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

November 2017

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 [21, 112]. 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 [35, 72]. 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 [97].

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 [45, 73]. 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.

In this thesis, I investigate a relationship between graph theory and formal language theory that allows for a Hyperedge 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 mean-

TABLE 1.1

Thesis Overview

Chapter	Title	Research Problem
3	HRG model inference and stochastic graph generation	Can an HRG capture graph features to grow graphs with properties matching the input or reference graph?
4	HRG using a new fixed-size graph generation approach	Can we grow graphs given a desired number of vertices?
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?

ing representations (AMR).

1.1 Contributions and Collaborations

This thesis is organized around the following themes: (I) HRG stochastic graph generation, (II) HRG for fixed-size graph generation, (III) measures of model resilience, and (IV) model bias resulting from tree decomposition algorithms. A summary of the key challenges for each theme is detailed in Table 1.1.

1.2 Impact

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. By proposing the use of the HRG graph model to engineer deep neural network architectures, I won a Department of Energy Student Research award in 2017.

1.2.1 Collaborations

1.2.1.1 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 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 [86]. Although not fully explored in this thesis, this illuminates another avenue for studying the application of HRG graph model.

1.2.1.2 Latent Variable Probabilistic Graph Grammars

This work address one of the limitations of the HRG model. The graph grammar encodes only sufficient information to ensure that the result is well-formed. In the tree decomposition step, we transform the the input graph into tree. The rules we extract come from the top, middle and bottom of the tree graph, however in the graph generation process we draw rules based on frequency. 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 is currently under review and is not delineated in this thesis, yet it illuminates yet another area of research.

CHAPTER 2

BACKGROUND

2.1 Graphs

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.

2.1.1 Hypergraphs

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.

2.1.2 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, 30]. 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 [77] to the more recent work describing scale-free power law distribution as a common property observed in large networks [9]. For more on the historical unfolding of social network analysis metrics see the survey's by Wasserman and Faust [110], Freeman [31], and Newman [84].

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 [9].

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 [69] 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 [91, 108]. 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 [113]. 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 [28, 83] 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 [27, 34]. Two decades ago Watts, Strogatz, Barabasi, and Albert introduced models of *Small World* networks [112] 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 [29]. 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. The famous PageRank algorithm, and the industry giant Google, is one such example of transformative real world implications.

Another class of network generators are Kronecker graphs. Using the Kronecker matrix operation, network scientists are able to learn a network model that can generate large, scale free graphs that "look like" their empirical counterparts [66, 73].

Exponential random graphs models (ERGMs) 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 [14, 38, 101]. 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 [32, 37].

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 [23]. 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.

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 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 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 [54]. Kukluk, Holder and Cook were able to define a grammar from frequent subgraphs [20, 44, 60–62], but their methods have a coarse resolution because *frequent* subgraphs only account for a small portion of the overall graph topology.

CHAPTER 3

GROWING GRAPHS

3.1 Introduction

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 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 easily understood, but whose global view is highly complex.

These networks exhibit a long and varied list of global properties, including heavy-tailed degree distributions [104] and interesting growth characteristics [69, 70], among others. Recent work has found that these global properties are merely products of a graph's local properties, in particular, graphlet distributions [108]. These small, local substructures often reveal the degree distributions, diameter and other global properties of a graph [91, 108], and have been shown to be a more complete way to measure the similarity between two or more graphs [113]. Our overall goal, and the goal of structural inference algorithms in general, is to learn the local structures that, in aggregate, help describe the observed interactions and generalize to explain further phenomena.

For example, physicists and chemists have found that many chemical interactions are the result of underlying structural properties of the individual elements. Similarly, biologists have agreed that simple tree structures are useful when organizing the evolutionary history of life, and sociologists find that clique-formation, *e.g.*, triadic closure, underlies community development [8, 15, 96]. In other instances, the structural organization of the entities may resemble a ring, a clique, a star, or any number of complex configurations.

In this work, we describe a general framework that can discover, from any large network, simple structural forms in order to make predictions about the topological properties of a network. In addition, this framework is able to extract mechanisms of network generation from small samples of the graph in order to generate networks that satisfy these properties. Our major insight is that a network's *clique tree* encodes simple information about the structure of the network. We use the closely-related formalism of *hyperedge replacement grammars* (HRGs) as a way to describe the organization of real world networks.

My Contribution

- Combine prior theoretical work on clique trees, tree decomposition and treewidth to automatically learn an HRG for real world graphs.
- Evaluate existing graph generators, like exponential random graphs, small world graphs, Kronecker graphs, and so on, learn parameters from some input graph to generate new graphs stochastically.
- This model is also able to stochastically generate different-sized graphs that share similar properties to the original graph.
- Contributed to the code repository

Unlike previous models that manually define the space of possible structures [53] or define the grammar by extracting frequent subgraphs [61, 63], 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.

After reviewing some of the theoretical foundations of clique trees and HRGs, we show how to extract an HRG from a graph and use it to reconstruct the original graph. We then show how to use the extracted grammar to stochastically generate generalizations of the original graph. Finally, we present experimental results that compare the stochastically generated graphs with the original graphs. We show that these generated graphs exhibit a wide range of properties that are very similar to the properties of the original graphs, and significantly outperform existing graph models at generating subgraph distributions similar to those found in the original graph.

3.2 Preliminaries

The new work in this paper begins where previous work [4, 16, 65, 92] 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 paper 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 clique trees and their relationship to hyperedge replacement grammars in order to support the claims made in sections 3 and 4.

3.2.1 Tree Decomposition

All graphs can be decomposed (though not uniquely) into a *clique tree*, also known as a tree decomposition, junction tree, join tree, intersection tree, or cluster graph. Within the pattern recognition community, clique trees are best known for their role in exact inference in probabilistic graphical models, so we introduce the preliminary work from a graphical modeling perspective; for an expanded introduction, we refer the reader to Chapters 9 and 10 of Koller and Friedman's textbook [57].

Definition 3.2.1. *A* clique tree *of a graph $H = (V, E)$ is a tree CT, 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 $\eta \in CT$ 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 $\eta \in CT$ 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 $\{\eta \in CT \mid v \in V_\eta\}$ is connected.*

Definition 3.2.2. *The* width *of a clique tree is* $\max(|V_\eta - 1|)$*, and the* treewidth *of a graph H is the minimal width of any clique tree of H.*

Unfortunately, finding the optimal elimination ordering and corresponding minimal-width clique tree is NP-Complete [7]. Fortunately, many reasonable approximations exist for general graphs: in this paper, we employ the commonly used maximum cardinality search (MCS) heuristic introduced by Tarjan and Yannikakis [107] to compute a clique tree with a reasonably-low, but not necessarily minimal, width.

Simply put, a clique tree 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 clique tree measures how tree-like the graph is. The reason for the interest in finding the clique tree 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 clique tree (showing V_η for each node η). We label nodes with lowercase Latin letters. We will refer back to this graph and clique tree as a running example throughout this paper.

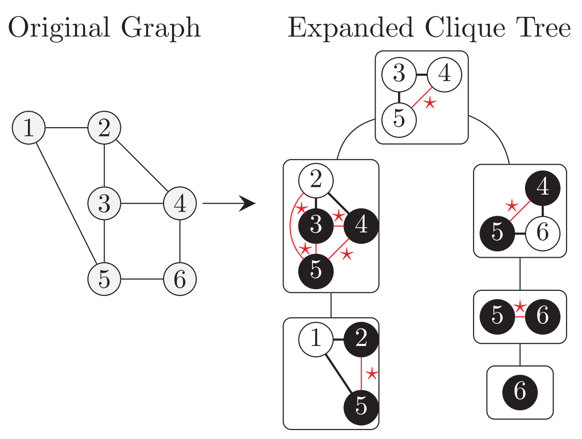


Figure 3.1. A graph and one possible minimal-width clique tree for it. Ghosted edges are not part of E_η; they are shown only for explanatory purposes.

3.2.2 Hyperedge Replacement Grammar

The key insight for this task is that a network's clique tree 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.* [25].

Definition 3.2.3. *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 3.2.4. *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.

3.2.3 Hyperedge Replacement Grammar

The key insight for this task is that a network's clique tree encodes robust and precise information about the network. An HRG, which is 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 represent 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.* [24].

Definition 3.2.5. *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 $A \rightarrow R$, 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*. 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 the 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.*

Definition 3.2.6. *Let G be an HRG and $P = (A \rightarrow R)$ 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 is repeated until there are no more nonterminal hyperedges. These definitions will be clearly illustrated in the following sections.

3.2.4 The Missing Link

Clique trees and hyperedge replacement graph grammars have been studied for 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 hand drawn to generate some specific graph or class of graphs. Very recently, Kemp and Tenenbaum developed an inference algorithm that learned probabilities from real world graphs, but still relied on a handful of rather basic hand-drawn production rules (of a related formalism called vertex replacement grammar) to which the learned probabilities were assigned [53].

The main contribution of this paper is to combine prior theoretical work on clique trees, tree decomposition and treewidth to automatically learn an HRG for real world graphs. Existing graph generators, like exponential random graphs, small world graphs, Kronecker graphs, and so on, learn parameters from some input graph to generate new graphs stochastically. Unlike these previous approaches, our model has the ability to reproduce the exact same graph topology where the new graph is guaranteed to be isomorphic to the original graph. Our model is also able to stochastically generate different-sized graphs that share similar properties to the original graph.

3.3 Learning Graph Grammars

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

3.3.1 Clique Trees and HRGs

Let η be an interior node of the clique tree T, let η' be its parent, and let $\eta_1, \ldots, \eta_m$ be its children. Node η corresponds to an HRG production rule $A \rightarrow R$ 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 3.2 shows an example of the creation of an HRG rule. In this example, we focus on the middle clique-tree node $V_\eta = \{\text{d}, \text{e}, \text{f}\}$, 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) $V_\eta = \{\text{d}, \text{e}, \text{f}\}$. 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 $E_\eta = \{(\text{d}, \text{f}), (\text{e}, \text{f})\}$. 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.

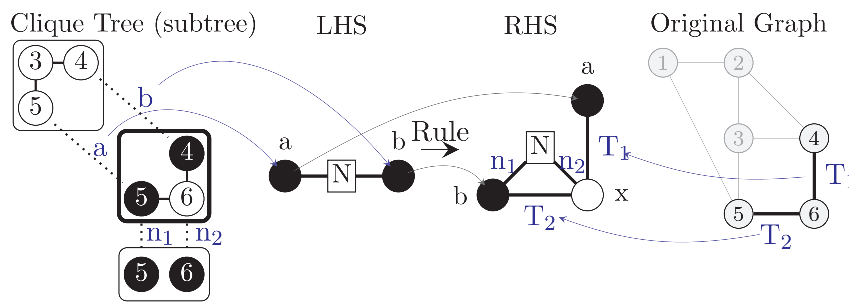


Figure 3.2. Example of hyperedge replacement grammar rule creation from an interior vertex of the clique tree. Note that lowercase letters inside vertices are for explanatory purposes only; only the numeric labels outside external vertices are actually part of the rule.

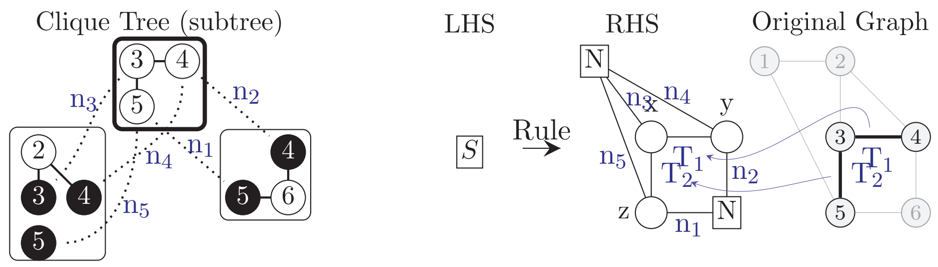


Figure 3.3. Example of hyperedge replacement grammar rule creation from the root node of the clique tree.

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 3.3.

The RHS is computed in the same way as the interior node case. For the example in Fig. 3.3, the RHS has vertices that are copies of c, d, and e. In addition, the 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.

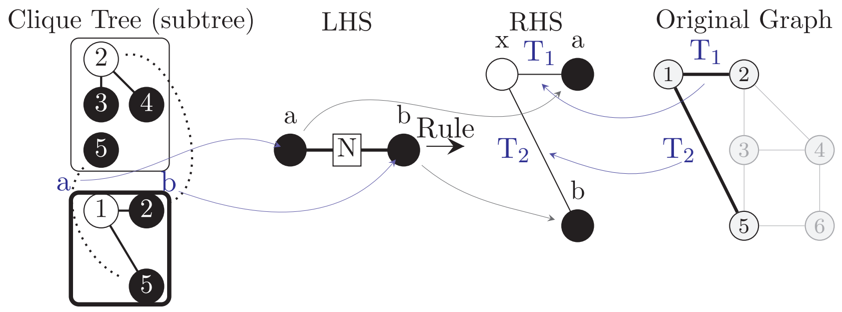


Figure 3.4. Example of hyperedge replacement grammar rule creation from a leaf vertex of the clique tree.

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. 3.4 (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.

3.3.2 Top-Down HRG Rule Induction

We induce production rules from the clique tree 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 clique tree.

Larger graphs will typically produce larger clique trees, especially sparse graphs because they are more likely to have a larger number of small maximal cliques. These larger clique trees will produce a large number of HRG rules, one for each clique in the clique tree. 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 are often repeated many times.

Figure 3.5 shows the 6 rules that are induced from the clique trees illustrated in Fig. 3.1 and used in the running example throughout this section.

3.3.3 Complexity Analysis

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

Unfortunately, finding a clique tree 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 clique tree [105] 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 clique tree as well as the number of vertices in each clique tree node. From Defn. 3.2.1 we have that the number of nodes in the clique tree is m. When minimal, the number of vertices in an the largest clique tree node $\max(|\eta_i|)$

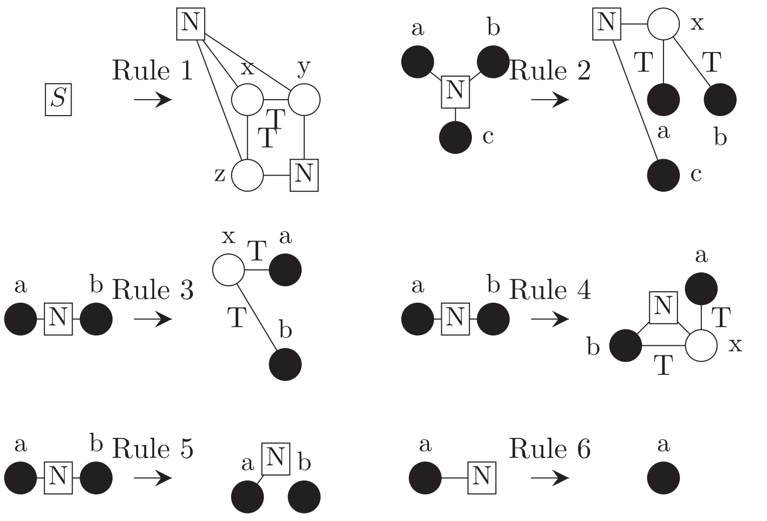


Figure 3.5. Complete set of production rules extracted from the example clique tree. Note that lowercase letters inside vertices are for explanatory purposes only; only the numeric labels outside external vertices are actually part of the rule.

(minus 1) is defined as the treewidth tw, however, clique trees generated by MCS have $\max(|\eta_i|)$ bounded by the maximum degree of H, denoted as Δ [33]. Therefore, given an elimination ordering from MCS, the computational complexity of the extraction process is in $O(m \cdot \Delta)$.

3.4 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.

3.4.1 Exact Generation

Exact generation operates by reversing the HRG extraction process. In order to do this, we must store the HRG rules $\mathcal{P}$ as well as the clique tree T (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 clique 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 clique tree T of some given hypergraph H, and Defn. 3.2.6 defines the application of a production rule $(A \to R)$ that transforms some hypergraph H' into a new hypergraph H^*. By applying the rules created from the clique tree 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.

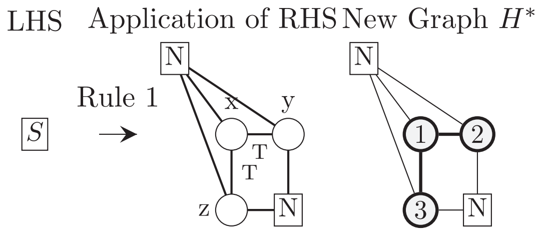


Figure 3.6. Application of Rule 1 to replace the starting nonterminal S with the RHS to create a new graph H^*.

Using the running example from the previous section, the application of Rule 1 illustrated in Fig. 3.6 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 clique tree.

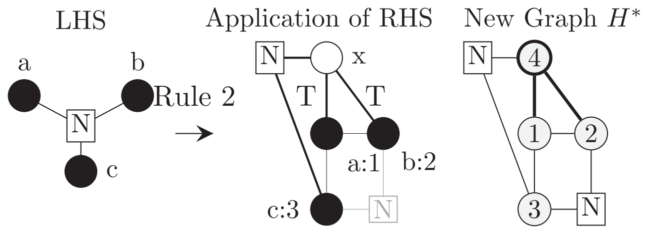


Figure 3.7. Application of Rule 2 to replace a size-3 nonterminal in H' with the RHS to create a new graph H^*.

We proceed down the left hand side of the clique tree, applying Rule 2 to H' as shown in Fig. 3.7. 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 clique tree.

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 3.8. 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 clique tree illustrated in Fig. 3.1

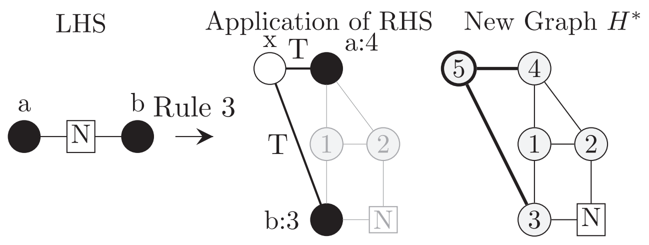


Figure 3.8. Application of Rule 3 to replace a size-2 nonterminal in H' with the RHS to create a new graph H^*.

has three further applications of the rules in $\mathcal{P}$. As illustrated in Fig. 3.9, Rule 4 adds the final vertex, two terminal edges and one nonterminal hyperedge to H^*. Rule 5 and Rule 6 do not create any more terminal edges or internal vertices in H^*, but are still processed because of the way the clique tree is constructed.

After all 6 rules are applied in order, we are guaranteed that H and H^* are isomorphic.

3.4.2 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 clique tree* 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 7 in Fig. 3.5 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

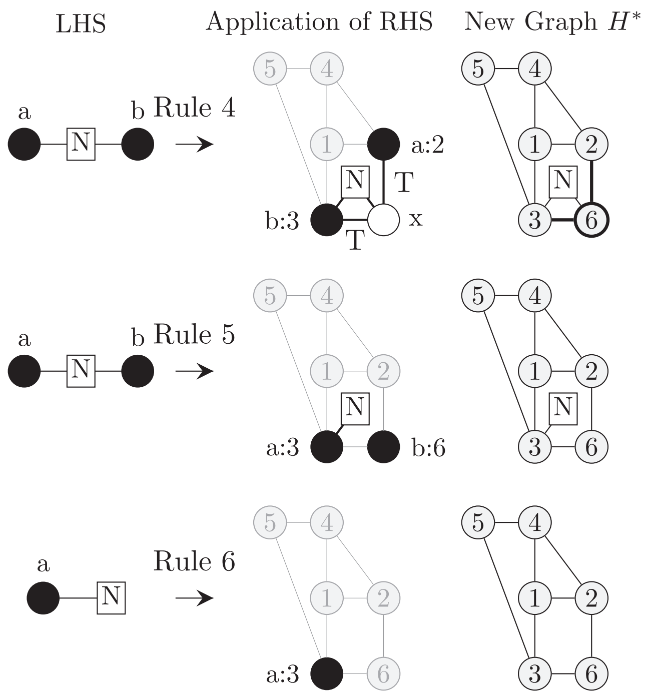


Figure 3.9. Application of Rules 4, 5 and 6 to create an H^* that is isomorphic to the original graph H.

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/`.

3.5 Experiments

HRGs contain rules that succinctly represent the global and local structure of the original graph. In this section, we compare our approach against some of the state-of-the-art graph generators. We consider the properties that underlie a number of real-world networks and compare the distribution of graphs generated using generators for Kronecker Graphs, the Exponential Random Graph, Chung-Lu Graphs, and the graphs produced by the stochastic hyperedge replacement graph grammar.

In a manner similar to 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 stochastically 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. Except in the case of exact HRG generation described above, the stochastically generated graphs are likely not isomorphic to the original graph. We can, however, still judge how closely the stochastically generated graph resembles the original graph by comparing several of their properties.

3.5.1 Real World Datasets

In order 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.

The four real-world networks considered in this paper are described in Table 4.1. The networks vary in their number of vertices and edges as indicated, but also differ in clustering coefficient, diameter, degree distribution and many other graph properties. Specifically, 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 represents traffic flows through Internet peers. And finally, DBLP is the co-authorship graph from the DBLP dataset. Datasets are downloaded from the SNAP and KONECT dataset repositories.

3.5.2 Methodology

We compare several different graph properties from the 4 classes of graph generators (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 due to limited space and 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 in order to create an adjacency matrix of the approximate graph. In our case, we use KronFit [73] 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 [17].

Exponential Random Graph Models (ERGMs) are a class of probabilistic models used to directly describe several structural features of a graph [93]. We used default parameters in R's ERGM package [46] 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 and Enron 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 [67]. 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 $G = \bigcup_i G_i$. For evaluation purposes, we generate 20 approximate graphs for the HRG, Chung-Lu, and Kronecker models and plot the mean values in the results section. We did compute the confidence internals for each of the models, but omitted them from the graphs for clarity. In general, the confidence intervals were very small for HRG, Kronecker and CL (indicating good consistency), but very big in the few ERGM graphs that we were able to generate because of the model degeneracy problem we encountered.

3.5.3 Graph Generation Results

Here we compare and contrast the results of approximate graphs generated from HRG, Kronecker product, and Chung-Lu. We use the graph properties described in Chapter 2 to compare the similarity between the real networks and their approximate counterparts.

Figure 4.8 in chapter 4 shows the results of the degree distribution property on the four real world graphs ($frequency$ or *count* as a function of degree k). Recall 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. Figure 4.9 in chapter 4 shows the eigenvector scores for each node ranked highest to lowest in each of the four real world graphs. Because the x-axis represents individual nodes, Fig. 4.9 also shows the size difference among the generated graphs. HRG performs consistently well across all four types of 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. Figure 4.10 in chapter 4 demonstrates that HRG graphs produce hop plots that are remarkably similar to the original graph. Chung-Lu performs rather well in most cases; Kronecker has poor performance on Arxiv and DBLP graphs, but still shows the correct hop plot shape.

Graphlet Correlation Coefficient

We computed the GCD between the original graph and each generated graph. Figure 3.10 shows the GCD results. Although they are difficult to see due to their small size, Fig. 3.10 includes error bars for the 95% confidence interval. The results here are clear: HRG significantly outperforms the Chung-Lu and Kronecker models.

3.5.4 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

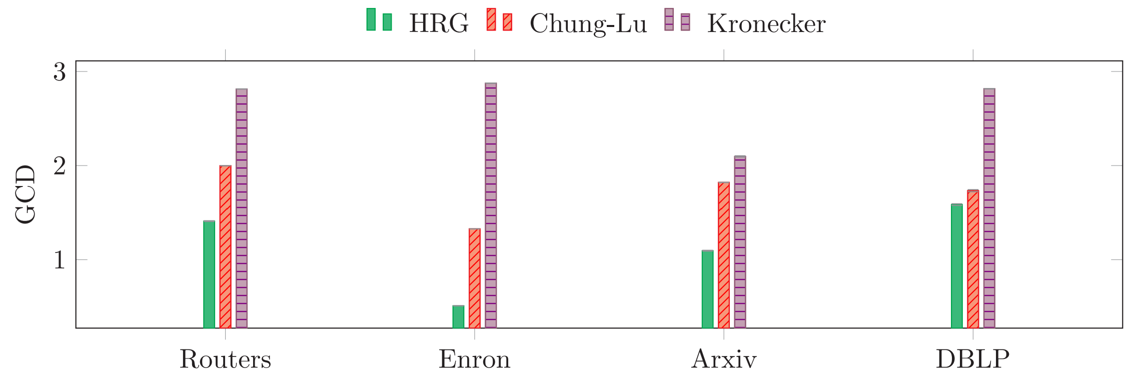


Figure 3.10. Graphlet Correlation Distance. A measure of the distance between the graphlet counts of both graphs, but also represents a canonical measure of graph similarity.

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 hypothetical Karate Franchise might look like?

Using two smaller graphs, Zachary's Karate Club (34 nodes, 78 edges) and the protein-protein interaction network of *S. cerevisiae* yeast (1,870 nodes, 2,240 edges), 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.

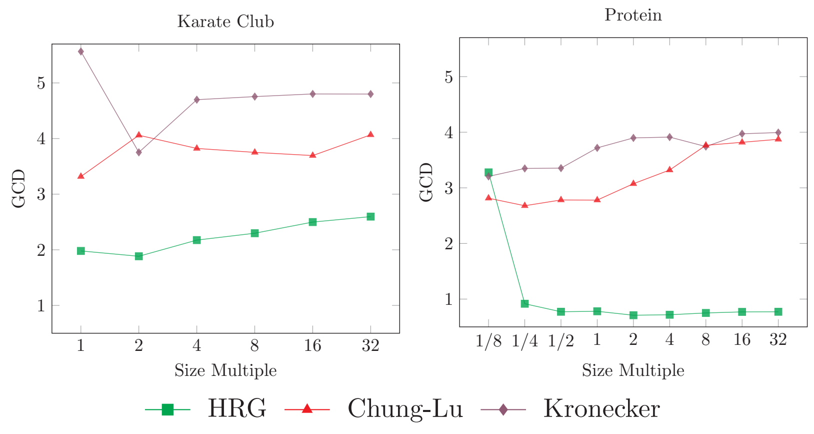


Figure 3.11. GCD of graphs extrapolated in multiples up to 32x from two small graphs. HRG outperforms Chung-Lu and Kronecker models when generating larger graphs. Lower is better.

Figure 3.11 in chapter 4 shows the mean GCD score and 95% confidence intervals for each graph model. Not only does HRG generate good results at $n^* = 1$x, the GCD scores remain mostly level as n^* grows.

3.5.5 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 clique tree will change causing the grammar to change in size and complexity. A large clique tree ought to create more rules and a more complex grammar, resulting in a larger model size and better performance; while a small clique tree 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 4.16 in chapter 4 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 clique tree 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 [43], *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.

3.5.5.1 Model size and Performance

One of the disadvantages of the HRG model, as indicated in Fig. 4.16, 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 4.17 in chapter 4 illustrates the relationship between model size and the GCD. We use the Router and DBLP graphs to shows the largest and smallest of our dataset; other graphs show similar results, but their plots are omitted due to of 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 generally 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 ($r = -0.08$, $p = 0.62$)

3.5.5.2 Runtime Analysis Revisited

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

We previously identified the runtime complexity of the rule extraction process to be $O(m \cdot \Delta)$. However, this did not include 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 and is linear in the number of edges in the output graph.

All experiments were performed on a modern consumer-grade laptop in an unoptimized, unthreaded python implementation. We recorded the extraction time while 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. 4.18 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.

3.5.6 Graph Generation Infinity Mirror

Lastly, we characterize the robustness of graph generators by introducing a new kind of test we call the *infinity mirror* [2]. One of the motivating questions behind this idea was to see if HRG holds sufficient information to be used as a reference itself. In this test, we repeatedly learn a model from a graph generated by the an earlier version of the same model. For HRG, this 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 thereby learning a new model from the generated graph recursively. We repeat this process ten times, and compare the output of the tenth recurrence with the original graph using GCD.

We expect to see that all models degenerate over 10 recurrences because graph generators, like all machine learning models, are lossy compressors of information. The question is, how quickly do the models degenerate and how bad do the graphs become?

Figure 4.14 in chapter 4shows the GCD scores for the HRG, Chung-Lu and Kronecker models at each recurrence. 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 honing in on rules that generate the necessary graph patterns. Yet again, further work is needed to study this important phenomenon.

3.6 Conclusions and Future Work

In this paper we have shown how to use clique trees (also known as junction trees, tree decomposition, intersection trees) constructed from a simple, general graph 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 clique tree is traversed during reconstruction. More practically, we show that a stochastic application of the grammar rules creates new graphs that have very similar properties to the original graph. The results of graphlet correlation distance experiments, extrapolation and the infinity mirror are particularly exciting because our results show a stark improvement in performance over existing graph generators.

In the future, we plan to investigate differences between the grammars extracted from different types of graphs; we are also interested in exploring the implications of finding two graphs which have a large overlap in their extracted grammars. Among the many areas for future work that this study opens, we are particularly interested in learning a grammar from the actual growth of some dynamic or evolving graph. Within the computational theory community there has been a renewed interest in quickly finding clique trees of large real world graphs that are closer to optimal. Because of the close relationship of HRG and clique trees shown in this paper, any advancement in clique tree algorithms could directly improve the speed and accuracy of graph generation.

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.

We encourage the community to explore further work bringing HRGs to attributed graphs, heterogeneous graphs and developing practical applications of the extracted rules. Given the current limitation related to the growth in the number of extracted rules as well as the encouraging results from small models, we are also looking for sparsification techniques that might limit the model's size while still maintaining performance.

CHAPTER 4

PROBABILISTIC GRAPH GENERATION

4.1 Introduction

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 [59]. For example, biologists have agreed that tree structures are useful when organizing the evolutionary history of life [22, 51], and sociologists find that triadic closure underlies community development [26, 39]. 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 [54, 75] or frequently reoccurring patterns [44, 50, 60, 62], 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 [104], and interesting community structures [99] to name a few. Recent work has found that these global properties are products of a graph's local properties [91, 109].

Stochastic graph generation exposed a computational vulnerability in the initial design. A new approach to how graph generation addresses the issue using concepts in probabilistic context-free grammars. We validate, and test and our experiments to show that we can consistently yield graphs of fixed-size. 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 *clique tree* encodes robust and precise information about the network. A *hyperedge replacement grammar* (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.

The HRG framework is divided into two steps: 1) graph model learning and 2) graph generation. After reviewing some of the theoretical foundations of clique trees 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.

Finally, we present experimental results that compare the generated graphs with the original graphs. We show that these generated structures exhibit a broad range of properties that are very similar to the properties of the reference (original) networks and outperform existing graph models across a variety of graph comparison metrics.

Preliminary concepts providing an auxiliary set of definitions is mentioned the previous chapter under section 3.2. In that section we define some concepts in detail.

4.2 Learning HRGs

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

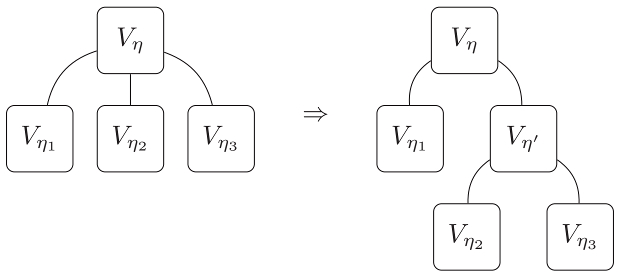


Figure 4.1. Binarization of a bag in a clique tree.

4.2.1 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 clique tree node η yields an HRG rule, and the number of children of η determines the number of nonterminals on the right-hand side of the resulting rule. Thus, it suffices for the clique tree to have a branching factor of at most two. Although the branching factor of a clique tree 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 clique tree 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. 4.1 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 clique tree whose nodes are at most binary-branching and still has the vertex cover, edge cover, and running intersection properties for H.

4.2.2 Clique Tree Pruning

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

The bottom-right clique tree node in Fig. 3.1 is such an example because f is an

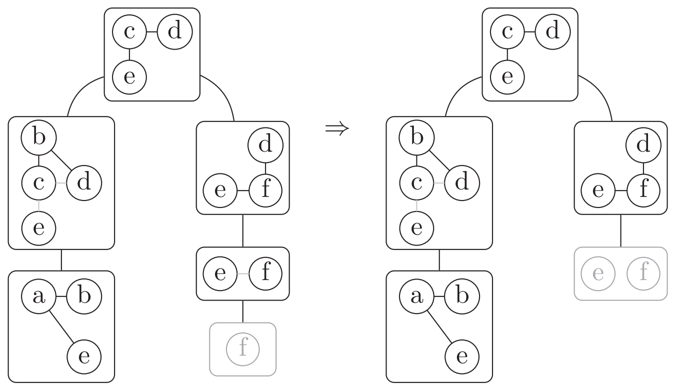


Figure 4.2. Pruning a clique tree to remove leaf nodes without internal vertices. Ghosted clique tree nodes show nodes that are pruned.

external vertex; that is, f exists in its parent. Because no internal vertices exist in this leaf node, it is removed from the clique tree. The clique tree node with vertices e and f is now a leaf, as illustrated in the left side of Fig. 4.2. Vertices e and f in the new leaf node are still both external vertices, so this clique tree node must also be removed creating a final clique tree illustrated in the right side of Fig. 4.2.

4.2.3 Clique Trees and HRGs

Here we show how to extract an HRG from the clique tree. Let η be an interior node of the clique tree CT, 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 4.3 shows an example of the creation of an HRG rule. In this example, we focus on the middle clique-tree node $V_\eta = \{\mathrm{b}, \mathrm{c}, \mathrm{d}, \mathrm{e}\}$, outlined in bold.

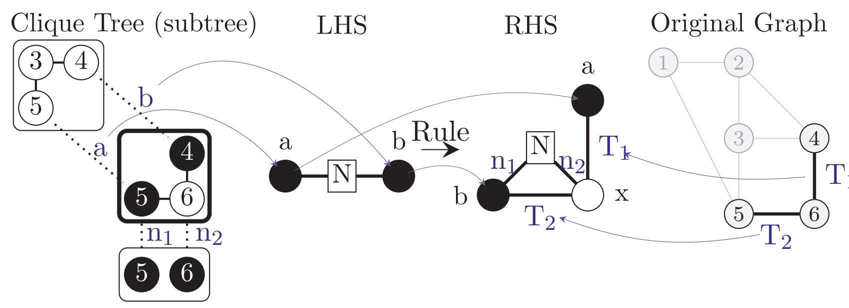


Figure 4.3. Example of hyperedge replacement grammar rule creation from an interior vertex of the clique tree. Note that lowercase letters inside vertices are for explanatory purposes only; only the numeric labels outside external vertices are actually part of the rule.

We choose nonterminal symbol N for the LHS, which must have rank 3 because η has 3 vertices in common with its parent. The RHS is a graph whose vertices are (copies of) $V_\eta = \{\mathrm{b}, \mathrm{c}, \mathrm{d}, \mathrm{e}\}$. Vertices c, d and e are marked external (and numbered 1, 2, and 3, arbitrarily) because they also appear in the parent node. The terminal edges are $E_\eta = \{(\mathrm{b}, \mathrm{c}), (\mathrm{b}, \mathrm{d})\}$. There is only one child of η, and the nodes they have in common are b and e, so there is one nonterminal hyperedge connecting b and e.

Next we deal with the special cases of the root and leaves.

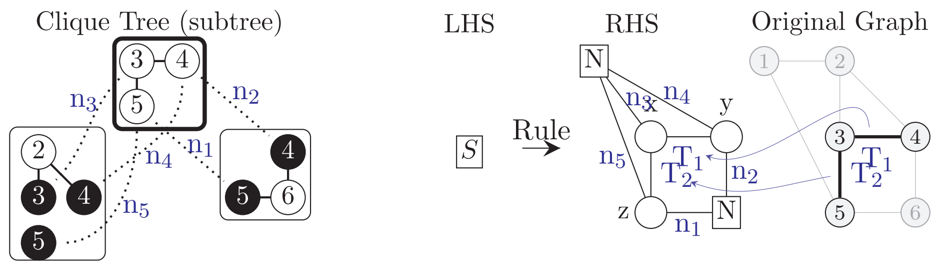


Figure 4.4. Example of hyperedge replacement grammar rule creation from the root node of the clique tree.

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 symbol S as the LHS, as shown in Fig. 4.4.

The RHS is computed in the same way as the interior node case. For the example in Fig. 4.4, the RHS has vertices that are copies of c, d, and e. In addition, the 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.

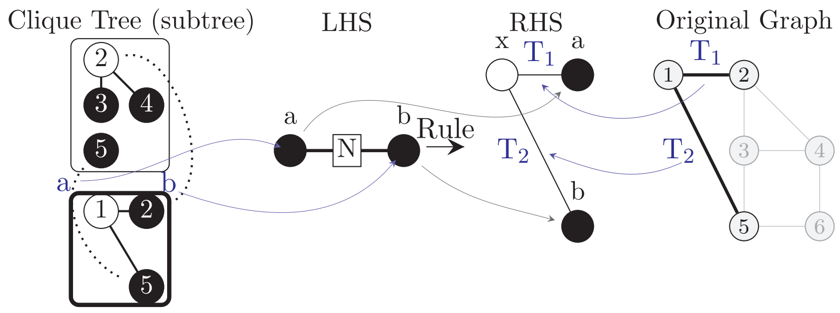


Figure 4.5. Example of hyperedge replacement grammar rule creation from a leaf vertex of the clique tree.

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. 4.5. 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.

4.2.4 Top-Down HRG Rule Induction

We induce production rules from the clique tree 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 clique tree.

Larger graphs will typically produce larger clique trees, especially sparse graphs because they are more likely to have a greater number of small maximal cliques. These larger clique trees will produce a large number of HRG rules, one for each clique in the clique tree. 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 4.6 shows the 4 rules that are induced from the clique tree illustrated in Fig. 3.1 and used in the running example throughout this section.

4.3 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

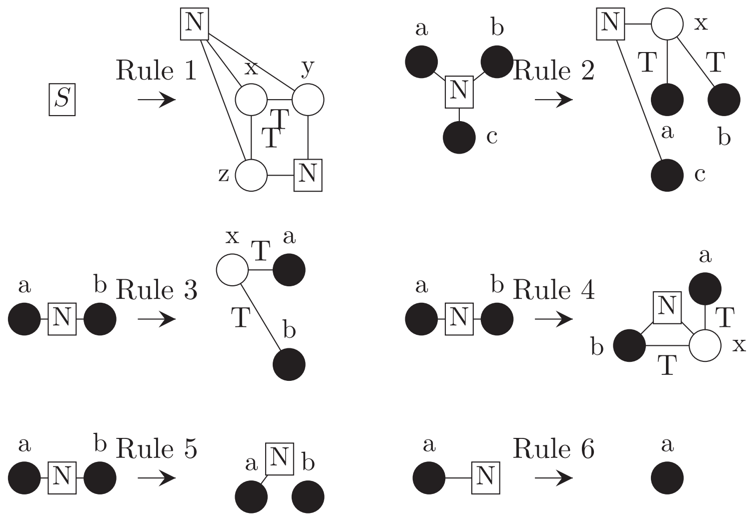


Figure 4.6. Complete set of production rules extracted from the example clique tree. Note that lowercase letters inside vertices are for explanatory purposes only; only the numeric labels outside external vertices are part of the rule.

be similar to or have features that are akin or analogous to the original graph H. We present two generation algorithms. The first generates random graphs with similar characteristics to the original graph. The second is like it but generates random graphs that have a specified number of nodes.

4.3.1 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. Here we describe a simple stochastic hypergraph generator that applies rules from the extracted HRG to efficiently create such graphs.

In larger HRGs we usually find many $A \rightarrow R$ production rules that are identical. We chose to consider rules that are identical *modulo* a permutation of their external vertices to be equivalent as well. 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 extra Rule #5 in Fig. 4.6 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 (PHRG), 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 ($A \rightarrow R$) associated with LHS A; (3) Randomly choose one of these rules with probability proportional to its count; (4) Choose an ordering of its external vertices with uniform probability; (5) Replace A in H' with R to create H^*; (5) Replace H' with H^* and repeat until there are no more nonterminal edges.

4.3.2 Fixed-Size Generation

A problem we find with the stochastic generation procedure is that, although the generated 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 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 [81]. 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 [103]. We precompute the strongly connected components of U, for example, using Tarjan's algorithm, and for each com-

```
compute digraph $U$ of unary size-zero rules;
topologically sort $U$;
assert ($U$ is acyclic);
for $\ell \leftarrow 1, \ldots, n$ do
    for $X \in N$ in reverse topological order do
        for rules $X \xrightarrow{p} R$ do
            $\ell' = \ell - \text{size}(R)$;
            if $R$ has no nonterminals and $\ell' = 0$ then
                $\alpha[X, \ell] \mathrel{+}= p$;
            end
            else if $R$ has nonterminal $Y$ then
                $\alpha[X, \ell] \mathrel{+}= p \times \alpha[Y, \ell']$;
            end
            else if $R$ has nonterminals $Y$ and $Z$ then
                for $k \leftarrow 1, \ldots, \ell' - 1$ do
                    $\alpha[X, \ell] \mathrel{+}= p \times \alpha[Y, k] \times \alpha[Z, \ell' - k]$;
                end
            end
        end
    end
end
```
Algorithm 1: Compute inside probabilities (no cycles of size-zero unary rules)

ponent C, we form the weighted adjacency matrix of C; call this U_C. The matrix $U_C^* = \sum_{i=0}^{\infty} U_C^i = (I - U_C)^{-1}$ 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 clique tree pruning (Sec. 4.2.2), every leaf of the clique tree has at least one internal vertex. In terms of HRG rules, this means that if R has no nonterminals, then size$(R) > 0$. Therefore, we have

```
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 $\ell \leftarrow 1, \ldots, n$ do
    for scc's $C$ in reverse topological order do
        for $X \in C$ do
            for rules $X \xrightarrow{p} R$ do
                $\ell' = \ell - \text{size}(R)$;
                if $R$ has no nonterminals and $\ell' = 0$ then
                    $\alpha[X, \ell] \mathrel{+}= p$;
                end
                else if $R$ has nonterminal $Y$ and $\ell' < \ell$ then
                    $\alpha[X, \ell] \mathrel{+}= p \times \alpha[Y, \ell']$;
                end
                else if $R$ has nonterminals $Y$ and $Z$ then
                    for $k \leftarrow 1, \ldots, \ell' - 1$ do
                        $\alpha[X, \ell] \mathrel{+}= p \times \alpha[Y, k] \times \alpha[Z, \ell' - k]$;
                    end
                end
            end
        end
        for $X \in C$ do
            $\alpha[X, \ell] = \sum_{Y \in C} [U^*_C]_{XY} \times \alpha[Y, \ell]$;
        end
    end
end
```

Algorithm 2: Compute inside probabilities (general)

$\alpha[X, 0] = 0$ for all X, and no problem arises.

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.

```
$H \leftarrow S^{(n)}$;
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
            weight$[R] = p$;
        end
        else if $R$ has nonterminal $Y$ then
            $R' = R\{Y \mapsto Y^{(\ell')}\}$;
            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
                $R' = R\{Y \mapsto Y^{(k)}, Z \mapsto Z^{(\ell'-k)}\}$;
                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

4.3.3 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 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 4.7). This is related to the fact, familiar with the study of algorithms, that random binary search trees tend to be highly unbalanced [98].

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.

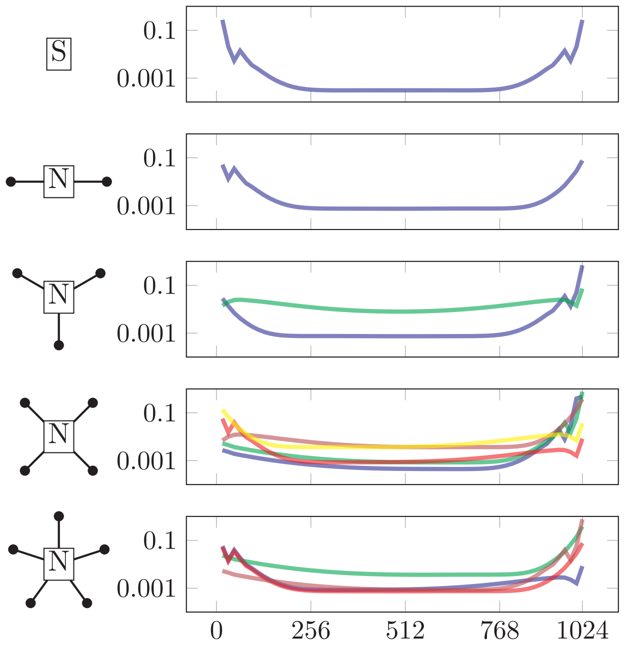


Figure 4.7. When an HRG rule has two nonterminal symbols, one is overwhelmingly likely to be much larger than the other. This plot shows, for various grammar rules (one LHS per row, one RHS per colored line), the probability (log scale) of apportioning 1024 nodes between two nonterminal symbols. This plot is best viewed in color.

4.4 Experiments

Here we show that HRGs contain rules that succinctly represent the global and local structure of the original graph. In this section, we compare our approach against some of the state-of-the-art graph generators. We consider the properties that characterize some real-world networks and compare the distribution of graphs generated using Kronecker Graphs, the Exponential Random Graph, Chung-Lu Graphs, and the graphs produced by the probabilistic hyperedge replacement graph grammar.

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. The probabilistically generated graphs are likely not isomorphic to the original graph. We can, however, still judge how closely the probabilistically generated graph resembles the original graph by comparing several of their properties.

4.4.1 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.

The six real-world networks considered in this paper 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. cere-*

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

visiae 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.

4.4.2 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 [73] 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 [19].

Exponential Random Graph Models are a class of probabilistic models. Their usefulness lies in that they directly describe several structural features of a graph [94]. We used default parameters in R's ERGM package [47] 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 [67]. 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.

4.4.3 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.

4.4.3.1 Global Measures

A key goal of graph modelling to preserve certain network properties of the original graph (*i.e.*, H as introduced in 3.2). 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.

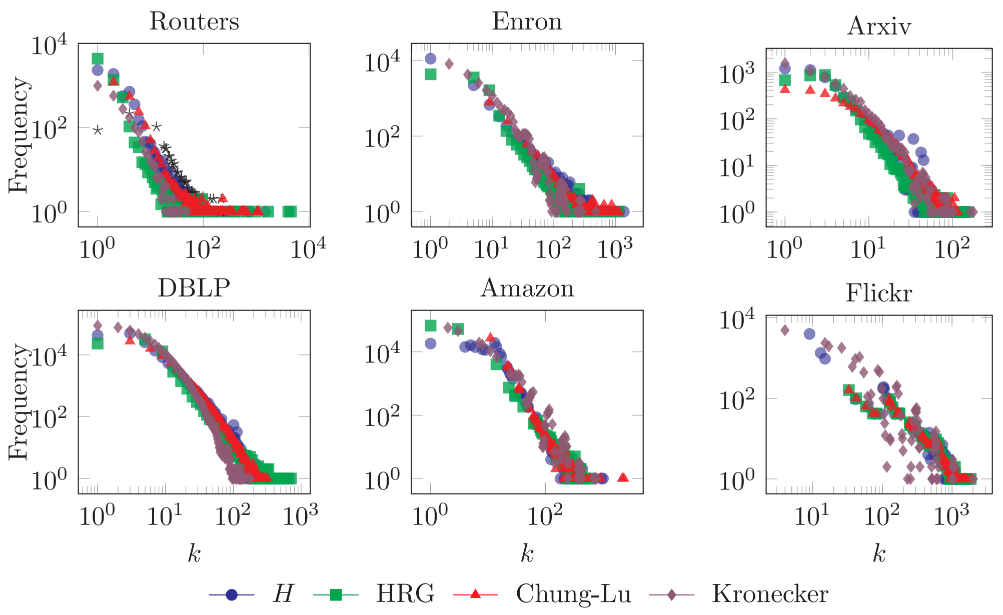


Figure 4.8. Degree Distribution. Dataset graphs exhibit a power law degree distribution that is well captured by existing graph generators as well as HRG.

Degree Distribution. The degree distribution of a graph is the distribution of the number of edges connecting to a particular vertex. Figure 4.8 shows the results of the degree distribution property on the six real-world graphs. Recall 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 4.9 shows the eigenvector scores

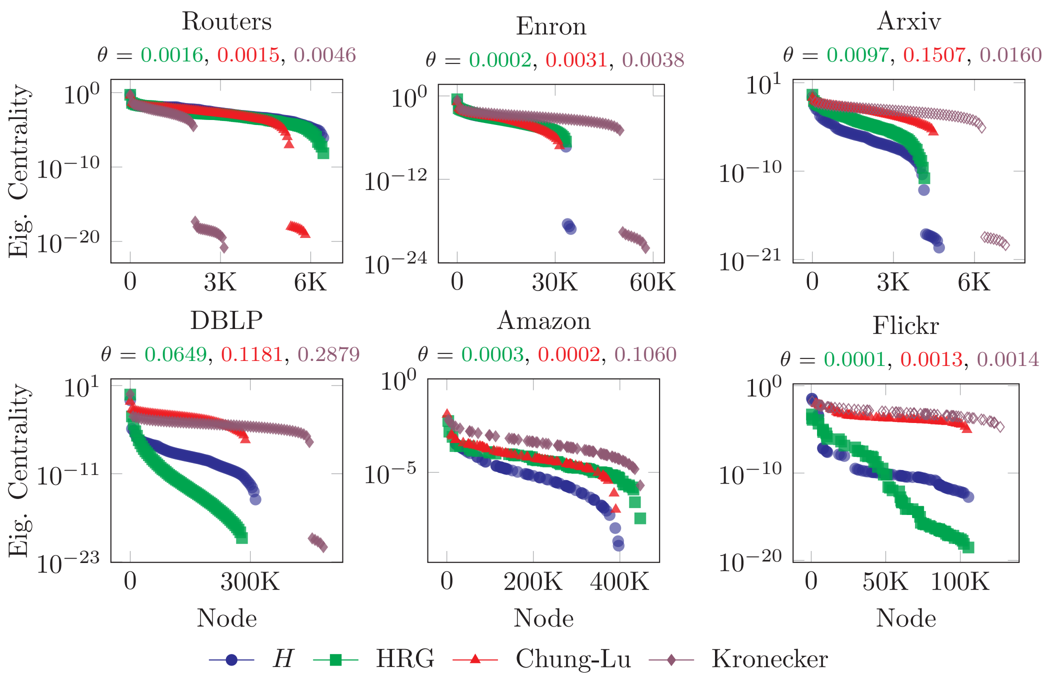


Figure 4.9. Eigenvector Centrality. Nodes are ordered by their eigenvector-values along the x-axis. Cosine distance between the original graph and HRG, Chung-Lu and Kronecker models are shown at the top of each plot where lower is better. In terms of cosine distance, the eigenvector of HRG is consistently closest to that of the original graph.

for each node ranked highest to lowest in each of the six real-world graphs. Because the x-axis represents individual nodes, Fig. 4.9 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

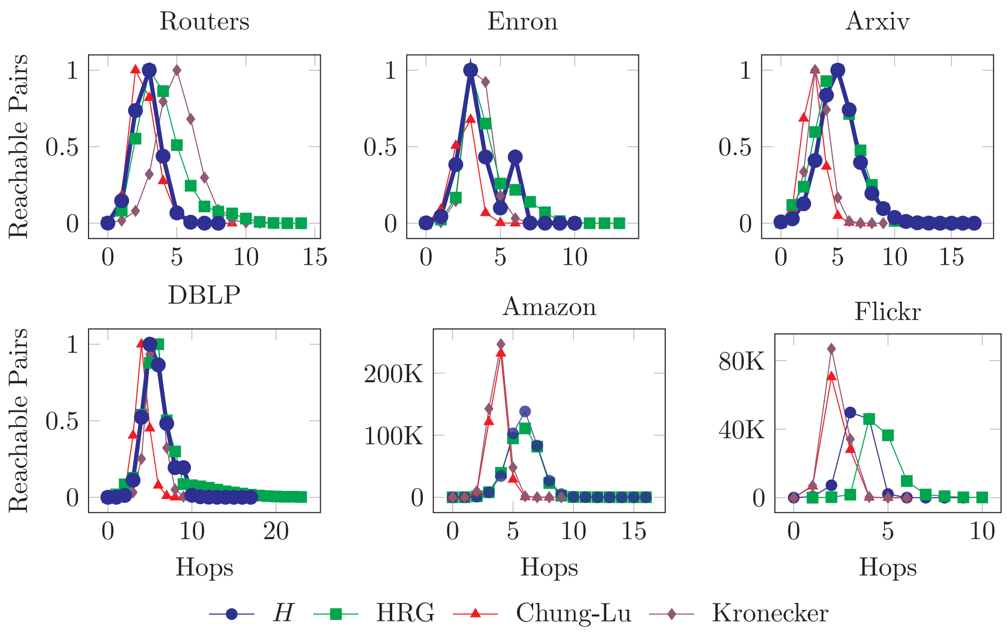


Figure 4.10. Hop Plot. Number of vertex pairs that are reachable within x-hops. HRG closely and consistently resembles the hop plot curves of the original graph.

quickly a vertex's neighborhood grows as the number of hops increases. As in related work [71] we generate a hop-plot by picking 50 random nodes and performing a complete breadth first traversal over each graph. Figure 4.10 demonstrates that HRG graphs produce hop-plots that are remarkably similar to the original graph.

Mean Clustering Coefficients. A vertex's clustering coefficient is a measure of how well connected its neighbors are [112]. 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 4.11 shows the average clustering coefficients for the sampled nodes as a function of its degree in the graph. Like the results from

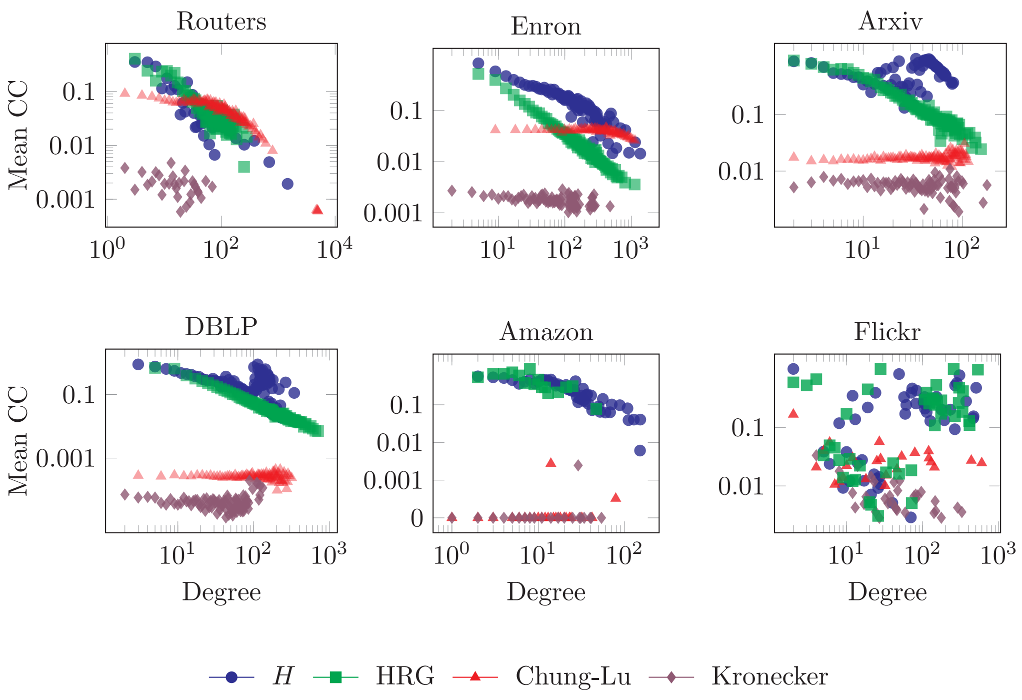


Figure 4.11. Mean Clustering Coefficient by Node Degree. HRG closely and consistently resembles the clustering coefficients of the original graph.

Seshadhri et al., we find that the Kronecker and Chung-Lu models perform poorly at this task [99].

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 4.12 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.

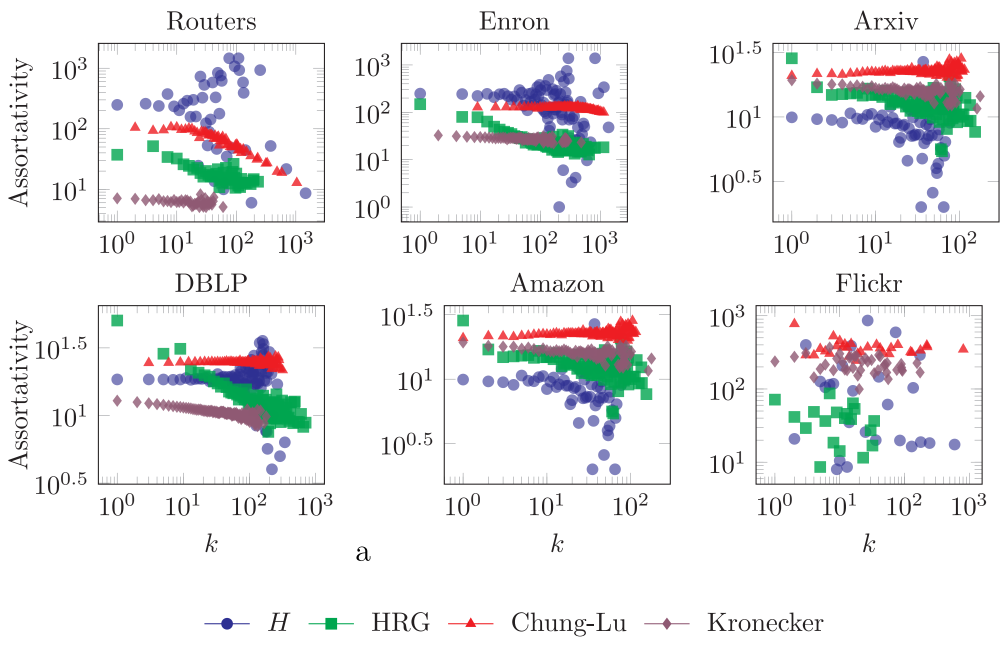


Figure 4.12. Local Degree Assortativity. HRG, Chung-Lu, and Kronecker graphs show mixed results with no clear winner.

4.4.4 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 [91, 109]. 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 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.

TABLE 4.2: Graphlet Statistics and Graphlet Correlation Distance (GCD) for 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.

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	–

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 [113]. 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 [113] could not process 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.

4.4.5 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 4.13 shows the mean GCD scores; not only does HRG generate good

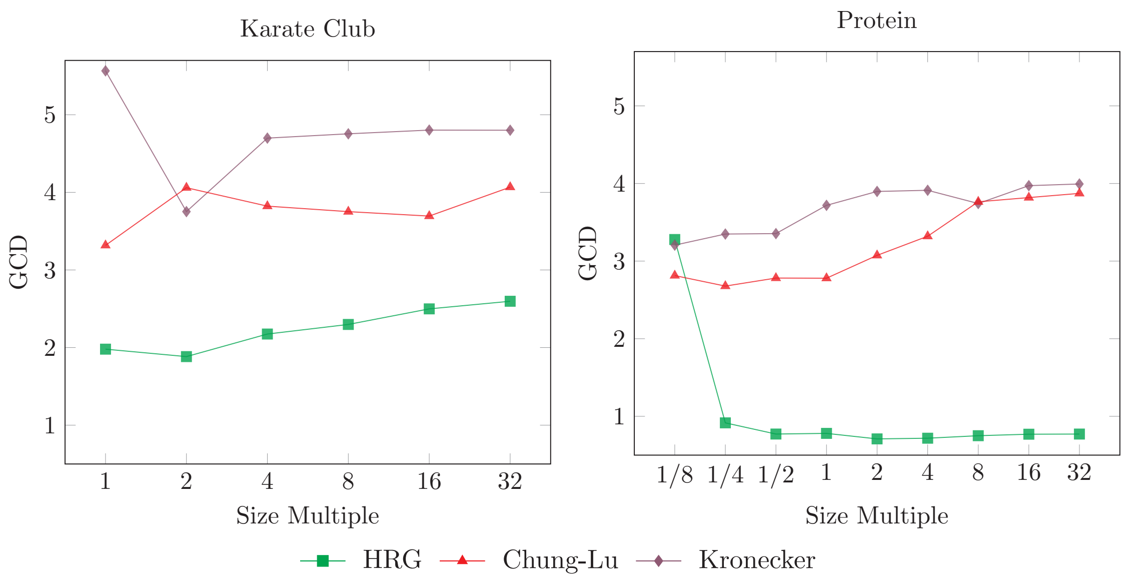


Figure 4.13. GCD of graphs extrapolated in multiples up to 32x from two small graphs. HRG outperforms Chung-Lu and Kronecker models when generating larger graphs. Lower is better.

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

4.4.6 Infinity Mirror

Next, we characterize the robustness of graph generators by introducing a new kind of test we call the *infinity mirror*.[1] One of the motivating questions behind this idea was to see if HRG holds sufficient information to be used as a reference itself. In this test, we repeatedly learn a model from a graph generated by an earlier version of the same model. For HRG, this 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.

[1] "Infinity mirror" gets its name from the novelty item with a pair of mirrors, set up to create a series of smaller and smaller reflections that appear to taper to an infinite distance.

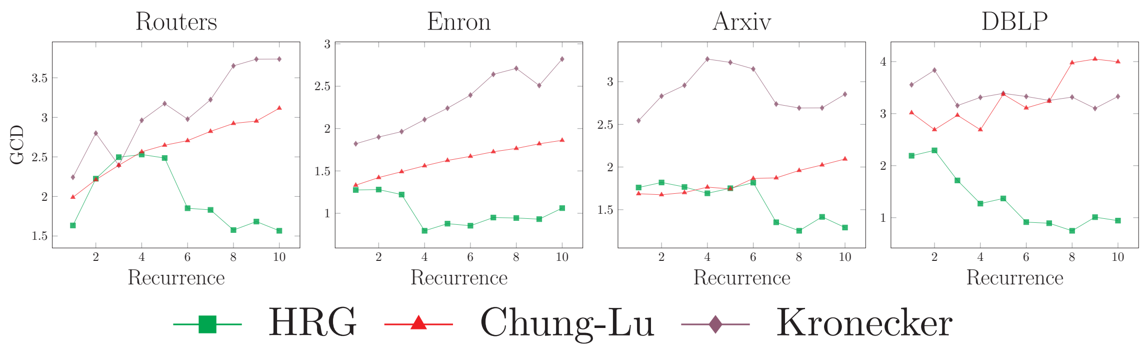


Figure 4.14. Infinity Mirror: GCD comparison after each recurrence. Unlike Kronecker and Chung-Lu models, HRG does not degenerate as its model is applied repeatedly.

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 4.14 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.

4.4.6.1 Infinity Mirror Model Size

The number of production rules derived from a given graph using Fixed-Size Graph Generation. Fig. 4.15 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.

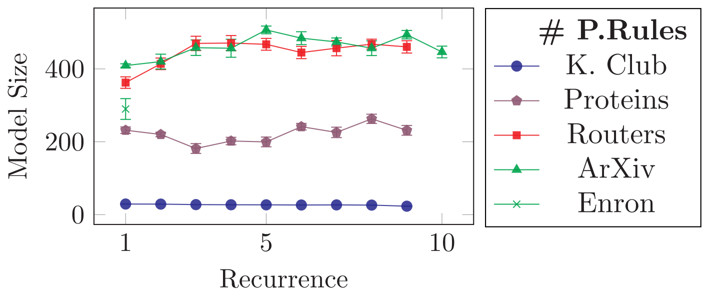


Figure 4.15. Number of rules (mean over 20 runs) derived as the number of recurrences increases.

4.4.7 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 clique tree will change causing the grammar to change in size and complexity. A large clique tree ought to create more rules and a more complex grammar, resulting in a larger model size and better performance; while a small clique tree 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 4.16 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

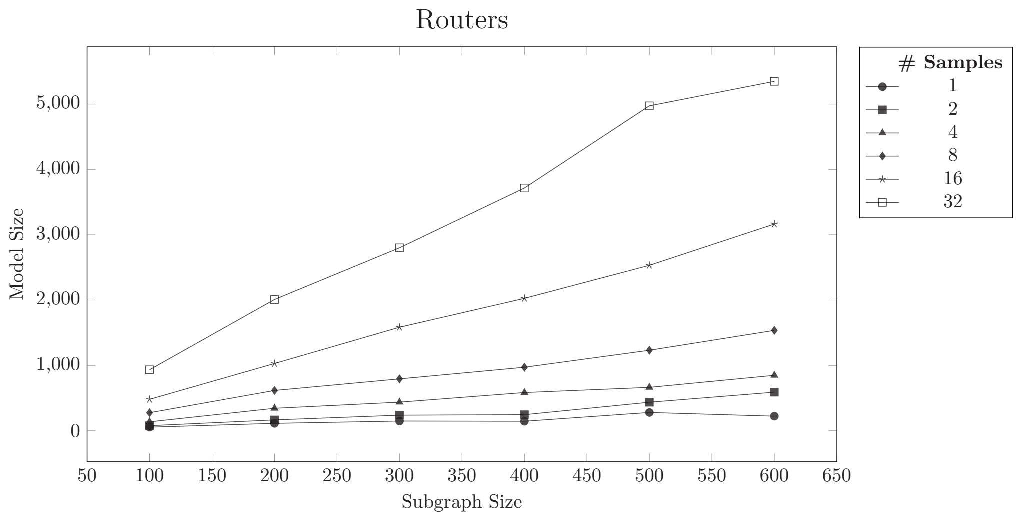


Figure 4.16. HRG model size as the subgraph size s and the number of subgraph samples k varies. The model size grows linearly with k and s.

choice of s affects the size of the clique tree 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 [43], *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.

4.4.7.1 Model size and Performance

One of the disadvantages of the HRG model, as indicated in Fig. 4.16, 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 4.17 illustrates the relationship between model size and the GCD. We use the Router and DBLP graphs

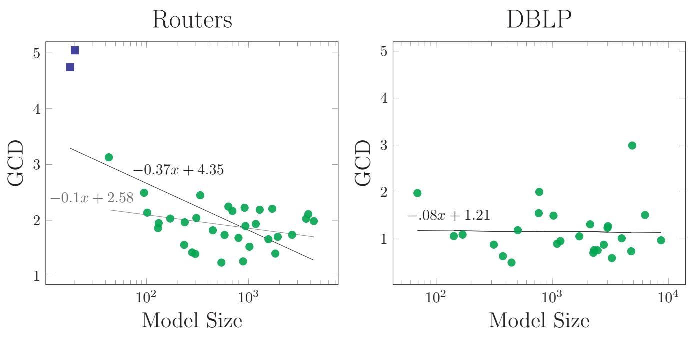
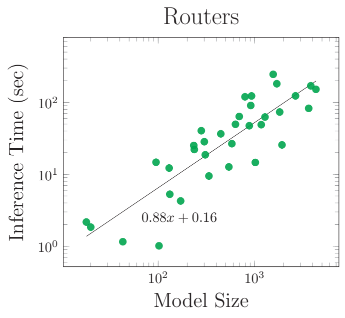


Figure 4.17. GCD as a function of model size. We find a slightly negative relationship between model size and performance, but with quickly diminishing returns. We show best-fit lines and their equations; the shorter fit line in the Routers plot ignores the square outlier points.

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 ($r = -0.08$, $p = 0.62$)

4.4.7.2 Runtime Analysis

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



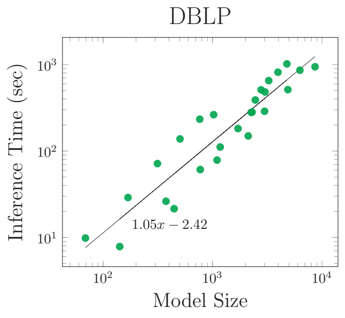


Figure 4.18. Total extraction runtime (*i.e.*, clique tree creation and rule extraction) as a function of model size. Best fit lines on the log-log plot show that the execution time grows linearly with the model size.

Unfortunately, finding a clique tree 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 usable clique trees [107] 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 clique tree as well as the number of vertices in each clique tree node. From Defn. 3.2.1 we have that the number of nodes in the clique tree is m. When minimal, the number of vertices in the largest clique tree node $\max(|\eta_i|)$ (minus 1) is defined as the treewidth tw. However, clique trees generated by MCS have $\max(|\eta_i|)$ bounded by the maximum degree of H and is denoted as Δ [33]. 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 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. 4.18 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 $O(n + m)$.

4.4.7.3 Graph Guarantees

In earlier work we showed that an application of HRG rules corresponding to a traversal of the clique tree 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 straightforward: 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.

4.5 Conclusions

This paper describes a new generative network framework that learns a hyperedge replacement grammar (HRG) given a simple, general graph and grows new graphs. The inference (or model learning) step uses clique trees (also known as junction trees, tree decomposition, intersection trees) to extract an HRG, which characterizes a set of production rules. We show that depending on how HRG grammar rules are applied, during the graph generation step, the resulting graph is isomorphic to the original graph if the clique tree is traversed during reconstruction. More significantly, we show that a stochastic application of the HRG grammar rules creates new graphs that have very similar properties to the original graph. The results of graphlet correlation distance experiments, extrapolation, and the infinity mirror are particularly exciting because our results show a stark improvement in performance over several existing graph generators.

Perhaps the most significant 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 into the nature of the extracted rules and their meaning (if any) is a top priority.

In the future, we plan to investigate differences between the grammars extracted from different types of graphs; we are also interested in exploring the implications of finding two graphs which have a large overlap in their extracted grammars. Among the many areas for future work that this study opens, we are particularly interested in learning a grammar from the actual growth of some dynamic or evolving graph. Within the computational theory community, there has been a renewed interest in quickly finding clique trees of large real-world graphs that are closer to optimal.

Because of the close relationship of HRG and clique trees are shown in this paper, any advancement in clique tree algorithms could directly improve the speed and accuracy of graph generation.

We encourage the community to explore further work bringing HRGs to attributed graphs, heterogeneous graphs and developing practical applications of the extracted rules. Given the current limitation related to the growth in the number of extracted rules as well as the encouraging results from small models, we are also looking for sparsification techniques that might limit the model's size while still maintaining performance.

CHAPTER 5

INFINITY MIRROR TEST FOR ANALYZING GRAPH GENERATORS

5.1 Introduction

Teasing out interesting relationships buried within volumes of data is one of the most basic challenges in data science research. When this data is viewed 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 in order to discover the fundamental principles that underlie a 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.

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

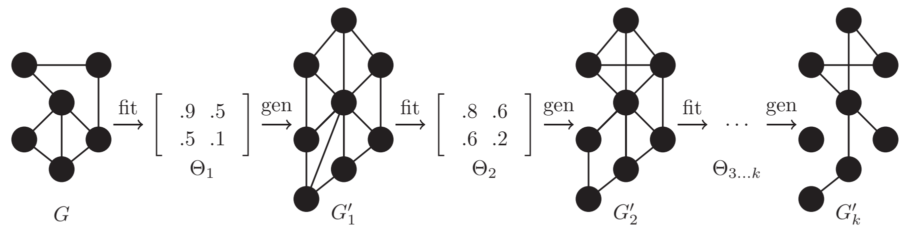


Figure 5.1: Example infinity mirror test on the Kronecker model. This test recursively learns a model and generates graphs. Although not apparent in G'_1, this example shows a particular type of degeneration where the model loses edges.

small perturbations caused by the implicit and inherent biases of each type of model may not be immediately visible using existing performance metrics.

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 G' holds sufficient information to be used as reference. Although a comparison between G and G' 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.

5.2 Graph Generators

Several graph generators have been developed for the tasks outlined above. We describe some of them here.

Kronecker Graph Kronecker graphs operate by learning a 2×2 initiator matrix K_1 of the form

$$K_1 = \begin{bmatrix} k_1 & k_2 \\ k_3 & k_4 \end{bmatrix}$$

and then performing a recursive multiplication of that initiator matrix in order to create a probability matrix P_{Kron} from which we can stochastically pick edges to create G'. Because of the recursive multiplication, the Kronecker product only creates graphs where the number of nodes is an exponential factor of 2, *i.e.*, 2^x [73].

The initiator matrix can be learned quickly, and the final graph shares many similarities with the original graph making the Kronecker graph model a natural fit for many graph modelling tasks.

Chung-Lu Models The Chung-Lu Graph Model takes, as input, some empirical (or desired) degree distribution and generates a new graph of the similar degree distribution and size [17, 18]. An optimized version called Fast Chung-Lu (FCL) was developed analogous to how the Kronecker model samples its final graph. Suppose we are given sequences of n-degrees d_1, d_2, $\ldots$ d_n where $\sum_i d_i = 2m$. We can create a probability matrix P_{FCL} where the edge e_{ij} has a probability $d_i d_j / m^2$ [89].

On average, the Chung-Lu model is shown to preserve the degree distribution of the input graph. However, on many graphs, the clustering coefficients and assortativity metrics of the output graphs do not match the original graph. Extensions of the Chung-Lu (CL) model, such as Transitive CL (TCL) [88], Binning CL (BCL) [79] and Block Two-Level Erdős-Rényi Model (BTER) [56], have been developed to further improve performance.

Exponential Random Graph Exponential Random Graph Models (ERGMs) are a class of probabilistic models used to directly describe several structural features of a graph [93]. Although ERGMs have been shown to model the degree distributions

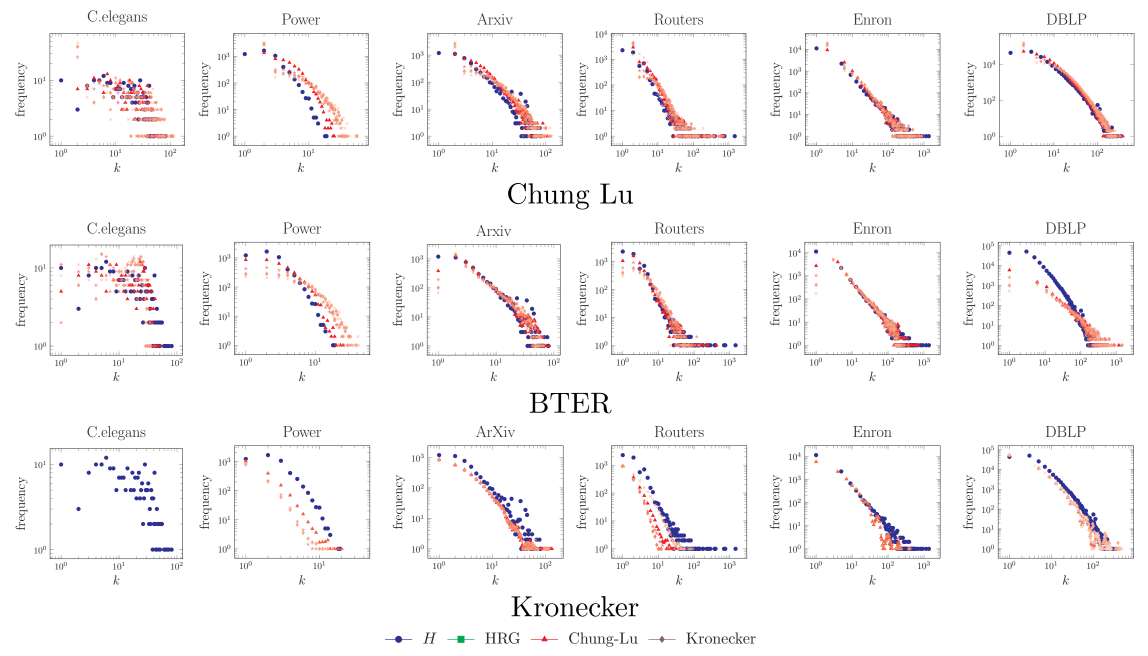


Figure 5.2: Degree distribution. G shown in blue. G'_2, G'_5, G'_8 and G'_{10} are shown in lighter and lighter shades of red. Degeneration is observed when recurrences increasingly deviate from G.

and other graph properties of small graphs, they simply do not scale to graphs of even moderate size. As a result we cannot include ERGM in the present work.

Existing approaches to graph modelling and generation perform well in certain instances, but each have their drawbacks. The Kronecker Model, for example, can only represent graphs with a power law degree distribution. Both Kronecker and the Chung-Lu models ignore local subnetwork properties, giving rise to more complex models like Transitive Chung-Lu for better clustering coefficient results [88] or Chung-Lu with Binning for better assortativity results [79, 80]. Exponential Random Graph Models (ERGMs) take into consideration the local substructures of a given graph. However, each substructure in an ERGM must be pre-identified by hand, and the complexity of the model increases (at least) quadratically as the size of the graph grows.

5.3 Infinity Mirror Test

We characterize the *robustness* of a graph generator by its ability to repeatedly learn and regenerate the same model. A perfect, lossless model (*e.g.*, $\Theta = G$) would generate G' as an isomorphic copy of the original graph. If we were to again apply the perfect model on the isomorphic G', we would again generate an isomorphic copy of the graph. On the other hand, a non-robust graph generator may generate a G' that is dissimilar from G; if we were to learn a new model from G' 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 G. 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 paper 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 G, a graph generator learns a model Θ_1 (where the subscript $\cdot_1$ represents the first recurrence) and generates a new graph G'_1. At this point, current works typically overlay graph properties like degree distribution, assortativity, etc. to see how well G matches G'_1. We go a step further and ask if the new graph G'_1 holds sufficient information to be used as reference itself. So, from G'_1 we learn a new model Θ_2 in order to generate a second-level graph G'_2. We repeat this recursive "learn a model from the model"-process k times, and compare G'_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 G is provided by the user. From G a model Θ_1 is fit, which is used to generate a new graph G'_1. Of course, G'_1 is only an approximation of G and is therefore slightly different. In the second recurrence a new model Θ_2 is fit from G'_1 and used to generate a new graph G'_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?

5.4 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 six real world networks considered in this paper are described in Table. 5.1. 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 [48]; the Power grid graph is the connectivity of the power grid in the Western United States [111]; the Enron graph is the email correspondence graph of the now defunct Enron corporation [55]; 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 recursively applied the Kronecker, Block Two-Level Erdos-Renyi (BTER), Exponential Random Graph (ERGM) and Chung-

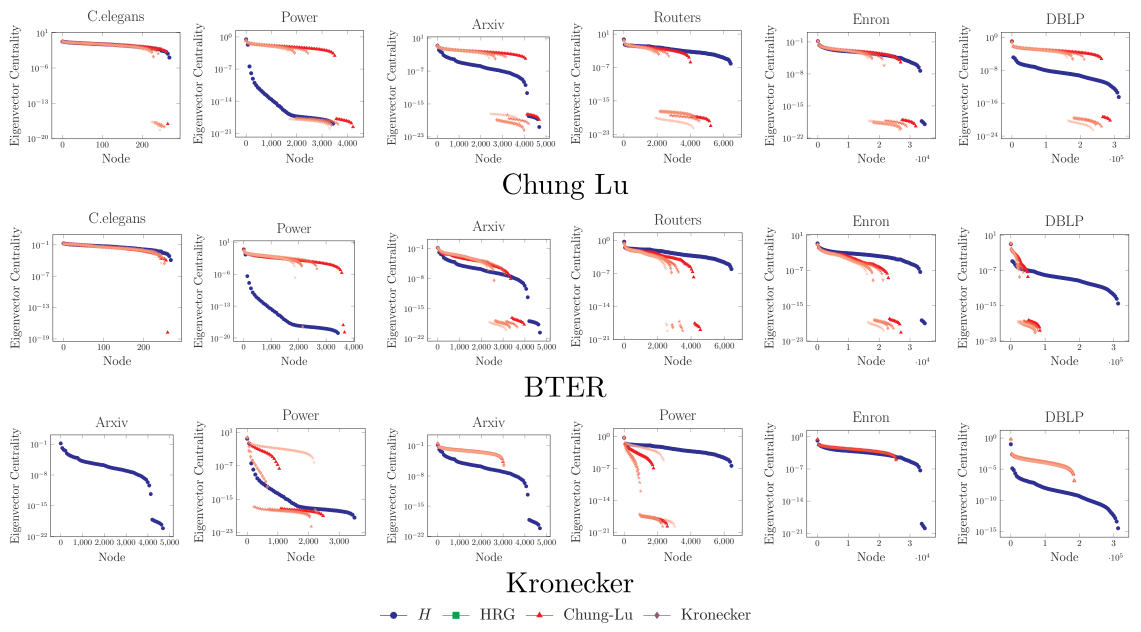


Figure 5.3: Eigenvector centrality. G shown in blue. Results for recurrences G'_2, G'_5, G'_8 and G'_{10} in lighter and lighter shades of red showing eigenvector centrality for each network node. Degeneration is shown by increasing deviation from G's eigenvector centrality signature.

Lu (CL) models 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 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 G'_2, G'_5, G'_8, G'_{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.

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

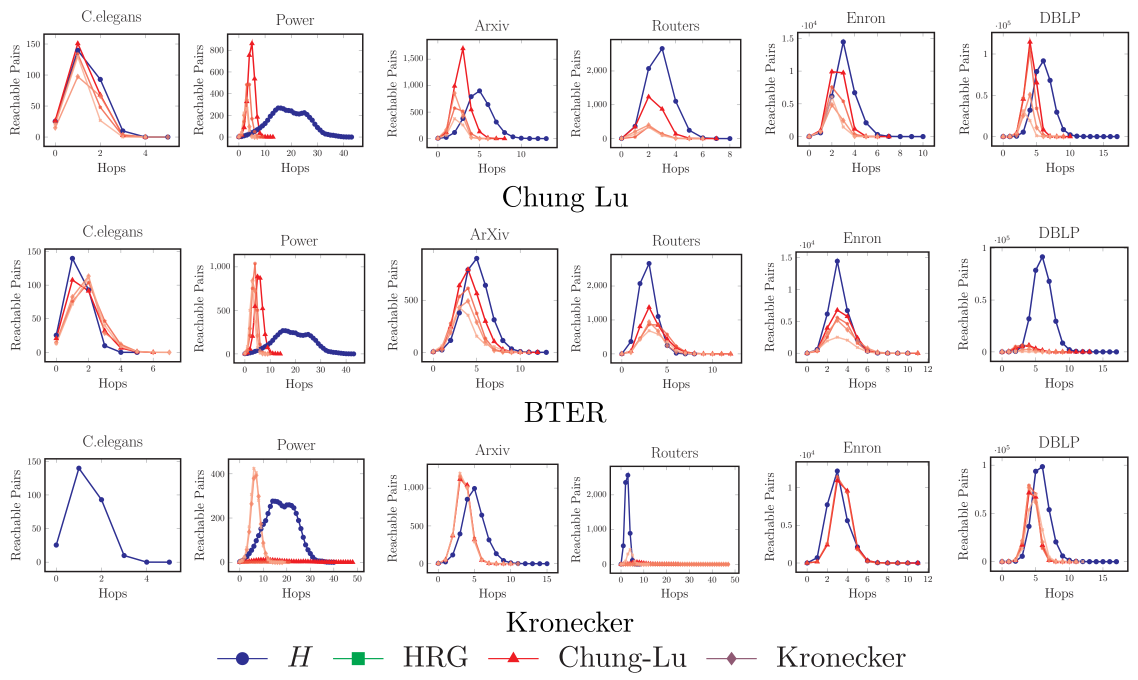


Figure 5.4: Hop plot. G shown in blue. Results for recurrences G'_2, G'_5, G'_8 and G'_{10} in lighter and lighter shades of red. Degeneration is observed when recurrences increasingly deviate from G.

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 [9].

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 G in blue first, then G'_2, G'_5, G'_8 and G'_{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.

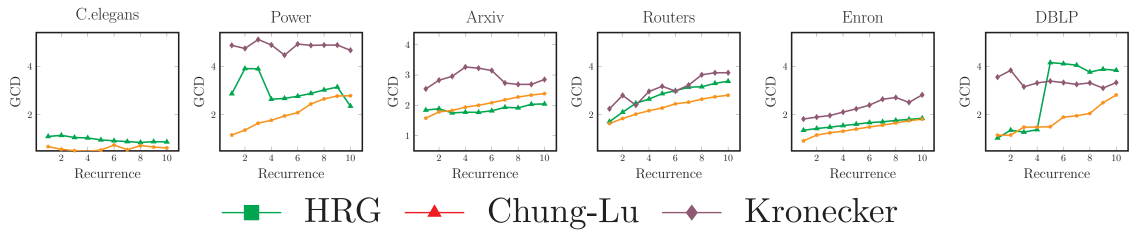


Figure 5.5: Graphlet Correlation Distance. All recurrences are shown for Chung Lu, BTER and Kronecker graph generators. Lower is better. Degeneration is indicated by a rise in the GCD values as the recurrences increase.

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

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

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 [69] 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 [91]. As a result, graphlet counting [5, 76, 108] and related comparison metrics [113] comprise the local-side of graph generator performance.

Thus a complete comparison of graph generator performance ought to include

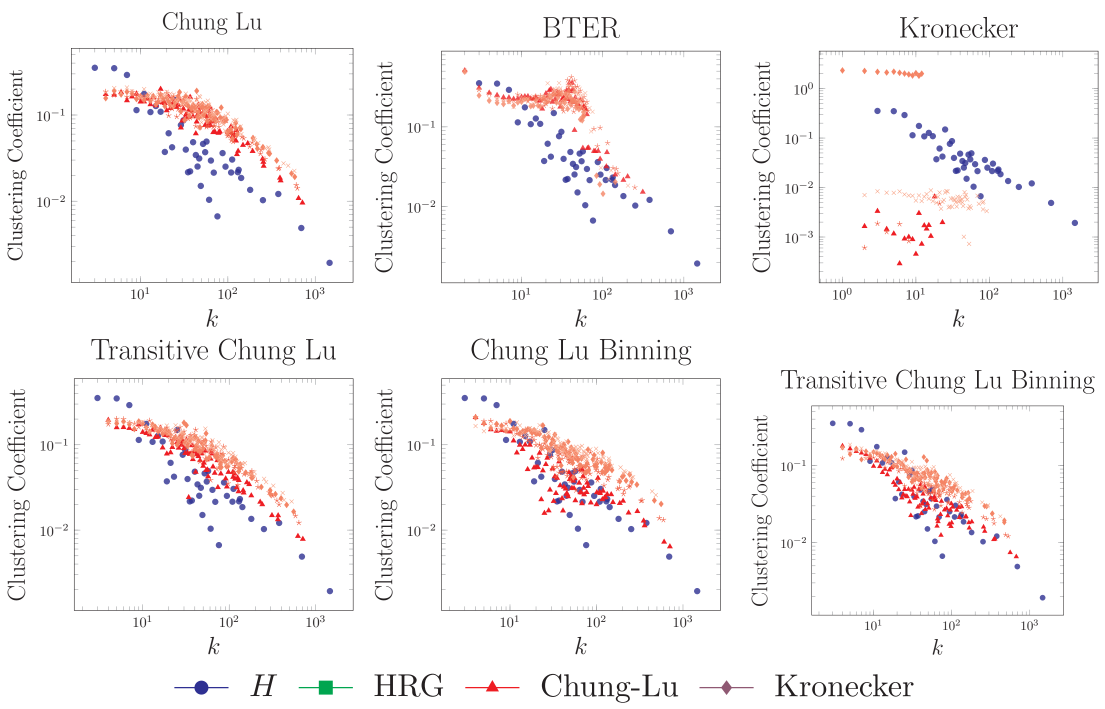


Figure 5.6. Clustering Coefficient. G is in blue. Results for recurrences G'_2, G'_5, G'_8 and G'_{10} in lighter and lighter shades of red. Degeneration is observed when recurrences increasingly deviate from G.

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 [91, 108]. 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.

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 [113]. 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.

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.

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 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

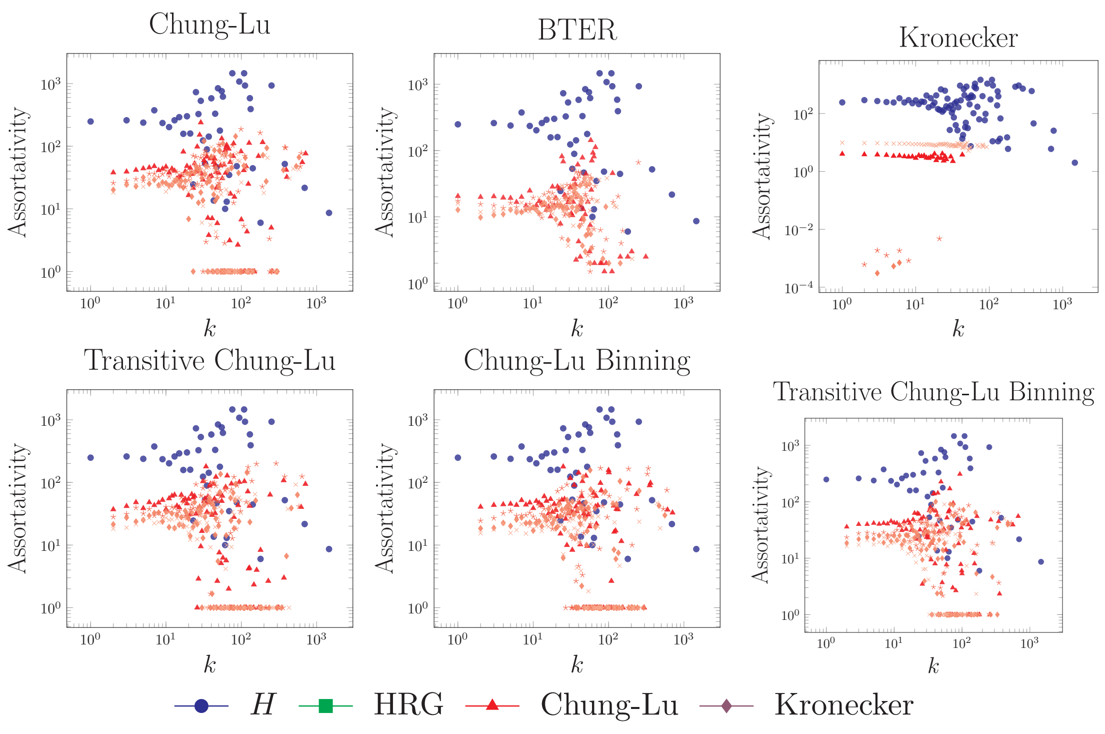


Figure 5.7. Assortativity. G is in blue. Results for recurrences G'_2, G'_5, G'_8 and G'_{10} in lighter and lighter shades of red. Degeneration is observed when recurrences increasingly deviate from G.

introduced the Transitive Chung Lu generator as an adaptation to the standard model [88].

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 [85].

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 [79]. 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.

5.5 Discussion and Conclusions

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 degeneration 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).

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

6.1 Introduction

The underlying mechanism of how social networks grow or, more generally, how real-world networks grow or evolve is a core area of research in computer science, mathematics, engineering, and physics. A recently introduced generative network modeling framework, Hyperedge Replacement Grammars (HRG), explores network structure to derive a set of replacement rules. These rules are also known as graph rewriting rules or as will be used throughout this work, production rules. These rules represent the grammars of the graph in question. Applying these recursively, we grow graphs with characteristic properties similar to the input graph. The rules derived are in response to the variable-elimination algorithm used in the decomposition step. The research question of interest is, what is the diversity of the productions rules we end up with in choosing the variable elimination algorithm during the decomposition step? We are motivated to explore this question because we want to examine if the rules are invariant to the choice of algorithms in this critical step. The original implementation of the HRG graph model yields TD characterized by relatively low treewidth (*tw*). In graph theory, *tw*is the size of the largest vertex set in a tree decomposition of a given graph. Vertex sets are also characterized as cliques, thus *tw*comes or is computed from the size of the largest clique in a chordal completion of the graph.

A key assumption in the tree decomposition step is computing tree decompositions with low (or close to optimal) *tw*. The motivation behind low *tw*is that with structures such as these trees, we can solve problems in probabilistic inference, constraint satisfaction, and matrix decomposition [41, 58, 58]. Unfolding how these structures are computed as a function of tree decomposition heuristics will be explored to gain better understanding on the structural patterns that underlie community and other network properties inherent to a diverse set of networks such as social, biological, and collaboration networks.

6.1.1 Core problem definition

We study the problem of graph decomposition. We examine patterns graph topology found in the resulting clique-trees (CT). The purpose is to infer a model for the given graph or a class of graphs. We have described how HRG takes the output of a tree decomposition and construct production rules, a process analogous to the productions derived in context-free grammars [4]. Formally, for any static graph, *i.e.*, a graphical representation of any complex system, let $G = (V, E)$ be the graph observed and $H = (X, E)$ its hypergraph. The graph $H^* = (X, E)$ is obtained through stochastic application of the production rules. Graph generation is split into two procedures, inference of HRG production rules and stochastic application of the graph grammars. The inference step may be visualized as follows, $H \xrightarrow{TD(H,\ell)} CT \xrightarrow{HRG} PR$ and the graph generation step $\mathcal{G}_{HRG}(PR, \nu) \rightarrow H^*$, where ν is the desired graph order (number of vertices), a$\mathcal{G}$ is the hyperedge replacement graph grammar, and ℓ is the variable elimination algorithm that characterizes the resulting clique-tree (CT). Synthetic graphs H^* reflect many of the network properties inherent in the reference or input graph. We interchangeably use the terms *reference graph* and *input graph* throughout this work when referring to a real-world graph.

Earlier work utilized the maximum cardinality search variable elimination algorithm described by Tarjan and Yannakakis [106] in the tree decomposition step. Thus, one of the open questions in the HRG graph model is how the choice of tree decomposition algorithm biases or affects the topology of the graph-fragment patterns used in the productions inferred. We answer this question and leverage multiple tree decompositions to hone in on patterns that might be invariant to tree decomposition and may offer insight into the fundamental patterns that contribute the global structure of a graphs.

We are interested in examining how the choice of ℓ contributes to the resulting characteristics of the CT and thus to the resulting graph grammar. Construction of the graph grammar is biased on the choice of variable elimination algorithm.

6.1.2 Outline of this work

In Section 6.2 we examine the background literature. In Section 6.4 we outline specific research questions and describe experiments and methods used to help answer them. Details of the results and a brief discussion on what conclusions we can draw are presented in Section 6.5 and conclude by looking ahead to some future direction in 6.5.2.

6.2 Background information and related work

A great deal of work has focused on applications of the algorithms that underlies 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. This work examines the use of tree decomposition heuristics for deriving HRG's production rules characterizing the features of the graph model.

Existing algorithms for constructing TDs range from those shown to be practically intractable due to their complexity to those characterized as computational for employing heuristic approaches whose deviation from optimality is guaranteed. Shoikhet and Geiger [100] developed a computational (*i.e.*, practical) algorithm, QuickTree, that can optimally triangulate graphs in a reasonable amount of time. This algorithm saw usefulness in problems in Bayesian inference and clique-tree inference algorithms. Gogate and Dechter [36] developed QuickBB, a branch and bound (BB) algorithm for computing the $tw$of undirected graphs. BB search algorithms fuel algorithm design trends in discrete and combinatorial optimization.

Jones *et al.*, describe very similar work where they extracted grammars using tree decomposition. Their work focuses on a class of tree decomposition called edge-mapped TD [49]. This tree decomposition is extended to use a topological sort that produces clique trees like those in HRG. However, they evaluate the model in contrast to other forms of the tree decomposition yielding linear trees. They rely on a measure of perplexity (akin to the entropy of text) 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 as well as edges.

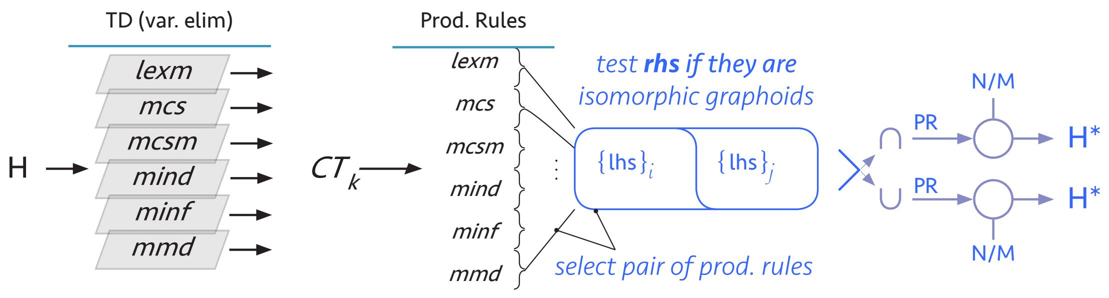


Figure 6.1. Proposed methodology: Tree Decomposition, clique-tree (CT), Production Rules (PR)

6.2.1 Tree Decomposition Influence on HRG

The influence of tree decomposition on graph grammars is explored to shed light on the bias or the assumptions inherent in the underlying algorithms. All graphs can be decomposed (though not uniquely) into a clique-tree, also known as a tree decomposition, junction tree, join tree, intersection tree, or cluster graph. clique-trees are found through a simple ordering the vertices in the input put graph. Computing a perfect ordering ensures graph chordality and finding this perfect ordering relies on maximum cardinality search or lexicographical BFS (lexBFS) algorithms. Minimal vertex separators is another algorithm for computing clique-trees are examined in this paper.

Constant factor approximation algorithms typically used for treewidth problems include triangulation heuristic algorithms such as min-degree, max-cardinality search, min fill, lexBFS, and min-vertex separators, we expand on these next with more detail. Treewidth (tw) is defined as a property for clique trees or cluster graphs, measuring the ''tree-likeness" of the reference graph.

6.3 Methods and proposed work

6.3.1 Tree Decomposition and Variable Elimination

The question of interest is how does tree decomposition contributes, affects, or biases the productions rules derived from the input graph? To address this we focus on deriving rules using variations of tree decomposition. In the decomposition tree step, a Adcock *et al.* [1] have developed tree decomposition that relies on different variable elimination ordering algorithms.

A battery of variable elimination heuristics will be used in the tree decomposition step. These heuristics include maximum cardinality search (MCS), minimum triangulation of H (MCS-M) [11], lexicographic search with minimal ordering (LEX M) [95],

TABLE 6.1

Summary of variable elimination algorithms used in this study.

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

minimum fill-in (MINF), Minimum degree (MIND), and Multiple Minimum Degree (MMD) [74]. These heuristics an others are described by Kemazi and Poole [52]. However, it must be noted that these heuristics do not add up the complete list of available algorithms [12, 87], but we choose this for this work.

The baseline tree decomposition variable elimination algorithm allowing or facilitating transformation of an input graph into clique-tree is MCS. In our earlier work, the HRG model uses Tarjan and Yanakakis' MCS algorithm. We examine and evaluate clique trees 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 exploring others as well in detail next.

Answering the core question explored in this work requires us to examine some properties of the clique trees. We start with (tw), a graph feature or property measuring how “treelike” a tree is.

6.3.2 Datasets

A number of empirical networks, otherwise known as real-world networks, are used to evaluate the different heuristics. We are interested in variable elimination and their effect on graph grammars. These networks vary in the number of nodes and edges. We consider small and large networks push the limits of tree decomposition, but more importantly we want a wide range and diverse set of graph grammars (graph fragments) to understand how the POE contributes to the network model. These networks are also characterized by other properties that we hope HRG is able to model accurately.

Small Network Datasets. Highland tribes (Gama) is a social network of tribes of the Gahuku-Gama alliance structure found in Eastern Central Highlands of New Guinea. Zachary’s Karate club (Karate) is a snapshot of a university karate club studied by Wayne Zachary. All real-world networks are public datasets [64, 68].

Larger Network Datasets. *EuroRoad* is an infrastructure network of Europe’s roads where nodes are cities and edges represent the road that connects them. *CollegeMsg* is a network of private message 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.

6.4 Experiments and Results

One of they ways we can inspect how the choice of variable elimination algorithm affects the production rules is to generate synthetic graphs and use a graph alignment

TABLE 6.2

Real-world networks

Datasets	nodes (n)	edges (m)	Avg. degree ($\bar{k}$)
Tribes (Gama)	16	58	7.25
Les Miserables	77	254	6.60
Hypertext	113	2196	38.87
arenas-jazz	198	2742	27.70
Contact	274	2124	15.50
email-Eu-core	309	1938	12.54
Infectious	410	2765	13.49
EuroRoad	1174	1417	2.41
College Msg	1899	13838	14.57

test to measure graphlet distance between graphs. Figure 6.2 shows that half the datasets show no significant difference in their GCD score. Complete GCD results are presented in Tab. 6.5.

6.4.1 Evaluation of Tree Decompositions

Methods of evaluating tree decomposition include tw. Derived productions rules undergo a isomorphic test to find subsets used to grow synthetic graphs. The resulting graph get the bulk of the metrics. These graphs will be evaluated based on degree distribution, clustering coefficients, hop-plot, and graphlet correlation distance (GCD).

One of our first goals is to evaluate HRG's variable elimination method of the resulting CT. Each dataset is transformed into a clique tree, the tree is binarized, only them we derive the production rules. The tree decomposition is obtained using

TABLE 6.3

tw as a function of variable elimination algorithm

Dataset	**Treewidth** (*tw*)						
	mcs* μ (σ)	mcs	lexm	mcsm	mind	minf	mmd
LesMis	9 (0)	11	9	11	9	9	9
contact	40	42	43	43	50	40	40
arenas-jazz	59 (0)	88	77	81	104	59	73
pdzbase	6	9	12	13	6	6	6
ucforum	126	326	361	341	282	276	279
Hypertext	76 (0)	80	89	89	76	76	76
Infectious	39 (0)	65	56	128	42	40	49
emailEuCore	34 (0)	41	46	45	35	34	35
EuroRoad	6.6 (2.6)	42	30	48	19	16	16
College Msg	87.6 (20.3)	459	602	543	404	394	403

the tool INDDGO [1] while each variable elimination heuristic is specified as one of the arguments in the execution.

6.4.2 Treewidth

Table 6.3 shows the (*tw*) that results from tree decomposition using different variable elimination algorithms. The first column shows the *tw* computed using the HRG model. We learn that our implementation of *mcs* is among the variable elimination algorithms that yield clique-trees with the smallest *tw* and that the next best algorithm is is *minf*. The baseline algorithm for *TD*is a Python implementation

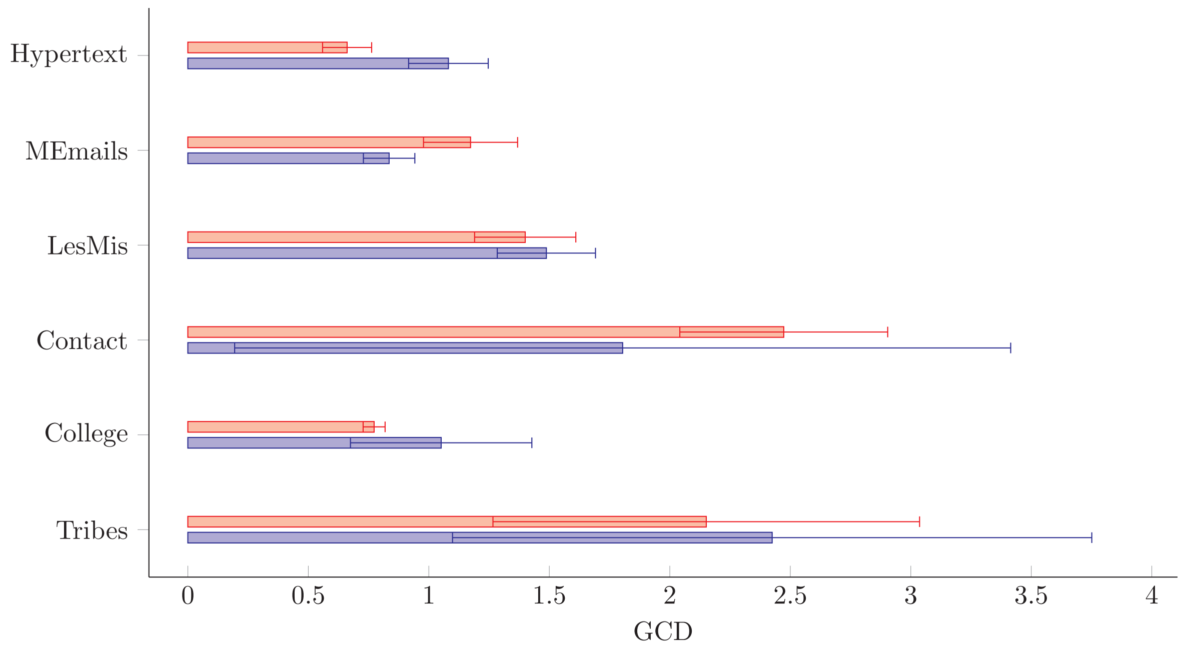


Figure 6.2. Graphlet correlation distance on graphs generated using productions derived from clique-trees computed with 🟪 MCS (baseline) and 🟧 MinF.

of QuickBB, which relies on *mcs*. All other elimination orderings will be computed using INDDGO [40].

6.4.3 Production Rules

Production rules are characterized by by the number of non-terminal nodes in the left-hand side (LHS) of the rule.A matrix showing the percentage of overlap between different.

6.4.4 Isomorphic Overlap

We are interested in patterns found in the various production rules. Each set of production rules set are independently derived. However, production rules still share

TABLE 6.4

Production rules overlap: email-Eu-core

	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						-

Figure 6.3. Isomorphic overlap via Jaccard Similarity

dependence on the source graph even if they are derived using different TD schemes. We examine graph structure as found in RHS rules. We use both approximate and complete isomorphic tests. We select rules that have the same number of terminal objects in the LHS. We compute Jaccard similarity to assess the ratio of isomorphic rules over a range of TD. Triangle heat dots show the Jaccard similarity between any pair of production rules. This result implies that certain variable elimination methods share an affinity. We intend to exploit this affinity to test if the intersection of isomorphic graph-fragments (multi-graph instances of the RHS of production) yields a smaller set of production rules that generates graphs.

6.4.5 Isomorphic graph-fragments

A subset of production rules from the concatenation of rules derived using several different tree decompositions is possible using isomorphic graph-fragment tests. This approach lets us hone in on automatically produced or derived rules. Some graph-fragments are observed to be found or produced by different tree decomposition algorithms. This collection of PHRG rules lead to productions that can generating graphs using a stochastic application of the rules. Fig 6.5 show the number of rules in a concatenated rules set (blue) and the subset resulting from isomorphic patterns. These smaller subset of production rules captures sufficient information to generate synthetic graphs. In Fig. 6.5 we show both the rules subset from the isomorphic test in contrast to the union of the rules for real-world networks and for random (Barabasi-Albert) graphs.

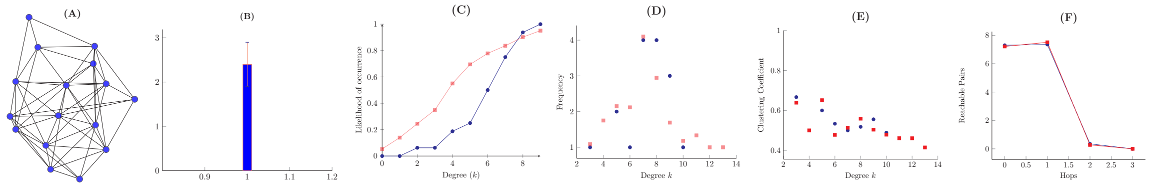


Figure 6.4. Degree distribution. (A) Highland tribes graph, (B) GCD, (C) Degree CDF, (D) Degree distribution, (E) Clustering coefficients, (F) Hop-plot

6.4.6 Generalization to Classes of Graphs

Formative patterns playing a key role in how networks grow might be found to be consistent across similar class of networks. And through a range of networks

Figure 6.5. Union and set intersection of the production rules derived using different variable elimination methods.

generated by the Barabasi-Albert random graph model we probe the usefulness of patterns found through a battery of tree decomposition algorithms. We want to examine if the production rules inferred from these artificially generated graphs hold the power to characterize the degree distribution in a single class of well defined graphs.

We generate random graphs of various sizes using the Barabasi-Albert model and we process each graph. In other words, we infer as set of production rules using the same set of tree decomposition algorithms used throughout our experimental setup. Furthermore, we hone in on RHS graph-fragments that are isomorphic regardless of the TD algorithm and test if this set holds sufficient information to generate similar graphs.

6.5 Conclusions

Generative network models using hyperedge replacement grammars are power tool in network science. The HRG graph model preserves many of the input graph's network properties such as degree distribution and others. One open question on the HRG generative model is how the choice of tree decomposition affects the inferred model. We examine a battery of tree decomposition algorithms to examine the production rules and find an answer to this question. Our initial experimental results highlight two important findings: not all variable elimination algorithms yield clique trees, and of those variable elimination algorithms that yield clique trees not all derived production rules fire, *i.e.*, can generate graphs.

TABLE 6.5

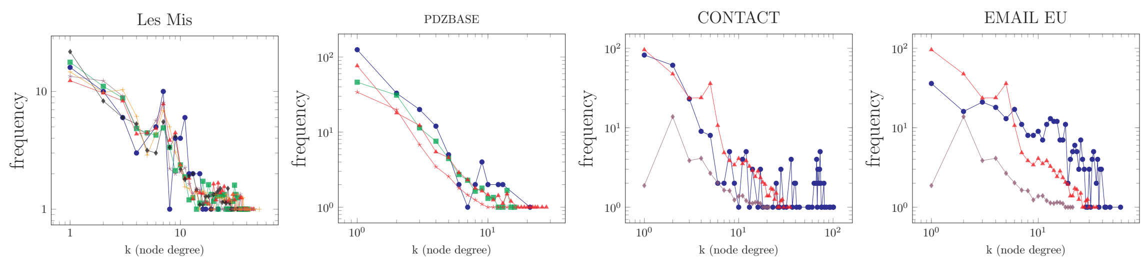
Clique-trees using variable elimination heuristics

	Graphlet Correlation Distance $\mu(\sigma)$					
Dataset	mcs	lexm	mind	mcsm	minf	mmd
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)
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)
Hypertext	0.707 (0.137)	0.582 (0.112)	0.606 (0.143)	0.576 (0.137)	0.661 (0.154)	0.653 (0.177)
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)
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)
pdzbase	1.8 (1.3)	2.2 (1.5)	4.0 (.40)	3.2 (1.3)	3.8 (0.7)	2.7 (0.27)
emailEuCore	3.0456 (0.118)	2.554 (0.100)	2.552 (0.248)	2.491 (0.151)	2.862 (0.269)	2.679 (0.267)
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)
College Msg		3.276 (0.275)	0.835 (0.036)	0.605 (0.047)	0.773 (0.059)	0.773 (0.059)

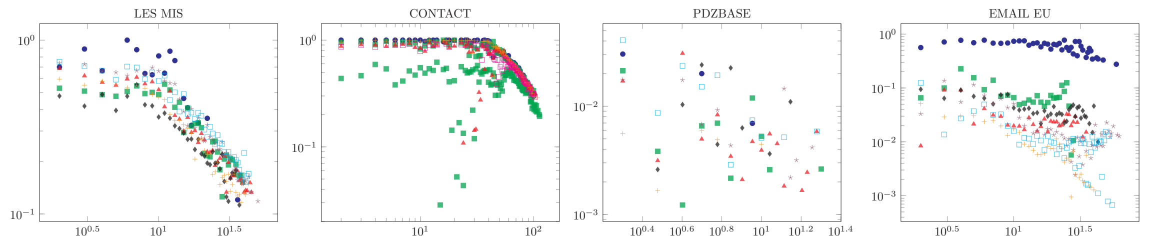
Where the derived production rules do fire, we explore the intersection of rules. If we further examine where the rules, or graph fragments, intersect we test if these subsets are smaller than any one individual rules set and test the set's ability to fire. When this small set of production rules can fire, we test the set to grow graphs. Our conclusion is that there are latent graph fragments with structure that when combined are compact enough to grow graphs. Using multiple variable elimination algorithms during the tree decomposition step flushes out these graph fragments. This finding hints to the need for a optimal method of finding a more compact set of production rules.

6.5.1 Limitations

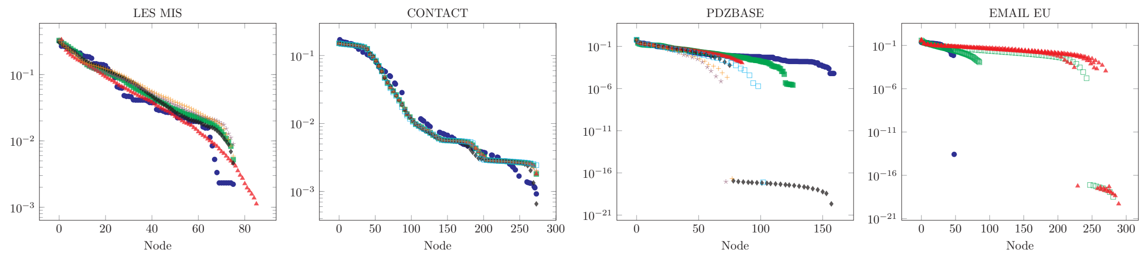
The challenges in this approach is the run-time increase required for each variable elimination algorithm. This naive approach does not offer a guarantee that we will end up production rules sets guaranteed to fire. The second limitation concerns scalability. An increase in the number of times we perform a tree decomposition given



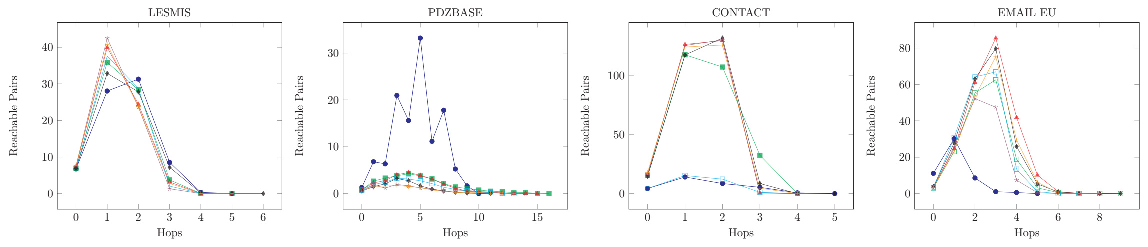
(a) Degree distribution.



(b) Clustering Coefficients



(c) Eigenvector Centrality



(d) Hop Plot.

Figure 6.6. Network properties

different variable elimination algorithms increases the complexity of this intermediate step. Processing large graphs increases run-time, but we can not guarantee the tree decomposition will yield viable production rules. The size of the graph is address in this work by sampling from the reference graph. These limitations, again hint at the need for new approaches combining smart graph sampling and optimization to an ensemble of variable elimination yielding richer rules compared to the naive approach.

6.5.2 Future Direction

Potential directions for this work include looking into the use of perplexity, a measure of entropy of a subgraph, to evaluate derived HRG models. Especially as we improve on the on the naive HRG implementation. Another future direction for this work may explore deeper an ensemble tree decomposition that relies on finding a fast and practical variable elimination methodology that guarantees both, a smaller subset of rules and rules that can fire. These future directions may lead to the development of new tools for natural language processing and graph mining applications.

CHAPTER 7

SUMMARY AND FUTURE DIRECTIONS

7.1 Summary

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 HRG to grow 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.

- **HRG Growing graphs.** Initial implementation derives a HRG, a set of production rules. Chapter 3 described the initial implementation of HRG as a graph model which showed the principled approach to transforming graphs into trees, deriving the graph grammar, and growing graphs using a stochastic growth algorithm.
- **Probabilistic HRG.** Improvements to the HRG model where the order (or size, in terms of the number of nodes) was specified. Chapter 4, described a new

procedure for growing graphs to address a limitation in the original implementation. Although the generated graphs have the same mean size as the original graph, the variance was too high to be useful. We used dynamic programming to sample a graph with specified size to illustrate this concept.

- **Infinity mirror tests model degeneration.** A new metric of model robustness examined model degeneration in Chapter 5: infinity mirror test. This procedure examined any graph generator's output and repeatedly fitted a model to the new output to see how well the model captured sufficient features to reconstruct graphs. We saw that after a few recursive runs some of the existing graph models tend to degenerate. Some models quickly lost the necessary features the model relies on to generate graphs with many of properties in the original graph.

- **Graph grammar bias.** We explored the core of the tree decomposition step, a critical step during model inference. Specifically, how the production rules are biased during tree decomposition, by examining the algorithms involved in transforming the input graph into tree-like graphs. Chapter 6 showed an examination of various tree decomposition algorithms. By studying the rules between sets, we evaluated various production rules from different tree decomposition algorithms. Moreover, where we had viable rules, we generated graphs allowing us to study the quality of the synthetic graph's rules produced in contrast to the input graph.

7.2 Vision and Future Work

By developing algorithms for graph generation models, we elucidate underlying mechanisms contributing to network growth. 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 multinomial degree distribution and or multinomial 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. On possible research path is, designing recurrent neural network controllers that optimize the selection of rules that form new production rules sets to generate neural network architectures. We evaluate the architecture on deep learning task and if it performs well, 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 [82, 102, 114].

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, approximating 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, 42, 78, 90]. 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*, (just-accepted), 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. L. Backstrom, D. Huttenlocher, J. Kleinberg, and X. Lan. Group formation in large social networks: membership, growth, and evolution. In *SIGKDD*, pages 44–54. ACM, 2006.

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

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

15. C. Canestro, H. Yokoi, and J. H. Postlethwait. Evolutionary developmental biology and genomics. *Nat Rev Genet*, 8(12):932–942, Dec. 2007. ISSN 1471-0056. doi: 10.1038/nrg2226. URL `http://dx.doi.org/10.1038/nrg2226`.

16. 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.

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

18. 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.

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

20. 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`.

21. N. R. Council et al. *Network science*. National Academies Press, 2006.

22. 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.

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

24. F. Drewes, B. Hoffmann, and D. Plump. Hierarchical graph transformation. *Journal of Computer and System Sciences*, 64(2):249 – 283, 2002. ISSN 0022-0000. doi: http://dx.doi.org/10.1006/jcss.2001.1790. URL `http://www.sciencedirect.com/science/article/pii/S0022000001917908`.

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

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

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

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

29. 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.

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

31. 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.

32. 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`.

33. 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.

34. 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`.

35. 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.

36. 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.

37. 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.

38. 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, Feb. 2009. ISSN 0070-3370. URL `http://www.ncbi.nlm.nih.gov/pmc/articles/PMC2831261/`. 19348111[pmid].

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

40. C. Groër, B. D. Sullivan, and D. Weerapurage. Inddgo: Integrated network decomposition & dynamic programming for graph optimization. *ORNL/TM-2012*, 176, 2012.

41. 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.

42. 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.

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

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

45. 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.

46. 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].

47. 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.

48. 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`.

49. 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.

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

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

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

53. 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`.

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

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

56. 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.

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

58. 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.

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

60. 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.

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

62. 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.

63. 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.

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

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

66. J. Leskovec. *9. Kronecker Graphs*, chapter 9, pages 137–204. Society for Industrial and Applied Mathematics, 2011. doi: 10.1137/1.9780898719918.ch9. URL `http://epubs.siam.org/doi/abs/10.1137/1.9780898719918.ch9`.

67. 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.

68. J. Leskovec and A. Krevl. SNAP Datasets: Stanford large network dataset collection. `http://snap.stanford.edu/data`, June 2014.

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

70. J. Leskovec, J. Kleinberg, and C. Faloutsos. Graph evolution: Densification and shrinking diameters. *ACM Trans. Knowl. Discov. Data*, 1(1), Mar. 2007. ISSN 1556-4681. doi: 10.1145/1217299.1217301. URL `http://doi.acm.org/10.1145/1217299.1217301`.

71. 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.

72. 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.

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

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

75. 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.

76. 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.

77. J. L. Moreno. Who shall survive?: A new approach to the problem of human interrelations. 1934.

78. 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.

79. 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.

80. S. Mussmann, J. Moore, J. J. Pfeiffer, III, and J. Neville. Incorporating assortativity and degree dependence into scalable network models. In *Proceedings of the Twenty-Ninth AAAI Conference on Artificial Intelligence*, AAAI'15, pages 238–246. AAAI Press, 2015. ISBN 0-262-51129-0. URL `http://dl.acm.org/citation.cfm?id=2887007.2887041`.

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

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

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

84. M. Newman. Networks: An introduction. 2010.

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

86. C. Pennycu, S. Aguinaga, and T. Weninger. A temporal tree decomposition for generating temporal graphs. 2017.

87. 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.

88. 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.

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

90. 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.

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

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

93. 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. doi: 10.1016/j.socnet.2006.08.002. URL `http://dx.doi.org/10.1016/j.socnet.2006.08.002`.

94. 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.

95. 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.

96. D. E. Sadava. *Life : the science of biology.* Sinauer Associates, Sunderland, MA, 2014. ISBN 9781429298643 1429298642 9781464136399 1464136394. ID: 811239088.

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

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

99. 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.

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

101. 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`.

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

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

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

105. 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`.

106. 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.

107. 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.

108. 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.

109. 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.

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

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

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

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

114. 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.*