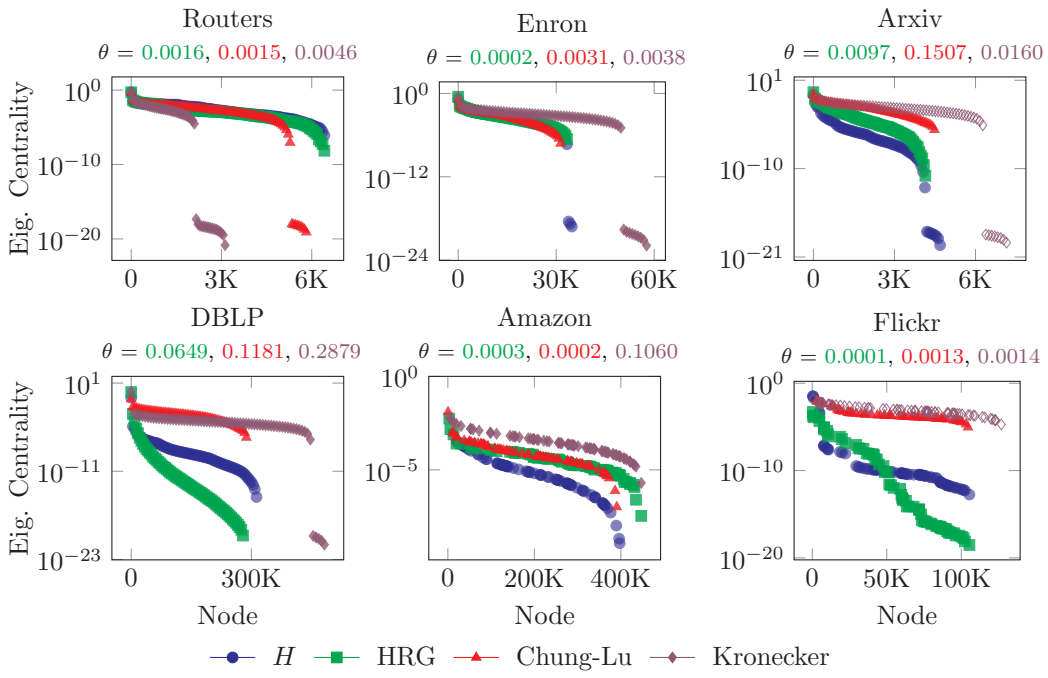

Figure 4.2. Eigenvector Centrality. Nodes are ordered by their eigenvector-values along the x-axis. Cosine distance between the original graph and HRG, Chung-Lu and Kronecker models are shown at the top of each plot where lower is better. In terms of cosine distance, the eigenvector of HRG is consistently closest to that of the original graph.

4.2.1.1 Global Measures

A key goal of graph modelling to preserve certain network properties of the original graph (*i.e.*, H as introduced in 3.1). Graphs generated using HRG, Kronecker, or Chung-Lu are analyzed by studying their fundamental network properties to assess how successful the model performs in generating graphs from parameters and production rules learned from the input graph. First, we look at the degree distribution, eigenvector centrality, local clustering coefficient, hop plot, and assortative mixing characteristics, and draw conclusions on these results.

- **Degree Distribution.** The degree distribution of a graph is the distribution of the number of edges connecting to a particular vertex. Figure 4.1 shows the results of the degree distribution property on the six real-world graphs. Recall

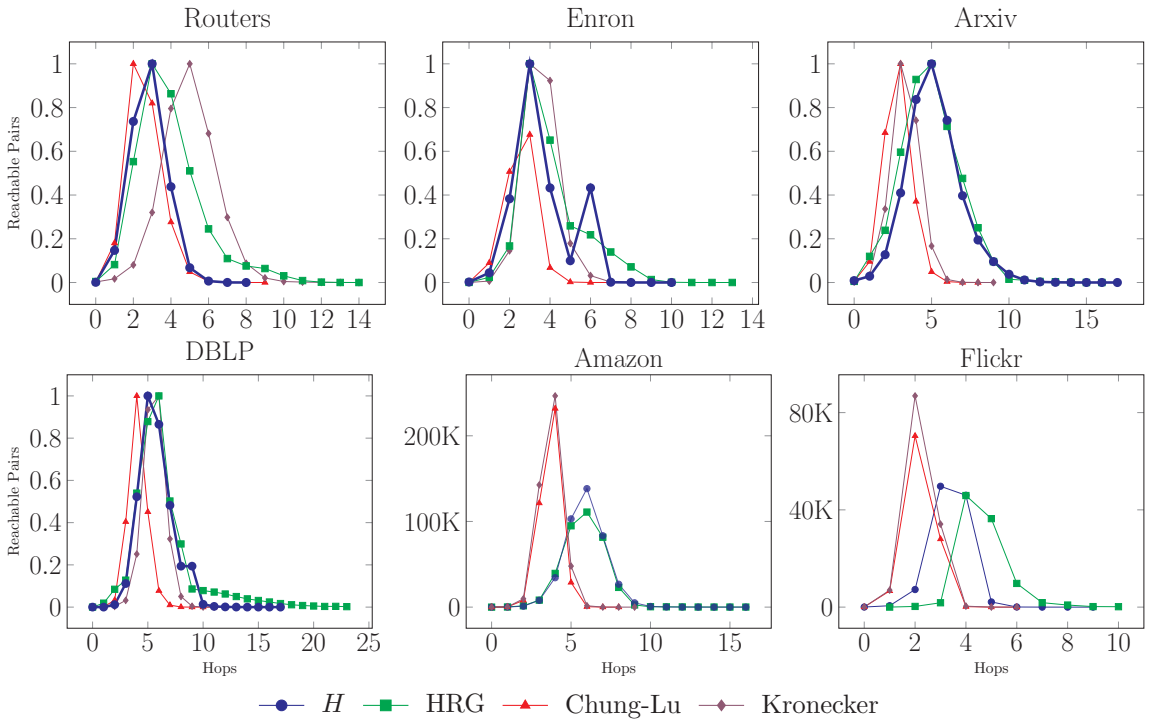

Figure 4.3. Hop Plot. Number of vertex pairs that are reachable within x-hops. HRG closely and consistently resembles the hop plot curves of the original graph.

that the graph results plotted here and throughout the results section are the mean averages of 20 generated graphs. Each of the generated graphs is slightly different from the original graphs in their own way. As expected, we find that the power law degree distribution is captured by existing graph generators as well as the HRG model.

- **Eigenvector Centrality.** The principal eigenvector is often associated with the centrality or "value" of each vertex in the network, where high values indicate an important or central vertex and lower values indicate the opposite. A skewed distribution points to a relatively few "celebrity" vertices and many common nodes.

  The principal eigenvector value for each vertex is also closely associated with the PageRank and degree value for each node. Figure 4.2 shows the eigenvector scores for each node ranked highest to lowest in each of the six real-world graphs. Because the x-axis represents individual nodes, Fig. 4.2 also shows the size difference among the generated graphs. HRG performs consistently well across all graphs, but the log scaling on the y-axis makes this plot difficult to discern. To more concretely compare the eigenvectors, the pairwise

cosine distance between eigenvector centrality of H and the mean eigenvector centrality of each model's generated graphs appear at the top of each plot in order. HRG consistently has the lowest cosine distance followed by Chung-Lu and Kronecker.

- **Hop Plot.** The hop-plot of a graph shows the number of vertex-pairs that are reachable within x hops. The hop-plot, therefore, is another way to view how quickly a vertex's neighborhood grows as the number of hops increases. As in related work [73] we generate a hop-plot by picking 50 random nodes and performing a complete breadth first traversal over each graph. Figure 4.3 demonstrates that HRG graphs produce hop-plots that are remarkably similar to the original graph.

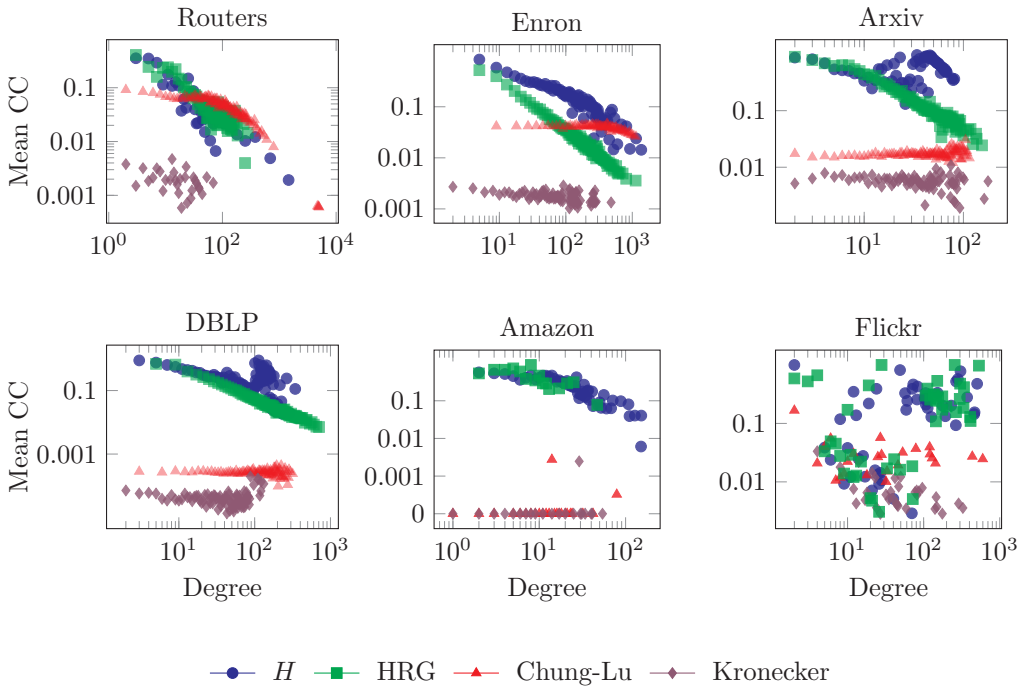

Figure 4.4. Mean Clustering Coefficient by Node Degree. HRG closely and consistently resembles the clustering coefficients of the original graph.

- **Mean Clustering Coefficients.** A vertex's clustering coefficient is a mea-

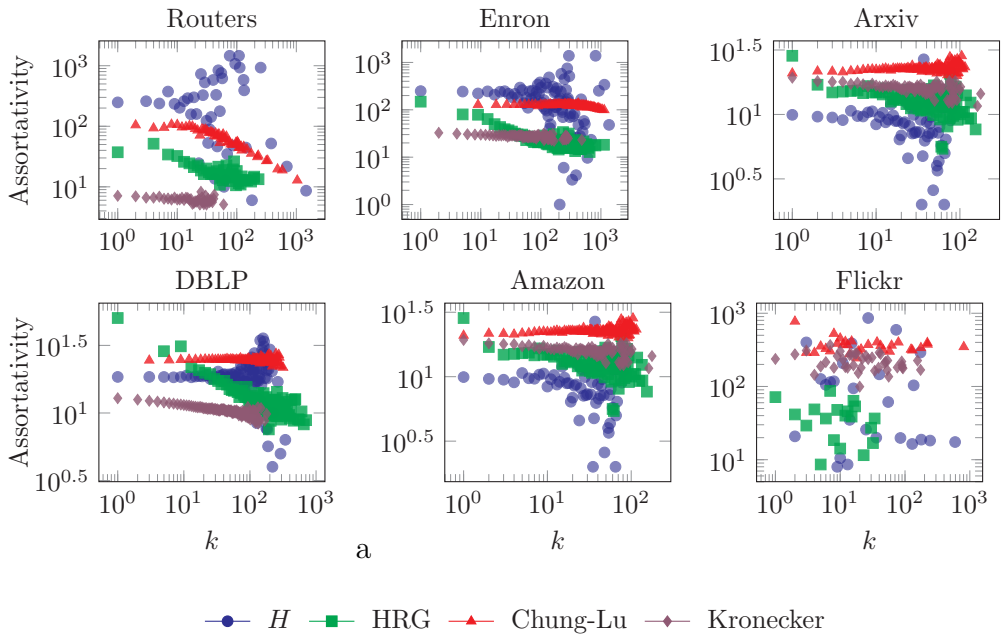

Figure 4.5. Local Degree Assortativity. HRG, Chung-Lu, and Kronecker graphs show mixed results with no clear winner.

sure of how well connected its neighbors are [119]. For each vertex in the graph, its clustering coefficient is the ratio of the number of edges in its ego-network (*i.e.*, local neighborhood) to the total number of possible edges that could exist if the vertex's neighborhood was a clique. Calculating the clustering coefficient for each node is a computationally difficult task and difficult plot aesthetically, so we sampled 100 nodes from the graph randomly. Figure 4.4 shows the average clustering coefficients for the sampled nodes as a function of its degree in the graph. Like the results from Seshadhri et al., we find that the Kronecker and Chung-Lu models perform poorly at this task [104].

- **Local Degree Assortativity.** The global degree assortativity of a graph measures its tendency to have high-degree vertices connect to high-degree vertices and vice versa measured as a Pearson correlation coefficient. The local degree assortativity is measured for each vertex as the amount that each vertex contributes to the overall correlation, *i.e.*, how different the vertex is from its neighbors. Figure 4.5 shows the degree assortativity for each vertex from each generated graph.

  The last three graph metrics, k-core, local clustering coefficient, and local degree assortativity, all showed a relatively poor performance of the Chung-Lu and Kronecker graph generators. HRG modelled the k-core and local clustering coefficients rather well but had inconsistent results in the local degree

assortativity plots.

TABLE 4.2: GRAPHLET STATISTICS AND CORRELATION DISTANCE

Graphs									GCD
Routers	13511	1397413	9863	304478	6266541	177475	194533149	18615590	
HRG	13928	1387388	9997	288664	6223500	174787	208588200	18398430	1.41
Kronecker	144	61406	0	80	10676	973	642676	551496	2.81
Chung-Lu	4787	356897	6268	81403	1651445	13116	35296782	4992714	2.00
Enron	727044	23385761	2341639	22478442	375691411	6758870	4479591993	1371828K	
HRG	79131	4430783	49355	554240	13123350	556760	688165900	54040090	0.51
Kronecker	2598	5745412	1	1011	608566	49869	1.89468000	141065K	2.88
Chung-Lu	322352	23590260	1191770	16267140	342570000	10195620	3967912K	2170161K	1.33
arXiv	89287	558179	320385	635143	4686232	382032	11898620	7947374	
HRG	88108	606999	320039	656554	5200392	455516	15691941	9162859	1.10
Kronecker	436	224916	1	293	47239	4277	3280822	2993351	2.10
Chung-Lu	927	232276	6	967	87868	11395	2503333	3936998	1.82
DBLP	2224385	15107734	16713192	4764685	96615211	203394	258570802	25244735	
HRG	1271520	7036423	1809570	2716225	26536420	296801	71099374	28744359	1.59
Kronecker	869	21456020	0	25	150377	11568	517370300	367981K	2.82
Chung-Lu	1718	22816460	740	91	306993	27856	453408500	495492K	1.73
Amazon	5426197	81876562	4202503	39339842	306482275	10982173K	11224584	1511382K	
HRG	4558006	90882984	3782253	35405858	275834K	12519677K	10326617	1556723K	–
Kronecker	11265	118261600	40	1646	4548699	350162	6671637K	4752968K	–
Chung-Lu	4535	71288780	21	6376	5874750	95323	11008170K	2134629K	–
Flickr	24553	3754965	1612	38327	2547637	63476	197979760	30734524	
HRG	24125	4648108	1600	39582	3130621	68739	409838400	41498780	–
Kronecker	679294	494779400	16068	4503724	951038K	78799K	96664230K	76331M	–
Chung-Lu	7059002	787155400	5003082	313863800	12826040K	1513807K	168423M	247999M	–

4.2.2 Canonical Graph Comparison

The previous network properties primarily focus on statistics of the global network. However, there is mounting evidence which argues that the graphlet comparisons are a complete way to measure the similarity between two graphs [98, 116]. The graphlet distribution succinctly describes the number of small, local substructures that compose the overall graph and therefore more completely represents the details of what a graph "looks like." Furthermore, it is possible for two very dissimilar graphs to have the same degree distributions, hop plots, etc., but it is difficult for two dissimilar graphs to fool a comparison with the graphlet distribution.

Table 4.2[1] shows the mean graphlet counts over 10 runs for each graph generator. We find that graphlet counts for the graphs generated by HRG follow the original counts more closely, and in many cases much more closely, than the Kronecker and Chung-Lu graphs.

4.2.2.1 Graphlet Correlation Distance

Recent work from systems biology has identified a new metric called the Graphlet Correlation Distance (GCD). The GCD computes the distance between two graphlet correlation matrices – one matrix for each graph [122]. It measures the frequency of the various graphlets present in each graph, *i.e.*, the number of edges, wedges, triangles, squares, 4-cliques, etc., and compares the graphlet frequencies of each node across two graphs. Because the GCD is a distance metric, lower values are better. The GCD can range from $[0, +\infty]$, where the GCD is 0 if the two graphs are isomorphic.

The rightmost column in Tab. 4.2 shows the GCD results. Unfortunately, the node-by-node graphlet enumerator used to calculate the GCD [122] could not pro-

[1]Six real-world graphs. First row of each section shows the original graph's graphlet counts, remaining row shows mean counts of 10 runs for each graph generator. We find that the HRG model generates graphs that closely approximate the graphlet counts of the original graph.

cess the large Amazon and Flickr graphs, so only the summary graphlet counts are listed for the two larger graphs [5]. The results here are clear: HRG significantly outperforms the Chung-Lu and Kronecker models. The GCD opens a whole new line of network comparison methods that stress the graph generators in various ways. We explore many of these options next.

4.2.3 Graph Extrapolation

Recall that HRG learns the grammar from $k = 4$ subgraph-samples from the original graph. In essence, HRG is extrapolating the learned subgraphs into a full-size graph. This raises the question: if we only had access to a small subset of some larger network, could we use our models to infer a larger (or smaller) network with the same local and global properties? For example, given the 34-node Karate Club graph, could we infer what a Karate Club might look like if it's membership doubled?

Using two smaller graphs, Zachary's Karate Club (34 nodes, 78 edges) and the protein-protein interaction network of *S. cerevisiae* yeast (see Table 4.1), we learned an HRG model with $k = 1$ and $s = n$, *i.e.*, no sampling, and generated networks of size-n^* = 2x, 3x, ..., 32x. For the protein graph, we also sampled down to $n^* = x/8$. Powers of 2 were used because the standard Kronecker model can only generate graphs of that size. The Chung-Lu model requires a size-n^* degree distribution as input. To create the proper degree distribution we fitted a Poisson distribution ($\lambda = 2.43$) and a Geometric Distribution ($p = 0.29$) to Karate and Protein graphs respectively and drew n^* degree-samples from their respective distributions. In all cases, we generated 20 graphs at each size-point.

Rather than comparing raw numbers of graphlets, the GCD metric compares the *correlation* of the resulting graphlet distributions. As a result, GCD is largely immune to changes in graph size. Thus, GCD is a good metric for this extrapolation task. Figure 4.6 shows the mean GCD scores; not only does HRG generate good

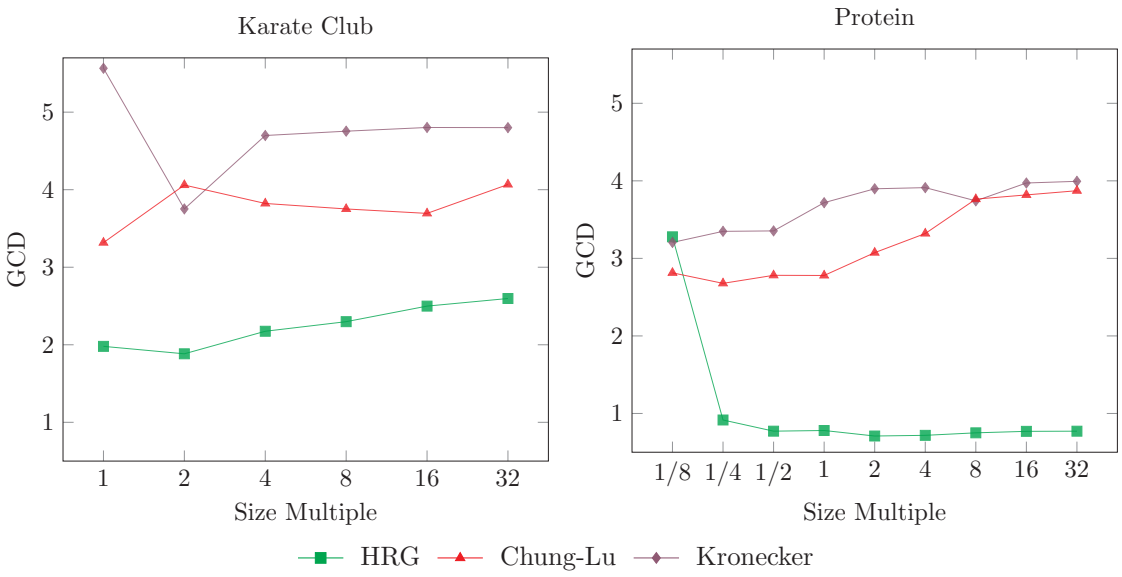

Figure 4.6. GCD of graphs extrapolated in multiples up to 32x from two small graphs. HRG outperforms Chung-Lu and Kronecker models when generating larger graphs. Lower is better.

results at $n^* = 1$x, the GCD scores remain mostly level as n^* grows.

4.2.4 Sampling and Grammar Complexity

We have shown that HRG can generate graphs that match the original graph from $k = 4$ samples of $s = 500$-node subgraphs. If we adjust the size of the subgraph, then the size of the tree decomposition will change causing the grammar to change in size and complexity. A large tree decomposition ought to create more rules and a more complex grammar, resulting in a larger model size and better performance; while a small tree decomposition ought to create fewer rules and a less complex grammar, resulting in a smaller model size and a lower performance.

To test this hypothesis, we generated graphs by varying the number of subgraph samples k from 1 to 32, while also varying the size of the sampled subgraph s from 100 to 600 nodes. Again, we generated 20 graphs for each parameter setting. Figure 4.7

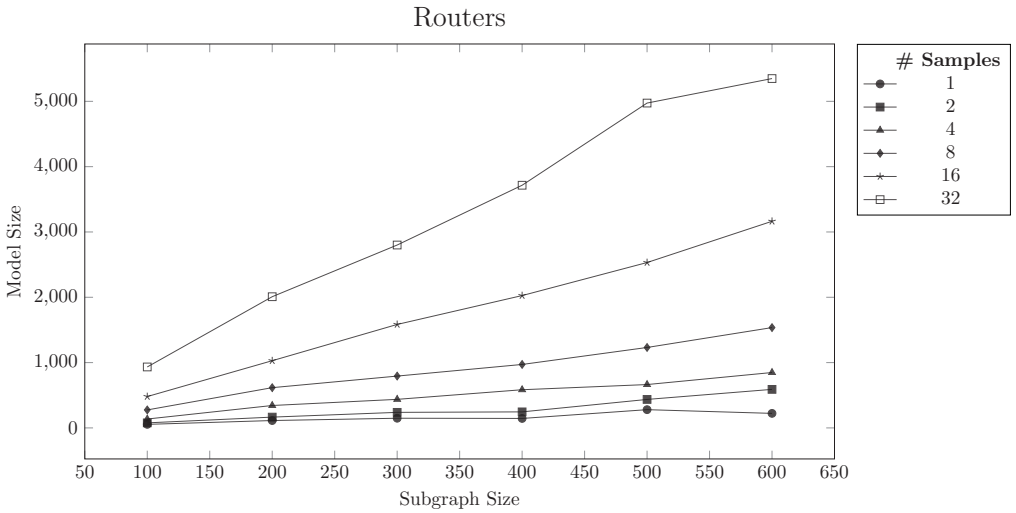

Figure 4.7. HRG model size as the subgraph size s and the number of subgraph samples k varies. The model size grows linearly with k and s.

shows how the model size grows as the sampling procedure changes on the Internet Routers graph. Plots for other graphs show a similar growth rate and shape but are omitted due to space constraints.

To test the statistical correlation we calculated Pearson's correlation coefficient between the model size and sampling parameters. We find that the k is slightly correlated with the model size on Routers ($r = 0.31$, $p = 0.07$), Enron ($r = 0.27, p = 0.09$), arXiv ($r = 0.21, p = 0.11$), and DBLP ($r = 0.29$, $p = 0.09$). Furthermore, the choice of s affects the size of the tree decomposition from which the grammars are inferred. So its not surprising that s is highly correlated with the model size on Routers ($r = 0.64$), Enron ($r = 0.71$), arXiv ($r = 0.68$), and DBLP ($r = 0.54$) all with $p \ll 0.001$.

Because we merge identical rules when possible, we suspect that the overall growth of the HRG model follows Heaps law [46], *i.e.*, that the model size of a graph can be predicted from its rules; although we save a more thorough examination of the grammar rules as a matter for future work.

4.2.4.1 Model Size and Performance

One of the disadvantages of the HRG model, as indicated in Fig. 4.7, is that the model size can grow to be very large. But this again begs the question: do larger and more complex HRG models result in improved performance?

To answer this question, we computed the GCD distance between the original graph and graphs generated by varying k and s. Figure 4.8 illustrates the relationship between model size and the GCD. We use the Router and DBLP graphs to shows the largest and smaller of our datasets; other graphs show similar results, but we omit their plots due to space. Surprisingly, we find that the performance of models with only 100 rules is similar to the performance of the largest models. In the Router results, two very small models with poor performance had only 18 and 20 rules each. Best fit lines are drawn to illustrate the axes relationship where negative slope indicates that larger models perform better. Outliers can dramatically affect the outcome of best-fit lines, so the faint line in the Routers graph shows the best fit line if we remove the two square outlier points. Without removing outliers, we find only a slightly negative slope on the best fit line indicating only a slight performance improvement between HRG models with 100 rules and HRG models with 1,000 rules. Pearson's correlation coefficient comparing GCD and model size similarly show slightly negative correlations on Routers ($r = -0.12$, $p = 0.49$), Enron ($r = -0.09, p = 0.21$), ArXiv ($r = 0.04, p = 0.54$), and DBLP ($r = -0.08$, $p = 0.62$)

4.2.4.2 Runtime Analysis

The overall execution time of the HRG model is best viewed in two parts: (1) rule extraction, and (2) graph generation.

Unfortunately, finding a tree decomposition with minimal width *i.e.*, the treewidth tw, is NP-Complete. Let n and m be the number of vertices and edges respectively in H. Tarjan and Yannikakis' Maximum Cardinality Search (MCS) algorithm finds

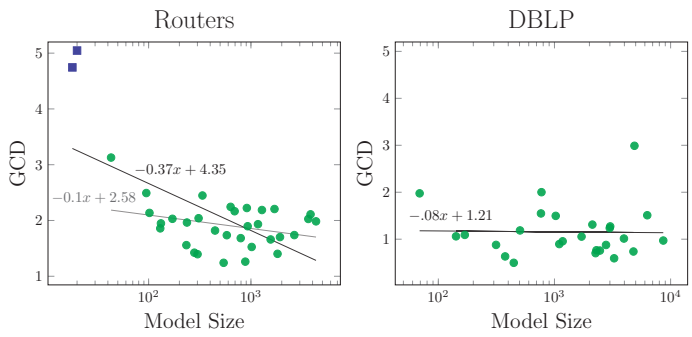

Figure 4.8. GCD as a function of model size. We find a slightly negative relationship between model size and performance, but with quickly diminishing returns. We show best-fit lines and their equations; the shorter fit line in the Routers plot ignores the square outlier points.

usable tree decompositions [113] in linear time $O(n + m)$, but is not guaranteed to be minimal.

The running time of the HRG rule extraction process is determined exclusively by the size of the tree decomposition as well as the number of vertices in each tree node. From Defn. 3.1.1 we have that the number of nodes in the tree decomposition is m. When minimal, the number of vertices in the largest tree node $\max(|\eta_i|)$ (minus 1) is defined as the treewidth tw. However, tree decompositions generated by MCS have $\max(|\eta_i|)$ bounded by the maximum degree of H and is denoted as Δ [36]. Therefore, given an elimination ordering from MCS, the computational complexity of the extraction process is in $O(m \cdot \Delta)$. In our experiments, we perform k samples of size-s subgraphs. So, when sampling with k and s, we amend the runtime complexity to be $O(k \cdot m \cdot \Delta)$ where m is bounded by the number of hyperedges in the size-s subgraph sample and $\Delta \leq s$.

Graph generation requires a straightforward application of rules that is linear in the number of edges in the output graph.

We performed all experiments on a modern consumer-grade laptop in an unoptimized, unthreaded python implementation. We recorded the extraction time while

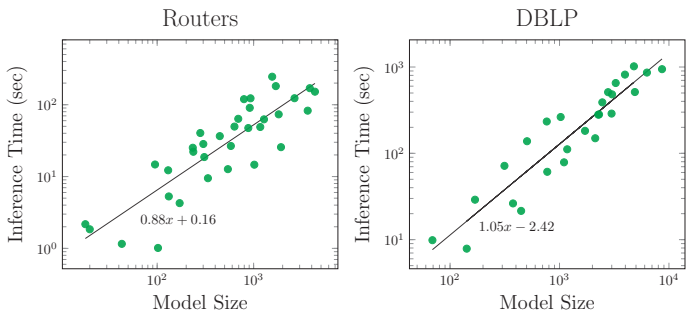

Figure 4.9. Total extraction runtime (*i.e.*, tree decomposition creation and rule extraction) as a function of model size. Best fit lines on the log-log plot show that the execution time grows linearly with the model size.

generating graphs for the size-to-GCD comparison in the previous section. Although the runtime analysis gives theoretical upper bounds to the rule extraction process, Fig. 4.9 shows that the extraction runtime is highly correlated to the size of the model in Routers ($r = 0.68$), arXiv ($r = 0.91$), Enron ($r = 0.88$), and DBLP ($r = 0.94$) all with $p \ll 0.001$. Simply put, more rules require more time, but there are diminishing returns. So it may not be necessary to learn complex models when smaller HRG models tend to perform reasonably well.

By comparison, the Kronecker graph generator learns a model in $O(m)$ and can create a graph in $O(m)$. The Chung-Lu model does not learn a model, but rather takes, as input, a degree sequence; graph generation is in $O(n + m)$.

4.2.4.3 Graph Guarantees

In earlier work we showed that an application of HRG rules corresponding to a traversal of the tree decomposition will generate an isomorphic copy of the original graph [4].

Unlike the Kronecker and Chung-Lu graph generators, which are guaranteed to generate graph with power-law degree distributions, there are no such guarantees that can be made about the shape of graphs generated by HRGs. The reason is straight-

forward: the HRG generator is capable of applying rules in any order, therefore, a wide variety of graphs are possible, although improbable, given an HRG grammar.

But the lack of a formal guarantees give the HRG model flexibility to model a large variety of graphs. For example, given a line-graph, the HRG model will generate a new graph that looks, more-or-less, like a line-graph. If given a random graph, characterized by a binomial degree distribution, then HRG is likely to generate a new graph with a binomial degree distribution.

4.3 Summary

This chapter describes results of various experiments that compare generated graphs against its original graph. We also performed experiments using graphlet correlation distance, graph extrapolation, and the infinity mirror test. The analysis of the results is particularly exciting because the results show a stark improvement in performance over several existing graph generators.

Next, we will investigate differences between the grammars extracted from different tree-decompositions. Within the computational theory community, there has been a renewed interest in quickly finding tree-decompositions of large real-world graphs that are closer to optimal. Because of the close relationship of HRG and tree-decompositions are shown in this thesis, any advancement in tree decomposition algorithms could directly improve the speed and accuracy of graph generation.

CHAPTER 5

INFINITY MIRROR TEST FOR ANALYZING GRAPH GENERATORS

Viewing data as an information network, the standard approach is to treat the network as a graph with some number of nodes and edges. Increasingly, researchers and practitioners are interested in understanding how individual pieces of information are organized and interact to discover the fundamental principles that underlie physical or social phenomena.

With this motivation, researchers have developed a suite of graph generation techniques that learn a model of a network in order to extrapolate, generalize or otherwise gain a deeper understanding of the data set. Early graph generators like the Erdős-Rényi, Watts-Strogatz, and Barabasi-Albert models produce random graphs, small world graphs, and scale-free graphs respectively. Although they are used to generate graphs given some hand-picked parameters, they do not learn a model from any observed real-world network [9].

We focus instead on graph model inducers, which take some observed network H, learn a model Θ and produce a new graph H'. These types of graph generators include the Kronecker Model [75], Chung-Lu Model [23], Exponential Random Graph Model (ERGM) [100] and Block Two-Level Erdős-Rényi Model (BTER) [96], and others.

The performance of a graph generator can be judged based on how well the new graph matches certain topological characteristics of the original graph. Unfortunately small perturbations caused by the implicit and inherent biases of each type of model may not be immediately visible using existing performance metrics.

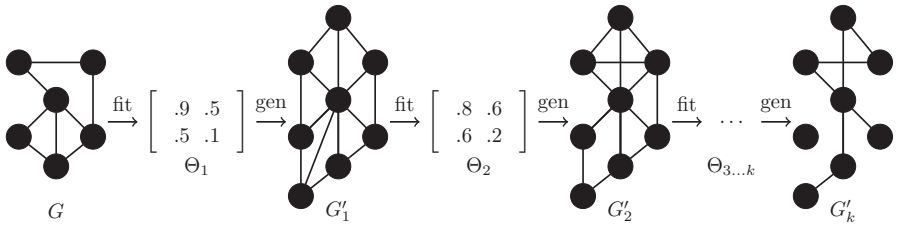

Figure 5.1: Example infinity mirror test on the Kronecker model. This test recursively learns a model and generates graphs. Although not apparent in H'_1, this example shows a particular type of degeneration where the model loses edges.

In the present work, we address this problem by characterizing the *robustness* of a graph generator via a new metric we call the *infinity mirror test*. The "infinity mirror" gets its name from the novelty item with a pair of mirrors, set up so as to create a series of smaller and smaller reflections that appear to taper to an infinite distance. The motivating question here is to see if a generated graph H' holds sufficient information to be used as reference. Although a comparison between H and H' may show accurate results, the model's biases only become apparent after recursive application of the model onto itself.

The details of the method are discussed later, but, simply put, the infinity mirror tests the robustness of a graph generator because errors (or biases) in the model are propagated forward depending on their centrality and severity. A robust graph generator, without severe biases or errors, should remain stable after several recurrences. However, a non-robust model will quickly degenerate, and the manner in which the model degenerates reveals the model-biases that were hidden before.

The infinity mirror test was conceived by myself with input from Prof. Weninger. The evaluation, methodology and result plots were created by myself.

5.1 Infinity Mirror Test

We characterize the *robustness* of a graph generator by its ability to learn and regenerate the same model repeatedly. A perfect, lossless model (*e.g.*, $\Theta = H$) would

generate H' as an isomorphic copy of the original graph. If we were to again apply the perfect model on the isomorphic H', we would again generate an isomorphic copy of the graph. On the other hand, a non-robust graph generator may generate a H' that is dissimilar from H. If we were to learn a new model from H' and create a second-level graph, we would expect this second graph to exacerbate the errors (the biases) that the first graph made and be even less similar to H. A third, fourth, fifth, etc. application of the model will cause the initial errors to accumulate and cause cascading effects in each successive layer.

Colored by this perspective, the robustness of a graph generator is defined by its ability to maintain its topological properties as it is recursively applied. To that end, this chapter presents the infinity mirror test. In this test, we repeatedly learn a model from a graph generated by the an earlier version of the same model.

Starting with some real world network H, a graph generator learns a model Θ_1 (where the subscript $\cdot_1$ represents the first recurrence) and generates a new graph H'_1. At this point, current works typically overlay graph properties like degree distribution, assortativity, etc. to see how well H matches H'_1. We go a step further and ask if the new graph H'_1 holds sufficient information to be used as reference itself. So, from H'_1 we learn a new model Θ_2 in order to generate a second-level graph H'_2. We repeat this recursive "learn a model from the model"-process k times, and compare H'_k with the original graph.

Figure 5.1 shows an example of the infinity mirror test for the Kronecker model. In this example some real world graph H is provided by the user. From H a model Θ_1 is fit, which is used to generate a new graph H'_1. Of course, H'_1 is only an approximation of H and is therefore slightly different. In the second recurrence a new model Θ_2 is fit from H'_1 and used to generate a new graph H'_2. This continues recursively k times.

With the infinity mirror test, our hypothetical, perfect model is perfectly robust and immune to error. A hypothetical "bad" model would quickly degenerate into

an unrecognizable graph after only a few recurrences. Despite their accurate performance, existing models are far from perfect. We expect to see that all models degenerate as the number of recurrences grow. The question is: how quickly do the models degenerate and how bad do the graphs become?

5.2 Experiments

In order to get a holistic and varied view of the robustness of various graph generators, we consider real-world networks that exhibit properties that are both common to many networks across different fields, but also have certain distinctive properties.

The real world networks considered in this chapter are described in Table. 5.1, many of which overlap from chapter 4. The networks vary in their number of vertices and edges as indicated, but also vary in clustering coefficient, degree distribution and many other graph properties. Specifically, C. elegans is the neural network of the roundworm of the named species [51]; the Power grid graph is the connectivity of the power grid in the Western United States [118]; the Enron graph is the email correspondence graph of the now defunct Enron corporation [59]; the ArXiv GR-QC graph is the co-authorship graph extracted from the General Relativity and Quantum Cosmology section of ArXiv; the Internet router graph is created from traffic flows through Internet peers; and, finally, DBLP is the co-authorship graph from the DBLP dataset. All datasets were downloaded from the SNAP and KONECT dataset repositories. On each of the six real-world graphs, we applied the Kronecker, Block Two-Level Erdos-Renyi (BTER), Exponential Random Graph (ERGM) and Chung- Lu (CL) models recursively to a depth of k=10.

Figures 5.2, 5.3, 5.4, and 5.5 show the results of the Chung-Lu, BTER and Kronecker graphs respectively. Different graph generators will model and produce graphs

TABLE 5.1

REAL NETWORKS

Dataset Name	**Nodes**	**Edges**
C. elegans neural (male)	269	2,965
Power grid	4,941	6,594
ArXiv GR-QC	5,242	14,496
Internet Routers	6,474	13,895
Enron Emails	36,692	183,831
DBLP	317,080	1,049,866

according to their own internal biases. Judging the performance of the generated graphs typically involves comparing various properties of the new graph with the original graph. In Figs. 5.2–5.5 we show the plots of the degree distribution, eigenvector centrality, hop plots and graphlet correction distance. Each subplot shows the original graph in blue and the generated graphs H'_2, H'_5, H'_8, H'_{10} in increasingly lighter shades of red.

In the remainder of this section we will examine the results one metric at a time, *i.e.*, figure-by-figure.

5.2.1 Network Statistics or Measures

5.2.1.1 Degree Distribution

The degree distribution of a graph is the ordered distribution of the number of edges connecting to a particular vertex. Barabási and Albert initially discovered that the degree distribution of many real world graphs follows a heavy-tailed power law distribution such that the number of nodes $N_d \propto d^{-\gamma}$ where $\gamma > 0$ and γ, called the power law exponent, is typically between 2 and 3 [8].

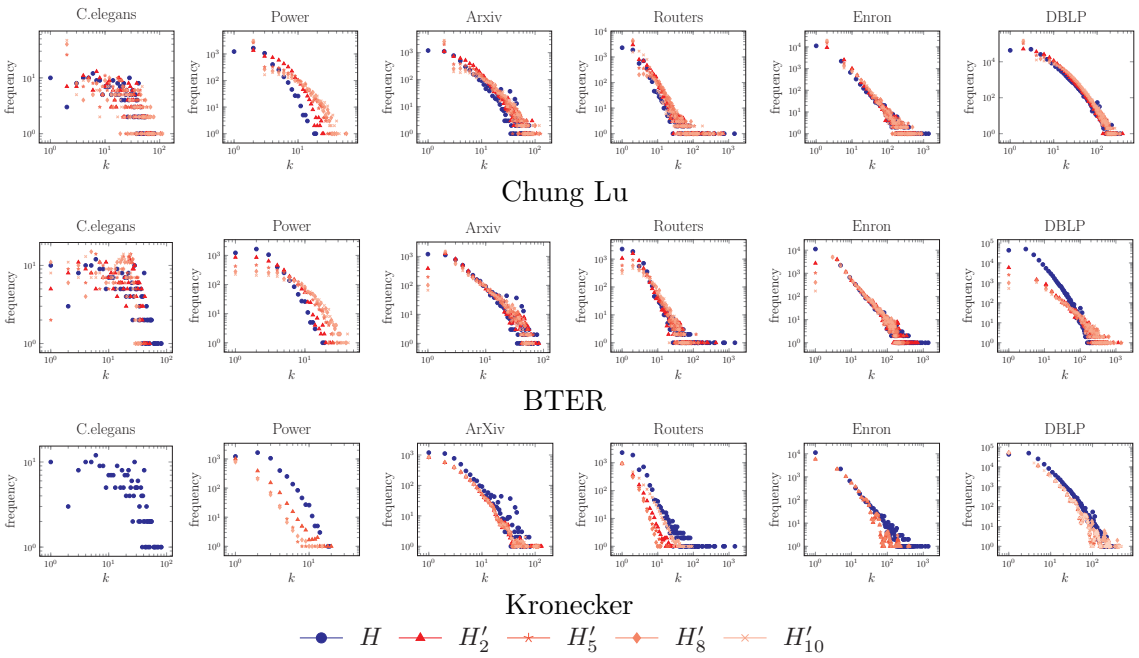

Figure 5.2. Degree distribution. H shown in blue. H'_2, H'_5, H'_8 and H'_{10} are shown in lighter and lighter shades of red. Degeneration is observed when recurrences increasingly deviate from H. The results for some Kronecker models are missing because they were unable to generate graphs for non-scale-free graphs.

Figure 5.2 shows the degree distribution of Chung Lu, BTER and Kronecker row-by-row for each of the six data sets. The Kronecker generator was unable to model the C. elegans graph because C. elegans does not have a power-law degree distribution, thus those results are absent. These plots are drawn with the original graph H in blue first, then H'_2, H'_5, H'_8 and H'_{10} are overlaid on top in that order; as a result, light-red plots often elide dark-red or blue plots indicating accurate results and non-degeneration. In general, we find that the degree distributions hold mostly steady throughout all 10 recurrences. One exception is present in the Power grid dataset for all three graph generators where the later graphs lose density in the head of their degree distribution. But overall the results are surprising stable.

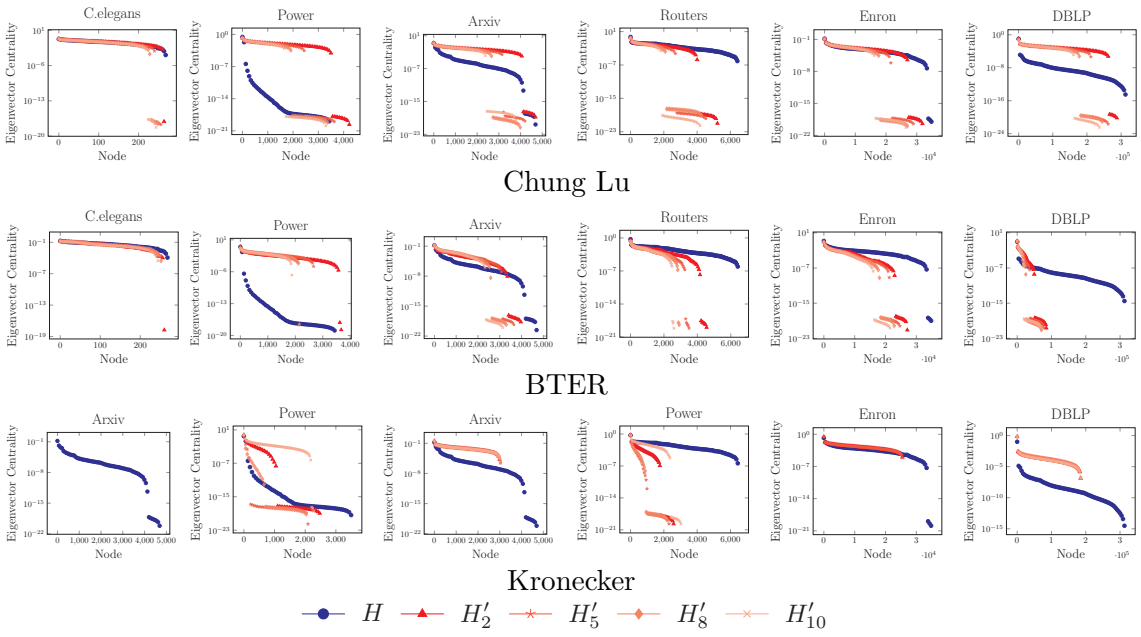

Figure 5.3. Eigenvector centrality. H shown in blue. Results for recurrences H'_2, H'_5, H'_8 and H'_{10} in lighter and lighter shades of red showing eigenvector centrality for each network node. Degeneration is shown by increasing deviation from H's eigenvector centrality signature.

5.2.1.2 Eigenvector Centrality

The principal eigenvector is often associated with the centrality or "value" of each vertex in the network, where high values indicate an important or central vertex and lower values indicate the opposite. A skewed distribution points to a relatively few "celebrity" vertices and many common nodes. The principal eigenvector value for each vertex is also closely associated with the PageRank and degree value for each node.

Figure 5.3 shows an ordering of nodes based on their eigenvector centrality. Again, the results of Kronecker on C. elegans is absent. With the eigenvector centrality metric we see a clear case of model degeneration in several data sets, but stability in others. The arXiv graph degenerated in Chung-Lu and BTER, but was stable in Kronecker. On the other hand, the Power grid and Routers graph had only a

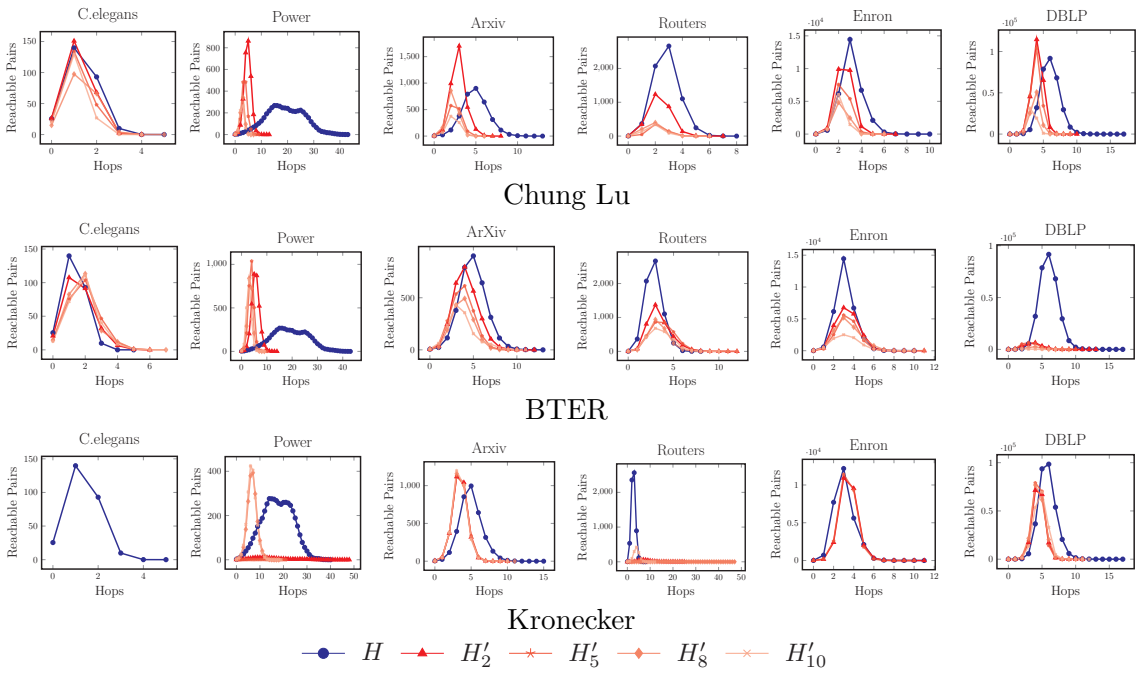

Figure 5.4. Hop plot. H shown in blue. Results for recurrences H'_2, H'_5, H'_8 and H'_{10} in lighter and lighter shades of red. Degeneration is observed when recurrences increasingly deviate from H.

slight degeneration with Chung Lu and BTER models, but severe problems with the Kronecker model.

5.2.1.3 Hop Plot

The hop plot of a graph shows the number of vertex-pairs that are reachable within x hops. The hop plot, therefore, is another way to view how quickly a vertex's neighborhood grows as the number of hops increases. As in related work [72] we generate a hop plot by picking 50 random nodes and perform a breadth first traversal over each graph.

Figure 5.3 shows the hop plots of each graph, model and recurrence level. Again we find mixed results. Model degeneration is clear in the arXiv results for Chung Lu and BTER: we see a consistent flattening of the hop plot line recurrence-level

increases. Yet the arXiv results are consistent with the Kronecker model.

The hop plot results are quite surprising in many cases. All of the models severely underestimate the shape of the power grid and routers graphs even in the first generation (not shown).

Of the many topological characteristics that could be compared, researchers and practitioners typically look at a network's *global properties* as in Figs 5.2–5.3. Although these metrics can be valuable, they do not completely test the performance of a graph generator.

In our view, a large network is essentially the combination of many small subnetworks. Recent work has found that the global properties are merely products of a graph's *local properties*, in particular, graphlet distributions [98]. As a result, graphlet counting [5, 80, 115] and related comparison metrics [122] comprise the local-side of graph generator performance.

Thus a complete comparison of graph generator performance ought to include both local and global metrics. In other words, not only should a generated graph have the same degree distribution, hop plot, etc. as the original graph, but the new graph should also have the same number of triangles, squares, 4-cliques, etc. as the original graph.

There is mounting evidence which argues that the graphlet distribution is the most complete way to measure the similarity between two graphs [98, 115]. The graphlet distribution succinctly describes the distribution of small, local substructures that compose the overall graph and therefore more completely represents the details of what a graph "looks like." Furthermore, it is possible for two very dissimilar graphs to have the same degree distributions, hop plots, etc., but it is difficult for two dissimilar graphs to fool a comparison with the graphlet distribution.

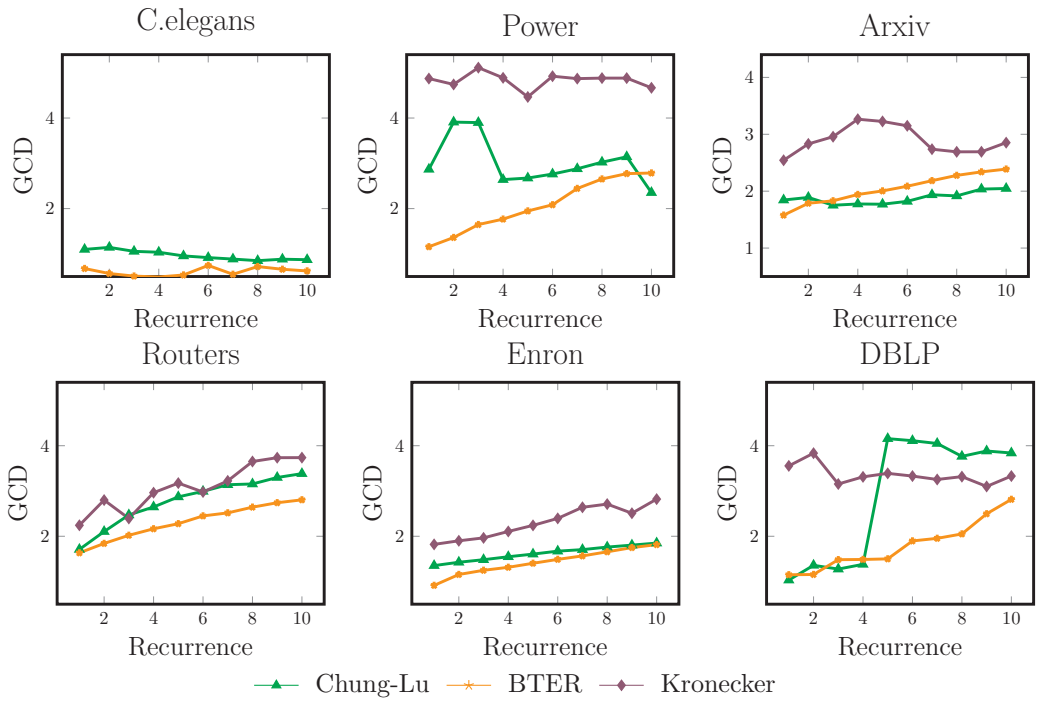

Figure 5.5. Graphlet Correlation Distance. All recurrences are shown for Chung Lu, BTER and Kronecker graph generators. Lower is better. Degeneration is indicated by a rise in the GCD values as the recurrences increase.

5.2.1.4 Graphlet Correlation Distance

Recent work from systems biology has identified a new metric called the Graphlet Correlation Distance (GCD). Simply put, the GCD computes the distance between two graphlet correlation matrices – one matrix for each graph [122]. Because GCD is a distance metric, lower values are better. The GCD can range from $[0, +\infty]$, where the GCD is 0 if the two graphs are isomorphic.

Figure 5.5 shows the GCD of each recurrence level. Because GCD is a distance, there is no blue line to compare against; instead, we can view degeneracy as an increase in the GCD as the recurrences increase. Again, results from the Kronecker model are absent for C. elegans. As expected, we see that almost all of the models show degeneration on almost all graphs.

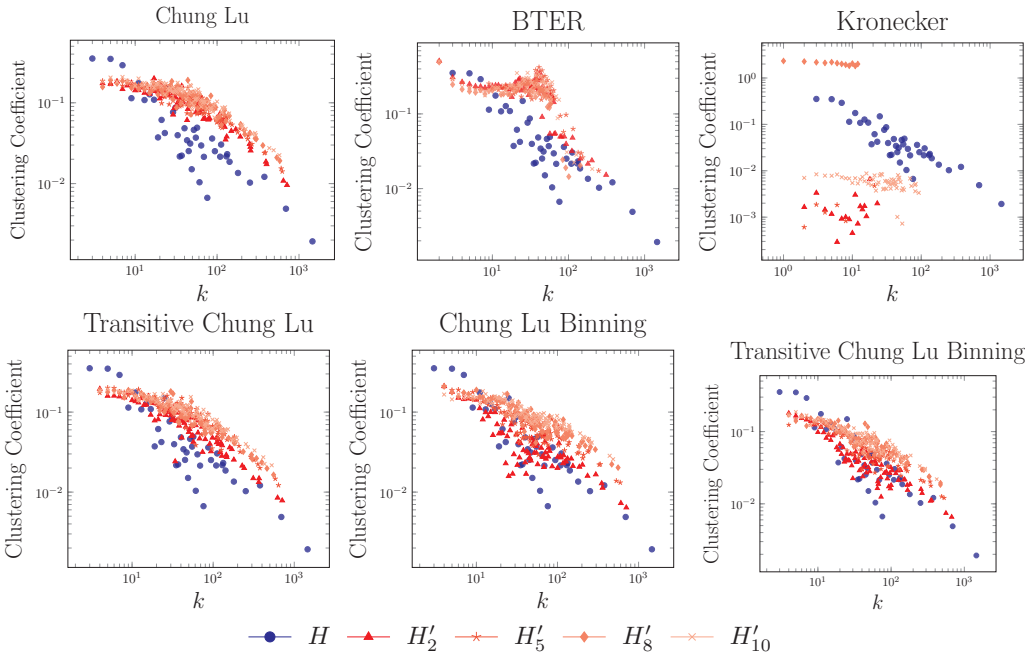

Figure 5.6. Clustering Coefficient. H is in blue. Results for recurrences H'_2, H'_5, H'_8 and H'_{10} in lighter and lighter shades of red. Degeneration is observed when recurrences increasingly deviate from H.

Kronecker's GCD results show that in some cases the GCD is slightly reduced, but in general its graphs deviate dramatically from the original. Chung-Lu and BTER show signs of better network alignment when learning a model from C. elegans. This result highlights biased assumptions in the Chung Lu and BTER models that seem to favor networks of this kind while struggling to handle networks with power-law degree distributions.

5.2.1.5 Clustering Coefficients

A node's clustering coefficient is a measure of how well connected a vertex's neighbors are. Specifically, a nodes's clustering coefficient, *i.e.*, the local clustering coefficient, is the number of edges that exist in a node's ego-network divided by the total number of nodes possible in the ego-network. The global clustering coefficient is

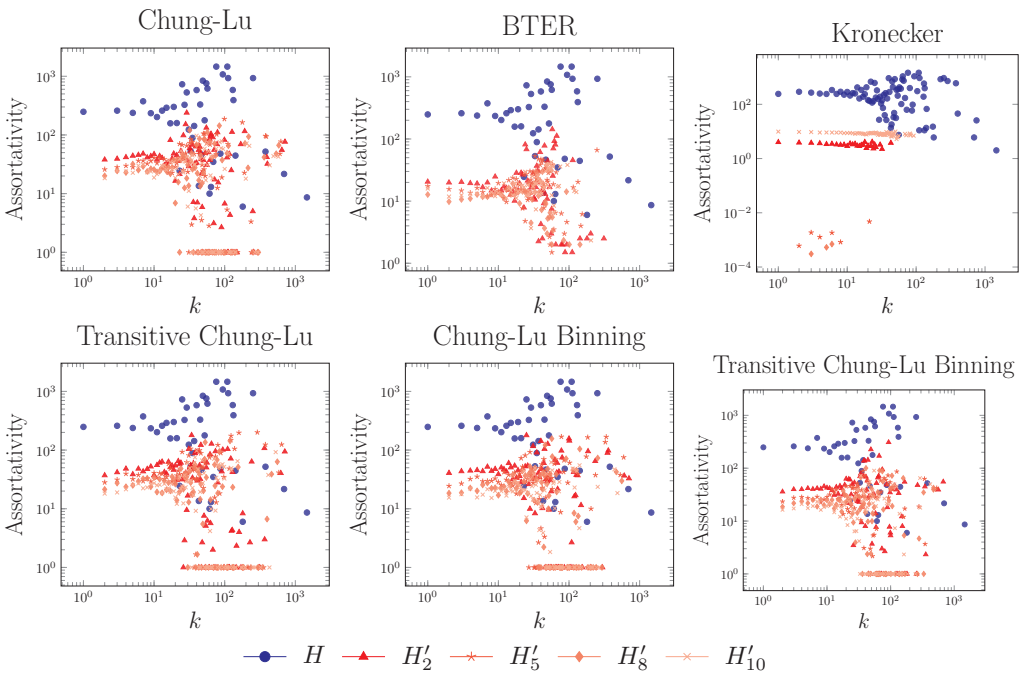

Figure 5.7. Assortativity. H is in blue. Results for recurrences H'_2, H'_5, H'_8 and H'_{10} in lighter and lighter shades of red. Degeneration is observed when recurrences increasingly deviate from H.

simply the average of all the local clustering coefficients.

The Chung Lu generator has been shown to model the degree distribution of some input graph, and our results bare this out. Eigenvector centrality, hop plot and graphlet correlation distances are also reasonably well modelled by the Chung Lu generator. However, Pfeiffer *et al.* recently showed that the standard Chung Lu generator does not well model a graph's local clustering coefficients; so they introduced the Transitive Chung Lu generator as an adaptation to the standard model [95].

5.2.1.6 Assortativity.

The assortativity of a network is its tendency to have edges between nodes with similar degree. For example, if high degree nodes primarily link to other high degree

nodes, and low degree nodes primarily link to low degree nodes, then the network's overall assortativity score will be high, and vice versa. The local assortativity for each node is the amount, positive or negative, that the node contributes to the overall global assortativity [91].

5.2.2 Robustness of Chung-Lu Extensions

Like in the case with the clustering coefficient, the standard Chung Lu model was found to not accurately model the assortativity of real world graphs. Mussmann *et al.* developed a Chung Lu with Binning adaptation that was shown to generate graphs with appropriate assortativity [85]. Even better is that the transitive and binning models can be combined to create a Transitive Chung Lu with Binning generator that models the degree distribution, clustering coefficient and assortativity of some input graph.

But the question remains, are these new generators robust?

We applied the infinity mirror test to the 6 graph generators, 3 original and 3 Chung Lu adaptations on the Routers dataset. All tests were performed on all graphs for all generators, but cannot all be shown because of space limitations. Figure 5.6 shows the clustering coefficient results. We find that transitive Chung Lu does nominally better than standard Chung Lu, but in all cases, the 5th, 8th and 10th recurrences seem to drift away (up and to the right) from original graph's plots demonstrating slight model degeneration as expressed through clustering coefficient. The Kronecker generator did rather poorly in this test. The Kronecker generator didn't seem to have a degeneration pattern, but was simply inconsistent.

The assortativity results are shown in Figure 5.7. We do not see any noticeable improvement in assortativity between the standard Chung Lu and the Chung Lu with Binning generators. We again find that the 5th, 8th and 10th recurrences seem to drift away (downward) from the original graph's assortativity plots demonstrating

slight model degeneration as expressed through assortativity. The Kronecker graph also performed poorly on this test, although it is unclear what the nature of the degeneration is.

5.3 Infinity Mirror for HRG

For HRG this infinity mirror test means that we learn a set of production rules from the original graph H and generate a new graph H^*; then we set $H \leftarrow H^*$ and repeat whereby learning a new model from the generated graph recursively. We repeat this process ten times and compare the output of the 10th recurrence with the original graph using GCD.

We expect to see that all models degenerate over ten recurrences. The question is, how quickly do the models degenerate and how badly do the graphs become?

Figure 5.8 shows the GCD scores for the HRG, Chung-Lu and Kronecker models at each recurrence (we have also validated the Infinity Mirror tests with other variations to the Chung-Lu model including the Block Two-Level Erdős-Rényi Model with similar results [3]). Surprisingly, we find that HRG stays steady, and even improves its performance while the Kronecker and Chung-Lu models steadily decrease their performance as expected. We do not yet know why HRG improves performance in some cases. Because GCD measures the graphlet correlations between two graphs, the improvement in GCD may be because HRG is implicitly homing in on rules that generate the necessary graph patterns.

5.3.1 Infinity Mirror Model Size

The number of production rules derived from a given graph using Fixed-Size Graph Generation. Fig. 5.9 shows the number of nodes in graphs after 1, 5, and 10 feedback iterations. The trend for each input graph varies slightly, but in general the model-size (*i.e.*, the number of production rules derived) stays flat.

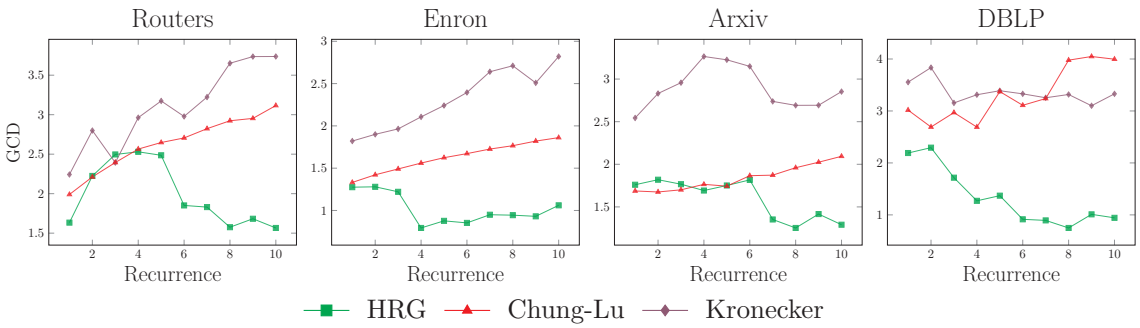

Figure 5.8. Infinity Mirror: GCD comparison after each recurrence. Unlike Kronecker and Chung-Lu models, HRG does not degenerate as its model is applied repeatedly.

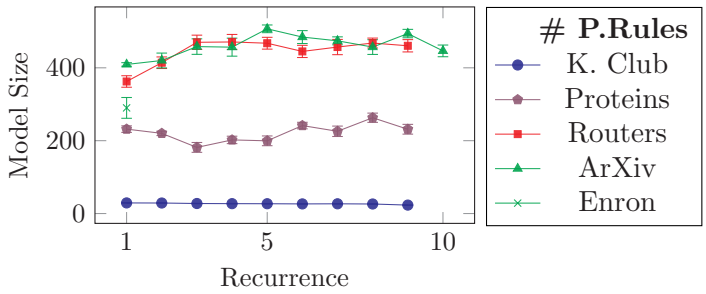

Figure 5.9. Number of rules (mean over 20 runs) derived as the number of recurrences increases.

5.4 Discussion

In the present work we introduced the infinity mirror test for graph generator robustness. This test operates by recursively generating a graph and fitting a model to the newly generated graph. A perfect graph generator would have no deviation from the original or ideal graph, however the implicit biases and assumptions that are cooked into the various models are exaggerated by the infinity mirror test allowing for new insights that were not available before.

Although the infinity mirror test shows that certain graph models show degen-

eration of certain properties in certain circumstances, it is more important to gain insight from how a model is degenerating in order to understand their failures and make improvements. For example, the BTER results in Figs 5.2-5.4 shows via the degree, eigenvector and hop plots that the BTER-generated graphs tend to become more spread out, with fewer and fewer cross-graph links, which, in retrospect, seems reasonable because of the siloed way in which BTER computes its model. Conversely, Chung Lu tends to generate graphs with an increasingly well connected core (indicated by the left-skewed hop plots and overestimated eigenvector centrality), but that also have an increasingly large portion of the generated graph that is sparsely connected (indicated by the odd shaped tail in the right-hand side of the eigenvector centrality plots).

Most importantly, we find that the HRG model actually gets better as the number of recurrences increases. It is still unclear why this improvement is happening contrary our intuition. The improvement in results may be evidence that the HRG model is learning to hone in on rules that succinctly describe the reference graph.

A better understanding of how the model degenerates will shed light on the inherent limitations. We hope that researchers and practitioners can consider using this method in order to understand the biases in their models and therefore create more robust graph generators in the future.

CHAPTER 6

TREE DECOMPOSITION

The goal of the HRG model is to derive a set of replacement rules that represent the structure of some input graph. Applying these iteratively, we grow graphs with characteristic properties similar to the input graph. Recall from prior chapters that finding an optimal tree decomposition and corresponding minimal-width tree decomposition is NP-Complete [7, 121]. Fortunately, many reasonable approximations exist for general graphs. Prior chapters employed the commonly used maximum cardinality search (MCS) algorithm introduced by Tarjan and Yannikakis [112] in 1985. MCS is a straightforward algorithm that creates a reasonable, but probably non-optimal, tree decomposition. A surge in recent theoretical and application-oriented projects has made a tremendous impact by finding bounded and near-optimal heuristics for real-world graphs [1, 15, 18, 19]. Each tree decomposition algorithm has certain heuristics and implementation decisions that are unavoidable; these decisions may introduce bias, which may affect the shape of the tree decomposition. For example, the MCS algorithm chooses an *elimination ordering*, *i.e.*, the ordering of nodes in the tree decomposition, based, in part, on the number of edges each node has.

This begs the question: How much does the choice of tree decomposition algorithm affect the shape of the tree decomposition?

The goal of this task is to understand the relationship between a tree decomposition algorithm, the resulting tree, and the extracted HRG. Even though the resulting tree decompositions may prove to be of different shapes, the extracted HRG may still be stable because the node labels are not copied into the grammar. It is therefore

possible that different tree decompositions will still produce very similar HRGs.

The original implementation of the MCS algorithm yielded tree decompositions with relatively low width. Recall that the treewidth is the size of the largest vertex set in an optimal/minimal-width tree decomposition of a given graph. Vertex sets are also characterized as cliques, thus the treewidth corresponds to the size of the largest clique in a chordal completion of the graph.

The goal of computing fast, heuristic, and non-optimal tree decompositions is finding a tree that has a width that is as close as possible to the treewidth. This is of broad interested because finding an optimal (or close to optimal) tree decomposition is used to solve problems in probabilistic inference, constraint satisfaction, and matrix decomposition [44, 62, 62].

In this chapter we study 6 different tree decomposition heuristics and describe their extracted HRGs. We also report the overlap of the intersection of the 6 individual HRGs. We call this intersection of HRGs a "super-HRG," and we report the characteristics of the graphs that are grown with such a model.

The tree decomposition test methodology was conceived by myself with input from Prof. Weninger. The evaluation, methodology and result plots were created by myself.

6.1 Background

A great deal of work has focused on applications of the algorithms that construct a graph tree decomposition. The concept of tree decomposition is also known by other other names depending on the specific branch of computer science and they include clique-trees, cluster graphs, and junction trees.

Typically, tree decomposition construction is based on algorithms for decomposing chordal graphs where a graph is deemed to be chordal if it has no induced cycles of length great than tree. That is, every cycle in the graph has a chord. Within chordal graphs, there always exists an ordering of vertices $v_1, \ldots, v_n$ such that for each v_i its

neighbors V_j with $j > i$ form a clique. This ordering is called a perfect elimination ordering, and it creates an optimal tree decomposition of the chordal graph [1].

Because of this property of chordal graphs, finding a tree decomposition for a general graph often begins with creating a chordal graph and then finding the associated tree decomposition. Unfortunately, finding an optimal chordal graph (*i.e.*, with the fewest fill-edges) is called graph triangulation and is NP-Complete. Thus, finding the optimal tree decomposition is also NP-Complete.

Existing algorithms for constructing tree decompositions range from those shown to be practically intractable due to their complexity to those characterized as being computationally efficient but whose trees are far from optimal. Shoikhet and Geiger developed a tractable algorithm, QuickTree, that can triangulate graphs in a reasonable amount of time [105]. This algorithm saw usefulness in problems in Bayesian inference and clique-tree inference algorithms. Gogate and Dechter developed QuickBB, a branch and bound algorithm for computing the tree decompositions on undirected graphs [39]. Jones *et al.*, describe very similar work where they extracted grammars using tree decomposition, but their work focuses on a class of tree decomposition called edge-mapped tree decompositions [53]. This tree decomposition is extended to use a topological sort that produces tree decompositions like those in HRG. However, they evaluate the model in contrast to other forms of the tree decomposition yielding linear trees, and they rely on a measure of perplexity to evaluate the production rules. Their work does not generate graphs using the derived rules and their approach is tested only on a series of graphs that have an average of 15 vertices.

Because of the intractability of exact/optimal algorithms, we resign to finding good elimination orderings through heuristics in order to create good, but not necessarily optimal, tree decompositions.

Many such elimination ordering heuristics exist. The first set of heuristics we

discuss are based on the graph triangulation. These are called lexicographic breadth first search (LEXBFS) and maximum cardinality search (MCS), as well as variations that find minimal triangulations on mcs (MCSM) and LEXBFS (LEXM). When used on chordal graphs, these heuristics produce a perfect elimination ordering, so it is thought that they would also produce near-optimal results on general graphs. However, these heuristics are computationally expensive and struggle to run on very large graphs, which is why the HRG extraction methods from the previous chapters sampled subgraphs from the input networks.

Alternatively, there exist many greedy heuristics that choose an elimination ordering in a greedy manner. There are many varieties of greedy elimination ordering algorithms. Two basic varieties chose an elimination ordering based on minimizing fill (*i.e.*, how many new edges would be added to a graph if a particular vertex is chosen next) or minimizing degree (*i.e.*, how many neighbors does the vertex have). We call these greedy measures min-fill (MINF) and min-degree (MIND) respectively. These greedy orderings are less computationally expensive than LEXM and MCS and there are ongoing efforts to further reduce their cost. For example, a multiple mindegree (MMD) algorithm has been proposed that eliminates multiple vertices in the same step [1].

6.2 Comparing Tree Decompositions

The question of interest is how does our choice of tree decomposition bias the productions rules derived from the input graph? To address this question we derive HRGs using various tree decompositions algorithms and compare the results.

A battery of elimination ordering heuristics are compared. These heuristics include maximum cardinality search (MCS), and its minimal triangulation variation (MCSM) [11], lexicographic search with minimal triangulation (LEXM) [101], minimum fill-in (MINF), minimum degree (MIND), and Multiple Minimum De-

TABLE 6.1

SUMMARY: ELIMINATION ORDERING ALGORITHMS.

VE Heuristic	Description
MCS	Maximum cardinality search is a simple heuristic that works well on chordal graphs.
MCSM	Minimum triangulation extension to MCS.
MIND	Minimum degree is a well known general-purpose ordering scheme and is widely used in sparse matrix computation.
MINF	Minimum fill consists of greedy node elimination with the fewest edges are added breaking ties arbitrarily.
LEXM	Derived from lexicographic breadth-first search for minimal triangulation.
MMD	Multiple minimum degree variation to MIND

gree (MMD) [78]. For a more complete description see the survey by Kemazi and Poole [56]. Also note that these heuristics do not represent the complete list of available algorithms [12, 94]. We used the INDDGO implementation of these algorithms [1].

We examine and evaluate tree decompositions from the heuristics in Table 6.1. We select a diverse set of real world networks. Table 6.2 shows some of the properties of these graphs.

6.2.1 Datasets

A number of empirical networks are used to evaluate the different heuristics. These networks vary in the number of nodes and edges. We consider small and large networks, but more importantly we want a wide range and diverse set of graph

grammars (graph fragments) to understand how the tree decomposition contributes to the HRG model. These networks are also characterized by other properties that we hope HRG is able to model accurately.

- **Small Network Datasets.** The PDZBase graph is a network of protein–protein interactions from PDZBase. The UCForum graph is a bipartite network of students at UC Irvine and an Online message board. The LesMis graph draws an edge between characters in the theatrical play Les Miserables if they appear in a scene together. The Conference graph describes face-to-face conversations during the Hypertext conference. The Jazz network is a network of jazz musicians where an edge is drawn if two musicians collaborated. The Email network is a graph of email communication between members of a large research institution in Europe. The Phone network is a network of phone calls made between students at a university. The Infectious network is like the Conference network in that it draws an edge between people if they interacted at a conference on infection diseases. These networks are publicly available from KONECT [68] or SNAP[71].
- **Large Network Datasets.** The EuroRoad network is an infrastructure network of Europe's roads where nodes are cities and edges represent the road that connects them. The CollegeMsg network is from private messages between college students using an online social network at the University of California, Irvine.

Table 6.2 illustrates the size of the graphs used in the experimental section.

Along with the treewidth (*i.e.*, the width of the optimal tree decomposition) we computed the width of each tree as constructed by the six tree decomposition algorithms. The results are illustrated in Tab. 6.3. The EuroRoad and CollegeMsg networks did not finish in a reasonable time, so we report the mean-average treewidth of 20 samples of 300-node subgraphs. We find that each elimination ordering performs reasonably close to optimal when the treewidth is small, but far worse when the treewith is large.

TABLE 6.2

REAL NETWORKS

Datasets	nodes (n)	edges (m)	Avg. degree ($\bar{k}$)
PDZBase	212	244	2.3
LesMis	77	254	6.60
Conference	113	2196	38.87
Jazz	198	2742	27.70
Phone	274	2124	15.50
Email	309	1938	12.54
Infectious	410	2765	13.49
UCForum	2320	7,089	47.46
EuroRoad	1174	1417	2.41
CollegeMsg	1899	13838	14.57

6.3 Methodology

Here we evaluate the affect the choice of elimination ordering heuristic has on the construction of new graphs. For each elimination ordering we generated 20 graphs and compared their distance to the original graph with GCD. Figure 6.4 shows that the choice between MCS and MINF, for example, results in little practical difference in the GCD score. The best performing elimination ordering heursitic was inconsistent across the different data sets.

In addition to the GCD results, Fig. 6.1 shows the results of various elimination orderings across various various graphs on different graph properties. As in the GCD results, these results show mixed results for various graph properties.

TABLE 6.3

WIDTH AS A FUNCTION OF ELIMINATION ORDERING

	Width						
Dataset	tw	MCS	LEXM	MCSM	MIND	MINF	MMD
PDZBase	6	9	12	13	6	6	6
LesMis	9	11	9	11	9	9	9
Conference	76	80	89	89	76	76	76
Jazz	59	88	77	81	104	59	73
Phone	40	42	43	43	50	40	40
Email	34	41	46	45	35	34	35
Infectious	39	65	56	128	42	40	49
UCForum	126	326	361	341	282	276	279
EuroRoad	6.6*	42	30	48	19	16	16
CollegeMsg	87.6*	459	602	543	404	394	403

6.3.1 Variance in Tree Decomposition

These results show that the choice of elimination ordering heuristic has little effect on the fidelity of the generated graph. Our next question asks: what is the overlap of the rules within the various HRGs produced by the various elimination orderings.

To answer this question we find the intersection of each HRG. Specifically, we say that a production rule from, say, HRG_{mcs} intersects with a production rules from, say, HRG_{mind} if the nonterminal hyperedges on both LHSs have the same arity, and if the graphlets on the RHSs are isomorphic.

We compute the Jaccard similarity to assess the ratio of isomorphic rules for each elimination ordering heuristic. The results of this test are illustrated in Tab. 6.5. This

TABLE 6.4

GCD USING AN ARRAY OF ELIMINATION ORDERING HEURISTICS

Dataset	MCS	LEXM	MIND	MCSM	MINF	MMD
PDZBase	**1.8 (1.3)**	2.2 (1.5)	4.0 (.40)	3.2 (1.3)	3.8 (0.7)	2.7 (0.27)
LesMis	1.46 (0.16)	1.6 (0.16)	1.7 (0.2)	1.8 (0.13)	**1.4 (0.15)**	1.774 (0.280)
Conference	0.707 (0.137)	0.582 (0.112)	0.606 (0.143)	0.576 (0.137)	0.661 (0.154)	0.653 (0.177)
Jazz	1.2 (0.1)	0.98 (0.03)	1.2 (1.6)	**0.75 (0.04)**	1.11 (0.09)	0.977 (0.083)
Email	3.0456 (0.118)	2.554 (0.100)	**2.552 (0.248)**	2.491 (0.151)	2.862 (0.269)	2.679 (0.267)
Infectious	0.760 (0.060)	0.889 (0.083)	0.824 (0.060)	0.684 (0.060)	0.857 (0.0613)	0.763 (0.085)
UCForum	0.985 (0.093)	1.030 (0.098)	1.198 (0.093)	0.948 (0.113)	1.242 (0.107)	1.002 (0.106)
EuroRoad	1.633 (1.059)	2.300 (1.237)	2.820 (1.231)	2.728 (1.239)	3.003 (1.186)	3.224 (1.102)
CollegeMsg		3.276 (0.275)	0.835 (0.036)	0.605 (0.047)	0.773 (0.059)	0.773 (0.059)

result implies that certain elimination ordering heuristics share an affinity, but we are uncertain what the exact nature of these results represent. Overall, we find that there is little overlap in the production rules extracted from various HRG models.

6.4 Discussion

Previous results show that the HRG model preserves many of the input graph's network properties. In this chapter we investigated the how the choice of tree decomposition affects the extracted model. We examined a battery of tree decomposition algorithms to examine the production rules and find an answer to this question. Our initial results highlight two important findings: elimination ordering heuristics produce mostly non-overlapping production rules, but the generated trees were largely immune to the differences in HRG models.

We intended to exploit this overlap in HRG models to test if a global intersection of isomorphic graph-fragments from all elimination ordering heuristics results in a robust, reduced set of production rules that generates graphs. This rule-intersection approach would lets us hone in on derived rules because the graph-fragments produced

TABLE 6.5

PRODUCTION RULES OVERLAP: EMAIL-EU DATASET

	MCS	MIND	MCSM	LEXM	MMD	MINF
mcs	-	0.182	0.182	0.161	0.194	0.186
mind		-	0.207	0.159	0.220	0.209
mcsm			-	0.180	0.201	0.191
lexm				-	0.157	0.155
mmd					-	0.184
minf						-

by different tree decomposition algorithms might indicate a stable set of meaningful production rules, rather than arbitrary rules extracted from a heuristically generated tree decomposition.

Unfortunately, the rule-intersection HRGs sometimes did not generate any graphs at all because the rule-intersection was so sparse so that there were no LHSs to match nonterminals during graph generation. Furthermore, finding graphlet intersections was a tedious and computationally expensive process.

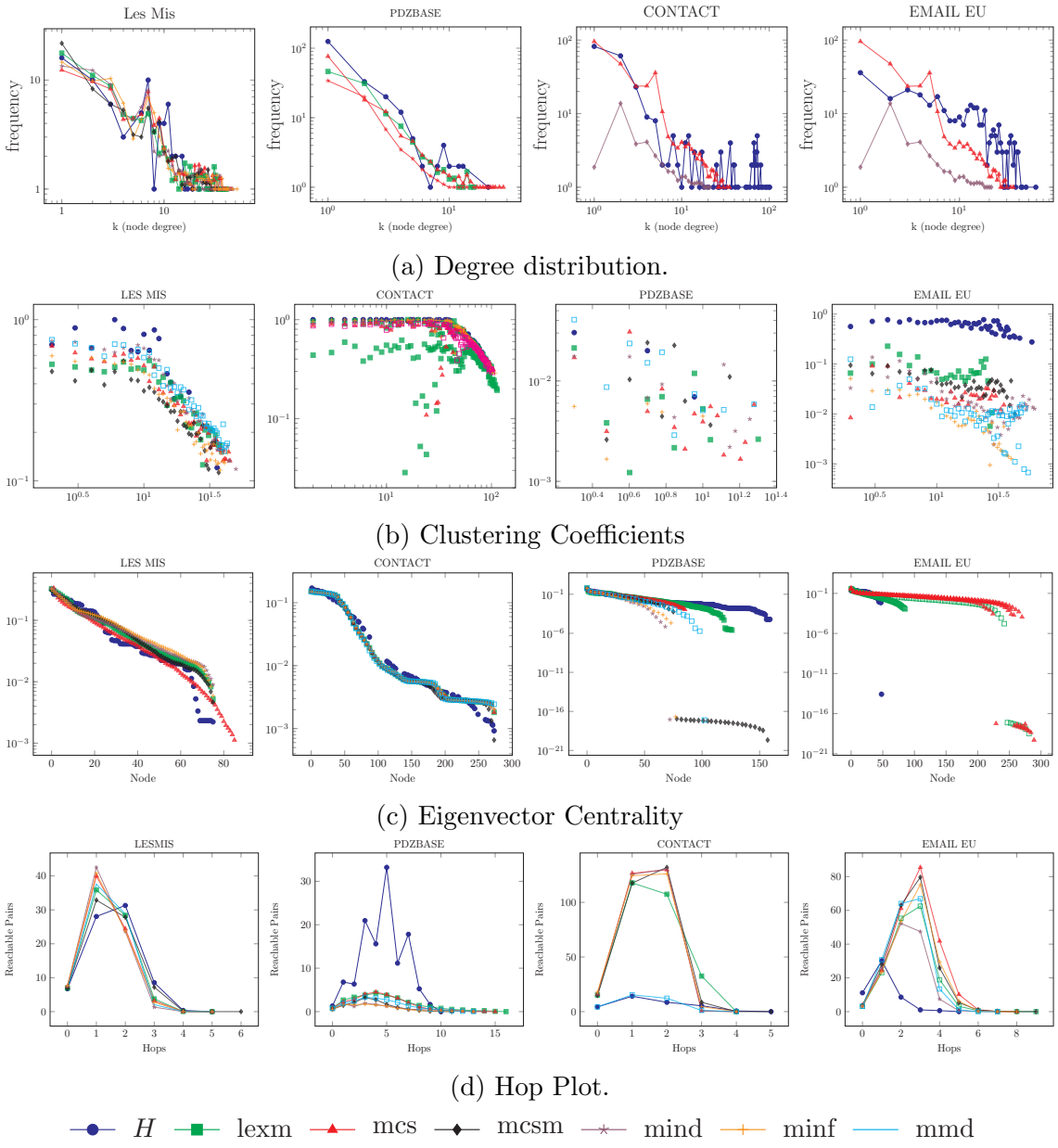

Figure 6.1. Results of generating graphs from various elimination orderings

CHAPTER 7

SUMMARY AND FUTURE DIRECTIONS

Here we have explored and evaluated principled techniques that learn the LEGO-like building blocks of real-world networks. We did so by exploiting techniques at the overlap where graph theory meets formal language theory. Building on these ideas, pioneered by my collaborators Tim Weninger and David Chiang, we examined graph decomposition, grammar extraction, model inference, and analysis of network patterns.

We focused on advancing HRG, a graph rewriting formalism, to extract graph grammars from any class of connected graphs. This model was able to learn the building blocks of networks and leveraged the generating power of HRGs to construct graphs that exhibit, or maintain, the network properties of interest found in the reference graph. The specific themes covered include model inference and stochastic graph generation, HRG fixed-size graph generation, measures of model resilience, and model bias resulting from the use of different tree decomposition algorithms. We expanded on these themes below in more detail.

- **Constructing Graphs with HRGs.** First we provide a description of how to derive an HRG by transforming the reference graph into a tree decomposition, deriving the graph grammar from the tree decomposition. We made further improvements to the HRG model so to create a fixed size algorithm, where the size, in terms of the number of edges or nodes) could be specified.
- **Evaluating HRGs** Next we evaluated the ability of the HRG model to generate graphs that are similar to the reference graph. In Chapter 4 we show that an application of the HRG rules creates new graphs that have very similar properties to the original graph. The results from standard network measures,

graphlet counts, and graphlet correlation distance metrics show a stark improvement in performance over several existing graph generators.

- **Infinity mirror tests for robustness.** A new metric of model robustness called the Infinity Mirror test examined model degeneration in Chapter 5. This procedure repeatedly learned a graph model from the output of a previously generated graph. We found that after a few recurrences most of the existing graph models degenerated, and some models quickly lost their ability to create cogent graphs. The HRG model, on the other hand, actually got better as the number of recurrences increased.

- **Tree decomposition bias.** We explored core tree decomposition step of the HRG model. Specifically, how the choice of elimination ordering during tree decomposition affected the construction and composition of the HRGs. Chapter 6 examined six different elimination orderings and found that they were equally able to generate good graphs, despite the observation that their rule set did not overlap.

7.1 Collaborations

7.1.1 HRG Extension to Temporal Graphs

A critical challenge in many applications is related to the constant changing nature of the data, for instance the dynamic properties that are observed along a temporal dimension. Consider for example the behavior of a friendship network. Depending on how a network of friends is defined, new friends may join the network over time. If we define the network more by the function of the ties (the interactions between friends) we can have at any one time only certain friends active and others dormant in their interactions with other members of the group. A graphical abstraction for this system is characterized by the ties that appear (active) and by those that disappear.

Consider another scenario that looks at the travelers at an airport. Modeling the incoming and outgoing flights of the passengers is best done by considering the temporal dynamics. Alternatively, we can potentially model the dynamics of how passengers move about a fixed space. The latter is concerned with a spatial component of the data, but still tightly coupled to the temporal dimension. How can

we incorporate the temporal properties of the nodes and edges into a model? This requires extending the HRG model to examine the time-stamps of edges. This idea was explored and the results presented at the Mining and Learning with Graphs Workshop [93]. Although not fully explored in this thesis, this illuminates another avenue for studying the application of HRG graph model.

7.1.2 Latent Variable Probabilistic Graph Grammars

This work address one of the limitations of the HRG model. The HRG model encodes only sufficient information to ensure that the result is properly formed. To do this, the tree decomposition step transforms the input graph into tree. The rules we extract come from the top, middle and bottom of the tree graph, however, may play different roles in the construction of the graph. The central question here is how do we provide context to the rules that is less aligned on their frequency and more on where in the tree they come from. This context could influence when the rules should fire when generating a new graph. My collaborator Xinyi Wang proposed a mechanism that corrects for the problem described above. We model an HRG-rule's context via latent variables and show that this approach can build better synthetic graphs. This work again illuminates avenue for additional research.

7.2 Vision and Future Work

By developing algorithms for graph generation models, we elucidate underlying mechanisms contributing to network composition. Understanding local patterns of large networks can lead to high impact applications, novel mathematical abstractions, and to the development of sorely needed tools for the advancement of field today. The concepts explored here naturally opens up new ground for further exploration.

7.2.1 Analytic Methods for the Network Properties of HRG Graphs

Analytically exploring the properties of HRG graphs is difficult, but important. Showing that HRG model yields graphs with properties that are analytically tractable might yield surprising results. Network properties such as degree distribution, diameter, and other spectral properties of the graphs should be explored further to determine if the grammar patterns model generated structures exhibit a certain degree distribution or any particular eigenvector distribution as is the case with Kronecker graphs. For example, in grammars obtained from connected graphs, can we analytically show that connected (or disconnected) graphs are possible?

7.2.2 Applications to Deep Learning

Exploiting the richness of the grammars for use in machine learning offers practical applications of the HRG model. One possible research path is in designing recurrent neural network controllers that optimize the selection of rules that generate neural network architectures. We could then evaluate the architecture on deep learning task. If it performs well, then we feed back the production rules to the recurrent neural network controller. This approach could help contribute to better-engineered high-performing neural network architectures over time [88, 107, 123].

7.2.3 Applications to Graph Engines

Our HRG model offers two potential applications in pyramid graph algorithms. First, to explore using hyperedge replacement for layer-to-layer contraction. Second, to investigate extracting rules set for selecting contraction algorithms according to the graph class in question. In combinatorial optimization tasks, approximation algorithms offer solutions to otherwise NP-hard or NP-Complete problems, if applied to a decision problem. A pyramid algorithm would be one example. In human and computer vision literature, these algorithms are well utilized, but only more recently

have been applied to problem-solving tasks. Multi-resolution graph pyramids, where the bottom of the pyramid contains the entire graph and successive layers have compressed (or contracted) graph information reduces the size of the input as we climb up the pyramid and reach a point where a combinatorial solution to the problem is feasible [13, 45, 84, 97]. How graphs contract from one pyramid level to the next depends on the class to which the network belongs.

BIBLIOGRAPHY

1. A. B. Adcock, B. D. Sullivan, and M. W. Mahoney. Tree decompositions and social graphs. *Internet Mathematics*, 12(5):315–361, 2016.

2. S. Aguinaga and T. Weninger. The infinity mirror test for analyzing the robustness of graph generators. In *ACM SIGKDD Workshop on Mining and Learning with Graphs*, MLG '16, New York, NY, USA, 2016. ACM.

3. S. Aguinaga and T. Weninger. The infinity mirror test for analyzing the robustness of graph generators. In *KDD Workshop on Mining and Learning with Graphs*. ACM, 2016.

4. S. Aguinaga, R. Palacios, D. Chiang, and T. Weninger. Growing graphs from hyperedge replacement grammars. In *CIKM*. ACM, 2016.

5. N. K. Ahmed, J. Neville, R. A. Rossi, and N. Duffield. Efficient graphlet counting for large networks. In *Data Mining (ICDM), 2015 IEEE International Conference on*, pages 1–10. IEEE, 2015.

6. R. Albert and A.-L. Barabási. Statistical mechanics of complex networks. *Reviews of modern physics*, 74(1):47, 2002.

7. S. Arnborg, D. G. Corneil, and A. Proskurowski. Complexity of finding embeddings in ak-tree. *SIAM Journal on Algebraic Discrete Methods*, 8(2):277–284, 1987.

8. A.-L. Barabási and R. Albert. Emergence of scaling in random networks. *science*, 286(5439):509–512, 1999.

9. A.-L. Barabasi, D. J. Watts, and M. Newman. *Princeton Studies in Complexity : Structure and Dynamics of Networks*. Princeton University Press, Princeton, NJ, USA, 201110 2011. ISBN 9781400841356. ID: 10578571.

10. M. Barthelemy. Betweenness centrality in large complex networks. *The European Physical Journal B-Condensed Matter and Complex Systems*, 38(2):163–168, 2004.

11. A. Berry, J. R. Blair, P. Heggernes, and B. W. Peyton. Maximum cardinality search for computing minimal triangulations of graphs. *Algorithmica*, 39(4): 287–298, 2004.

12. A. Berry, R. Pogorelcnik, and G. Simonet. Organizing the atoms of the clique separator decomposition into an atom tree. *Discrete Applied Mathematics*, 177: 1–13, 2014.

13. M. Bister, J. Cornelis, and A. Rosenfeld. A critical view of pyramid segmentation algorithms. *Pattern Recognition Letters*, 11(9):605–617, 1990.

14. H. L. Bodlaender. A linear-time algorithm for finding tree-decompositions of small treewidth. *SIAM Journal on computing*, 25(6):1305–1317, 1996.

15. H. L. Bodlaender and A. M. Koster. Treewidth computations i. upper bounds. *Information and Computation*, 208(3):259–275, 2010.

16. E. Bullmore and O. Sporns. Complex brain networks: graph theoretical analysis of structural and functional systems. *Nature Reviews Neuroscience*, 10(3):186–198, 2009.

17. D. Chakrabarti, Y. Zhan, and C. Faloutsos. R-mat: A recursive model for graph mining. In *SDM*, volume 4, pages 442–446. SIAM, 2004.

18. F. Chang, C.-Y. Guo, X.-R. Lin, and C.-J. Lu. Tree decomposition for large-scale svm problems. *Journal of Machine Learning Research*, 11(Oct):2935–2972, 2010.

19. C. Chekuri and J. Chuzhoy. Large-treewidth graph decompositions and applications. In *Proceedings of the forty-fifth annual ACM symposium on Theory of computing*, pages 291–300. ACM, 2013.

20. D. Chiang, J. Andreas, D. Bauer, K. M. Hermann, B. Jones, and K. Knight. Parsing graphs with hyperedge replacement grammars. In *ACL (1)*, pages 924–932, 2013.

21. F. Chung and L. Lu. Connected components in random graphs with given expected degree sequences. *Annals of combinatorics*, 6(2):125–145, 2002.

22. F. Chung and L. Lu. The average distances in random graphs with given expected degrees. *Proceedings of the National Academy of Sciences*, 99(25):15879–15882, 2002.

23. F. Chung and L. Lu. Connected components in random graphs with given expected degree sequences. *Annals of combinatorics*, 6(2):125–145, 2002.

24. D. J. Cook and L. B. Holder. Substructure discovery using minimum description length and background knowledge. *Journal of Artificial Intelligence Research*, pages 231–255, 1994. URL `http://www.jair.org/media/43/live-43-1384-jair.pdf`.

25. W. F. Doolittle and E. Bapteste. Pattern pluralism and the tree of life hypothesis. *Proceedings of the National Academy of Sciences*, 104(7):2043–2049, 2007.

26. F. Drewes, H.-J. Kreowski, and A. Habel. Hyperedge replacement, graph grammars. *Handbook of Graph Grammars*, 1:95–162, 1997.

27. F. Drewes, B. Hoffmann, and D. Plump. Hierarchical graph transformation. *Journal of Computer and System Sciences*, 64(2):249–283, 2002.

28. D. Easley and J. Kleinberg. *Networks, crowds, and markets: Reasoning about a highly connected world*. Cambridge University Press, 2010.

29. P. Erdos and A. Rényi. On the evolution of random graphs. *Publ. Math. Inst. Hungar. Acad. Sci*, 5:17–61, 1960.

30. P. Erdos and A. Rényi. On the evolution of random graphs. *Bull. Inst. Internat. Statist*, 38(4):343–347, 1961.

31. L. Euler. Solutio problematis ad geometriam situs pertinentis. *C. Petr.*, 8:128, 1736(1741).

32. M. Faloutsos, P. Faloutsos, and C. Faloutsos. On power-law relationships of the internet topology. In *ACM SIGCOMM computer communication review*, volume 29, pages 251–262. ACM, 1999.

33. L. C. Freeman. Centrality in social networks conceptual clarification. *Social networks*, 1(3):215–239, 1978.

34. L. C. Freeman. The development of social network analysis-with an emphasis on recent events. *The sage handbook of social network analysis*, 21(3):26–39, 2011.

35. P. Fronczak, A. Fronczak, and M. Bujok. Exponential random graph models for networks with community structure. *Phys. Rev. E*, 88:032810, Sept. 2013. doi: 10.1103/PhysRevE.88.032810. URL `http://link.aps.org/doi/10.1103/PhysRevE.88.032810`.

36. S. Geman and M. Johnson. Dynamic programming for parsing and estimation of stochastic unification-based grammars. In *ACL*, pages 279–286. Association for Computational Linguistics, 2002.

37. E. N. Gilbert. Random graphs. *Ann. Math. Statist.*, 30(4):1141–1144, 12 1959. doi: 10.1214/aoms/1177706098. URL `http://dx.doi.org/10.1214/aoms/1177706098`.

38. M. Girvan and M. E. Newman. Community structure in social and biological networks. *Proceedings of the national academy of sciences*, 99(12):7821–7826, 2002.

39. V. Gogate and R. Dechter. A complete anytime algorithm for treewidth. In *Proceedings of the 20th conference on Uncertainty in artificial intelligence*, pages 201–208. AUAI Press, 2004.

40. A. Goldenberg, A. X. Zheng, S. E. Fienberg, and E. M. Airoldi. A survey of statistical network models. *Foundations and Trends® in Machine Learning*, 2 (2):129–233, 2010.

41. S. M. Goodreau, J. A. Kitts, and M. Morris. Birds of a feather, or friend of a friend? using exponential random graph models to investigate adolescent social networks. *Demography*, 46(1):103–125, 2009.

42. G. Grahne and J. Zhu. Fast algorithms for frequent itemset mining using fp-trees. *TKDE*, 17(10):1347–1362, 2005.

43. M. S. Granovetter. The strength of weak ties. *American journal of sociology*, pages 1360–1380, 1973.

44. H. Guo and W. Hsu. A survey of algorithms for real-time bayesian network inference. In *AAAI/KDD/UAI02 Joint Workshop on Real-Time Decision Support and Diagnosis Systems*. Edmonton, Canada, 2002.

45. Y. Haxhimusa, W. G. Kropatsch, Z. Pizlo, A. Ion, and A. Lehrbaum. Approximating tsp solution by mst based graph pyramid. *GbRPR*, 4538:295–306, 2007.

46. H. S. Heaps. *Information retrieval: Computational and theoretical aspects*. Academic Press, Inc., 1978.

47. L. B. Holder, D. J. Cook, S. Djoko, et al. Substucture discovery in the subdue system. In *KDD workshop*, pages 169–180, 1994.

48. P. W. Holland and S. Leinhardt. An exponential family of probability distributions for directed graphs. *Journal of the american Statistical association*, 76 (373):33–50, 1981.

49. D. R. Hunter, M. S. Handcock, C. T. Butts, S. M. Goodreau, and M. Morris. ergm: A package to fit, simulate and diagnose exponential-family models for networks. *J Stat Softw*, 24(3):nihpa54860–nihpa54860, May 2008. ISSN 1548-7660. URL `http://www.ncbi.nlm.nih.gov/pmc/articles/PMC2743438/`. 19756229[pmid].

50. D. R. Hunter, M. S. Handcock, C. T. Butts, S. M. Goodreau, and M. Morris. ergm: A package to fit, simulate and diagnose exponential-family models for networks. *Journal of statistical software*, 24(3):nihpa54860, 2008.

51. T. A. Jarrell, Y. Wang, A. E. Bloniarz, C. A. Brittin, M. Xu, J. N. Thomson, D. G. Albertson, D. H. Hall, and S. W. Emmons. The connectome of a decision-making neural network. *Science*, 337(6093):437–444, 2012. ISSN 0036-8075. doi: 10.1126/science.1221762. URL `http://science.sciencemag.org/content/337/6093/437`.

52. C. Jiang, F. Coenen, and M. Zito. A survey of frequent subgraph mining algorithms. *The Knowledge Engineering Review*, 28(01):75–105, 2013.

53. B. K. Jones, S. Goldwater, and M. Johnson. Modeling graph languages with grammars extracted via tree decompositions. In *Proceedings of the 11th International Conference on Finite State Methods and Natural Language Processing*, pages 54–62, 2013.

54. I. Jonyer. Graph grammar learning. *Mining Graph Data*, pages 183–201, 2006.

55. S. A. Kauffman. *The origins of order: Self organization and selection in evolution*. Oxford University Press, USA, 1993.

56. S. M. Kazemi and D. Poole. Elimination ordering in lifted first-order probabilistic inference. In *AAAI*, pages 863–870, 2014.

57. C. Kemp and J. B. Tenenbaum. The discovery of structural form. *PNAS*, 105(31):10687–10692, 2008. doi: 10.1073/pnas.0802631105. URL `http://www.pnas.org/content/105/31/10687.abstract`.

58. C. Kemp and J. B. Tenenbaum. The discovery of structural form. *Proceedings of the National Academy of Sciences*, 105(31):10687–10692, 2008.

59. B. Klimt and Y. Yang. Introducing the enron corpus. In *CEAS*, 2004.

60. T. G. Kolda, A. Pinar, T. Plantenga, and C. Seshadhri. A scalable generative graph model with community structure. *SIAM Journal on Scientific Computing*, 36(5):C424–C452, 2014.

61. D. Koller and N. Friedman. *Probabilistic graphical models: principles and techniques*. MIT press, 2009.

62. A. M. Koster, S. P. van Hoesel, and A. W. Kolen. Solving partial constraint satisfaction problems with tree decomposition. *Networks*, 40(3):170–180, 2002.

63. T. S. Kuhn. *The structure of scientific revolutions*. University of Chicago press, 2012.

64. J. Kukluk, L. Holder, and D. Cook. Inferring graph grammars by detecting overlap in frequent subgraphs. *International Journal of Applied Mathematics and Computer Science*, 18(2):241–250, 2008.

65. J. P. Kukluk, L. B. Holder, and D. J. Cook. Inference of node replacement recursive graph grammars. In *SDM*, pages 544–548. SIAM, 2006.

66. J. P. Kukluk, C. H. You, L. B. Holder, and D. J. Cook. Learning node replacement graph grammars in metabolic pathways. In *BIOCOMP*, pages 44–50, 2007.

67. J. P. Kukluk, L. B. Holder, and D. J. Cook. Inference of edge replacement graph grammars. *International Journal on Artificial Intelligence Tools*, 17(03): 539–554, 2008.

68. J. Kunegis. Konect: the koblenz network collection. In *Proceedings of the 22nd International Conference on World Wide Web*, pages 1343–1350. ACM, 2013.

69. C. Lautemann. Decomposition trees: structured graph representation and efficient algorithms. In *CAAP'88*, pages 28–39. Springer, 1988.

70. J. Leskovec and C. Faloutsos. Sampling from large graphs. In *Proceedings of the 12th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 631–636. ACM, 2006.

71. J. Leskovec and A. Krevl. SNAP Datasets: Stanford large network dataset collection. `http://snap.stanford.edu/data`, June 2014 (Retrieved: 11/Nov/2017).

72. J. Leskovec, J. Kleinberg, and C. Faloutsos. Graphs over time: densification laws, shrinking diameters and possible explanations. In *SIGKDD*, pages 177–187. ACM, 2005.

73. J. Leskovec, J. Kleinberg, and C. Faloutsos. Graph evolution: Densification and shrinking diameters. *ACM Transactions on Knowledge Discovery from Data (TKDD)*, 1(1):2, 2007.

74. J. Leskovec, K. J. Lang, A. Dasgupta, and M. W. Mahoney. Statistical properties of community structure in large social and information networks. In *WWW*, pages 695–704. ACM, 2008.

75. J. Leskovec, D. Chakrabarti, J. Kleinberg, C. Faloutsos, and Z. Ghahramani. Kronecker graphs: An approach to modeling networks. *Journal of Machine Learning Research*, 11(Feb):985–1042, 2010.

76. B. Li, F. Z. Moataz, N. Nisse, and K. Suchan. Minimum size tree-decompositions. *Electronic Notes in Discrete Mathematics*, 50:21–27, 2015.

77. W. Lin, X. Xiao, and G. Ghinita. Large-scale frequent subgraph mining in mapreduce. In *ICDE*, pages 844–855. IEEE, 2014.

78. J. W. Liu. Modification of the minimum-degree algorithm by multiple elimination. *ACM Transactions on Mathematical Software (TOMS)*, 11(2):141–153, 1985.

79. M. H. Luerssen. Graph grammar encoding and evolution of automata networks. In *Proceedings of the Twenty-eighth Australasian conference on Computer Science-Volume 38*, pages 229–238. Australian Computer Society, Inc., 2005.

80. D. Marcus and Y. Shavitt. Efficient counting of network motifs. In *2010 IEEE 30th International Conference on Distributed Computing Systems Workshops*, pages 92–98, June 2010. doi: 10.1109/ICDCSW.2010.41.

81. D. Marcus and Y. Shavitt. Rage–a rapid graphlet enumerator for large networks. *Computer Networks*, 56(2):810–819, 2012.

82. O. J. Mengshoel. Understanding the scalability of bayesian network inference using clique tree growth curves. *Artificial Intelligence*, 174(12):984–1006, 2010.

83. J. L. Moreno. *Who shall survive?: A new approach to the problem of human interrelations*. Nervous and Mental Disease Publishing Co, 1934.

84. S. F. Mousavi, M. Safayani, and A. Mirzaei. Graph pyramid embedding in vector space. In *Computer and Knowledge Engineering (ICCKE), 2014 4th International eConference on*, pages 146–151. IEEE, 2014.

85. S. Mussmann, J. Moore, J. J. Pfeiffer, and J. Neville III. Assortativity in chung lu random graph models. In *Proceedings of the 8th Workshop on Social Network Mining and Analysis*, page 3. ACM, 2014.

86. S. Mussmann, J. Moore, J. J. Pfeiffer III, and J. Neville. Incorporating assortativity and degree dependence into scalable network models. In *AAAI*, pages 238–246, 2015.

87. M.-J. Nederhof and G. Satta. Probabilistic parsing as intersection. In *Proc. International Workshop on Parsing Technologies*, 2003.

88. R. Negrinho and G. Gordon. Deeparchitect: Automatically designing and training deep architectures. *arXiv preprint arXiv:1704.08792*, 2017.

89. M. Newman. *Networks: An Introduction*. Oxford University Press, Inc., New York, NY, USA, 2010. ISBN 0199206651, 9780199206650.

90. M. Newman. *Networks: An Introduction*. Oxford University Press, Inc., 2010.

91. M. E. Newman. Mixing patterns in networks. *Physical Review E*, 67(2):026126, 2003.

92. S. Nijssen and J. N. Kok. The gaston tool for frequent subgraph mining. *Electronic Notes in Theoretical Computer Science*, 127(1):77–87, 2005.

93. C. Pennycu, S. Aguinaga, and T. Weninger. A temporal tree decomposition for generating temporal graphs. *KDD Workshop on Mining and Learning with Graphs*, 2011.

94. N. Peyrard, S. De Givry, A. Franc, S. Robin, R. Sabbadin, T. Schiex, and M. Vignes. Exact and approximate inference in graphical models: variable elimination and beyond. *arXiv preprint arXiv:1506.08544*, 2015.

95. J. J. Pfeiffer, T. La Fond, S. Moreno, and J. Neville. Fast generation of large scale social networks while incorporating transitive closures. In *Privacy, Security, Risk and Trust (PASSAT), 2012 International Conference on and 2012 International Confernece on Social Computing (SocialCom)*, pages 154–165. IEEE, 2012.

96. A. Pinar, C. Seshadhri, and T. G. Kolda. The similarity between stochastic kronecker and chung-lu graph models. *SDM*, 2011.

97. Z. Pizlo and Z. Li. Pyramid algorithms as models of human cognition. In *Electronic Imaging 2003*, pages 1–12. International Society for Optics and Photonics, 2003.

98. N. Pržulj. Biological network comparison using graphlet degree distribution. *Bioinformatics*, 23(2):e177–e183, 2007.

99. N. Robertson and P. D. Seymour. Graph minors. ii. algorithmic aspects of tree-width. *Journal of algorithms*, 7(3):309–322, 1986.

100. G. Robins, P. Pattison, Y. Kalish, and D. Lusher. An introduction to exponential random graph (p*) models for social networks. *Social networks*, 29(2): 173–191, 2007.

101. D. J. Rose, R. E. Tarjan, and G. S. Lueker. Algorithmic aspects of vertex elimination on graphs. *SIAM Journal on computing*, 5(2):266–283, 1976.

102. E. Schmidt and J. Cohen. *The new digital age: Transforming nations, businesses, and our lives*. Vintage, 2014.

103. R. Sedgewick and P. Flajolet. *Introduction to the Analysis of Algorithms*. Addison-Wesley Professional, 2nd edition, 2013. URL `http://aofa.cs.princeton.edu`.

104. C. Seshadhri, T. G. Kolda, and A. Pinar. Community structure and scale-free collections of Erdős-Rényi graphs. *Physical Review E*, 85(5):056109, 2012.

105. K. Shoikhet and D. Geiger. A practical algorithm for finding optimal triangulations. In *AAAI/IAAI*, pages 185–190, 1997.

106. S. L. Simpson, S. Hayasaka, and P. J. Laurienti. Exponential random graph modeling for complex brain networks. *PLoS ONE*, 6(5):e20039, 05 2011. doi: 10.1371/journal.pone.0020039. URL `http://dx.doi.org/10.1371%2Fjournal.pone.0020039`.

107. A. Sinha, M. Sarkar, A. Mukherjee, and B. Krishnamurthy. Introspection: Accelerating neural network training by learning weight evolution. *arXiv preprint arXiv:1704.04959*, 2017.

108. A. Stolcke. An efficient probabilistic context-free parsing algorithm that computes prefix probabilities. *Computational linguistics*, 21(2):165–201, 1995.

109. S. H. Strogatz. Exploring complex networks. *Nature*, 410(6825):268–276, 2001.

110. Z. Sun, H. Wang, H. Wang, B. Shao, and J. Li. Efficient subgraph matching on billion node graphs. *PVLDB*, 5(9):788–799, 2012.

111. R. E. Tarjan and M. Yannakakis. Addendum: Simple linear-time algorithms to test chordality of graphs, test acyclicity of hypergraphs, and selectively reduce acyclic hypergraphs. *SIAM Journal on Computing*, 14(1):254–255, 1985. doi: 10.1137/0214020. URL `http://dx.doi.org/10.1137/0214020`.

112. R. E. Tarjan and M. Yannakakis. Simple linear-time algorithms to test chordality of graphs, test acyclicity of hypergraphs, and selectively reduce acyclic hypergraphs. *SIAM Journal on computing*, 13(3):566–579, 1984.

113. R. E. Tarjan and M. Yannakakis. Addendum: Simple linear-time algorithms to test chordality of graphs, test acyclicity of hypergraphs, and selectively reduce acyclic hypergraphs. *SIAM Journal on Computing*, 14(1):254, 1985.

114. M. Thoma, H. Cheng, A. Gretton, J. Han, H.-P. Kriegel, A. Smola, L. Song, P. S. Yu, X. Yan, and K. M. Borgwardt. Discriminative frequent subgraph mining with optimality guarantees. *Statistical Analysis and Data Mining*, 3(5): 302–318, 2010.

115. J. Ugander, L. Backstrom, and J. Kleinberg. Subgraph frequencies: Mapping the empirical and extremal geography of large graph collections. In *WWW*, pages 1307–1318, 2013.

116. J. Ugander, L. Backstrom, and J. Kleinberg. Subgraph frequencies: Mapping the empirical and extremal geography of large graph collections. In *Proceedings of the 22nd international conference on World Wide Web*, pages 1307–1318. ACM, 2013.

117. S. Wasserman and K. Faust. *Social network analysis: Methods and applications*, volume 8. Cambridge university press, 1994.

118. D. J. Watts and S. H. Strogatz. Collective dynamics of small-world networks. *Nature*, 393(6684):440–442, 1998. ISSN 0028-0836. doi: 10.1038/30918. URL `http://dx.doi.org/10.1038/30918`.

119. D. J. Watts and S. H. Strogatz. Collective dynamics of ''small-world'networks. *nature*, 393(6684):440, 1998.

120. X. Yan and J. Han. gspan: Graph-based substructure pattern mining. In *ICDM*, pages 721–724. IEEE, 2002.

121. M. Yannakakis. Computing the minimum fill-in is np-complete. *SIAM Journal on Algebraic Discrete Methods*, 2(1):77–79, 1981.

122. Ö. N. Yaveroğlu, T. Milenković, and N. Pržulj. Proper evaluation of alignment-free network comparison methods. *Bioinformatics*, 31(16):2697–2704, 2015.

123. B. Zoph and Q. V. Le. Neural architecture search with reinforcement learning. *arXiv preprint arXiv:1611.01578*, 2016.

This document was prepared & typeset with pdfLaTeX*, and formatted with* NDdiss2$_\varepsilon$ *classfile (v3.2013[2013/04/16]) provided by Sameer Vijay and updated by Megan Patnott.*