Network Similarity Measures and Automatic Construction of Graph Models using Genetic Programming

Kyle Robert Harrison

Submitted in partial fulfillment
of the requirements for the degree of

Master of Science

Department of Computer Science
Faculty of Mathematics and Science
Brock University
St. Catharines, Ontario

Abstract

A complex network is an abstract representation of an intricate system of interrelated elements where the patterns of connection hold significant meaning. One particular complex network is a social network whereby the vertices represent people and edges denote their daily interactions. Understanding social network dynamics can be vital to the mitigation of disease spread as these networks model the interactions, and thus avenues of spread, between individuals. To better understand complex networks, algorithms which generate graphs exhibiting observed properties of real-world networks, known as graph models, are often constructed. While various efforts to aid with the construction of graph models have been proposed using statistical and probabilistic methods, genetic programming (GP) has only recently been considered. However, determining that a graph model of a complex network accurately describes the target network(s) is not a trivial task as the graph models are often stochastic in nature and the notion of similarity is dependent upon the expected behavior of the network.

This thesis examines a number of well-known network properties to determine which measures best allowed networks generated by different graph models, and thus the models themselves, to be distinguished. A proposed meta-analysis procedure was used to demonstrate how these network measures interact when used together as classifiers to determine network, and thus model, (dis)similarity. The analytical results form the basis of the fitness evaluation for a GP system used to automatically construct graph models for complex networks. The GP-based automatic inference system was used to reproduce existing, well-known graph models as well as a real-world network. Results indicated that the automatically inferred models exemplified functional similarity when compared to their respective target networks. This approach also showed promise when used to infer a model for a mammalian brain network.

Acknowledgments

I would like to specifically acknowledge and thank the following individuals and organizations for their guidance and contributions:

- First and foremost, my supervisor Dr. Beatrice Ombuki-Berman for her continued support and guidance throughout the course of this work
- Dr. Mario Ventresca for his collaborate efforts, especially regarding the analytical study of network measures
- Alexander Bailey for his insightful comments at various points
- Michael Medland, a colleague and collaborator on many aspects
- The members of my supervisory committee for their insightful comments
- The members of my examination committee for taking the time to examine this work
- Brock University and the Government of Ontario for providing funds to support this research
- Lastly, my wife and son who waited patiently while I worked countless hours on my research

K.R.H.

Contents

List of Tables

List of Figures

List of Algorithms

Chapter 1

Introduction

A *complex network*, as defined by Newman [1], is a collection of related elements in which the emergent patterns of connections hold significant meaning. The elements within such networks are the vertices of the network while the relationships or connections among these entities form the edges. While often unmanageably large in size, complex networks are referred to as *complex* due to their intricate and tightly coupled structure and semantics, not their size alone [1]. The study of complex networks is a broad, inter-disciplinary research topic which brings together efforts from computer science, mathematics, biology, the social sciences, and many others. For example, socialization patterns have long been studied in the social sciences, and the resulting networks are referred to as social networks. Social networks [2, 3, 4, 5], dating back as early as the 1930's [6], are used to study how people interact, learn, and as a byproduct, can model the way in which infectious diseases are spread.

Imagine a scenario where a new, highly contagious disease is discovered. Understanding the network dynamics of a social network depicting the daily interactions of individuals, whereby the constituent network elements are people and the connections between them denote their daily iterations, can be vital to the mitigation of the spread of such a disease. By understanding how people interact, preventative measures can be taken to select strategic recipients of vaccines. In fact, recent efforts have used such an approach to propose and evaluate strategies of disease control based on social networks [7]. Further examples of complex networks arise in both natural and artificial contexts. Social networks, for example, are naturally occurring networks in that they emerge as a result of a natural process, human interaction. A further example of naturally occurring networks are biological networks, which aim to describe biological processes such as protein-protein interaction [8, 9], predator-prey relationships, i.e., food-webs [10, 11], and neural networks [12, 13]. Technological networks on the other hand, describe artificially constructed systems such as the Internet [14, 15], power-grid networks [16], and even the spread of computer viruses [17]. Clearly, complex networks arise in all sorts of applications, however, despite their widespread use, complex networks are still not well understood.

By the very nature of their intricate connections, understanding both the structure and dynamics of growth is crucial to understanding complex networks as a whole. However, one of the primary mechanisms used in over 250 years of graph theory [18], visualization and manual inspection of networks [19], is impractical on the scale of networks under consideration. Consider trying to visualize, for example, the human brain which is estimated to have 86.1 billion neurons [20]. Surely, one cannot visualize such a network meaningfully and thus the usefulness of visualization is lost when considering complex networks. Furthermore, analyzing the topological structure alone, as visualization would encompass, is insufficient to claim an understanding of a complex network as this ignores the semantics defining their growth and dynamics [21]. Comprehending a complex network effectively requires one to grasp the reasoning behind the connections, i.e., *how* and *why* the connections formed as such. Devising algorithms which explain the growth patterns of networks has been a topic of interest for over 50 years [22]. Such algorithms are known as *graph models* and are capable of generating networks of arbitrary size which replicate statistical and structural behaviors of networks.

Graph model algorithms have a tremendous number of applications across many domains as they allow both interpolation of previous, captured states of a network and the equally important extrapolation of future states. Recall the example above in which social networks were considered for modeling the spread of diseases. Having a graph model which accurately models the interaction of humans would allow for simulations of its spread to be conducted on synthetic populations. A graph model algorithm could also be used to generate populations of varying sizes and dynamics, allowing the analysis of disease spread to be more rigorous with regards to populations types whereas the use of a single network limits the generality of such a study. Consider, as a further example, the Internet when examining new protocols, i.e., sets of digital rules. Although knowing how a particular protocol will run on the current state of the Internet is necessary, the behavior on future states should also be taken into consideration. With an accurate graph model describing the growth mechanics of the Internet, the future structure can be approximated, facilitating a way to test the protocol on (simulated) future states of the Internet. Furthermore, graph models produce synthetic networks which facilitate a number of benefits over real networks.

Synthetic networks allow for simulations to be performed, possibly even on subsets of the entire network, in a sandbox environment. Similarly, the emergent effects of various parameter alterations can be readily examined using graph models and synthetic networks. Graph model algorithms proposed to model such real-world phenomenon are often the result of years of intensive research and analysis. It is also important to note that the graph model algorithms arising from such specific focus may not be applicable outside of their intended domain as they tend to model only a single property and neglect others [21, 23, 24].

While it would be ideal to have a graph model algorithm tailored to the network being examined, such a situation is prohibited by the difficulty and time-consuming process of constructing graph models, which has typically been done manually. To produce a graph model model, i.e., simulate the growth of a network, one must understand how the network grows. However, graph model algorithms are used to study the growth of complex networks, thus we have arrived at a circular dependency – graph models are a tool used to better understand complex networks but the construction of appropriate graph models requires an understanding of the networks which they are to model. This circular dependency which makes the construction of graph models difficult has lead to a tendency of researchers to reuse existing models which may or may not be fit for their particular application. While reusing existing graph models is beneficial in the sense that previous analytics regarding the model are (typically) readily available, the issue of model and parameter selection still exists. Furthermore, a model selected to fit the current network may be a good fit but there is no guarantee that the growth mechanics will be similar. The selected model may accurately describe the current state of the network but not future states as a result of emergent behaviors attributed to differing growth mechanics between the model and the network of interest.

While statistical methods which generate meta-models, often based on parameter fitting mechanisms, have existed for quite some time [25, 26, 27, 28] and somewhat alleviate the problem of parameterization, only recently has the topic of automatic inference of graph models for complex networks been considered [29, 30, 31]. An automated approach to the construction of graph models would significantly alleviate many of the aforementioned drawbacks of reusing a known model. Ideally, an automated approach would allow for a graph model to be directly tailored to the growth dynamics of the target network, thereby negating the problem of selecting an inappropriate, and thus unfit, existing model. Therefore, the utility value of constructing graph models automatically is undeniable. However, designing and using an automated approach to the inference of graph models is not without its own caveats.

With the above observations, the motivating factors for this research are as follows.

1. Graph model algorithms are crucial to understanding complex networks across a wide variety of applications, but are difficult and time consuming to create. Often, these algorithms are the culmination of many years of study and incorporate a substantial amount of expert knowledge in a specific domain. Facilitating the accurate construction of graph model algorithms will greatly reduce the research efforts needed to adequately model observed phenomenon. Furthermore, automatically generated graph models would, ideally, be tailored to the specific target network and would eliminate the necessity to make trade-off decisions arising from the use of an existing model which may not be an ideal fit.

2. An ideal set of evaluation criteria is unknown and without a study of such criteria, the search process of an automated approach may be misleading. That is, without proper fitness evaluation of candidate models, the results of an automated construction process may be even more unfit than existing models when compared to their intended target. Thus, an analysis of network measures used to both guide an automated approach, in the form of heuristics, and post-validate the models is merited.

3. Many existing graph models, while different in principle, construct networks in a similar way. That is, in one way or another, many construction techniques can be described as follows. For some number of iterations, a new vertex is added and a number of existing vertices are selected for consideration. For each selected vertex, a connection may be formed in some fashion depending on some criteria. Finally, actions which are direct result of forming a connection (or not forming a connection) are performed. This process naturally leads to a definition of a generalized graph model, which can be and, as shown later, is used to aid in their automatic inference.

A well suited candidate for the automatic inference of complex networks is genetic programming (GP) [32] due to its demonstrated ability to build complex, constructive solutions even in poorly understood domains [33]. Genetic programming is a process inspired by Darwinian evolution [34] to automatically generate computer programs through repeated genetic operations in a survival-of-the-fittest evolutionary style. However, the implementation of a genetic programming system for the automatic inference of complex networks is not a trivial task. One major difficulty resides in the fitness evaluation of the evolved models. Since graph models cannot be directly compared (see Chapter 5 for an explanation), they must be analyzed in terms of their generated networks. However, what structural or statistical properties of networks are best used to carry out this procedure? Similarly, how can an evolved graph model be validated given that there may only be one target network for comparison? Designing a fitness function, and thus the heuristic to guide the search process, is thus a major challenge associated with the use of GP. An inappropriate choice of fitness may guide the evolutionary process towards an unfit model, which leads to a further difficulty – what is a good graph model and how can the "goodness" be determined? How can one be reasonably sure that the graph model being used accurately describes the target network? An even more fundamental challenge associated with the use of GP lies in the definition of the language available to the evolutionary process. While GP effectively evolves programs, it must be provided with a set of language elements, such as functions, which can be used to this end. One must provide the GP system with a set of language elements which is sufficiently expressive without being too biased to specific networks yet not too unmanageably large as to complicate the search process.

This thesis provides an analysis of network measures to address the issue of determining model similarity. The results of the analytic study of network measures are then used as a basis of fitness evaluation in a genetic programming system for the automatic construction of graph models for complex networks. The genetic programming system is validating by evolving graph models for four well-known graph models, namely the growing random model [35], the Barabasi-Albert model [35], the Erdos-Reyni model [22], and the Forest-Fire model [36]. Furthermore, the GP system is used to automatically infer a graph model for a real-world brain network of a cat [12]. The results of the GP system are post-validated both by manual deconstruction of their growth algorithms (in the case of the known models) and analytically by a comparison of their structural and statistical properties.

1.1 Contributions

The major contributions of this work can be summarized as follows:

1. This thesis provides an analysis of ten measures of network similarity, and all subsets thereof, for the purposes of quantifying the discriminatory power of each set to distinguish networks generated by different models. Based on the results of this analysis, a set of fitness functions for the GP system to use during the evolution of graph models for complex networks is proposed.

2. A recently proposed, object-oriented linear GP [37, 38] system is used which allows the representation of a candidate graph model to be naturally decomposed into an abstract form. This abstract representation serves as a generalized graph model which expresses the network construction process of a graph in terms of three logically distinct subprocesses.

3. This thesis proposes a GP language which allows recursively defined construction mechanisms to be built. These recursive construction mechanisms allow for the rudimentary generation of community structures, as demonstrated when inferring a model for the Forest-Fire model [36] in Section 8.4. Although not intended to be a focus, this thesis presents the first attempt at evolving community structure within the GP system without the use of an external mechanism. Furthermore, this work presents the evolution of a graph model which generates networks exhibiting community structure for a real-world mammalian brain network. Note that while a graph model for a cortical network has been automatically inferred elsewhere [30], this work differs in that the target network used here is nearly twice as large.

1.2 Thesis Outline

The remainder of this thesis is structured as follows.

Chapter 2 introduces the topic of complex network research. A number of real-world networks are examined, giving an overview of the broad, inter-disciplinary study of networks. The representation of complex networks as graphs is described and a variety of network measures used to quantify the behaviors of networks are presented.

Chapter 3 introduces the topic of graph models and a details a number of existing graph models used to reproduce various properties observed in real-world networks.

Chapter 4 gives an overview of genetic programming and the various components of the traditional algorithm. Various alternative paradigms are discussed, providing justification for the recently proposed GP system used in this work.

Chapter 5 presents results an initial analysis on network measures introduced in Chapter 2. A visualization of the spatial distribution quantifying the dissimilarity between each measure is provided is provided.

Chapter 6 presents a more robust study of centrality measures whereby the network measures are considered not only individually, but each possible subset of measures is examined to determine the discriminatory power of the set. This study provides empirical evidence for the centrality measures selected as fitness functions.

Chapter 7 outlines the experimental procedures for the automatic inference of graph models for complex networks. This chapter presents the fitness functions selected, the GP language employed, as well a description of the generalized abstract graph model which the GP representation used.

Chapters 8 and 9 present the experimental results and a detailed discussion of using GP to automatically infer known graph models and a model for a real-world cortical network, respectively. Observations regarding network measures from Chapters 5 and 6 are incorporated, providing empirical evidence of their merit.

Finally, Chapter 10 offers concluding remarks and summarizes the major contributions and findings of this thesis. Furthermore, limitations of this work and potential areas of future work are highlighted.

Chapter 2

Complex Networks and Network Measures

Complex networks are a valuable tool used to represent and understand intricate systems across many fields. Many complex networks arise naturally, such as social networks as a result human behavior, while artificial networks, such as technological networks, are engineered and have arisen as part of societal development. While the underpinnings of complex network research have been largely rooted in graph theory, the fields have diverged significantly. The study of complex networks focuses on large networks, which limits the usefulness of graph theoretic approaches, many of which are meant for relatively small networks [19]. To give a brief introduction to complex networks and where they have been applied, this chapter gives an overview of four categories of complex networks. Basic definitions of graph representation along with various important properties of graphs are introduced. Finally, this chapter introduces network measures which are used to quantify various characteristics of networks. As later chapters (namely Chapters 5 and 6) will demonstrate, statistical methods applied to the measures described here can be used to quantify the (dis)similarity of networks and, by extension, graph models. While many of the measures described in this chapter can be applied to directed and/or weighted networks, the measures are defined in this chapter with respect to unweighted, undirected networks. Thus, the reader is cautioned that assumptions made in various definitions may only hold for such networks.

2.1 Technological Networks

While many complex systems are a result of natural processes, this is not the case with *technological networks*, as these networks have been engineered as a result of technological growth. Such networks form the underlying infrastructure of modern technological society [1]. Perhaps the most prominent complex network which falls into the category of technological networks is the Internet[1]. The Internet consists of a worldwide network of physical data connections between computers and thus the vertices in such a network represent computers while the edges between them represent physical data connections. The Internet makes for an excellent example of a complex network due to its decentralized nature. No single organization is responsible for determining the physical connections which make up the network, and thus the Internet is considered to be self-organizing [39]. A byproduct of the decentralized, self-organizational structure of the Internet is its robustness to lost connections caused by power loss, equipment failure, traffic overload, etc.. A large number of alternative pathways exist within the Internet to allow pathways which have been lost, whether they be temporary or permanent, to autonomously be bypassed. In fact, about 0.3% of routers (the hardware responsible for routing connections) regularly fail, which due to the scale of the Internet is a significant number, and the nodes which contain the largest number of connections are often subject to attack [40]. Furthermore, it is not uncommon for small bits of information to be lost during transmission. Thus, the Internet must be robust to a reasonable amount of data loss by being able to reconstruct the original message in such a scenario. The robustness of the Internet is a property which many models of the Internet do not exhibit despite its importance [21].

A second well-known example of a technological network, with drastically different structure and behavior, is a telephone network. Note that a telephone network refers to the physical network of telephones, whereas the network of who calls whom would be categorized as a social network (seen later). Telephone networks exhibit a much different connection scheme than most networks as the connections between their nodes are not permanent. Connections in telephone networks are only initiated when needed, i.e., a connection is formed between telephones (the vertices) when a phone call is placed between them. When a call is initiated, the network operators must initiate a connection between the caller and the intended recipient for the call to

[1]Not be confused with the World Wide Web, the network of web pages and hyperlinks – such a network is categorized as an information network (seen later).

take place. Due to this connection scheme, the telephone network can be seen as a fully connected network, as every phone can (in theory) call every other phone, with the caveat that only a fraction of the such connections are present at any given time. Caused by their geographical dependence, telephone networks, and similarly the Internet, exhibit community structure [41] (see Section 2.8 for an overview of community structure), where highly connected niches are formed with relatively few connections between them. Such community structure can be used to form a simplified meta-network representation whereby the communities form the vertices [41].

2.2 Social Networks

Interested in the social interactions of people, Moreno [6] is often credited as the pioneer of the study of networks known today as *social networks*. Despite this credit, antecedent work does exist [42, 43] which is, arguably, the precursor to Moreno's work. An example of the early work by Moreno consisted of the manual generation of social networks, then called *sociograms*, to represent the friendship relationships of schoolchildren. A result that was both interesting and inspiring was made: friendships frequently existed among two boys or two girls, but friendships between a boy and a girl were far less common. While studies involving friendship (or acquaintance) relationships are quite common [2, 44, 45, 46], the study of social networks encompasses many more types of relationships. Further examples of social networks are scientific collaboration networks [47, 48], sexual contact networks [49, 50], interactions of drug users [51], and online communities, e.g. Facebook, [52].

One particular, groundbreaking study of social networks is known as Milgram's "small-world" experiment [45]. Milgram's experiment was quite simple in nature, but resulted in monumental observations regarding human society. Although several variants of the experiment were performed, each variant followed a similar pattern. In one particular variant [53], 296 randomly selected individuals in the state of Nebraska, USA (196 individuals) and the city of Boston (100 individuals), MA, USA, were asked to participate in the study. The purpose of the study was to create an acquaintance chain from each individual to an arbitrarily chosen target person in the state of Massachusetts, USA. Each of the participants was given a document outlining the details of the experiment and was asked to forward the document to another recipient, after affixing their name, with the stipulation that they must know this recipient on a first-name basis. Of the 296 initially distributed documents, 64 reached the target destination with an average of 5.2 intermediate steps along their pathways.

Furthermore, 48% of the chains passed through the same three people before reaching the destination. Thus, the term “small-world”, arising from these experiments, is used to refer to networks which exhibit low average geodesic path lengths.

Data for social networks can be gathered in many different ways. Rapoport and Horvath [54], for example, gathered information about friendships among schoolchildren by providing a questionnaire. Although cumbersome, direct observation can be used to construct social networks. An example of direct observation was Zachary’s [2] karate club network, which was formed by observing the interactions of students within a university karate club. As an equally cumbersome approach, Davis *et al.* [55] used newspaper reports of social events to build a network of affiliations. A final technique introduced here is coined as *snowball sampling*, whereby investigators probe members of the target population. These members then recursively probe others, and a “snowball” effect leads to a reasonable sample of the target population. This technique, while known to cause erroneous conclusions due to the noisy nature [1], is used in hidden populations where standard techniques cannot be effectively applied. For example, in the interaction networks of drug users [51], this sampling technique is used due to the sensitive nature of the behaviors being examined.

2.3 Information Networks

Information networks are, as the name implies, networks of information flow. There is a strong relationship between information networks and social networks as it is completely reasonable, and intuitive, that information can be propagated by means of a social network. Consider a network of email communications. Clearly, such a network depicts a flow of information and could be considered an information network. In a social context, emails are conversations occurring between individuals and thus could be reasonably considered to be social networks. Many such examples of networks overlapping these two categories exist, which causes a fuzzy boundary between social networks information networks in these scenarios [1]. However, not all information networks share such a close relationship with social networks. Consider, for example, the World Wide Web (WWW), a global network of web pages (the vertices) and hyperlinks (the edges) between them. While social networks can reside on top of the WWW, e.g. Facebook, the WWW in and of itself is not a social network but rather a network of information.

Citation networks, i.e., networks which depict the citations among academic papers, are a widely studied type of information network [56, 57, 58, 59] dating as early as 1965 [60]. Typically, an academic paper will reference one or more other papers, which in turn reference other papers, the complete network of which forms a citation network. In such networks, academic papers are the vertices and a (directed) edge is placed between papers when one references the other. Note that by definition, citation networks typically will not have bidirectional edges as it is uncommon to have papers which mutually cite each other. Citation networks often form a power-law distribution where the proportions of citations for papers is extremely non-linear (see Section 3.3 for a brief overview of power-law distributions). According to Newman [1], about 47% of all papers have not been cited at all, while only 9% of the remaining papers have two citations and only 6% have two citations. Furthermore, Newman states that 21% of papers have more than 10 citations and only 1% have greater than 100 citations while one of the most highly cited papers (Lowry *et al.* [61]) has more than 250,000.

2.4 Biological Networks

Biologists have recognized that many biological processes can be modeled using complex networks. *Biological networks* are used as convenient representation whereby the patterns of interaction between various biological process can be analyzed. The most notable applications of biological networks can be loosely grouped into three categories, namely biochemical networks, neural networks, and ecological networks. Biochemical networks are used to represent molecular level processes occurring within cells. One prominent biochemical network is that of the metabolic system, known as a metabolic network. Metabolism is a chemical process by which nutrients are disassembled by cells into building blocks which are then be reassembled to form molecules necessary for various tasks [1]. Reassembling the blocks into usable molecules involves a series of intermediate steps, the complete set of which form a metabolic network.

Neural networks arise from the study of the brains and nervous systems. The main function of the brain is to process information, with its primary unit of processing being known as the neuron. A neuron is a special type of cell which has multiple inputs and generates a single output. To give the scale of neural networks, an adult male brain is estimated to contain roughly 86.1 billion neurons and an additional 84.6 billion non-neural cells [20]. Due to the overwhelmingly large scale of neural networks, it is common to examine them at a higher level by grouping cortical areas into single vertices [12, 13, 62, 63].

The final type of biological networks introduced are ecological networks which model the various interactions between species. Consider a network where individual species are the vertices and the edges represent predator-prey relationships. Networks of this form are referred to as food webs and are typically directed networks, with an edge originating from the prey and ending at the predator. The direction of the edges is a conventional choice as food webs are commonly used to study the flow of energy within ecosystems. When examining a food web network, it not uncommon for a hierarchical community structure to arise [11, 64, 65]. Intuitively, ecological niches will implicitly form community structures in such networks. In a similar fashion to the grouping of neurons in neural networks, food webs are often simplified by grouping related species together [1].

2.5 Representation of Graphs

A graph can be used as an abstraction of a complex network whereby the network representation is reduced to include only information regarding the connection patterns and little else [1]. A graph, G, is defined as $G = (V, E)$, where V is the set of vertices[2] and E is the (mutli-)set of edges[3], with respective sizes denoted by $|V| = n$ and $|E| = m$. An edge within a graph is represented by a pair such that an edge between vertices v and u is denoted as (v, u). An *undirected graph* is a graph in which all edges are assumed to be bidirectional, i.e., the edge pairs are not ordered, whereas a *directed graph* refers to a graph without this assumption, i.e., the edges pairs are ordered. More concretely, for an undirected graph, $(v, u) \in E$ implies that the connection between v and u is reciprocal, i.e., v is connected to u and u is connected to v. However, with directed graphs, $(v, u) \in E$ only implies that v is connected to u, and not necessarily the reverse, although a reciprocal connection can still exist when $\{(v, u), (u, v)\} \subseteq E$. An edge which connects a vertex to itself, i.e., (v, v), is referred to as a *self-edge* or *loop* while edges between two vertices which exist more than once, i.e., when $\{(v, u), (v, u)\} \subseteq E$, are referred to as *multi-edges*. A graph which exhibits neither self-edges nor multi-edges is called a *simple graph* [1]. For the purposes of this thesis, only undirected, simple graphs will be considered.

The *degree* of a vertex, k, is defined as the number of edges connected to it. For example, a vertex v which is connected to two other vertices is said to have degree 2, i.e., $k_v = 2$. When considering directed networks, it is often necessary to distinguish between the in-degree and out-degree of vertices, which correspond to the number

[2]Vertices are also commonly referred to as nodes, sites, and actors.

[3]Edges are also commonly referred to as links, bonds, and ties.

of incoming and outgoing edges, respectively. Because every edge must have two vertices associated with it, a noteworthy relation exists between the number of edges and vertices in a graph given by

$$2m = \sum_{v \in V} k_v \quad . \tag{2.1}$$

Using Equation (2.1), the mean degree of a vertex, $\langle k \rangle$, can be readily obtained as

$$\langle k \rangle = \frac{2m}{n} \quad . \tag{2.2}$$

2.5.1 Paths

A *path* within a network refers to a sequence of vertices, $P = (v_1, v_2, ..., v_l)$, such that for each consecutive pair of vertices $(v_i, v_{i+1}) \in P$ there is an edge from v_i to v_{i+1}. The *length* of a path is then defined as the number of edges traversed or, equivalently, $|P| - 1$. While the general definition of a path allows the revisiting of vertices, a *self-avoiding path* explicitly disallows such duplicated vertices and edges. A *geodesic path* is a path of shortest length between two vertices. By definition, a geodesic path must be self-avoiding as it is trivial to see that any non-avoiding path can be shortened by removing the duplicated vertices/edges. While the geodesic paths themselves need not be unique, the length of such paths between vertices v and u, denoted d_{vu}, is always well-defined. Vertices which are not connected to the remainder of the graph are conventionally assigned an infinite geodesic path length [1].

2.5.2 Components

A *component* is a subset of vertices such that a path exists between every pair of vertices within the subset. More concretely, a component is a subset of vertices such that there is at least one path from each vertex to each other vertex within the subset and that no other vertex within the graph can be added to the subset while preserving this property [1]. Consider Figure 2.1, which depicts two distinct components, and note that no path exists between vertices in the top component and vertices within the bottom component. Thus, adding any vertex from the bottom component to the subset of vertices in the top component would violate this property as no path would exist to this vertex from the other vertices. A network in which there is only a single component, i.e., there exists a path from every vertex to every other vertex, is said to be *connected* while a network in which there exists multiple components, i.e., there

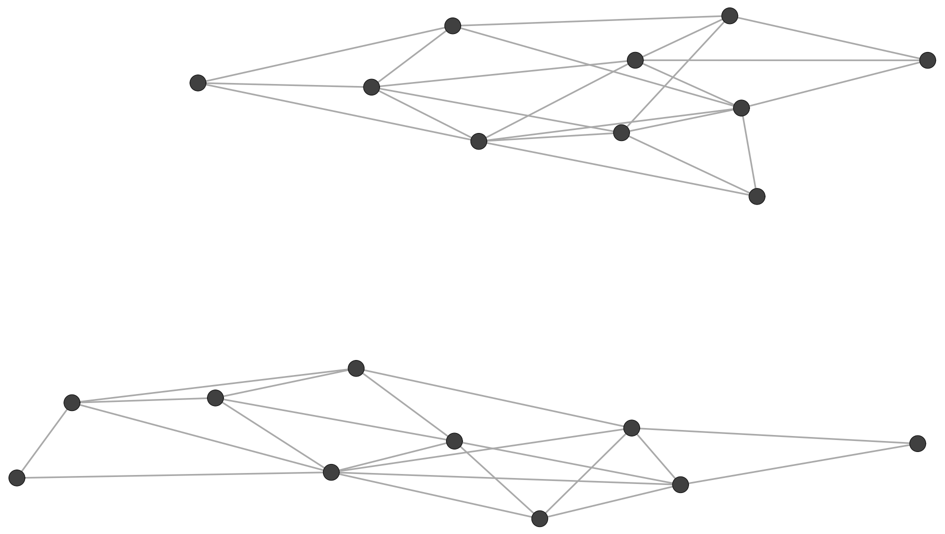


Figure 2.1: Graph with two components.

exist at minimum two vertices in which no path exists between them, is said to be *disconnected*.

2.6 Global Network Measures

In this section, four measures which assign a single value to the entire network are presented. Such measures are considered for their ease of comparison. The information provided by them, however, is somewhat limited as much of the intrinsic local properties of individual vertices is disregarded. Nonetheless, networks which function in a similar fashion should be expected to have similar values for te following measures.

2.6.1 Measures Based on Path Length

The longest geodesic path length in a network is referred to as the *network diameter*. The *eccentricity* of a vertex is defined as the shortest path from that vertex to the farthest other vertex in the graph, with the shortest such eccentricity referred to as the *network radius*. Both the network radius and network diameter can be computed in $\mathcal{O}(nm)$ time.

An alternative measure of network diameter, *effective diameter*, is defined by finding the geodesic path length, d, such that 90% of geodesic path lengths are shorter than d [36]. Note that since geodesic path lengths are natural numbers for unweighted networks, proportions are linearly interpolated between successive geodesic path lengths. Thus, the effective diameter can be thought of as the point in which the linearly interpolated empirical distribution function is 0.9.

The average geodesic path length of a vertex v within a network containing n vertices, denoted l_v, is defined as

$$l_v = \frac{1}{n} \sum_{u \in V} d_{vu} \ . \tag{2.3}$$

Extending this definition across an entire network, the *average geodesic path length* of a network, denoted by l, is defined as the mean of all such geodesic path lengths and is given as

$$l = \frac{1}{n} \sum_{v \in V} l_v \ . \tag{2.4}$$

Care must be taken as this definition (loosely) assumes that the network consists of a single connected component. Note that by convention, vertices with no path between them (i.e., they reside in different components) have an infinite geodesic path length [1]. Thus, when calculating the average geodesic path length, it is common to only include paths with non-infinite length [1]. The average geodesic path length can be computed in $\mathcal{O}(nm)$ time.

2.6.2 Network Transitivity

Transitivity, an important property especially in social networks, is similar in principle to the transitivity property of mathematical operators. To illustrate the notion of transitivity, consider the mathematical operator "=". In mathematics it is trivial to see, using the colloquial definition of equality, that if $a = b$ and $b = c$, it follows that $a = c$. Such a relation is said to be *transitive*, that is, the relation between a and b, and b and c implies a relationship between a and c.

The notion of transitivity arises in a network where it is the case that if vertex u is connected to vertex v and vertex v is connected to vertex w, then vertex u is also connected to vertex w, i.e., a triangular structure is formed by three vertices. The proportion of connections which form such transitive relations are a useful measure of network structure [1], known as the *network transitivity* or *global clustering coefficient*. The transitivity is defined [1], informally, as

$$C = \frac{(\text{number of triangles}) \times 3}{(\text{number of connected triples})} \tag{2.5}$$

or, alternatively

$$C = \frac{(\text{number of triangles}) \times 6}{(\text{number of paths of length two})} \tag{2.6}$$

where a connected triple refers to a triangle as above, with the caveat that only two of the three edges need be present[4]. The factors of 3 and 6, respectively, account for the redundant counting of the triangles with respect to the denominator measure. That is, when counting the number of connected triples, each triangle accounts for three such triples, while each triangle also accounts for six paths of length two. The global transitivity can be calculated in $\mathcal{O}(m\langle k\rangle^2)$ time.

2.7 Measures of Centrality

A large amount of research effort has been devoted to the concept of *centrality*, which stems from the idea that some vertices in a network are more important than others. This concept follows naturally from the inherent semantics of networks. Centrality refers to how central, or "important", a vertex is within a network. The importance of a vertex is, however, subjectively based on the perception of "importance". As such, many definitions of importance, and corresponding measures of centrality, have been proposed. This section introduces a number of such measures of centrality.

2.7.1 Degree

The *degree* of a vertex, while quite possibly the simplest and most crude, leads to a rather intuitive definition of importance. It is completely reasonable to assume that a vertex which is connected to a large number of other vertices will have a large influence on the network and, as such, is an important vertex within the network. For example, in a social network, a highly influential person is typically one which has a large number of connections thereby facilitating information to be disseminated over a larger group of people. Similarly, in a citation network, a paper which has a large number of citations is typically a very influential paper. As the degree of each vertex can be computed in $\mathcal{O}(1)$ time, the degree distribution of the network can be calculated in $\mathcal{O}(n)$ time.

[4]The third edge may or may not be present, but only two need be present to be considered a connected triple.

2.7.2 Betweenness Centrality

The *betweenness centrality* [66] of a vertex measures the extent to which it lies within the shortest paths between vertices given by

$$x_v = \sum_{st} \frac{n_{st}^v}{g_{st}} \tag{2.7}$$

where n_{st}^v is the number of geodesic paths between s and t which pass through v and g_{st} is the total number of geodesic paths from s to t. Vertices which have a high betweenness can have considerable influence on the network as they possess a sense of control over the flow of information within the network [1]. The betweenness centrality differs from many other measures of centrality as it is not directly related to the connectedness of vertices, but rather the extent to which the vertex is between other vertices. The betweenness centrality can be calculated in $\mathcal{O}(nm)$ time.

2.7.3 Closeness Centrality

The average geodesic path length of a vertex, l_v, given in Equation (2.3) can be seen as a measure of centrality. However, the average geodesic path length would assign unintuitive centrality scores in the sense that vertices which are more central, i.e., closer to other vertices, are assigned lower scores. Therefore, it is common to measure a vertices' centrality as the inverse of l_v. This inverse measure is known as the *closeness centrality* [67], denoted C_v, and is calculated as

$$C_v = \frac{1}{l_v} = \frac{n}{\sum\limits_{u \in V} d_{vu}} \ . \tag{2.8}$$

While the closeness centrality closely resembles a natural definition of importance, it suffers from a major drawback – values within the measure have very little variance. That is, the range of values in the closeness centrality tends to be small due to the fact that geodesic paths typically increase logarithmically with network size [1]. This small range of values makes it difficult to distinguish the centrality of vertices. Furthermore, issues similar to those discussed in Section 2.6.1 arise when dealing with networks that have more than one component. The closeness centrality can be calculated in $\mathcal{O}(nm)$ time.

2.7.4 Local Transitivity

Following from the definition of the network transitivity in Section 2.6.2, a ratio of transitivity can also be defined for each vertex in a network. The *local transitivity*, or *local clustering coefficient*, is given by[5] [1]

$$CC_v = \frac{\text{(number of pairs of neighbors of } v \text{ that are connected)}}{\text{(number of pairs of neighbors of } v\text{)}} \quad . \tag{2.9}$$

The numerator of Equation (2.9) is found by counting the number of distinct pairs of the neighbors of vertex v that are connected. This quantity is divided by the number of total number of pairs of neighbors, which can be alternatively given as $\frac{k_v(k_v-1)}{2}$. Similar to the network transitivity, the local transitivity can be computed in order $\mathcal{O}(m\langle k \rangle^2)$ time.

The local transitivity has been empirically found to have a rough, inverse correlation with degree [1]. Furthermore, the local transitivity is an indicator of “structural holes” [68], which refers to the missing edges which, if present, would form a transitive relation. Such holes in the structure can cause degradations in the passing of information through a network because of the lack of alternative pathways [68].

2.7.5 Eigenvector Centrality

Bonacich [69] proposed a measure of vertex importance based on the relative importance of its connections. That is, the *eigenvector centrality* score of a vertex is based the centrality score of its neighbors with the justification that being connected to a lower number of highly influential nodes in a network is fundamentally different than being connected to a higher number of less influential nodes. In such a case, one can argue that the node connected to fewer, highly influential nodes has more impact on the network as whole.

The eigenvector centrality scores are calculated by solving the eigenvector equation

$$Ax = \lambda x \tag{2.10}$$

where A is the adjacency matrix of the graph. While there can be many eigenvalues for which a solution to the above equation exists, the eigenvector corresponding to the greatest eigenvalue is the desired vector of centrality scores. Note that the ith component of this eigenvector is the centrality score of the ith vertex. The eigenvector

[5]While the notation C_v is commonly used for the local transitivity, CC_v is used here to avoid confusion with the closeness centrality.

centrality can be computed in $\mathcal{O}(n)$ time.

2.7.6 PageRank

Brin and Page proposed PageRank [70] as a measure of a websites importance based on the simulated behavior of a typical user of the web. The authors claim the PageRank corresponded well with people's subjective view of importance [70]. PageRank was initially used by the Google search engine as a method of ranking each web link based on the link's relative probability of being visited. The PageRank algorithm works on the analogy that a user will browse the web by following a random walk of web pages, but also has a probability of randomly jumping to a new page. Thus, PageRank corresponds to a random walk with occasional jumps. At each step of the random walk, the probability of continuing is given by the damping factor, d, and the probability of a random jump is thus $(1 - d)$. When the damping factor is 0, PageRank is directly proportional to the degree, while a damping factor of 1 corresponds to every step being a random jump, and thus the PageRank of every vertex will be $\frac{1}{n}$ [71].

The PageRank of a vertex is recursively defined by the PageRank of its predecessors, leading naturally to an iterative calculation process give by

$$PR_v = \frac{1-d}{n} + d\left(\sum_{u \in P(v)} \frac{PR_u}{|S(u)|}\right) \tag{2.11}$$

where d is the damping factor, $P(v)$ is the set of predecessors of vertex v (i.e., vertices withe edges that point to v), and $|S(u)|$ refers to the number of successors of vertex u (i.e., vertices which are pointed to by u). Note that the importance of the predecessors has a significant influence on the PageRank of a vertex, meaning that vertices with a lower number of "important" connections may have a larger PageRank score than vertices with more, less "important" connections. The PageRank measure can be computed in $\mathcal{O}(nm)$ time.

Figure 2.2 demonstrates the PageRank scores on a small network. The percentages on each vertex reflect the likelihood of reaching a node using a damping factor of 85% (or 0.85), i.e., the PageRank scores are expressed as percentages. A number of noteworthy properties are demonstrated by this image. First, node C has fewer connections than node E, but receives a higher score. This demonstrates that not all connections are treated equally when calculating PageRank: the incoming connection for node C is deemed more influential than all of the incoming connections of E

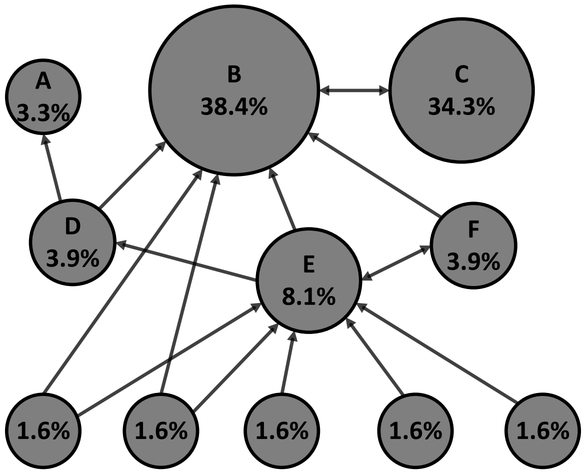


Figure 2.2: PageRank scores expressed as percentages.

combined. The necessity for a damping factor is also demonstrated. The damping factor allows nodes with no incoming connections, i.e., vertices which no random walk could reach (unless they were the starting point), to be reached as a result of the random jumps.

2.8 Community Structure

Often it is beneficial to divide a network into groups or clusters, whereby these clusters have many inter-cluster edges and few edges between clusters. These clusters are commonly referred to as *communities* and members of a community typically share a closer relationship with other members of the same community than with members from different communities. In a social network context, a community commonly represents an organizational structure, or hierarchy, such as a department within a university. Similarly, communities tend to also represent geographical regions as it is more common to form friendships with individuals within close proximity. Communities within a web network may indicate similar content while a metabolic network might have different functional units form distinct communities [1]. An example demonstrating a network with its community structure highlighted can be found in Figure 2.3.

The separation of a network into its respective communities is referred to as a *community detection* algorithm. Note that a similar technique, known as *graph partitioning*, also separates graphs into clusters, however, with graph partitioning the number of clusters to form is known in advance. Roughly speaking, community detection is the process of dividing a network into distinct subdivisions, i.e., commu-

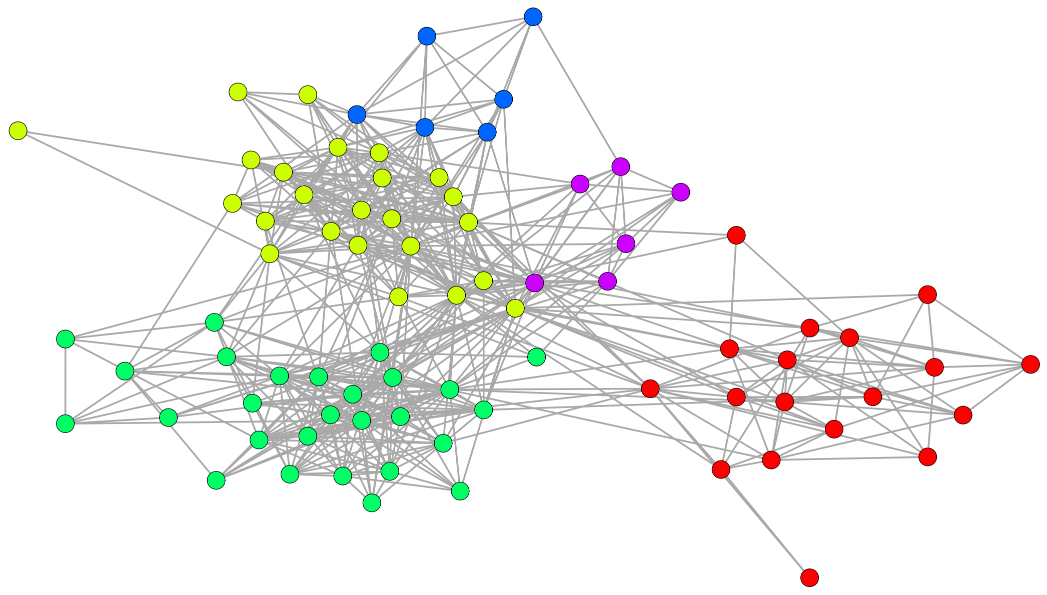


Figure 2.3: The community structure of a friendship network within a UK university faculty [3]. Each color represents a community detected by a modularity optimizing method.

nities, such that there are many edges within communities and few edges between communities. However, the definitions of “many” and “few” are subjective and have lead to the proposal of a wide variety of community detection algorithms based on various definitions [1].

A measure of how logically distinctly a community structure is separated from the remainder of the vertices can be devised based on the density of edges between communities, relative to the expected edge density in a null model with an equal distribution of vertex degrees [1]. This measures is known as *modularity* and assigns positive values to networks where there are more edges between vertices of the same type, i.e., communities, than one can expect by random chance and, conversely, assigns negative values if there are less such edges than expected. Note that optimizing the modularity is, by virtue, a method of detecting communities. However, finding the optimal modularity takes exponential time and is thus infeasible in general [1]. One common approach to community detection uses spectral properties of the network to form a modularity matrix, of which an eigenanalysis is performed to subdivide the network into communities based on the signs of the leading eigenvector [72]. This algorithm, known as the *leading eigenvector* community detection algorithm, is used later in this work to determine the number of communities within networks.

Chapter 3

Graph Models

A *graph model* is an algorithm which, as a result of execution, produces a graph that exhibits properties of interest but is otherwise random [1]. Graph models are crucial to the understanding of complex network behavior. Although complex networks, by definition, have an intricate structure, many of these structural properties can be replicated to some extent using very basic construction rules. Graph models are typically stochastic or probabilistic algorithms which, through repeated execution, are capable of producing a set of graphs that exhibit similarity in some fashion. The similarity of the resulting graphs is dependent upon the context in which the model was designed. That is, graphs produced by a model designed to replicate degree distributions will exhibit similar degree distributions, while models meant to replicate path lengths will generate graphs with similar path length distributions. However, the degree distribution of a graph from the former will most likely differ from the degree distribution of a graph produced by the latter, and vice versa when considering path lengths. Note that a graph model is not expected to reproduce any specific graph, i.e., a graph model is not meant to generate *isomorphic* graphs, but rather to reproduce a family of graphs which behave in a similar fashion with respect to certain properties. Many graph models have been proposed to model various phenomenon not present in random networks [35, 36, 73, 74]. This chapter introduces a variety of graph models which, although some are quite simplistic in principle, are able to model a wide variety of structural properties observed in real-world networks. A further, in-depth overview of various graph models can be found in [1], [75], and [76].

3.1 Erdos-Reyni Random Graph Model

Consider a graph model in which the number of vertices, n, and the (independent) probability of an edge existing between each pair of vertices is given by p. Often referred to as the Erdos-Reyni (ER) random graph model, such a model was first studied by Solomonoff and Rapoport [77] in 1951, but is commonly associated with Erdos and Reyni [22] due to their celebrated work involving the model. The Erdos-Reyni model is quite possibly the simplest yet most widely studied graph model [1]. More importantly, comparing the output of the Erdos-Reyni model to real-world networks lead to the realization that complex networks are not truly random in nature and that using a random graph model is problematic [1, 73]. The general algorithm, as popularized by Erdos and Reyni, is given in Algorithm 1 and denoted as the ER model. In Algorithm 1, a new vertex, v, is added each iteration and each existing vertex, u, is considered for connection with an independent probability, p.

Algorithm 1 Erdos-Renyi Model

```
Require: $0 \leq p \leq 1$
  function ER($n, p$)
      for $i \leftarrow 1$ to $n$ do
          $v \leftarrow g$.AddVertex( )
          for $j \leftarrow 1$ to $i$ do
              if RandomDouble() $< p$ then
                  $u \leftarrow g$.GetVertex($j$)        // Vertex at index $j$
                  $g$.AddEdge($v, u$)
              end if
          end for
      end for
  return $g$
  end function
```

The ER model, while useful as a null-model for comparison, does not serve well as a network model [1]. One clear shortcoming of random graphs is the lack of transitivity, which tends toward zero as the network size grows. Another problem lies in the emergent patterns with regards to the degrees of vertices, which are unlike those observed in real world networks. Since the edges are randomly assigned, there is no correlation between the degrees of adjacent vertices, which prevents the emergence of community structures. Furthermore, many real world networks exhibit right-tailed degree distributions while random graphs exhibit Poisson distributions. In the following section, a graph model proposed to exhibit the transitivity of real-world networks is described.

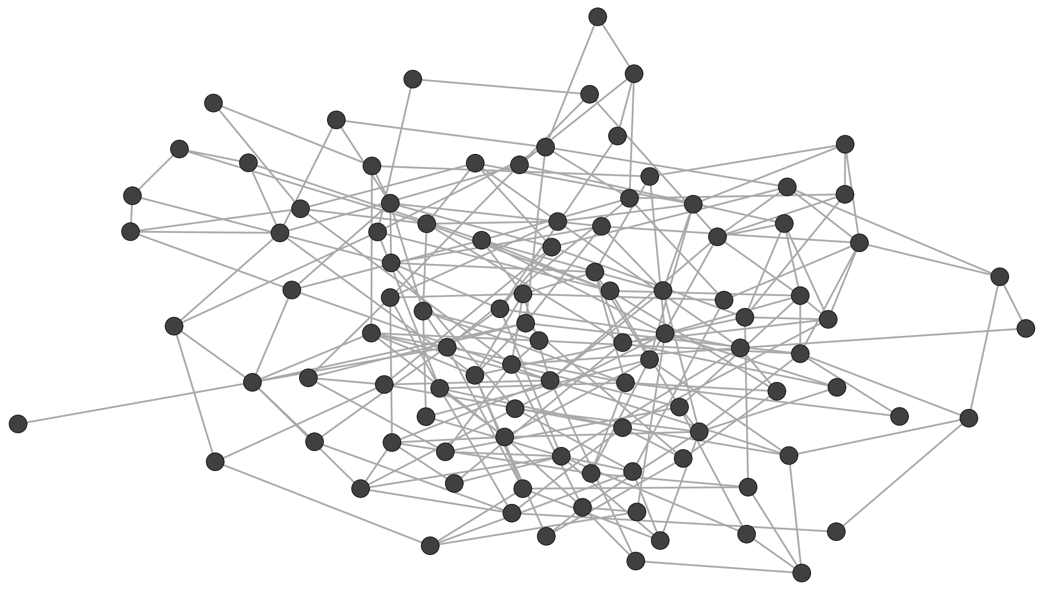


Figure 3.1: Graph constructed using the ER model with 100 vertices ($p = 0.05$).

3.2 Watts-Strogatz Small World Model

To alleviate the shortcomings of the random graph model with regards to the lack of transitivity, Watts and Strogatz [73] proposed a model which exhibits a "small world" [45] effect. Watts and Strogatz noted that many naturally occurring networks were neither random nor regular, but rather fell somewhere between these two extremes and often exhibited high clustering coefficients coupled with low average geodesic path lengths [73]. Networks which exhibit such behaviors are classified as "small-world" networks. The Watts-Strogatz (WS) model, shown in Algorithm 2, constructs a network by first creating a ring structure where each vertex is connected to its m_0 nearest neighbors. Each of the initial connections are then rewired according to probability p. By varying the rewire probability, the randomness of the network can be controlled; when $p = 0$, an entirely regular network is constructed while setting $p = 1$ generates a completely random network as every edge in the network is randomly rewired.

While many real world networks, such as the neural network of the *C. elegans* worm, the US power grid, and a collaborative network of film actors, have been shown to exhibit a small-world effect [73], the WS model is known to generate degree distributions which are unlike those of real-world networks [1]. However, this result is not surprising as the WS model was proposed solely to replicate the transitivity and average path lengths of networks. The following section outlines a model which was proposed to replicate real-world degree distributions.

Algorithm 2 Watts-Strogatz Small World Model

```
Require: $0 \le p \le 1$
  function WS($n, m_0, p$)
    $g \leftarrow$ RingGraph($n, m_0$)    // Ring graph with n vertices connected to $m_0$ neighbors
    for 1 to $|E|$ do
      $e \leftarrow g$.GetEdge($i$)    // Get the $i$th edge
      if RandomDouble( ) $< p$ then
        $e$.Rewire( )
      end if
    end for
  return $g$
  end function
```

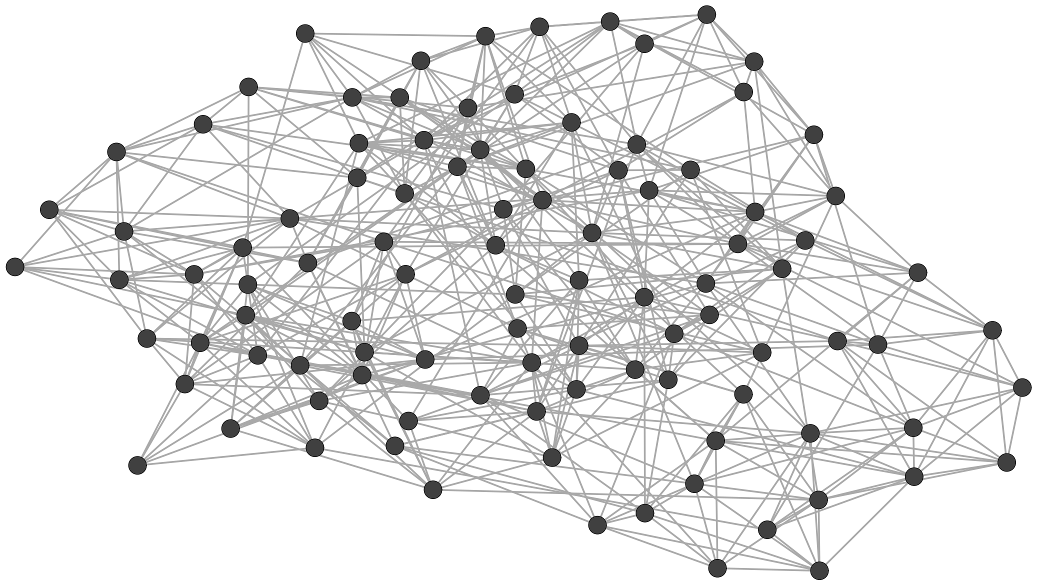


Figure 3.2: Graph constructed using the WS model with 100 vertices ($m_0 = 5$, $p = 0.15$).

3.3 Barabasi-Albert Model

The Barabasi-Albert (BA) [35] model is shown in Algorithm 3, where a new vertex v is added each iteration and connects to m preferentially selected vertices, according to Equation (3.1). The BA model was premised on two key properties of real-world networks. The first property, *growth*, being that network sizes are rarely static – they grow over time. The second, arguably more important, property was that highly connected vertices have a higher probability of attaining new connections, a phenomenon known as *preferential attachment*. The preferential attachment defines a sort of "rich get richer" effect. The modeling of these two properties leads to the two defining mechanisms of the BA model:

1. *Growth:* Starting with m_0 initial vertices, create $m < m_0$ new edges at each time step. This is demonstrated in Algorithm 3 by a new vertex, v, being added each iteration.

2. *Preferential Attachment:* The probability of connecting to a vertex u is dependent upon the (in-)degree of u. An additional factor, known as the *zero degree appeal*, is added to enable vertices with no connections to have a non-zero probability of attaining a new connection. The preferential attachment mechanism is outlined in Equation (3.1), giving the calculation of a vertices relative probability of attaining a new connection. In Algorithm 3, the *SelectVertex* function implements the preferential attachment mechanism according to Equation (3.1).

$$P(v \to u) \sim InDegree(u)^{\alpha} + z \tag{3.1}$$

Algorithm 3 Barabasi-Albert Model

```
function BA($g, n, m, \alpha, z$)        // An initial graph, $g$, is provided
    for 1 to $n$ do
        $v \leftarrow g$.AddVertex( )
        for 1 to $m$ do
            $u \leftarrow$ SelectVertex($\alpha, z$)        // See Equation (3.1)
            $g$.AddEdge($v, u$)
        end for
    end for
return $g$
end function
```

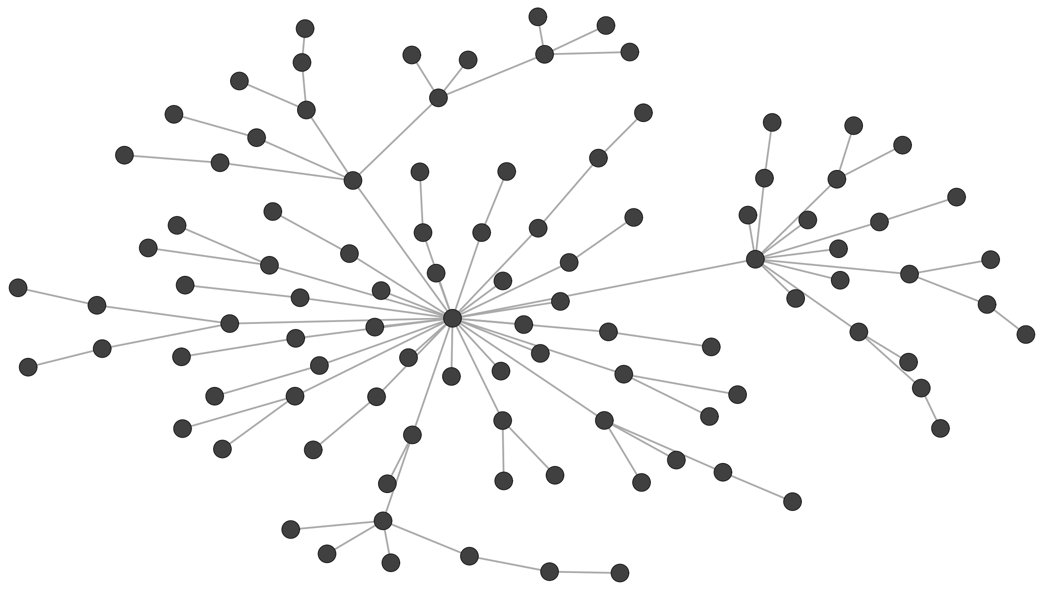


Figure 3.3: Graph constructed using the BA model with 100 vertices ($m = 1$, $m_0 = 0$, $\alpha = 1$).

When $\alpha = 1$, the BA model exhibits a particular type of degree distribution, governed by a power law of the form

$$p_k = Ck^{-\alpha} \tag{3.2}$$

where k is a particular degree, p_k denotes the probability of a vertex having degree k, and α[1], C are constants. Networks exhibiting such power-law distributions are sometimes referred to as *scale-free* networks and are of particular interest due to their tendency to arise in a variety of networks [1].

3.4 Growing Random Model

The growing random (GR) model was a null-model used by Barabasi and Albert [35] to demonstrate that the emergent, scale-free behavior in their proposed model must be attributed to the preferential attachment mechanism and not random chance. As such, the GR model, demonstrated in Algorithm 4, is simply a BA model with the removal of the preferential attachment selection mechanism. As Algorithm 4 depicts, the GR model simply adds a new vertex, v, each iteration and randomly selects m vertices to which v will form a connection.

[1]This is not to be confused with the α parameter of the BA model.

Algorithm 4 Growing Random Model

```
function GR($g, n, m$)
    for 1 to $n$ do
        $v \leftarrow g$.AddVertex( )
        for 1 to $m$ do
            $u \leftarrow g$.RandomVertex( )
            $g$.AddEdge($v, u$)
        end for
    end for
return $g$
end function
```

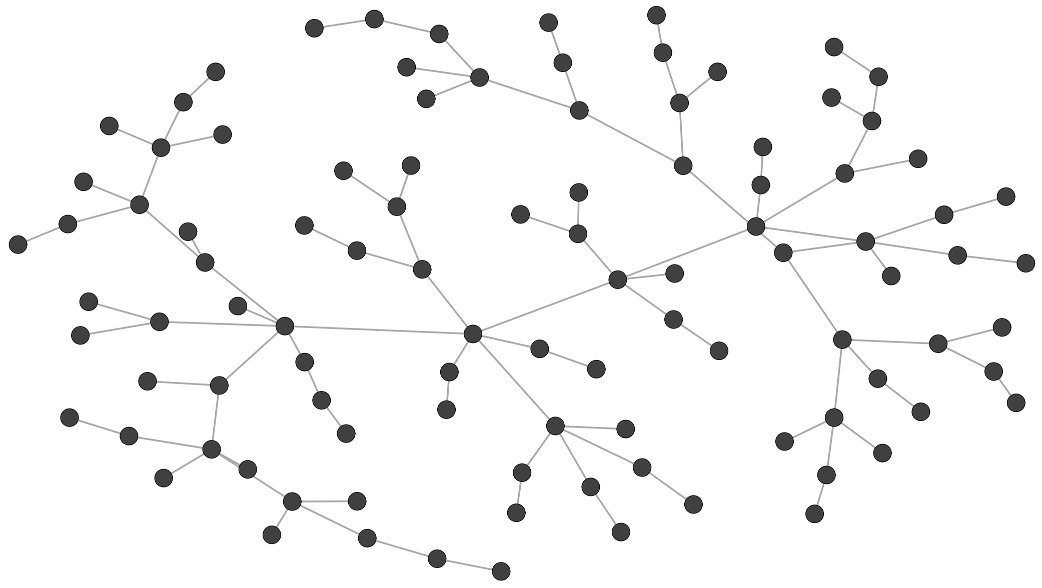


Figure 3.4: Graph constructed using the GR model with 100 vertices ($m = 1$, $m_0 = 0$).

3.5 Aging Preferential Attachment Model

Dorogovtsev and Mendes [74] proposed an extension to the BA model, the Aging Preferential Attachment (APA) model, whereby the preferential attachment mechanism is influenced by the age of vertices. However, the realization of the effect of vertex age on the selection process had been previously noted by Price [60]. Price, working on scientific citation networks, observed a phenomenon which he labeled the "immediacy factor" of academic papers. This observation noted that recent papers were more likely to attain new citations, while older papers were considered obsolete and, as such, were much less likely to attain new citations. Thus, the probability of connecting to a vertex u is proportional to both the (in-)degree and age of u, as shown in Equation (3.3). Furthermore, it is common to define a number of bins for the age – the age used in Equation (3.3) then becomes the binned of the vertex. The APA model is presented in Algorithm 5. Note that the algorithm for the APA model builds a network in the exact same manner as the BA model with the exception of the influence of vertex age in the preferential attachment mechanism. That is, a new vertex v is added each iteration and connects to m preferentially selected vertices, according to Equation (3.3).

$$P(v \to u) \sim (\text{InDegree}(u)^{\alpha} + z) \times (\text{Age}(u)^{\beta} + y) \tag{3.3}$$

Algorithm 5 Aging Preferential Attachment Model

```
function APA($g, n, m, \alpha, z, \beta, y$)
    for 1 to $n$ do
        $v \leftarrow g$.AddVertex( )
        for 1 to $m$ do
            $u \leftarrow$ SelectVertex($\alpha, z, \beta, y$)        // See Equation (3.3)
            $g$.AddEdge($v, u$)
        end for
    end for
return $g$
end function
```

3.6 Forest Fire Model

Proposed by Leskovec *et al.* [36], the Forest Fire (FF) model generates networks based on the spread patterns of forest fires. The model aims to incorporate the following

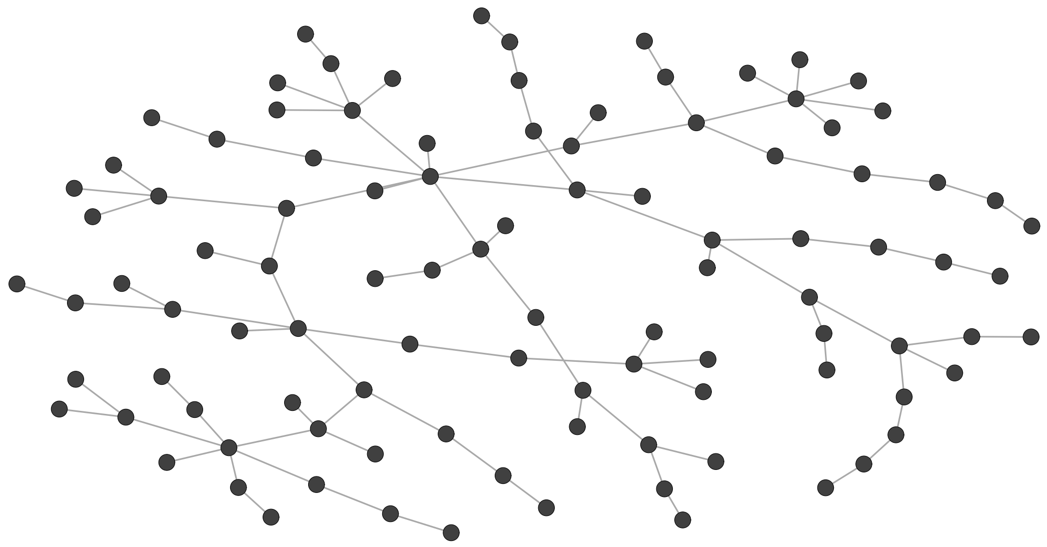


Figure 3.5: Graph constructed using the APA model with 100 vertices ($m = 1$, $\alpha = 1$, $z = 1$, β = -1, $y = 0$, bins = 25).

properties observed in real-world networks: 1) heavy-tailed degree distributions, 2) community structures, 3) power-law densification – the network becomes more dense over time, according to a power-law, and 4) shrinking (effective) diameter (see Section 2.6.1).

Assuming p and r are the forward and backward-burning probabilities, respectively, and v is a vertex joining the graph at time $t > 1$, a graph is generated as follows:

1. v chooses m ambassador nodes at random and creates edges to each
2. For each ambassador node, w
   (a) Generate two random numbers, x and y, from geometric distributions with means $\frac{p}{1-p}$ and $\frac{1}{rp(1-rp)}$, respectively
   (b) v selects x outgoing edges and y incoming edges incident to vertices which are not yet visited, denoted $w_1, w_2, ..., w_{x+y}$
      - If there are not enough vertices to select, v selects the maximal number possible
   (c) v forms outgoing edges to $w_1, w_2, ..., w_x$ and recursively applies steps 2 to each

Algorithm 6 presents the general algorithm for generating a graph according to the forest fire model. Thia algorithm uses a queue data structure to simulate the recursive burning procedure where the number of successors and predecessors which are connected to and subsequently added to the “burning” queue are selected according to geometric distributions given by ρ_p and ρ_r.

Algorithm 6 Forest Fire Model

```
function FF($n, m, p, r$)
    $g \leftarrow$ EmptyGraph( )
    $\rho_p \leftarrow$ GeometricDistribution($\frac{p}{1-p}$)
    $\rho_r \leftarrow$ GeometricDistribution($\frac{1}{rp(1-rp)}$)
    for 1 to $n$ do
        $v \leftarrow g$.AddVertex( )
        $Q \leftarrow$ Queue( )
        for 1 to $m$ do
            $w \leftarrow g$.RandomVertex( )
            $Q$.Enqueue($w$)
        end for
        while Q.Count $> 0$ do        // Recursive burning procedure
            $u \leftarrow Q$.Dequeue( )
            for 1 to $\rho_p$.Sample() do        // Forward burning
                $s \leftarrow u$.RandomSuccessor( )
                $g$.AddEdge($v, s$)
                $Q$.Enqueue($s$)
            end for
            for 1 to $\rho_r$.Sample() do        // Backward burning
                $s \leftarrow u$.RandomPredecessor( )
                $g$.AddEdge($v, s$)
                $Q$.Enqueue($s$)
            end for
        end while
    end for
return $g$
end function
```

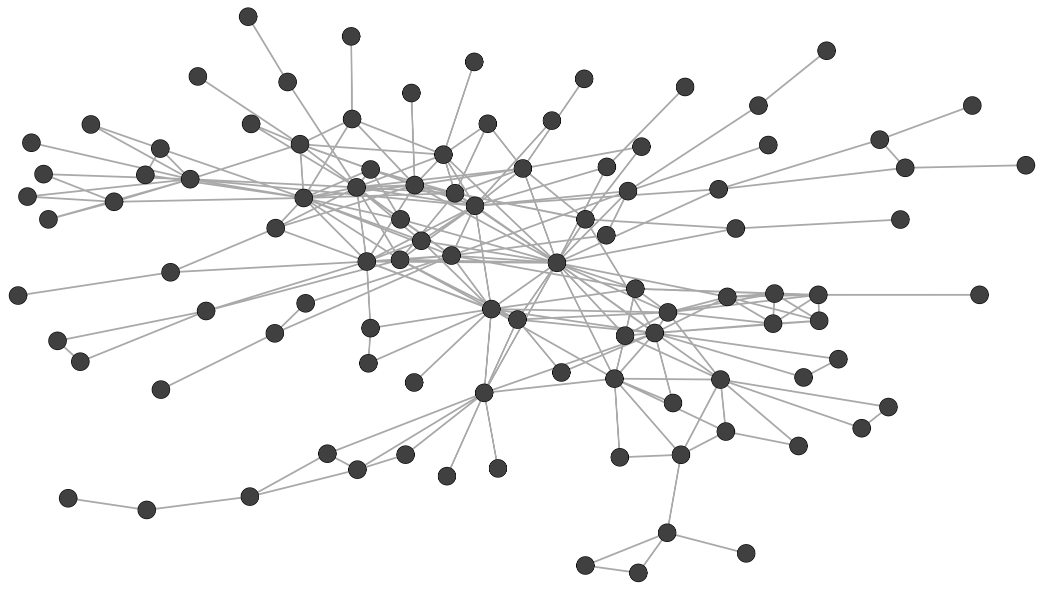


Figure 3.6: Graph constructed using the FF model with 100 vertices (m = 1, p = 0.37, r = $\frac{0.32}{0.37}$).

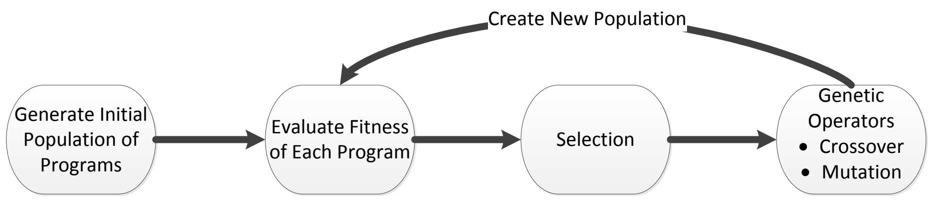
Chapter 4

Genetic Programming

Genetic programming (GP) [32] is an artificial intelligence paradigm which uses the concept of Darwinian evolution [34] to automatically synthesize computer programs. In genetic programming, a population of candidate programs, called *chromosomes*, are evolved over some number of generations with the intention of producing a fit program [33]. Inspired by natural evolution, the population of chromosomes undergo a repeated process of fitness-based selection and application of genetic operators, as visualized in Figure 4.1. Fitness of an individual in GP is typically determined by a user-supplied procedure, thereby allowing the user to provide a heuristic which guides the search process. The selection of this fitness function is vital to the performance of the optimizer as an inappropriate fitness can hinder the search process. The remainder of this chapter provides further information about the traditional genetic programming paradigm as well as various alternative paradigms proposed to alleviate the various drawbacks associated with traditional GP.

4.1 Representation

Genetic programming, as originally popularized by Koza [32], used a tree-based representation analogous to a parse tree. Each element of the tree consists of some provided



Create New Population
Generate Initial Population of Programs
Evaluate Fitness of Each Program
Selection
Genetic Operators
• Crossover
• Mutation

Figure 4.1: Visualization of the genetic programming optimization procedure.

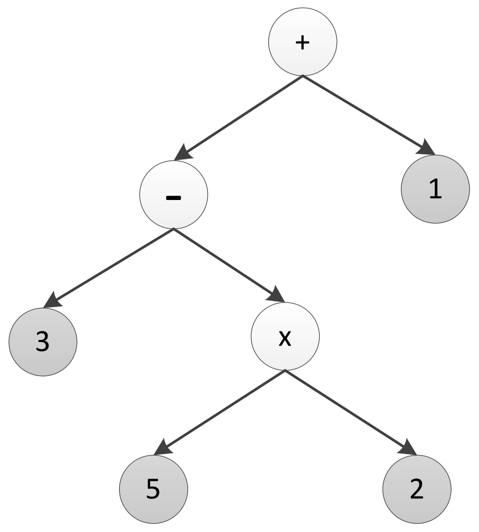


Figure 4.2: A chromosome in tree-based GP. The lighter elements are *functions* while the darker elements are *terminals*.

primitive; the collective set of such primitives is known as the *language*. The language primitives are typically categorized into two distinct sets based on their arity. Language elements which have non-zero arity, and typically perform data manipulation operations, are known as *functions*. In contrast, a *terminal* refers to a language element which is nullary, that is, it takes no arguments. Terminals often take the form of constants or randomly generated values. To construct a program from such a tree structure, child nodes are recursively applied as arguments to the parent node, starting from the root. Figure 4.2 gives an example GP tree representing the expression $(3-(5\times 2))+1$. Note that by definition, leaf nodes must be terminals while non-leaf nodes must necessarily be functions.

To begin the optimization process, a population of candidate programs, and thus program trees, must be generated. A straightforward approach to construct a tree is to select a root node and then recursively select additional nodes for each argument in a predefined pattern, up to a specified *depth*. The *depth* of a node n refers to the number of edges required to reach n from the root node. Note that by this definition, the root node has a depth of 0. The depth of a tree is similarly defined as the depth of the deepest leaf node. The maximum allowable depth of a GP tree is typically provided as a user parameter. Although various techniques exist for creation of initial trees, the two most common approaches are the *full* and *grow* methods [33]. The *full* method, as the name implies, attempts to create trees which are full, meaning the depth of each leaf node is the maximum allowable depth. This is achieved by recursively selecting from only the function set at each step until the depth limit is reached, at which point terminals are selected. In contrast, the *grow* method allows for a wider variety of shapes and sizes. At each stage of the construction process, the grow method selects from either the functions or terminals, subject to the maximum depth. Thus, the grow method can construct trees which have nodes that are not at the maximum depth.

Because the shapes of both construction methods are somewhat limited [32], a procedure known as *ramped half-and-half* initialization is often used. The ramped half and half procedure allows the user to define a range for the initial depth of trees, whereby the construction mechanism "ramps" the maximum depth of the generated trees along this range and constructs both grow and full trees. This procedure allows for a much wider variety of sizes and shapes to be constructed in the initial population [33].

Initially, the language elements used by GP systems were typeless, i.e., all elements of a program had the same type. Strongly-typed GP [78] introduced the ability to represent more sophisticated programs by facilitating the construction of trees with more than one type. That is, each language element, including the arguments of functions, have a type associated with them and the tree structure must adhere to the compatibility of language elements based on their types. Thus, in a strongly typed GP system, the set of suitable language elements available for selection at each stage of the tree creation is context-sensitive.

4.2 Fitness-Based Selection

The genetic operators are applied to individuals which have been selection probabilistically according to their fitness. This allows for a competitive evolution scheme whereby more fit individuals are given a higher chance at having their genetic material preserved through offspring. Therefore, the fitness evaluation of individuals plays an important role in the search behavior. The *selection pressure* of a selection operation describes the degree to which the fitness is used to select individuals. Operators which exhibit strong selection pressure tend to create a skewed probability distribution whereby individuals with preferable fitnesses are given a much higher probability of selection while a weaker selection pressure provides a more even probability distribution for all individuals [33]. The two most common selection operators in, which demonstrate drastically different selection pressures, are *tournament* and *roulette* selection.

The *tournament* selection operator is premised, as the name implies, by a tournament style competition for selection. A number of individuals, denoted k, are randomly chosen to participate in the "tournament", the best of these k individuals is taken as the selected individual. Such an operator is implicitly inherent to noise due to the random selection, however, tournament selection effectively rescales the fitness to maintain a constant selection pressure throughout the run [33]. Tournament selection, due to the preferable selection pressure, is the selection operator employed in this work.

4.3 Genetic Operators

Crossover is the process of recombining the genetic material of two or more chromosomes to produce offspring. The selection of the chromosomes to be recombined during crossover, known as the *parents*, is done probabilistically based on the fitness of the chromosomes. This effectively allows more fit individuals to receive a higher pressure for selection. The increased selection pressure causes a bias towards more fit programs which, ideally, will create offspring with better fitness. Crossover on tree based chromosomes is done by swapping randomly selected subtrees from the parents adhering to the type constraints. The crossover procedure can be visualized in Figure 4.3.

While the crossover operator allows for large, drastic movements through the search space early in the run, it can only recombine genetic material which is already present in the population. Thus, if a language element which is necessary to construct a correct solution does not exist in any individual in the population, GP with only crossover will never find this correct solution. To facilitate the integration of new genetic material, *mutation* is also performed during the evolutionary process. The typical mutation operator is defined by randomly replacing a subtree of an individual with another, randomly generating subtree. This allows for new genetic material to be introduced and typically causes smaller, less drastic changes than crossover.

4.3.1 Replication

While crossover and mutation are used to produce new candidate chromosomes, sometimes it is beneficial to directly copy chromosomes to the next generation. As such, some GP implementations make use of a *replication* operator which simply copies a selected individual to the new population without altering its genetic material. A common variant of replication, whereby some proportion of the most fit individuals

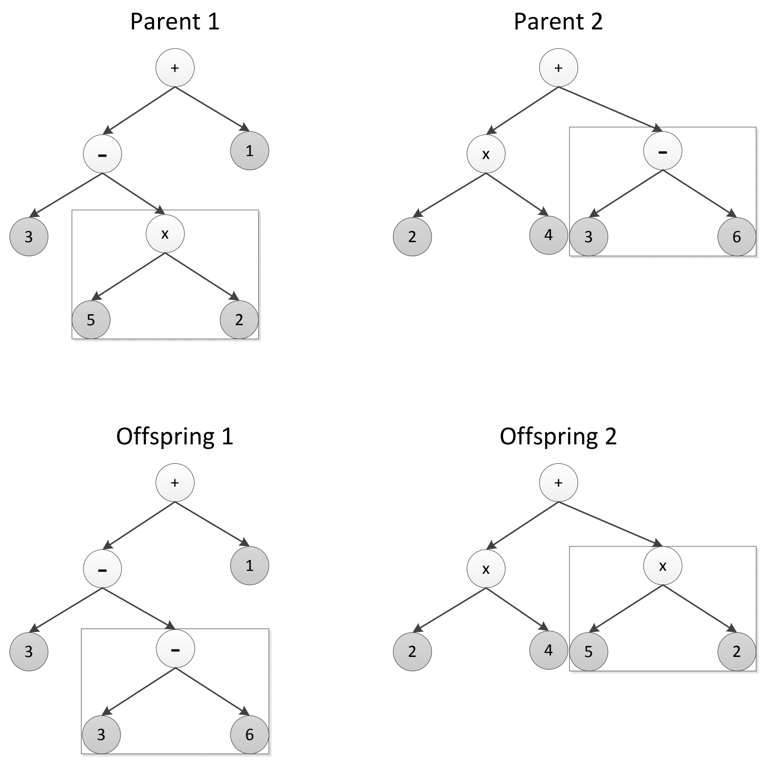


Figure 4.3: Crossover operation on tree-based chromosomes. The selected subtrees (shown in boxes) are swapped between the parents resulting in two offspring.

are copied to the new population, is known as *elitism*. Elitism allows the genetic material of the best individuals to be preserved, and thus, prevents the fitness of the best solution from degrading.

4.4 Alternate Genetic Programming Paradigms

GP has been shown to produce fit programs in a wide variety of different applications such as symbolic regression [79, 80], circuit production [81], evolutionary art [82], modeling [29, 30, 81, 83], and the construction of artificial agents [84, 85], among others. Likewise, there have been a variety of methodologies introduced to extend and enhance the capabilities of GP [86, 87]. This section presents an overview of the most commonly used, non-traditional GP paradigms.

4.4.1 Linear Genetic Programming

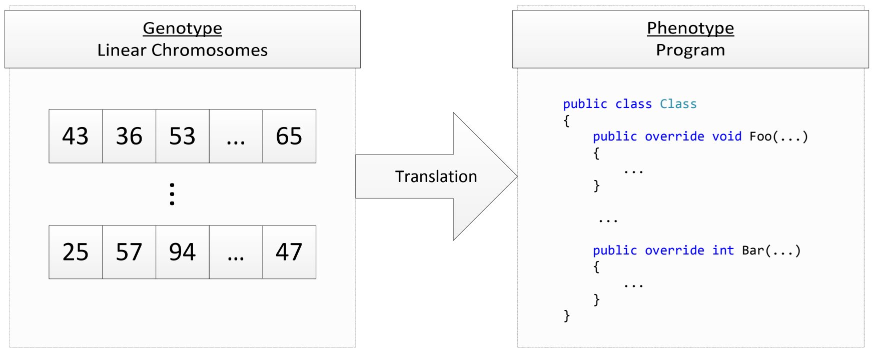
Programs in linear GP are represented as a linear array of instructions [88], rather than a tree structure. The linear representation, analogous to a sequence of instructions, ultimately produces an imperative-styled program. Furthermore, the linear representation creates a clear distinction between genotype and phenotype [89]. Many techniques have been proposed for linear GP [90], each with their own advantages and disadvantages. However, an important advantage of linear GP over traditional GP is the ability to vary the destructiveness of crossover and mutation [87]. Furthermore, the linear representation allows traditional crossover and mutation operators defined for genetic algorithms to be employed.

4.4.2 Object-Oriented Genetic Programming

Object-oriented programming is a widely used programming paradigm where types are represented as entities with behaviors. In the object-oriented paradigm, instances of types, known as objects, are mutable by way of methods which are analogous to behaviors. The object-oriented genetic programming (OOGP) methodology [86] provides a framework to produce programs in such a paradigm. OOGP makes use of a multi-tree representation, whereby a chromosome consisting of several trees, each corresponding to a single method, is evolved. This multi-tree representation used by OOGP allows the simultaneous optimization of multiple methods, leading to the evolution of more complex programs.

4.4.3 Linear Object-Oriented GP

The structure of individuals in linear object-oriented GP [37] is inspired by both linear GP and OOGP representations – individuals are constructed using a collection of linear chromosomes. Each of these chromosomes directly translates to a method in the resulting object, as seen in Figure 4.4. As such, this paradigm combines the benefits of the linear and OOGP paradigms by allowing the simultaneous optimization of multiple, imperative style methods. Furthermore, this GP paradigm allows the use of partially-implemented classes, i.e., abstract classes, as a basis whereby the user may incorporate their domain knowledge of the problem to provide the implementation of methods where the desired functionality is known *a priori*. The GP system used in this work is of this sort and further implementation details of the system, including the crossover and mutation, are provided in Chapter 7.



```
public class Class
{
    public override void Foo(...)
    {
        ...
    }

    ...

    public override int Bar(...)
    {
        ...
    }
}
```

Figure 4.4: Visualization of the genotype to phenotype translation in linear object-oriented GP where a linear array of integers is translated to form a fully functional object.

Chapter 5

Analysis of Network Properties

A natural issue to be addressed in the context of automatic inference of graph models is how to evaluate the evolved models. Graph models cannot be directly compared for two reasons: 1) it is well known in computability theory that determining if algorithms produce the same output is impossible [91]; and 2) in real-world scenarios there will not be a target model for comparison. Although replicating the underlying target model exactly would be ideal, this would require knowledge of the target model, therefore making the process of automated inference unnecessary. As Figure 5.1 demonstrates, the evaluation of graph models must be done by comparing the graphs generated by each of the models.

Comparison of generated graphs allows insight about the respective models to be made. However, Fan *et al.* [21] demonstrated that using only topological characteristics to evaluate complex network models can be misleading. Their study demonstrated this claim by showing instances where using topological measures to evaluate mod-

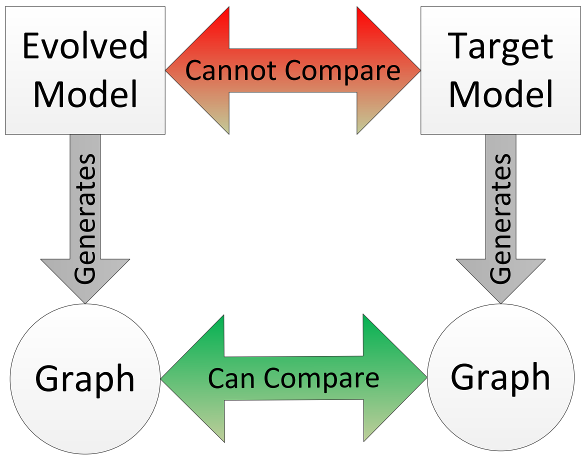


Figure 5.1: Graph model comparison process.

els of the Internet could lead to false conclusions – the examined models were able to closely match the topological characteristics but were unable to replicate crucial emergent properties of the network. Thus, one must be aware that an inferred model may not be ideal with respect to all network properties, but rather only those used to evaluate its similarity. As such, this chapter focuses on the analysis of network properties and how they can be used to determine similarity.

Graph Models and Parameters

The six graph models examined, as well as the parameters used (see Chapter 3), are given below.

BA	Barabasi-Albert (m = 1, α = 1)
APA	Aging Preferential Attachment (m = 1, α = 1, z = 1, β = -1, y = 0, bins = 25)
GR	Growing Random (m = 1)
FF	Forest Fire (m = 1, p = 0.37, r = $\frac{0.32}{0.37}$)
ER	Erdos-Renyi (p = 0.05)
WS	Watts-Strogatz / Small World (m = 5, p = 0.15)

5.1 Global Network Properties

This section examines global network properties, i.e., measures which assign a single value to an entire network. Global properties are useful to quantify the overall structure and behavior of the network, such as the average geodesic path length which provides a sense of the information propagation time, but are limited in that they generally disregard the emergent local behaviors of individual vertices.

5.1.1 Average Geodesic Path Length

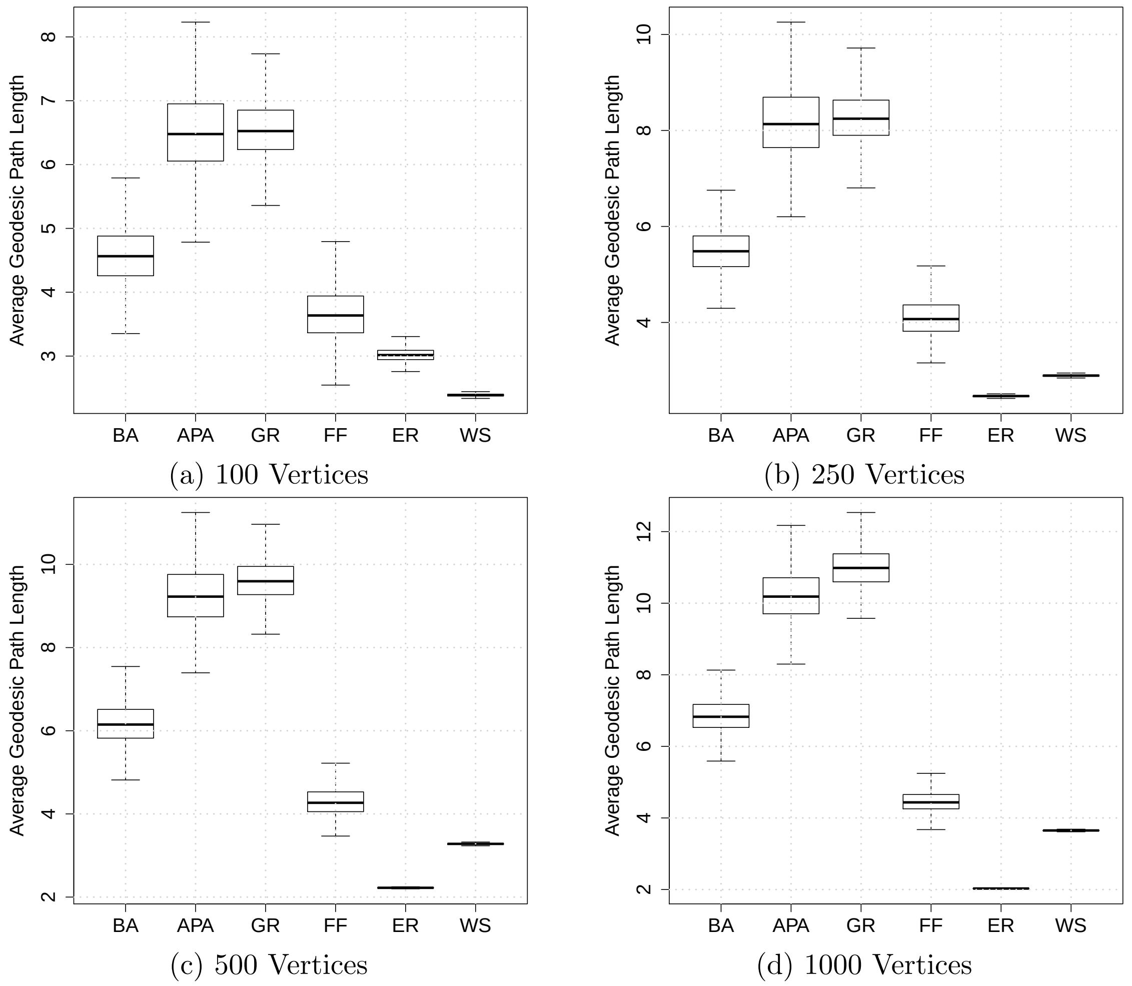
Figure 5.2 shows the average geodesic path length of each model using 1000 generated graphs of varying sizes. An immediate observation with 100 vertex networks was the amount of overlap between graphs from different models. For example, these plots indicate that an average geodesic path length of 5.5 could be generated by any of the BA, APA, or GR models. An even more extreme example lies in the APA and GR models as the entire range of values for the GR graphs was enveloped by those from the APA model, while both models showed nearly identical median values. However, the APA model demonstrated a larger variance, allowing both higher and lower path lengths than those of the GR model to be generated.

As network sizes increased, all networks observed an increase in average geodesic path length with the exception of the ER model, which demonstrated shrinking AGP as network size increased. Both the FF and WS models demonstrated quite slow growth of the AGP relative to the network size, with the FF model showing the slower growth of the two. Neither of the FF or WS results are unexpected as the models were designed to have a shrinking diameter and the small-world property, respectively, both of which suggest slow growth of path lengths. Examining Figure 5.2d showed that the expected median of GR graphs grew quicker than that of APA networks. Furthermore, the range of observed values for the GR model was no longer completely enveloped by the APA model – at the 1000 vertex level, the GR model was able to generate networks with a higher AGP than the APA model. Similarly, the observed average geodesic path lengths for the FF model grew significantly quicker than the ER model as the network size increased – even at the 250 vertex level, the ER model was consistently producing significantly lower geodesic path length values than the FF model.

The relative difference between the average geodesic path lengths was visualized using MDS on 30 networks generated at various network sizes, shown in Figure A.1 (see Appendix A). The MDS procedure demonstrated that the relative difference between the average geodesic path lengths from different models was quite small, especially with 100 vertex networks. It is interesting to note that at all network sizes, there were instances of the FF model which were more similar to WS networks than other instances of the FF model. This same observation can be made for the BA, APA, and GR networks, although the BA model becomes dissimilar more rapidly relative to the APA and GR networks as the network sizes increase – there is no visible overlap at the 500 and 1000 network sizes. Thus, the average geodesic path length of a network is clearly insufficient to determine dissimilarity between smaller networks, especially the preferential attachment networks. However, as these networks grew, their emergent behaviors became more evident and their respective path lengths began to diverge.

5.1.2 Network Transitivity

Figure 5.3 shows the network transitivity ratio of each model using 1000 generated graphs of varying sizes. It should be noted here that the preferential attachment



(a) 100 Vertices

(b) 250 Vertices

(c) 500 Vertices

(d) 1000 Vertices

Figure 5.2: Boxplots of the average geodesic path length.

networks (BA, APA, GR) could not generate triangles[1], thus their network transitivity was always 0. An interesting immediate observation was made with the ER model – the transitivity of the ER model was centered around 0.5 (note that 0.05 was the probability of an edge existing between any two vertices) for all network sizes, with a decreasing variance as the network size increased. At the 100 vertex level, shown in Figure 5.3a, the observed median for the WS model was below the expected range for the FF model. However, as Figure 5.3d depicts, the transitivity of the FF and WS models becomes extremely similar for 1000 vertex networks with the entire range of the WS model enveloped by the FF model. As expected, both the FF and WS models showed significantly higher network transitivities than the ER model, while both models showed shrinking network transitivities as the network size increased.

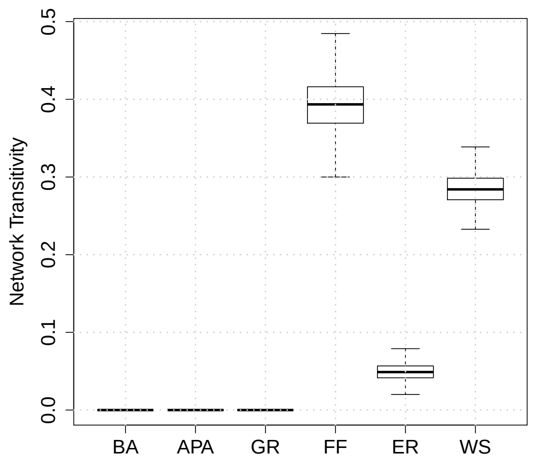
The above observations are further visualized by the application of MDS on network transitivities of 30 networks generated at various network sizes, as depicted in Figure A.2 (see Appendix A). The BA, APA, and GR networks all occupy a single point as their differences were all 0. At all network sizes, the ER model was more similar to the zero transitivity of the BA, APA, and GR models than it is to the FF and WS models. However, as the network sizes increased, the network transitivities of the FF and WS model both decreased which effectively shrunk the relative difference between the ER and FF/WS models.

From these observations, it can be concluded that the network transitivity is not an effective measure to quantify dissimilarity for the examined networks, especially on the larger networks. The first reason being that three of the six models, by definition, had no transitivity. For the same reason, this can be argued as both a good and bad measure to use for these networks. That is, one can be assured that any graph with a non-zero transitivity was not generated by the BA, APA, or GR models, however, instances of these networks can never be distinguished with respect to the network transitivity. Furthermore, the transitivity of the FF and WS models became quite similar with 500 vertex network and nearly indistinguishable with 1000 vertex networks even though the generating mechanisms used by these models were vastly different.

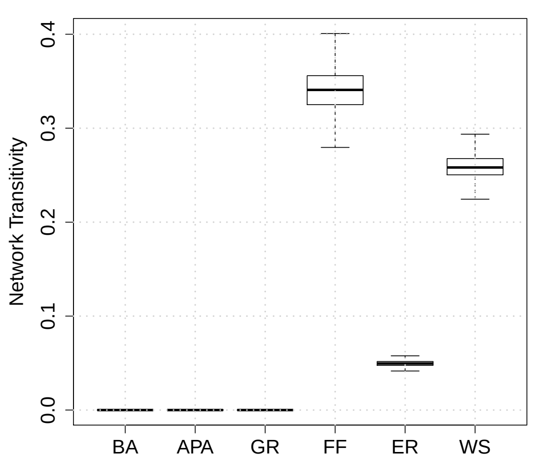
5.1.3 Network Diameter

Figure 5.4 shows the network diameter of each model using 1000 generated graphs of varying sizes. Immediately, the network diameter stands out as being much more

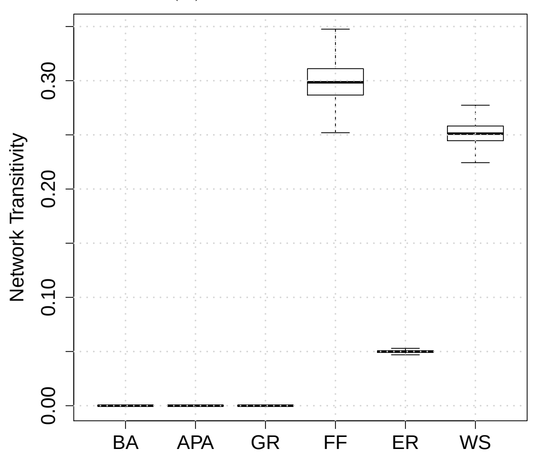
[1]This is only true for the parameters used in this study. In general, each of these models can produce non-zero network transitivities.



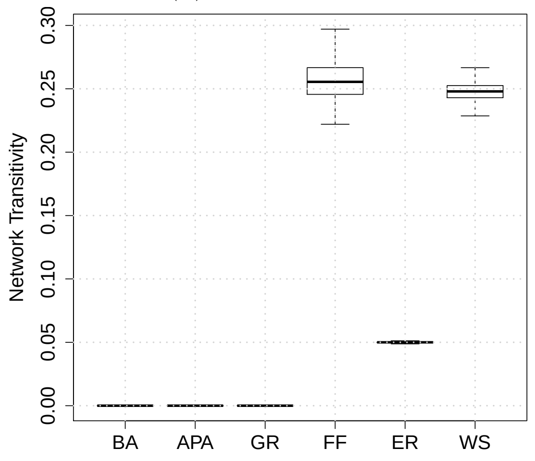
(a) 100 Vertices



(b) 250 Vertices



(c) 500 Vertices



(d) 1000 Vertices

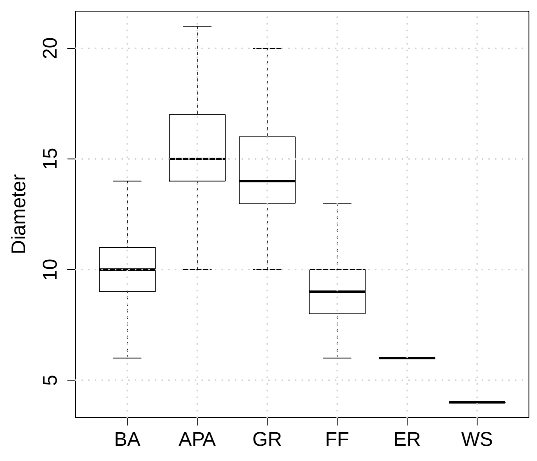
Figure 5.3: Boxplots of network transitivity.

similar between models than the previously examined measures. A large part of this was due to the discrete nature of the diameter[2]. The network diameter is not a continuous measure, thus the range of values which it can take is severely limited. As such, we see that many networks can produce networks with the same diameter. To illustrate this point, a 100 vertex network with a diameter of 13 could have been reasonably produced, albeit not expected in the average case, by any of the BA, APA, GR, or FF models.

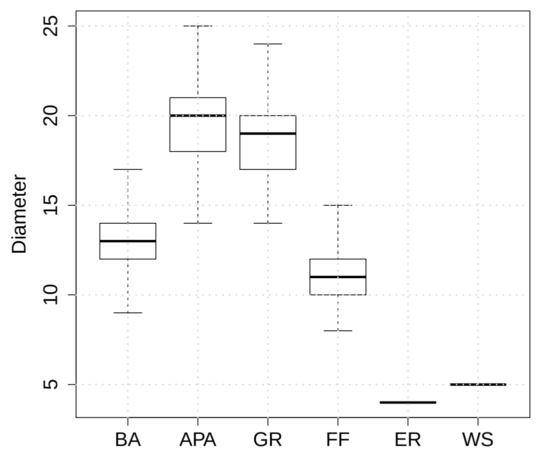
As observed with the AGP measure, the APA and GR models were quite similar with respect to the network diameter. With all network sizes, the range of values for the GR model were completely enveloped by the APA model. Less similar than the GR and APA models but more unexpectedly, the BA and FF models demonstrated similar network diameters. However, the BA model demonstrated far quicker growth with respect to the network diameter compared to the FF model. When examining the 1000 vertex networks, it is seen that the FF model would consistently produce networks with smaller diameters than the BA model. The reader is reminded here that the network diameter of the FF was increasing with network size, even the FF model was proposed to have shrinking diameters, due to an alternative definition of diameter, referred to as the effective diameter, used by Leskovec *et al.* [36]. The ER model demonstrated the smallest diameters for all network sizes, with the exception of the 100 vertex networks, where the WS networks had significantly smaller diameters. For all network sizes, the ER and WS models produced significantly shorter diameters than the other networks. However, this is not unexpected as both the ER and WS models had relatively small average geodesic path lengths.

Presented in Figure A.3 (see Appendix A) is the MDS procedure applied to the network diameter of 30 networks generated at various network sizes. These plots further reinforce the observation of the high degree of overlap between the diameters for networks from different models. For each of the four network sizes, there existed points which indicated no difference between instances of the BA and FF models, and the BA, APA, GR models (with the exception of 1000 vertex networks, where the BA did not overlap with either model). In contrast, the ER and WS models demonstrated almost no overlap with any other model for any size networks, indicating that these models consistently generated networks with distinct diameters relative to the other models. Due to the discrete nature coupled with the high degree of overlap, the diameter is concluded to be an extremely inefficient measure to quantify the both the dissimilarity between networks generated by different models as well as the similarity

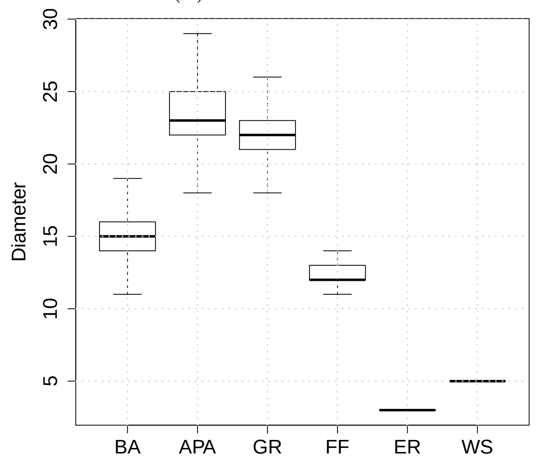
[2]This is only true, in general, for unweighted networks.



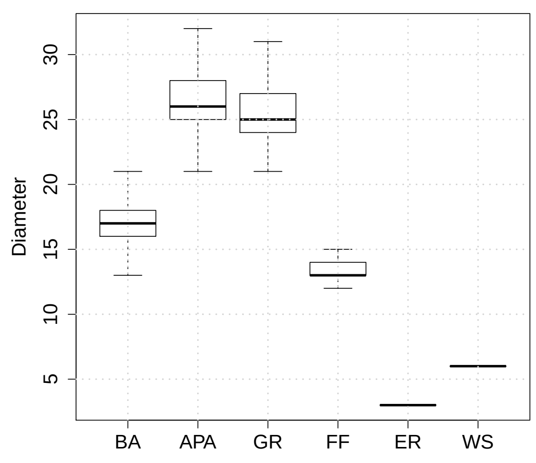
(a) 100 Vertices



(b) 250 Vertices



(c) 500 Vertices



(d) 1000 Vertices

Figure 5.4: Boxplots of network diameter.

between networks generated by the same model.

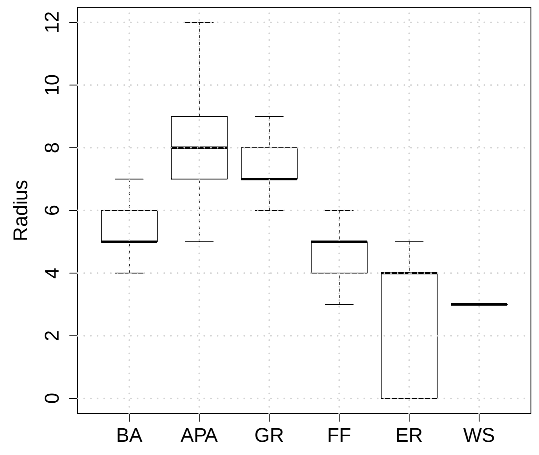
5.1.4 Network Radius

Figure 5.5 shows the network radius of each model using 1000 generated graphs of varying sizes. As with the network diameter, values of the network radius are discrete, leading to a high degree of overlap. Due to the related nature of the measures, the results obtained using network radius were quite similar to those observed with network diameter. Notably, the entire range of values for the GR model were enveloped by the range for the APA model, with the interesting exception of the 1000 vertex networks, where the two models produced nearly identical distributions of network radii. As with the diameter, the radii of the ER and WS were the smallest of all network models, with the ER model showing the smaller radii in general. However, on 100 vertex networks, as seen in Figure 5.5a, the ER had quite a large range of possible radii, leading to a higher median that the WS model. While the BA and FF models were quite similar with respect to the radius of the generated instances, the BA model demonstrated slightly larger values than the FF for all network sizes. Furthermore, most of the range of values for FF networks were within the expected range for BA networks on all sizes. For all network sizes, the boxplots for the WS model shown in Figure 5.5 were all straight lines, indicating that little to no variance was observed in the networks generated by this model. A similar observation was made for the ER model at network sizes of 250, 500, and 1000 vertices.

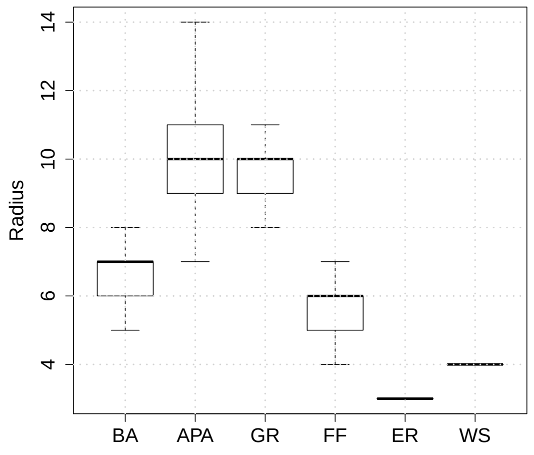
Presented in Figure A.4 (see Appendix A) is the MDS procedure applied to the network radius of 30 networks generated at various network sizes. Again, these results appeared quite similar to those obtained when the network diameter was examined. However, one striking difference was in the number of distinct points visible – the radius had far fewer such points. This suggested that the variance among instances from all models was less than when the diameter was considered, i.e., there were less values observed for the radius than for the diameter. Furthermore, the reduced variance among values observed with radius lead to the degree of overlap between graphs generated by different models to be higher. Thus, the MDS procedure produced a higher number of points situated directly on top of other points. In all cases, the APA model had the largest number of distinct points on the MDS plot, with at least six such points in every plot. The same argument against the similarity measuring potential made for diameter can be made for radius; the low variance and high degree of overlap leads to many cases of networks from different models being indistinguishable with respect to network radius. However, the radius is arguably a worse measure than the diameter due to the smaller range of observed values.

5.2 Vertex Centrality Measures

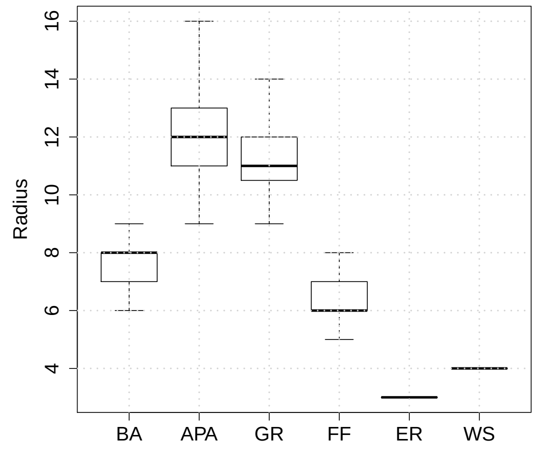
In contrast to the measures examined in the previous section, this section examines vertex centrality measures. While single measures for a network are beneficial for reducing the problem of similarity to a comparison of singular values, they are limited by the same nature. Assigning a single value to an entire network will, to a certain extent, disregard the local properties of the constituent vertices in the network. With some networks, the local properties of the vertices are crucial to the functionality of the network. Fan *et al.* [21] demonstrated that models which best replicated topological characteristics of the Internet, such as transitivity and distances, were



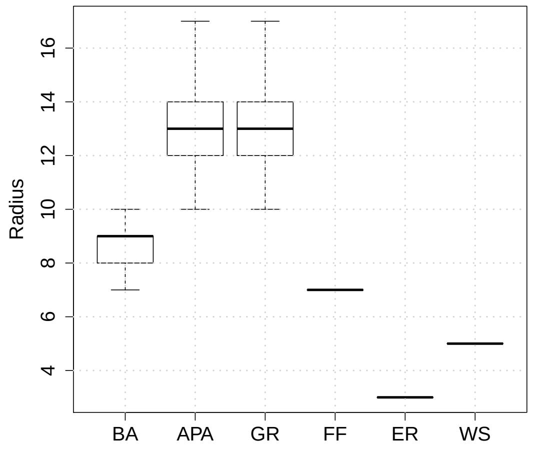
(a) 100 Vertices



(b) 250 Vertices



(c) 500 Vertices



(d) 1000 Vertices

Figure 5.5: Boxplots of network radius.

Table 5.1: Critical Values for the KS Test Assuming a 95% Confidence Level

Sample Sizes	Critical Value
100	0.19233
250	0.12164
500	0.08601
1000	0.06082

not necessarily the best models when more emergent, vertex level characteristics were considered. As such, the focus of this section is to quantify the similarity of network models using vertex centrality measures.

To quantify the similarity of network models using network measures, 1000 pairwise KS tests were performed between networks generated by each model. All tests were performed at a 95% confidence level with the critical values presented in Table 5.1. A resulting D statistic below the critical threshold (bolded) signified that the distributions were insignificantly different.

5.2.1 Degree Distribution

The average D statistic over 1000 pairwise KS tests of the degree distributions are presented in Table 5.2. In all cases where networks from the same model were compared (i.e., entries along the diagonal), the average KS statistic was well below the critical value. This result indicated that the degree distribution was consistent for all models. For each network size examined, the GR model showed the most consistent networks while the FF and ER were by far the least consistent models – the FF model being slightly worse than ER. Nonetheless, the average D statistic for the FF and ER models was still well below the critical threshold at all network sizes.

Although the degree distribution was consistent among models, there were a number of observed cases where different models were deemed to have an insignificant difference in their degree distributions. When the 100 vertex networks were examined, the BA, APA, and GR were all insignificantly different from each other. As the network sizes grew to 250 vertices, the BA model became significantly different from the GR model, but not the APA. At all network sizes, the APA and GR models demonstrated insignificant differences. Furthermore, the degree distributions of the APA and GR models become more similar, on average, as the network sizes increased. With 100 vertex networks, the APA and GR models comparisons attained an average D statistic of 0.069, while only differing by an average of 0.057 when 1000 vertex networks were considered. However, with a few slight exceptions (e.g., comparing BA and FF models), each of the models became more distinct, and thus demonstrated lower D statistics when compared to other models, as the network size increased. A noteworthy example which was contrary to the general trend was observed for 250 vertex networks when comparing the ER and WS models. When the WS and ER models were compared at the 100, 500, and 1000 vertex networks, the average D statistics were 0.823, 0.984, and 1.000, respectively. Note that the values increase as the network size increased. However, when 250 vertex networks were considered, the average D statistic dropped to 0.427 – a value significantly lower than those observed at other network sizes.

Examining the results of the MDS procedure applied to the degree distributions, shown in Figure A.5 (see Appendix A), demonstrated the above observations in a more visual fashion. A striking difference was observed when the MDS plot for degree distribution was viewed in a comparative fashion to those produced using global network measures – far more clustering was observed for the degree distribution. However, the lack of relative difference between the BA, APA, and GR models lead them to essentially form a single cluster in each of the four plots. Regardless of the similarity between BA, APA, and GR networks, it was quite apparent that the degree distribution was more suited to differentiate networks generated by different models.

5.2.2 Local Clustering Coefficient

Table 5.3 depicts the average D statistic observed for pairwise comparisons when the local transitivity was considered. Due to the implicit relationship between the local and the network transitivities, the results for local transitivity quite closely resembled their network counterparts. As expected, the BA, APA, and GR models demonstrated absolutely no local clustering, thus the distribution would be zero for each vertex causing each of these three models to be indistinguishable from one another. Despite this expected shortcoming of the local clustering coefficient, the remaining three models were easily distinguished using the local clustering coefficient. More specifically, each of the comparisons between instances of the same models lead to average D statistics which were below the critical value, albeit they were higher than those observed with the degree distribution, while comparisons between different models were always significantly above the critical threshold. These results indicated that the local clustering coefficient was both consistent among each model and significantly different between models.

Figure A.6 presents the MDS procedure applied to local clustering coefficient.

Table 5.2: Average D Statistic - Degree Distribution

(a) 100 Vertices

	BA	APA	GR	FF	ER	WS
BA	**0.050**	**0.120**	**0.165**	0.481	0.710	0.952
APA		**0.052**	**0.069**	0.419	0.657	0.965
GR			**0.045**	0.401	0.636	0.977
FF				**0.123**	0.276	0.717
ER					**0.105**	0.823
WS						**0.059**

(b) 250 Vertices

	BA	APA	GR	FF	ER	WS
BA	**0.032**	**0.113**	0.164	0.481	0.940	0.952
APA		**0.033**	**0.060**	0.417	0.948	0.961
GR			**0.029**	0.399	0.960	0.973
FF				**0.083**	0.681	0.689
ER					**0.071**	0.427
WS						**0.039**

(c) 500 Vertices

	BA	APA	GR	FF	ER	WS
BA	**0.023**	0.112	0.165	0.480	0.987	0.951
APA		**0.023**	**0.057**	0.419	0.995	0.960
GR			**0.021**	0.399	0.999	0.972
FF				**0.062**	0.865	0.676
ER					**0.053**	0.984
WS						**0.027**

(d) 1000 Vertices

	BA	APA	GR	FF	ER	WS
BA	**0.016**	0.110	0.167	0.479	0.997	0.951
APA		**0.016**	**0.057**	0.423	0.999	0.960
GR			**0.015**	0.400	1.000	0.971
FF				**0.046**	0.948	0.667
ER					**0.038**	1.000
WS						**0.019**

As seen in Figure A.6a, there is a relatively large variance in the local clustering coefficients of the ER model compared to the other models, which all demonstrated quite consistent distributions for this measure. However, as the network sizes increase, the variance among instances of the ER model became more similar to those observed in the FF and WS models. Interestingly, Figures A.6c and A.6d appeared quite similar, which implied the relative difference among the networks did not have any dramatic change when the network size increased from 500 to 1000 vertices.

In contrast to the network transitivity, the local clustering coefficient demonstrated the qualities which a good measure of (dis)similarity should possess – consistency among networks generated by the same model alongside significant differences between networks generated by different models. Furthermore, the local transitivity perfectly exemplifies how global network measures can disregard the local properties of the vertices. By comparing Figures A.2d and A.6d, it was noted that the FF and WS models were far more similar when network transitivity was considered. However, the differences in transitivity between these two models were much more pronounced with the local clustering coefficient. This demonstrated that examining networks with a finer grain, i.e., using vertex centrality measures, can provide more information than network level properties.

5.2.3 Betweenness Centrality

Table 5.4 depicts the average D statistic observed for pairwise comparisons when betweenness centrality was considered. In all instances the betweenness measure was consistent among networks generated by the same model. However, with the 100 vertex networks, there were a number of instances where the average D statistic was well below the critical threshold of 0.19233 while the networks being compared were from different models. Although not terribly surprising due to their inherent structural similarities, each of the BA, APA, and GR networks demonstrated insignificant differences, on average, from each other. Interestingly, both the APA and GR models were insignificantly different from the FF model, on average, while the BA model was significantly different. The FF, ER, and WS models were all significantly different when 100 vertex networks were considered.

As the network size increased, the discriminatory power of the betweenness centrality measure became more apparent. When 250 vertex networks were considered, only the BA-APA and APA-GR comparisons obtained average D statistics below the critical threshold. The BA-APA comparisons netted, on average, a D statistic of

Table 5.3: Average D Statistic - Local Transitivity

(a) 100 Vertices

	BA	APA	GR	FF	ER	WS
BA	**0.000**	**0.000**	**0.000**	0.933	0.407	1.000
APA		**0.000**	**0.000**	0.936	0.409	1.000
GR			**0.000**	0.932	0.405	1.000
FF				**0.136**	0.879	0.570
ER					**0.125**	0.869
WS						**0.155**

(b) 250 Vertices

	BA	APA	GR	FF	ER	WS
BA	**0.000**	**0.000**	**0.000**	0.937	0.921	0.999
APA		**0.000**	**0.000**	0.936	0.922	0.999
GR			**0.000**	0.937	0.923	0.999
FF				**0.090**	0.929	0.553
ER					**0.094**	0.926
WS						**0.097**

(c) 500 Vertices

	BA	APA	GR	FF	ER	WS
BA	**0.000**	**0.000**	**0.000**	0.937	1.000	0.999
APA		**0.000**	**0.000**	0.937	1.000	0.999
GR			**0.000**	0.937	1.000	0.999
FF				**0.066**	0.936	0.535
ER					**0.068**	0.964
WS						**0.070**

(d) 1000 Vertices

	BA	APA	GR	FF	ER	WS
BA	**0.000**	**0.000**	**0.000**	0.938	1.000	0.999
APA		**0.000**	**0.000**	0.938	1.000	0.999
GR			**0.000**	0.938	1.000	0.999
FF				**0.050**	0.938	0.515
ER					**0.054**	0.979
WS						**0.049**

0.120 – a value just marginally below the critical threshold of 0.12164. When 500 and 1000 vertex networks were considered, the only comparisons of different models which attained an average D below the critical threshold were between the APA and GR models. A noteworthy observation was made with respect to the BA, APA, and GR models. Contrary to the general trend, the average D statistic obtained when comparing models from each of these networks tended to decrease as the network size increased. This indicated that the betweenness centrality of the BA, APA, and GR models becomes more similar as network size increased. The largest such decrease was observed between the BA and APA models, which were different by an average D statistic of 0.133 on 100 vertex networks, while only different by an average of 0.112 on 1000 vertex networks – a 0.021 decrease in the average D statistic. As expected, the remaining three models, namely FF, ER, and WS, tended to become more dissimilar as network size increased.

Figure A.7 (see Appendix A) presents the MDS procedure applied to the betweenness centralities. An immediate observation was the densification of the clusters as the network size increased, reinforcing the previous observation of the increased discriminatory power on larger networks. Furthermore, when Figure A.7a was considered, it was apparent that the variance of the betweenness centrality was quite high among the BA, APA, and GR networks. This can be explained by the models, in a stochastic fashion, generating different numbers of hub nodes, causing major differences in the betweenness centrality among instances of these networks to be observed. When generating larger networks, the expected number of hub nodes stabilizes to an extent, causing far less variance of the betweenness to be observed. Interestingly, as observed with the degree distribution in Section 5.2.1, the ER and WS models appeared more similar when 250 vertex networks were examined than on any other size. Table 5.4a confirms that with betweenness centrality, the average D statistic when comparing the ER and WS networks was 0.219, while on 100, 500, and 1000 vertex networks, the average D statistic was 0.264, 0.494, and 0.714, respectively.

5.2.4 Closeness Centrality

Table 5.5 depicts the average D statistic observed for pairwise comparisons when closeness centrality was considered. In contrast to the previously examined centrality measures, the closeness centrality was completely ineffective as distinguishing instances generated by both the same model and different models. When 100 vertex networks were examined, the only model which demonstrated any consistency among

Table 5.4: Average D Statistic - Betweenness Centrality

(a) 100 Vertices

	BA	APA	GR	FF	ER	WS
BA	**0.058**	**0.133**	**0.168**	0.282	0.622	0.665
APA		**0.070**	**0.076**	**0.179**	0.509	0.551
GR			**0.062**	**0.156**	0.456	0.499
FF				**0.095**	0.418	0.487
ER					**0.108**	0.264
WS						**0.095**

(b) 250 Vertices

	BA	APA	GR	FF	ER	WS
BA	**0.038**	**0.120**	0.167	0.291	0.665	0.664
APA		**0.044**	**0.062**	0.181	0.555	0.554
GR			**0.039**	0.147	0.500	0.500
FF				**0.063**	0.488	0.495
ER					**0.064**	0.219
WS						**0.062**

(c) 500 Vertices

	BA	APA	GR	FF	ER	WS
BA	**0.027**	0.115	0.167	0.298	0.666	0.666
APA		**0.032**	**0.058**	0.187	0.556	0.557
GR			**0.028**	0.145	0.500	0.501
FF				**0.046**	0.511	0.504
ER					**0.046**	0.494
WS						**0.045**

(d) 1000 Vertices

	BA	APA	GR	FF	ER	WS
BA	**0.019**	0.112	0.166	0.302	0.667	0.666
APA		**0.023**	**0.057**	0.194	0.557	0.557
GR			**0.020**	0.146	0.506	0.500
FF				**0.036**	0.527	0.514
ER					**0.031**	0.714
WS						**0.032**

the closeness centrality, i.e., an average D statistic below the critical threshold, was the WS model. This was, in fact, the only case where the average D statistic was below the critical threshold for 100 vertex networks. When the results from larger networks were examined, the ER model was noted to have obtained average D statistics below the critical threshold when compared to itself on the 250, 500, and 1000 vertex networks while the same could be said for the WS model only when 250 and 500 vertex networks were considered. When 1000 vertex networks were considered, the average D statistic between instances of the WS model was just marginally (i.e., 0.00618) above the critical threshold.

When the MDS plots applied to closeness centrality, shown in Figure A.8 (see Appendix A) were considered, vastly different observations were made than with previous measures. In general, there was not much visible clustering in the 100, 250, and 500 vertex network plots, while some clustering structure, indicating consistency among instances from the models, became apparent in Figure A.8d. Although each of the models became more distinct as network size increased, there was still an extremely large relative variance among instances generated by the same model, especially when the BA model was considered. The 100 vertex plot demonstrated a very large amount of overlap, which indicated that there was relatively minute differences in closeness centrality between the models, with the exception of the WS model, which was visibly distinct from the other models. However as the network size increased, the FF and WS networks were located much closer in proximity to each other.

With the above observations, it was quite apparent that closeness was not a suitable centrality measure for discriminating between networks generated by different models. Furthermore, the closeness centrality was not consistent among networks generated by the same model. Thus, in the context of this work, no valuable information could be obtained when the closeness centrality was used to measure the (dis)similarity of networks.

5.2.5 PageRank

Table 5.6 depicts the average D statistic observed for pairwise comparisons when PageRank was considered. With respect to all models, the PageRank measure was consistent among networks generated by the same model. For all network sizes, the BA model showed the least consistent values for PageRank, as demonstrated by the highest average D statistic when compared to other instances from the BA model. With the exception of the APA-GR comparisons on 100 and 250 vertex networks, the

Table 5.5: Average D Statistic - Closeness Centrality

(a) 100 Vertices

	BA	APA	GR	FF	ER	WS
BA	0.330	0.630	0.654	0.482	0.686	0.991
APA		0.292	0.265	0.812	0.889	1.000
GR			0.234	0.823	0.906	1.000
FF				0.301	0.548	0.881
ER					0.643	0.947
WS						**0.133**

(b) 250 Vertices

	BA	APA	GR	FF	ER	WS
BA	0.284	0.674	0.723	0.554	0.998	0.994
APA		0.246	0.239	0.875	1.000	1.000
GR			0.202	0.901	1.000	1.000
FF				0.238	0.945	0.805
ER					**0.108**	0.921
WS						**0.109**

(c) 500 Vertices

	BA	APA	GR	FF	ER	WS
BA	0.254	0.673	0.767	0.631	1.000	0.996
APA		0.209	0.231	0.913	1.000	1.000
GR			0.178	0.941	1.000	1.000
FF				0.206	0.997	0.712
ER					**0.080**	1.000
WS						**0.083**

(d) 1000 Vertices

	BA	APA	GR	FF	ER	WS
BA	0.237	0.667	0.801	0.707	1.000	0.997
APA		0.182	0.250	0.935	1.000	1.000
GR			0.160	0.968	1.000	1.000
FF				0.163	1.000	0.592
ER					**0.055**	1.000
WS						0.067

PageRank measure was significantly different, on average, among networks generated by different models. Although the PageRank was, on average, insignificantly different among instance of the APA and GR models when 100 and 250 vertex networks were considered, the average D statistic was above the critical threshold when 500 and 1000 vertex networks were examined. However, the average D was still decreasing as the network size increased. Similarly, the average D statistic for the BA-GR, BA-APA, and ER-WS comparisons also decreased as the network size increased. Note that the PageRank is one of the only measures which obtained an average D statistic above the critical threshold when the APA and GR models were compared. More concretely, the PageRank was the only measure which was consistent among instances of the same model which was also significantly different among instances of the APA and GR, although only for 500 and 1000 vertex networks.

Figure 5.6 presents the MDS procedure applied to the PageRank, which reinforces the observations made regarding the average D statistic. With the exception of the 100 vertex networks (Figure 5.6a), the PageRank demonstrated compact, distinct clusters. This effectively meant the PageRank was both consistent among networks from the same model while being sufficiently different among networks generated by different models. When the 100 vertex networks were considered, there was a lot more visible overlap between instances from different models. However, even with the 100 vertex networks, the PageRank was still an extremely effective discriminator for different networks. When the plots for 500 and 1000 vertex networks were examined, the overlap of the ER and WS models was apparent. However, these models are still, on average, significantly different when 500 and 100 vertex networks were considered. The visible similarity is attributed to two factors. Firstly, the WS and ER models were quite similar when larger networks were considered – comparisons between these models resulted in an average D statistic of 0.107 and 0.094, respectively. Furthermore, the reader is reminded that the distance in an MDS plot between points represents a relative difference, not an absolute difference. Thus, the small distance only signified the relative difference between instances of the ER and WS models was much smaller compared to other models.

The above observations demonstrated the effectiveness of the PageRank measure for discriminating networks generated by different networks. Even with the similarity of the ER and WS models observed on the larger networks, the PageRank was the best measure examined for quantifying (dis)similarity of networks.

Table 5.6: Average D Statistic - PageRank

(a) 100 Vertices

	BA	APA	GR	FF	ER	WS
BA	**0.152**	0.209	0.272	0.332	0.532	0.661
APA		**0.119**	**0.146**	0.279	0.415	0.547
GR			**0.105**	0.267	0.363	0.498
FF				**0.112**	0.260	0.456
ER					**0.110**	0.289
WS						**0.099**

(b) 250 Vertices

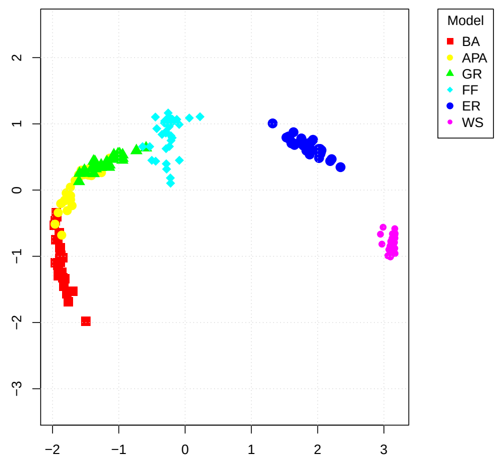
	BA	APA	GR	FF	ER	WS
BA	**0.101**	0.181	0.253	0.313	0.623	0.663
APA		**0.080**	**0.121**	0.279	0.507	0.551
GR			**0.069**	0.278	0.453	0.497
FF				**0.081**	0.378	0.480
ER					**0.073**	0.182
WS						**0.063**

(c) 500 Vertices

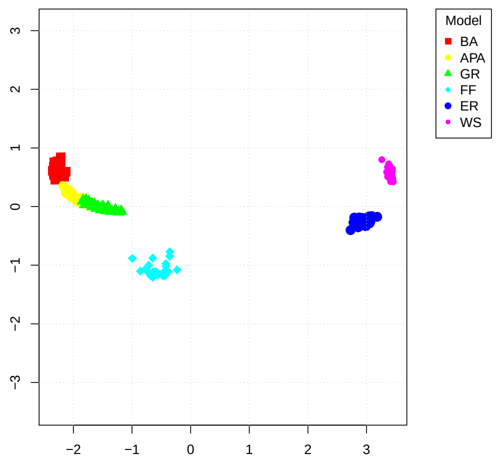
	BA	APA	GR	FF	ER	WS
BA	**0.075**	0.167	0.243	0.309	0.657	0.663
APA		**0.057**	0.107	0.290	0.546	0.553
GR			**0.050**	0.291	0.490	0.498
FF				**0.064**	0.464	0.500
ER					**0.055**	0.107
WS						**0.046**

(d) 1000 Vertices

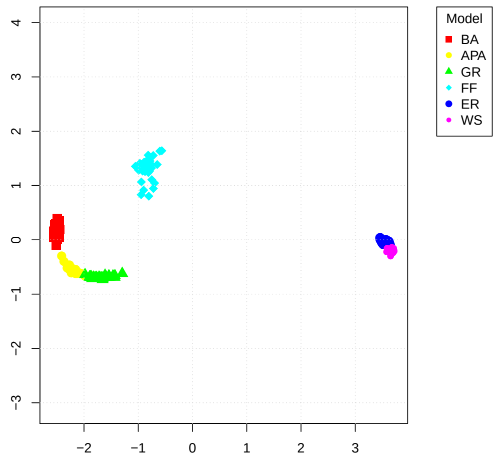
	BA	APA	GR	FF	ER	WS
BA	**0.055**	0.157	0.237	0.318	0.666	0.663
APA		**0.041**	0.101	0.305	0.556	0.555
GR			**0.035**	0.306	0.500	0.498
FF				**0.052**	0.536	0.519
ER					**0.040**	0.094
WS						**0.032**



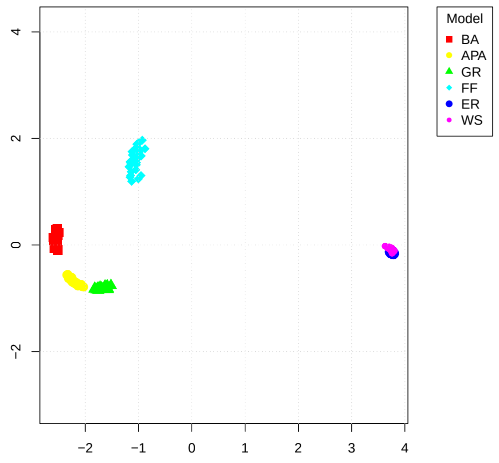
(a) 100 vertex networks. 93.04% variance accounted for on each axis.



(b) 250 vertex networks. 93.83% variance accounted for on each axis.



(c) 500 vertex networks. 95.88% variance accounted for on each axis.



(d) 1000 vertex networks. 96.23% variance accounted for on each axis.

Figure 5.6: MDS applied to PageRank for various network sizes.

5.2.6 Eigenvector Centrality

Table 5.7 depicts the average D statistic observed for pairwise comparisons when eigenvector centrality was considered. The eigenvector centrality measure was ineffective at distinguishing networks generated by different models. There were only two comparisons which resulted in an average D statistic below the critical threshold, namely when comparing networks from the FF model on 100 and 250 vertex networks and comparing GR networks on 100 vertex networks. All other comparisons resulted in an average D statistic above the critical threshold. In general, networks from the same model became more similar (i.e., the average D decreased), while networks from different models became more dissimilar as the network size increased. A noteworthy exception from this trend was observed with the ER-WS comparison. The average D statistic was significantly lower for the ER-WS comparisons when 250 and 500 networks were examined than with 100 and 1000 vertex networks. Interestingly, a similar observation was made regarding the degree distribution, albeit the unexpected similarity was only noted for 250 vertex networks. For all network sizes examined, the FF model demonstrated the most consistency among networks with respect to eigenvector centrality, while the BA model demonstrated the largest differences, on average.

Figure A.9 (see Appendix A) presents the MDS procedure applied to the eigenvector centrality. Similar to the closeness centrality, shown in Figure A.8, a large amount of overlap is depicted when the eigenvector centrality was considered. For all network sizes examined, the BA, APA, GR, and FF models were essentially indistinguishable – the networks from these models were all overlapped, and formed one large cluster. Furthermore, no model, with respect to any size network, demonstrated a compact cluster indicating consistency among the model. The relatively small distance between the networks from the ER and WS networks in Figures A.9b and A.9c demonstrated the increased relative similarity on the mid-sized networks, as observed with the average D statistic.

Thus, it can be safely concluded that the eigenvector centrality was completely ineffective at quantifying (dis)similarity of networks. Furthermore, no valuable information can be obtained by measuring and comparing the eigenvector centrality.

Table 5.7: Average D Statistic - Eigenvector Centrality

(a) 100 Vertices

	BA	APA	GR	FF	ER	WS
BA	0.232	0.237	0.217	0.266	0.716	0.976
APA		0.217	0.245	0.246	0.701	0.954
GR			**0.174**	0.217	0.655	0.926
FF				**0.158**	0.537	0.820
ER					0.194	0.645
WS						0.225

(b) 250 Vertices

	BA	APA	GR	FF	ER	WS
BA	0.207	0.240	0.214	0.258	0.972	0.990
APA		0.192	0.272	0.280	0.939	0.979
GR			0.159	0.197	0.916	0.956
FF				**0.118**	0.817	0.876
ER					0.153	0.289
WS						0.194

(c) 500 Vertices

	BA	APA	GR	FF	ER	WS
BA	0.193	0.233	0.218	0.271	0.996	0.995
APA		0.172	0.302	0.318	0.992	0.989
GR			0.149	0.200	0.977	0.972
FF				0.092	0.918	0.906
ER					0.149	0.198
WS						0.180

(d) 1000 Vertices

	BA	APA	GR	FF	ER	WS
BA	0.182	0.237	0.226	0.291	0.999	0.998
APA		0.155	0.334	0.350	0.998	0.995
GR			0.135	0.208	0.994	0.982
FF				0.072	0.965	0.929
ER					0.137	0.531
WS						0.155

Chapter 6

Analysis of Network Property Subsets

While the results from Chapter 5 demonstrated that a number of centrality measures could be reasonably used to distinguish between networks generated by different models, how the measures interact was not considered. As such, the focus of this chapter is to determine which network centrality measures, and subsets thereof, are most capable of distinguishing between graphs generated by different models. A good set of measures should consistently and accurately be able to determine that a graph was not generated by a given model, as the measures should provide significantly different centrality values for graphs generated by different models. Similarly, a good set of measures should have relatively consistent results among graphs generated the same model.

6.1 Experimental Procedure

To evaluate a subset of measures, a meta-analysis procedure was carried out which made use of Fisher's method [92] to combine p-values. Furthermore, receiver operating characteristic (ROC) curves were generated to visualize the performance of the best subsets of measures. This section outlines the experimental procedures used to evaluate a subset of measures.

6.1.1 Fisher's Method

Fisher's method [92] is a meta-analysis test used to combine the p-values from independent hypothesis tests. The independent p-values are combined to form a test statistic using the formula

$$\chi^2_{2k} \sim -2 \sum_{i=1}^{k} \ln(p_i) \tag{6.1}$$

where k is the number of independent tests being combined and p_i is the p-value of the i test. When the p-values of the independent tests are small, the test statistic is large, suggesting the null hypotheses were not true for all of the independent tests. In contrast, when the p-values are large, the test statistic is small and follows a chi-squared distribution with $2k$ degrees of freedom. This property allows a p-value to be attributed to χ^2_{2k}, thus resulting in a combined p-value.

In the context of graph model evaluation, Fisher's method was used to determine similarity of networks, and thus their respective models, using a set of measures. Thus, Fisher's method was used to combine individual p-values, resulting from KS-tests, to determine whether this set of measures agreed upon the null hypothesis of similarity.

6.1.2 Combining the Performance of a Set of Measures

To determine the performance of a subset of measures, a method of combining performance evaluations was necessary as each measure must be examined independently. For this purpose, each centrality measure was calculated and compared to that of the target network, the results of which were combined in various ways (i.e., using each subset of measures) with Fisher's method. In more detail, for a given target graph G, model M, and set of measures F, N instances of M were generated. For each of the N instances of M, all six centrality measure was calculated and compared, using a KS test, to the corresponding centrality measure of the target graph, G. The resulting p-values from the KS test, performed at a 95% confidence level using critical thresholds given in Table 5.1, were recorded. Once all instances of M had been generated and compared to the target graph, the combination phase took place. The power set (excluding the empty set) of F was generated, denoted by $\mathcal{P}_{\geq 1}(F)$, in order to examine all non-empty subsets of measures. For each subset of measures s, the p-values from the KS tests belonging to the members of s were retrieved and combined using Fisher's method. Thus, for each subset of measures, a combined p-value was generated. This procedure is demonstrated in Algorithm 7.

The procedure outlined in Algorithm 7 only compared a single target to a single model. To compare multiple models to a single target graph, a meta analysis pro-

Algorithm 7 Combining the Performance of a Set of Measures

```
function COMBINE($G, M, F, N$)        // $G$ - target graph, $M$ - graph model, $F$ - set of measures
    for $i$ from 1 to $N$ do
        $g_i \leftarrow M$.generate()
        for all $f \in F$ do
            $p[f]$.append(KS-TEST($f(g_i), f(G)$))
        end for
    end for
    for all $s \in \mathcal{P}_{\geq 1}(F)$ do
        $result[s] \leftarrow$ FISHERSMETHOD($\{p[x] \mid x \in s\}$)        // Combine the p-values for each measure in $s$
    end for
return $result$
end function
```

cedure was used. This procedure, depicted in Algorithm 8, generated ROC curves for each subset of measures. As ROC curves only plot the discriminatory power of a binary classifier, a classification system had to be derived for this procedure to be applied. To construct such a classifier, an assumption was made that if two graphs were generated by the same model, they would exhibit similar properties and thus, when their centrality measures were compared, a high p-value would be obtained. Following from this, a good subset of centrality measures should have, similarly, produced a high p-value when the constituent p-values were combined using Fisher's method. If two graphs were produced by the same model, the p-value resulting from Fisher's method over each subset of measures was expected, ideally, to be 1. Conversely, if the graphs were produced by different models, the p-value resulting from Fisher's method was expected, ideally, to be 0 for each subset of measures. This reasoning was used to derive a binary classification system where the observed p-values from this procedure were taken as an approximation (response) to the expected outcome, which allowed the performance of each subset of measures to be examined using an ROC curve.

To further quantify the performance of each subset of measures, the area under the curve (AUC) of each ROC curve was calculated. The AUC measure, while failing to capture individual trade-off information about a classifier, reduces an ROC curve to a single value which represents the general performance of the classifier. More formally, the AUC of a classifier represents the probability that a randomly chosen positive instance will be given a higher rank than a randomly chosen negative instance [93]. Similarly, due to the ranking system used when constructing an ROC curve, there exists a close relationship between the AUC measure and the Mann-Whitney U test

[94, 95]. In the context of this study, this property of AUC translates to the probability that a graph G, originating from model M, will receive a higher p-value using Fisher's method when compared to graphs generated by M than when compared to graphs which were not generated by M.

Algorithm 8 Meta Analysis

```
function MetaAnalysis($G, l_M, F, N$)
    for all $M \in l_M$ do
        $results[M] \leftarrow$ Combine($G, M, F, N$)        // See Algorithm 7
        if $G \in M$ then        // If $G$ was generated by $M$
            $expected[M] \leftarrow 1$        // The expected p-value
        else
            $expected[M] \leftarrow 0$
        end if
    end for
    for all $s \in \mathcal{P}_{\geq 1}(F)$ do
        Roc-Curve($expected, results$.get($s$))        // Compute an ROC curve using
the expected outcome and the observed Fisher's p-values
    end for
end function
```

6.2 Results

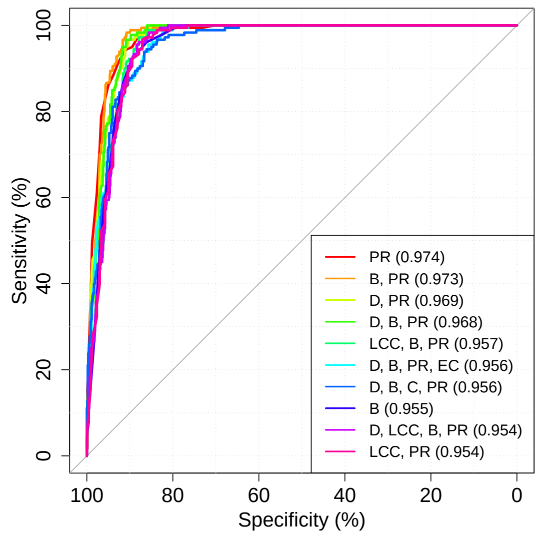
This section presents the results from experiments described in Section 6.1. Finer grained results, depicting the ROC curves generated by the top ten sets of measures when each model was used as a target are given in Appendix B. For the remainder of this chapter, the following abbreviations are used to refer to each of the network measures:

D	– Degree distribution
LCC	– Local clustering coefficient/transitivity
B	– Betweenness centrality
C	– Closeness centrality
PR	– PageRank
EC	– Eigenvector centrality

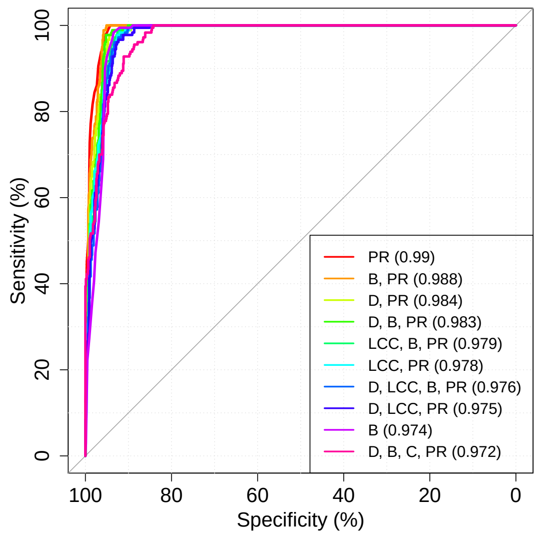
These results were obtained by repeating the process outlined in Algorithm 8 using a target graph generated by each of the six models, respectively, and aggregating the results. These aggregated results demonstrated the overall classification power of each of the centrality measure subsets. Figure 6.1 presents the ROC curve results for various network sizes while Tables 6.1 – 6.5 outline the AUC measure for each subset of measures. Two immediate observations were made using these results. First, the PageRank and betweenness centrality measures, in some combination, had attained the highest AUC for each network size. At each of the four network sizes, the PageRank measure was included in each of the top five sets of measures. Furthermore, PageRank was present in nine of the top ten measure sets for the 100, 250, and 500 vertex networks and eight of the top ten sets when 1000 vertex networks were examined. Similarly, betweenness was present in at least five of the top ten fitness sets for each network size. The discriminatory power of the betweenness and PageRank measures further demonstrated the effectiveness of these measures, as observed in Sections 5.2.3 and 5.2.5, respectively. The second observation, albeit more expected, was that the networks were easier to distinguish as the network size increased. Note that the highest AUC measured on 100 vertex networks was 0.974 – the same AUC measure on the ninth best subset when 250 vertex networks were considered. Similar observations regarding the relative differences among performance were made for each of the network size increases.

6.2.1 Single Measure Sets

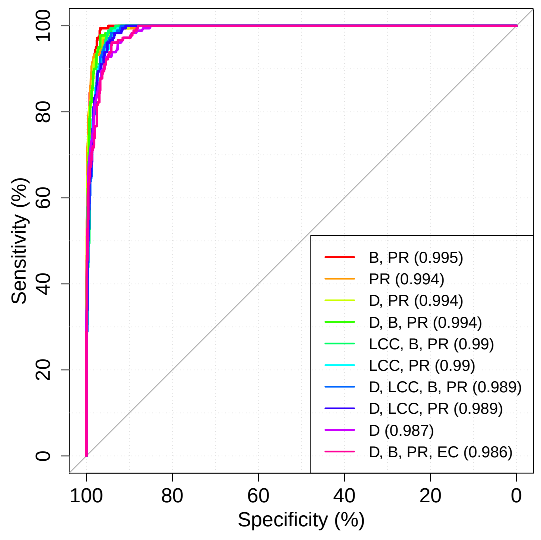
Table 6.1 presents the AUC results when the centrality measure were examined independently. With all network sizes, the PageRank measure demonstrated the highest AUC, with the degree and betweenness being only slightly lower, especially when the larger networks were considered. The degree, betweenness, and PageRank measures were immediately noted to have significantly higher AUC values than the other measures. As expected from the results in Sections 5.2.4 and 5.2.6, the closeness and eigenvector centrality measures demonstrated the lowest AUC values on all network sizes examined. When 100 and 150 vertex networks were considered, the closeness centrality was worse than the eigenvector centrality. However, the opposite was true with 500 and 1000 vertex networks – the eigenvector centrality performed worse than the closeness centrality. Furthermore, with 1000 vertex networks, the eigenvector centrality obtained an AUC of 0.593, only slightly (0.093) better than a completely



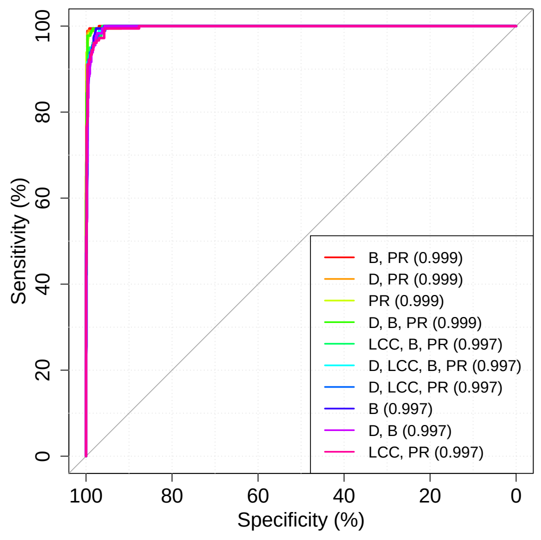
(a) 100 vertex networks.



(b) 250 vertex networks.



(c) 500 vertex networks.



(d) 1000 vertex networks.

Figure 6.1: ROC curves depicting the ten measure sets with the highest area under the curve (AUC) values, shown in the legend, for various network sizes.

Table 6.1: AUC for 1-Measure ROC Curves on Aggregated Results

(a) 100 Vertex Networks

Measure Set	AUC
D	0.953
LCC	0.844
B	0.955
C	0.731
PR	**0.974**
EC	0.809

(b) 250 Vertex Networks

Measure Set	AUC
D	0.968
LCC	0.850
B	0.974
C	0.682
PR	**0.990**
EC	0.687

(c) 500 Vertex Networks

Measure Set	AUC
D	0.987
LCC	0.850
B	0.985
C	0.672
PR	**0.994**
EC	0.671

(d) 1000 Vertex Networks

Measure Set	AUC
D	0.996
LCC	0.850
B	0.997
C	0.629
PR	**0.999**
EC	0.593

random classifier[1]. This further demonstrates the ineffectiveness of the eigenvector centrality when used to compare networks. On all network sizes, the local transitivity measure demonstrated a moderate AUC value. With each network size examined, the AUC for the local transitivity was significantly lower AUC than the degree, betweenness, and PageRank measures, but significantly higher than the closeness and eigenvector centrality measures. Interestingly, the AUC of the local transitivity was 0.850 for each of the 250, 500, and 1000 vertex network experiments, which demonstrated this measure was quite consistent in terms of its discriminatory power as the network size increased.

6.2.2 Two Measure Sets

When Table 6.2 was examined, it was observed that combining the betweenness and PageRank lead to the highest AUC on all network sizes. Note that with 1000 vertex networks, a tie was observed with the subset which combined the degree distribution and PageRank measures. The subsets which contained one or more of the degree, betweenness, and PageRank measures were, in general, the best classifiers. The subsets which contained the local transitivity, especially when combined with the PageRank,

[1]A random classifier has an equal probability of ranking a positive event as positive or negative, leading to an AUC of 0.5.

also had relatively high AUC values. The local transitivity and PageRank set obtained an AUC value at least as great, or greater than, the set containing degree and betweenness on all network sizes. The lowest AUC values on each network were observed, as expected, when the closeness and eigenvector centrality measures were combined. However, with the exception of 1000 vertex networks, the AUC value when the closeness and eigenvector centralities were combined was higher than when each was considered individually. Although together they still perform significantly worse than other measures, it was interesting to note that more information can be gathered by using these measures in combination than either can provide individually – this observation alone attests to the merit of examining the combinations of different measures.

6.2.3 Three Measure Sets

Table 6.3 depicts the AUC value results when centrality measures were examined in subsets of size three. Following previous observations, the subset with the degree, betweenness, and PageRank measures had the highest AUC for each of the network sizes considered. While the PageRank measure itself had a higher AUC (see Table 6.1) than the subset containing degree, betweenness, and PageRank, this subset had a higher AUC than either of the degree or betweenness measures when considered independently. Again, this demonstrates how more discriminatory information is made available when more than just a single measure is considered. This observation can further be explained by Fisher's method producing "exaggerated" p-values when the independent p-values are correlated[2] [96]. The subset which attained the lowest AUC on 100 vertex networks was the set containing betweenness, closeness, and eigenvector centrality, while the set containing local transitivity, closeness, and eigenvector centrality obtained the lowest AUC values on each of the remaining network sizes examined.

6.2.4 Four Measure Sets

The AUC results using subsets containing four measures are presented in Table 6.4. Counter-intuitively, the highest AUC value for 100 vertex networks was tied between two subsets of measures, one contained degree, betweenness, PageRank, and closeness

[2]The independent p-values are considered correlated as each of the degree, betweenness, and PageRank measures demonstrated high p-values when comparing networks generated by the same model while networks generated by different models obtained low p-values. Thus, these measure can be reasonably considered to be correlated.

Table 6.2: AUC for 2-Measure ROC Curves on Aggregated Results

(a) 100 Vertex Networks

Measure Set	AUC
D, LCC	0.920
D, B	0.954
D, C	0.895
D, PR	0.969
D, EC	0.925
LCC, B	0.924
LCC, C	0.877
LCC, PR	0.954
LCC, EC	0.894
B, C	0.899
B, PR	**0.973**
B, EC	0.890
C, PR	0.905
C, EC	0.832
PR, EC	0.930

(b) 250 Vertex Networks

Measure Set	AUC
D, LCC	0.948
D, B	0.970
D, C	0.879
D, PR	0.984
D, EC	0.870
LCC, B	0.954
LCC, C	0.857
LCC, PR	0.978
LCC, EC	0.837
B, C	0.877
B, PR	**0.988**
B, EC	0.856
C, PR	0.897
C, EC	0.692
PR, EC	0.893

(c) 500 Vertex Networks

Measure Set	AUC
D, LCC	0.968
D, B	0.986
D, C	0.885
D, PR	0.994
D, EC	0.906
LCC, B	0.969
LCC, C	0.855
LCC, PR	0.990
LCC, EC	0.865
B, C	0.890
B, PR	**0.995**
B, EC	0.908
C, PR	0.894
C, EC	0.706
PR, EC	0.914

(d) 1000 Vertex Networks

Measure Set	AUC
D, LCC	0.984
D, B	0.997
D, C	0.884
D, PR	**0.999**
D, EC	0.917
LCC, B	0.985
LCC, C	0.806
LCC, PR	0.997
LCC, EC	0.840
B, C	0.890
B, PR	**0.999**
B, EC	0.923
C, PR	0.872
C, EC	0.622
PR, EC	0.911

Table 6.3: AUC for 3-Measure ROC Curves on Aggregated Results

(a) 100 Vertex Networks

Measure Set	AUC
D, LCC, B	0.933
D, LCC, C	0.932
D, LCC, PR	0.952
D, LCC, EC	0.918
D, B, C	0.941
D, B, PR	**0.968**
D, B, EC	0.937
D, C, PR	0.947
D, C, EC	0.922
D, PR, EC	0.953
LCC, B, C	0.930
LCC, B, PR	0.957
LCC, B, EC	0.917
LCC, C, PR	0.940
LCC, C, EC	0.909
LCC, PR, EC	0.942
B, C, PR	0.937
B, C, EC	0.901
B, PR, EC	0.942
C, PR, EC	0.917

(b) 250 Vertex Networks

Measure Set	AUC
D, LCC, B	0.959
D, LCC, C	0.933
D, LCC, PR	0.975
D, LCC, EC	0.925
D, B, C	0.945
D, B, PR	**0.983**
D, B, EC	0.940
D, C, PR	0.958
D, C, EC	0.862
D, PR, EC	0.955
LCC, B, C	0.934
LCC, B, PR	0.979
LCC, B, EC	0.921
LCC, C, PR	0.952
LCC, C, EC	0.834
LCC, PR, EC	0.946
B, C, PR	0.958
B, C, EC	0.845
B, PR, EC	0.956
C, PR, EC	0.868

(c) 500 Vertex Networks

Measure Set	AUC
D, LCC, B	0.976
D, LCC, C	0.953
D, LCC, PR	0.989
D, LCC, EC	0.944
D, B, C	0.967
D, B, PR	**0.994**
D, B, EC	0.961
D, C, PR	0.969
D, C, EC	0.862
D, PR, EC	0.963
LCC, B, C	0.955
LCC, B, PR	0.990
LCC, B, EC	0.947
LCC, C, PR	0.964
LCC, C, EC	0.846
LCC, PR, EC	0.957
B, C, PR	0.967
B, C, EC	0.869
B, PR, EC	0.965
C, PR, EC	0.852

(d) 1000 Vertex Networks

Measure Set	AUC
D, LCC, B	0.990
D, LCC, C	0.960
D, LCC, PR	0.997
D, LCC, EC	0.947
D, B, C	0.975
D, B, PR	**0.999**
D, B, EC	0.982
D, C, PR	0.972
D, C, EC	0.837
D, PR, EC	0.979
LCC, B, C	0.963
LCC, B, PR	0.997
LCC, B, EC	0.946
LCC, C, PR	0.958
LCC, C, EC	0.786
LCC, PR, EC	0.945
B, C, PR	0.975
B, C, EC	0.852
B, PR, EC	0.980
C, PR, EC	0.824

while the other consisted of degree, betweenness, PageRank, and eigenvector centrality. For all other network sizes, the subset which obtained the highest AUC consisted of, expectedly, the degree, betweenness, PageRank, and local transitivity measures. Although with 100 vertex networks, the difference was extremely small (the subset containing local transitivity had an AUC of only 0.002 lower than those with closeness and eigenvector centrality), it was still unexpected for the subsets with closeness and eigenvector centrality to have obtained higher AUC values. It was also noted that the subsets which contained four measures all had AUC values above 0.9. The AUC of the best subsets, however, were lower when four measures were combined than when only three measures were combined.

6.2.5 Five Measure Sets and All Measures

For convenience, the subsets containing five measures, as well as the combination containing all six measures, are presented together in Table 6.5. When five measures were combined, the subset which obtained the highest AUC for each network size was not consistent. When 100 vertex networks were considered, the subset which contained the (abbreviations used for brevity) D, LCC, B, C, and PR measures along with the subset which consisted of the D, B, C, PR, and EC measures obtained the highest AUC value. Interestingly, when the sixth measure was added to the above two subsets, the AUC did not decrease. When 250 vertex networks were considered, the highest AUC was observed when the D, LCC, B, C, and PR measures were combined, while the subset containing D, LCC, B, PR, and EC obtained the highest AUC on 500 vertex networks. Finally, with 1000 vertex networks, there was a tie for the highest AUC between the subset containing the D, LCC, B, C, and PR measures and the subset containing the D, LCC, B, PR, and EC measures. Note that with all network sizes, the subset of measures which obtained the highest AUC values always contained the degree, betweenness, and PageRank measures. In contrast, the lowest AUC was always obtained by the subset which consisted of the D, LCC, B, C, and EC measures.

6.2.6 Summary

In summary, when single measures are considered, the PageRank measure has the most discriminatory power among the models examined. While PageRank being the most explanatory was somewhat unintuitive, an explanation for this behavior is due to the correlation between PageRank and degree in undirected networks. The

Table 6.4: AUC for 4-Measure ROC Curves on Aggregated Results

(a) 100 Vertex Networks

Measure Set	AUC
D, LCC, B, C	0.943
D, LCC, B, PR	0.954
D, LCC, B, EC	0.928
D, LCC, C, PR	0.951
D, LCC, C, EC	0.936
D, LCC, PR, EC	0.945
D, B, C, PR	**0.956**
D, B, C, EC	0.940
D, B, PR, EC	**0.956**
D, C, PR, EC	0.950
LCC, B, C, PR	0.949
LCC, B, C, EC	0.930
LCC, B, PR, EC	0.947
LCC, C, PR, EC	0.941
B, C, PR, EC	0.936

(b) 250 Vertex Networks

Measure Set	AUC
D, LCC, B, C	0.955
D, LCC, B, PR	**0.976**
D, LCC, B, EC	0.949
D, LCC, C, PR	0.966
D, LCC, C, EC	0.927
D, LCC, PR, EC	0.962
D, B, C, PR	0.972
D, B, C, EC	0.937
D, B, PR, EC	0.969
D, C, PR, EC	0.948
LCC, B, C, PR	0.968
LCC, B, C, EC	0.921
LCC, B, PR, EC	0.964
LCC, C, PR, EC	0.940
B, C, PR, EC	0.949

(c) 500 Vertex Networks

Measure Set	AUC
D, LCC, B, C	0.970
D, LCC, B, PR	**0.989**
D, LCC, B, EC	0.973
D, LCC, C, PR	0.979
D, LCC, C, EC	0.932
D, LCC, PR, EC	0.982
D, B, C, PR	0.983
D, B, C, EC	0.945
D, B, PR, EC	0.986
D, C, PR, EC	0.943
LCC, B, C, PR	0.981
LCC, B, C, EC	0.934
LCC, B, PR, EC	0.983
LCC, C, PR, EC	0.938
B, C, PR, EC	0.945

(d) 1000 Vertex Networks

Measure Set	AUC
D, LCC, B, C	0.984
D, LCC, B, PR	**0.997**
D, LCC, B, EC	0.983
D, LCC, C, PR	0.991
D, LCC, C, EC	0.926
D, LCC, PR, EC	0.989
D, B, C, PR	0.994
D, B, C, EC	0.955
D, B, PR, EC	0.995
D, C, PR, EC	0.954
LCC, B, C, PR	0.992
LCC, B, C, EC	0.929
LCC, B, PR, EC	0.991
LCC, C, PR, EC	0.923
B, C, PR, EC	0.955

Table 6.5: AUC for 5- and 6-Measure ROC Curves on Aggregated Results

(a) 100 Vertex Networks

Measure Set	AUC
D, LCC, B, C, PR	**0.954**
D, LCC, B, C, EC	0.943
D, LCC, B, PR, EC	0.949
D, LCC, C, PR, EC	0.952
D, B, C, PR, EC	**0.954**
LCC, B, C, PR, EC	0.948
D, LCC, B, C, PR, EC	0.954

(b) 250 Vertex Networks

Measure Set	AUC
D, LCC, B, C, PR	**0.970**
D, LCC, B, C, EC	0.947
D, LCC, B, PR, EC	0.967
D, LCC, C, PR, EC	0.958
D, B, C, PR, EC	0.964
LCC, B, C, PR, EC	0.959
D, LCC, B, C, PR, EC	0.963

(c) 500 Vertex Networks

Measure Set	AUC
D, LCC, B, C, PR	0.982
D, LCC, B, C, EC	0.967
D, LCC, B, PR, EC	**0.985**
D, LCC, C, PR, EC	0.974
D, B, C, PR, EC	0.978
LCC, B, C, PR, EC	0.975
D, LCC, B, C, PR, EC	0.978

(d) 1000 Vertex Networks

Measure Set	AUC
D, LCC, B, C, PR	**0.993**
D, LCC, B, C, EC	0.970
D, LCC, B, PR, EC	**0.993**
D, LCC, C, PR, EC	0.979
D, B, C, PR, EC	0.987
LCC, B, C, PR, EC	0.979
D, LCC, B, C, PR, EC	0.983

damping factor causes the correlation between degree and PageRank to exhibit some variability, the amount of which depends upon the degree of the vertex [71]. Thus, the PageRank measure amplifies the differences among the degrees of a network, causing smaller differences in degree to be perceived as larger differences in PageRank. It is interesting to note that the AUC values obtained when using PageRank alone were higher than when subsets of measures were considered. This behavior can be attributed to the way in which Fisher's method combines p-values. Since PageRank was the best single measure, it would have, in general, produced the highest p-value when two networks from the same model were compared. Thus when lower p-values, resulting from comparisons using other measures, were combined with the p-values for PageRank, the combined p-value from Fisher's method would be lower. This lower combined p-value would be propagated to the meta-analysis, causing an apparent decrease in the discriminatory power of the subset of measures.

When two measures were combined, it was noted that combining the betweenness and PageRank measures would insignificantly change the AUC value relative to only using the PageRank measure, namely the AUC was different by at most 0.002. When three measures were considered, the subset which contained the degree, betweenness, and PageRank measures obtained the highest AUC. As the number of measures combined was increased beyond three, the subsets which attained the highest AUC became less intuitive. However, it was noted that the subsets which obtained the highest AUC always contained the degree, betweenness, and PageRank measures. Based on the previous observations, these three measures were used as measures of evolutionary fitness within the GP system to construct graph models. The subsequent chapters detail how these measures were incorporated and demonstrates their merit as fitness measures.

Chapter 7

Evolving Graph Models for Complex Networks

The GP system used to infer graph models for complex networks was developed using LinkableGP[1] [38, 37]. The style of GP system used in this work differs from the tree-based GP system used in previous work [29, 31] although the LinkableGP system has been applied to the problem of inferring graph models [38]. Justification for the choice of GP system is provided in the following section. Parameters used by the GP system are presented with the experimental results in Chapters 8 and 9, respectively. During the evolutionary process, the SwiftGraph[1] library was used to compute network measures. Target networks were generated using the *igraph* library for network analysis [97].

7.1 LinkableGP

The LinkableGP system used in this work is based on a hybrid of two existing GP paradigms, object-oriented GP to construct class-based programs and linear GP to evolve imperative style methods (see Chapter 4 for an overview of these terms). The LinkableGP system also proposed a mechanism whereby the user can supply a partially-implemented class, i.e., an abstract class, to provide the implementation of methods where the desired functionality is known *a priori*. This partially-implemented class allows for known implementation details to be included and elegantly combined with the unknown methods which are to be evolved by the GP system. As such, the evolved program is a fully functional object which includes the

[1]LinkableGP and SwiftGraph were co-developed by the author as part of this thesis.

a priori specified implementations.

Unlike a traditional GP tree structure, which represents a single function, individuals in the LinkableGP population represent types or classes, which are implementations of a parent type. This parent type is provided by the user via an abstract class. Due to its influence from both object-oriented and linear methodologies, individuals are constructed using a collection of linear chromosomes. Each of these chromosomes corresponds to a single abstract method defined in the parent type and can define its own collection of language elements. Individual chromosomes are represented in a linear fashion by an array of integers and are used to construct a phenotype representing the physical implementation of a method. The construction process is an iterative one whereby each successive integer in the chromosome is used to select a suitable language element. It should be noted that all operations in the LinkableGP system are strongly typed [78]. Selection operations are performed by enumerating the suitable language elements and using the respective chromosome value in a modulo fashion as follows. Assume the chromosome value being used for selection is c and there are m language elements which are suitable for selection, i.e., all their arguments can be satisfied with the available terminals. The m suitable language elements are enumerated and assigned identifiers $\{0, 1, 2, ..., m-1\}$. The modulo selection proceeds by returning the language element with identifier c **mod** m.

To construct a method, the first value from a chromosome is used to select a function from the language such that all of its arguments can be satisfied. Successive chromosome values are then used to select correctly-typed terminals to satisfy each argument of the function. If the function selected for construction has a non-void return type, the next chromosome value is used to select either a mutable variable or a new variable to store the result of the function, which is then added to the set of available terminals. Terminals in the LinkableGP system take the form of either variables or constants. The initial set of terminals contains the user-specified constant generators and any arguments to the method being constructed. The methods arguments are immediately made available as (immutable) variables. New variables can be created at any time during method construction. This means that the set of terminals can drastically change during method construction. Note that while nested functions cannot be explicitly constructed, their behavior can be replicated by storing the result of a function execution in a variable and then using this variable as an argument for another function. The above construction process continues until all chromosome values have been exhausted. In the event that all chromosome values have been exhausted and the program being constructed is not complete, the chromosome is extended to facilitate the completion of the method. That is, the construction process guarantees a valid, type-safe program will be constructed, provided there are sufficient language elements to do so. Finally, if the method being constructed has a non-void return type, a single chromosome value is retained to select the return value of the method. If there exists variables of the correct type, a selection operation is performed to select the return type from such variables. If no variable of the correct type is present, an additional function with the correct return type is constructed and used as the return value of the method.

While graph models for complex networks have been automatically inferred using both tree-based [29, 30, 31] and LinkableGP [38] approaches, the motivations for selecting the LinkableGP system were twofold. Firstly, due to the object-oriented nature of LinkableGP, distinct areas of the evolved program can be logically separated into methods which are simultaneously evolved. Each of these methods can be provided with its own language elements similar to the concept of automatically defined functions (ADFs) [33], thus eliminating the need for type-manipulations to control the program flow. Furthermore, each method is only provided language elements related to its intended functionality, and so the search space is not muddled by language elements which are irrelevant to the current context. The second motivation was provided by previous work (co-authored as part of this thesis) [38] whereby a generalized graph model was proposed and demonstrated to be promising for the evolution of graph models for complex networks.

The evolutionary capabilities of the LinkableGP system are provided by the use of a typical, generational genetic algorithm which performs crossover, mutation, elitism, and selection operations. However, the crossover and mutation operations were modified to work with the DNA structure. A proportion of the best performing individuals from the previous generation are directly copied to the new population, via elitism, while the remainder of the new population results from the offspring of genetic operations.

7.1.1 Genetic Operators

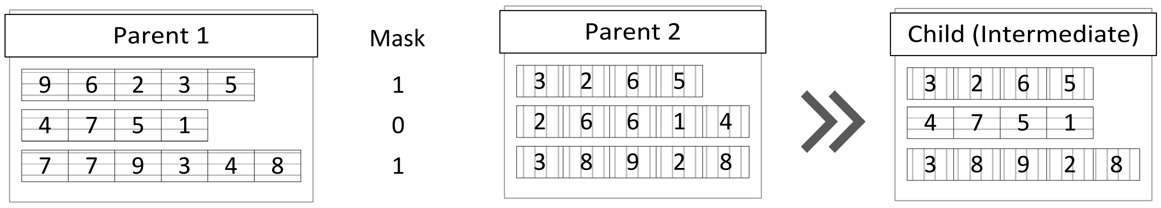
Crossover in LinkableGP, visualized in Figure 7.1, is a two phased operation. The mating phase selects two parents using any standard selection operator (tournament selection was used in this work) and constructs a random bit-mask to determine the chromosomes which are to be initially inherited from each parent. Each bit in the randomly generated bit-mask acts a parent discriminator, i.e., the chromosome at index i is selected from parent 1 or parent 2 based upon whether the bit at index i is a 0 or 1. The mating phase is depicted in Figure 7.1a. Although this phase provides a functional offspring, only the inheritance portion of crossover is accomplished. The recombination occurs in the second phase.

Following the mating phase, mixing occurs on each chromosome of the child with probability ρ. During the mixing phase, a chromosome in which the mixing occurs is replaced by the offspring resulting from a one-point crossover operation between the corresponding parent chromosomes. First, a random cut-point is selected. The mixing phase then proceeds in one of two ways, with equal probability: 1) the genetic material up to the cut-point is taken from parent 1 while the genetic material after the cut-point is taken from parent 2, or, 2) the genetic material up to the cut-point is taken from parent 2 while the genetic material after the cut-point is taken from parent 1. The mixing phase is depicted in Figure 7.1b.

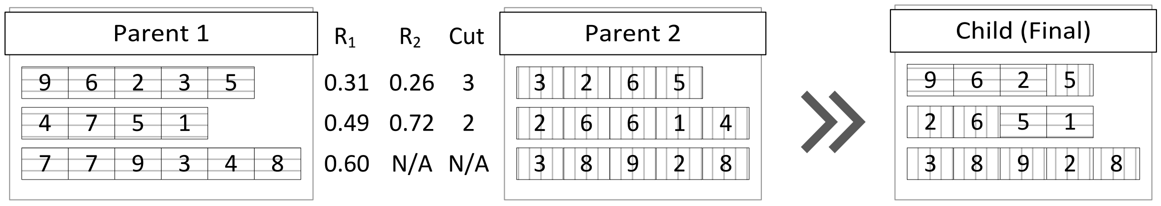
When mutation was applied to an individual, a mutation operation was performed to only one of the individuals chromosomes, selected at random. Mutation was applied according to the mutation rate in one of two places throughout the evolutionary process. Each individual selected via the selection process, for which crossover is not applied, had a chance of being mutated before it became a part of the new population. Similarly, the mutation operator could be invoked on the offspring of the crossover operation before they were placed in the new population. The mutation operator used by the LinkableGP system had two possible operations, *resize* or *random reset*, which could be performed when mutation was invoked. Each mutation operation had an equal probability of being selected. The resize mutation allowed the chromosome length to be altered, by either increasing or shrinking the length of the chromosome. The random reset mutation would select a random location within the chromosome, and regenerate the value at this location.

7.2 Representation

LinkableGP is a linear, object-oriented GP system which makes use of a user-supplied abstract class to guide the search process. As such, an abstract class representing a generalized graph model was provided to define the structure of the evolved models. Each abstract method within the supplied abstract class was evolved by the GP system, ultimately resulting in a fully-functional, standalone graph model object. To guide the evolution of graph models, LinkableGP was provided with the generalized graph model shown in Algorithm 9. This generalized algorithm was con-



(a) Phase 1: Mating. A mask value of 0 denotes the corresponding chromosome is inherited from parent 1 while a mask value of 1 denotes the chromosome is inherited from parent 2.



(b) Phase 2: Mixing, with $\rho = 0.5$. $R_1 < \rho$ denotes the mixing phase occurs. $R_2 < 0.5$ denotes the first portion of the genetic material is taken from parent 1, while $R_2 \geq 0.5$ denotes the first portion of genetic material is taken from parent 2.

Figure 7.1: Visualization of the two-phase crossover operation in the LinkableGP system applied to a three-chromosome individual. Genetic material is patterned according to the parent to visually aid in following where the genetic material in the offspring is inherited.

structed to be a more robust version of the initial generalized algorithm proposed by Medland *et al.* [38]. Algorithm 9 depicts three abstract methods, namely *SelectVertices*, *CreateEdges*, and *SecondaryActions*, each of which were evolved using GP. The *SelectVertices* method was responsible for selecting potential target vertices to which the newly created vertex could form an edge. The methods *CreateEdges* and *SecondaryActions* were invoked for each vertex selected by *SelectVertices*. The *CreateEdges* method was responsible for creating edges to be added to the graph while *SecondaryActions* was responsible for adding additional vertices to the collection, S, and provided the means to generate recursively defined construction mechanisms. The functions and terminals which were made available to each of these methods are presented in Section 7.3.

7.3 Genetic Programming Language

Each of the three methods to be evolved by the GP system were assigned their own set of language elements. The functions and terminals provided to each of these abstract methods are described in detail below. All operations were strongly typed and, contrary to most linear GP systems, no grammar is required.

Algorithm 9 Generalized Graph Model

```
function GENERATE($t$)
    $g \leftarrow$ EmptyGraph()
    for $i$ from 1 to $t$ do
        $v \leftarrow$ NewVertex()        // Create a new vertex
        $S \leftarrow$ SelectVertices($g$)
        ADDEDGES($g, v, S$)        // Execute the AddEdges subroutine, see below
        $g$.AddVertex($v$)        // Added after processing to prevent self edges
    end for
end function

function ADDEDGES($g, newVertex, S$)
    while $S$ is not empty do
        $v \leftarrow S$.GetNext()        // Get and remove the next element from the collec-
tion
        $E \leftarrow$ CreateEdges($newVertex, v$)
        SecondaryActions($v, S$)
        for all $e \in E$ do
            $g$.AddEdge($e$)
        end for
    end while
end function
```

7.3.1 SelectVertices Method

This section describes the language elements which were made available to the *SelectVertices* method. The current graph being constructed was provided as an argument to this method and thus was available as a terminal to the functions in this language. For each of the methods listed below, a specialized, *taboo* collection was returned in either stack or queue form. That is, each method below had two formulations – one which returned a stack and one which returned a queue. The collections are referred to as *taboo* collections because each vertex added to the collection was also added to a taboo list, preventing it from being added to the collection a second time.

- **TabooVertexCollection GetAll{Stack, Queue}(g)** – Adds all vertices from g to a taboo stack or queue.
- **TabooVertexCollection GetRandom{Stack, Queue}(g)** – Adds a random vertex from g to a taboo stack or queue.
- **TabooVertexCollection GetRandom{Stack, Queue}(g, n)** – Adds n ran-

dom vertices from g to a taboo stack or queue.

- **TabooVertexCollection GetRoulette{Stack, Queue}(g, f)** – Adds a vertex from g, selected via roulette selection, to a taboo stack or queue where a vertex evaluator function, *f*, is used to assign the probability of selection for each vertex.

- **TabooVertexCollection GetRoulette{Stack, Queue}(g, f, n)** – Adds n vertices from g, selected via roulette selection, to a taboo stack or queue where a vertex evaluator function, *f*, is used to assign the probability of selection for each vertex.

The following vertex evaluator functions were available to the selection process. Each method below returns a function of type Vertex $\rightarrow$ Double, denoted $\mathrm{F}_{\mathrm{V}\rightarrow\mathrm{D}}$, for use in the selection functions above. That is, the method returns a function which takes a vertex and returns a real value.

- **$\mathbf{F}_{\mathbf{V}\rightarrow\mathbf{D}}$ GetDegree( )** – Returns a function which computes the degree of the vertex.

- **$\mathbf{F}_{\mathbf{V}\rightarrow\mathbf{D}}$ GetLocalTransitivity( )** – Returns a function which computes the local transitivity ratio of the vertex.

- **$\mathbf{F}_{\mathbf{V}\rightarrow\mathbf{D}}$ GetAge( )** – Returns a function which computes the age of the vertex. The age is defined as the number of iterations which have passed since the vertex was added to the graph.

The evaluator functions allow for much more robust selection procedures to be carried out. For example, the use of *GetRouletteQueue*(g, f), where f is the *GetDegree*( ) function, would lead to a preferential attachment mechanism similar to the one seen in the Barabasi-Albert model [35]. The following arithmetic operations, acting upon evaluator functions, were also provided. Each operation below would take one or more functions and return another function.

- **$\mathbf{F}_{\mathbf{V}\rightarrow\mathbf{D}}$ Add(f1, f2)** – Returns a function which computes the sum of *f1* and *f2* evaluated on a vertex, i.e., $f_1(v) + f_2(v)$.

- **$\mathbf{F}_{\mathbf{V}\rightarrow\mathbf{D}}$ Add(f, d)** – Returns a function which computes *1* evaluated on a vertex and adds a floating-point value *d*, i.e., $f(v) + d$.

- **$F_{V \to D}$ Mult(f1, f2)** – Returns a function which computes the product of *f1* and *f2* evaluated on a vertex, i.e., $f_1(v) \times f_2(v)$.
- **$F_{V \to D}$ Mult(f, d)** – Returns a function which computes f evaluated on a vertex and multiplies by the floating-point value d, i.e., $f(v) \times d$.
- **$F_{V \to D}$ Pow(f, d)** – Returns a function which computes f evaluated on a vertex to the power of a floating-point value d, i.e., $f(v)^d$.
- **$F_{V \to D}$ InversePow(f, d)** – Returns a function which computes f evaluated on a vertex to the inverse power of a floating-point value d, i.e., $f(v)^{-d}$.

In addition to the above functions, two constant generators, used to produce terminal values, were provided – one generated integers between 1 and 10 while the other produced floating point values between 0.0 and 1.0, both inclusive.

7.3.2 CreateEdges Method

This section describes the language elements made available in the *CreateEdges* method. The *CreateEdges* method had two parameters made available as terminals, namely the newly constructed vertex and the vertex from the taboo collection currently being considered for connection. Each of the functions below returned a list of edges, which were added to the graph after the secondary selection took place. Edges were not created immediately to prevent interference with the secondary vertex selection.

- **List⟨Edge⟩AddEdge(v1, v2)** – Return a list with a single edge between *v1* and *v2*.
- **List⟨Edge⟩EmptyEdge( )** – A special function to signify that no edge was to be created.
- **List⟨Edge⟩AddEdgeWithProbability(v1, v2, p)** – With probability p, return a list with a single edge between *v1* and *v2*. Otherwise, an empty list is returned.
- **List⟨Edge⟩AddTriangle(v1, v2)** – Returns a list with two edges forming a triangle, if possible, which includes both *v1* and *v2*. The first edge in the list is between *v1* and *v2*. If *v2* has any neighbors, the second edge is between *v1* and a randomly selected neighbor of *v2*. If *v2* has no neighbors, a randomly

selected neighbor of *v1* is chosen, if any, and an edge is created between this neighbor and *v2*. In the event that neither *v1* nor *v2* have neighbors, only the edge between *v1* and *v2* is returned.

- **List⟨Edge⟩AddTriangleWithProbability(v1, v2, p)** – Returns a list with an edge between *v1* and *v2*. With probability *p*, attempts to create a second edge, forming a triangle, in the same fashion as *AddTriangle*.
- **List⟨Edge⟩Duplicate(v1, v2, p)** – Returns a list of edges between *v1* and each neighbor of *v2*. With probability *p*, an edge is also created between *v1* and *v2*.

A random number generator which produced floating-point values between 0.0 and 1.0, inclusive, was also provided to generate the probability parameters, i.e., terminals, used in the various functions.

7.3.3 SecondaryActions Method

The *SecondaryActions* method was responsible for performing actions as a direct result of adding a vertex and/or edge(s). This method had two arguments available as terminals, namely the current vertex being considered for connection and the (taboo) collection of vertices for future consideration. For the purpose of this work, this procedure was used only to add additional vertices to the taboo collection using the function below. However, for more difficult inference problems, such as with weighted networks, the *SecondaryActions* procedure could be used to alter connection weights of existing edges when a new edge is added, analogous to diverting some traffic to the new edge.

- **void AddNeighbours(c, n, v)** – Adds *n* randomly selected neighbors of vertex *v* to the taboo collection *c*. This method had no return type.

To provide the number of neighbors, the following random integer providers were available alongside two constant generators used to produce terminal values – one which generated integers between 0 and 10 while the other produced floating point values, used as probabilities, between 0.0 and 1.0, both inclusive.

- **Integer GetRandomValue(a, b)** – Returns a uniformly random integer between *a* and *b*, inclusive, where *a* and *b* are integer arguments.
- **Integer GetGeometricValue(p)** – Returns an integer generated according to a geometric distribution with probability *p*.

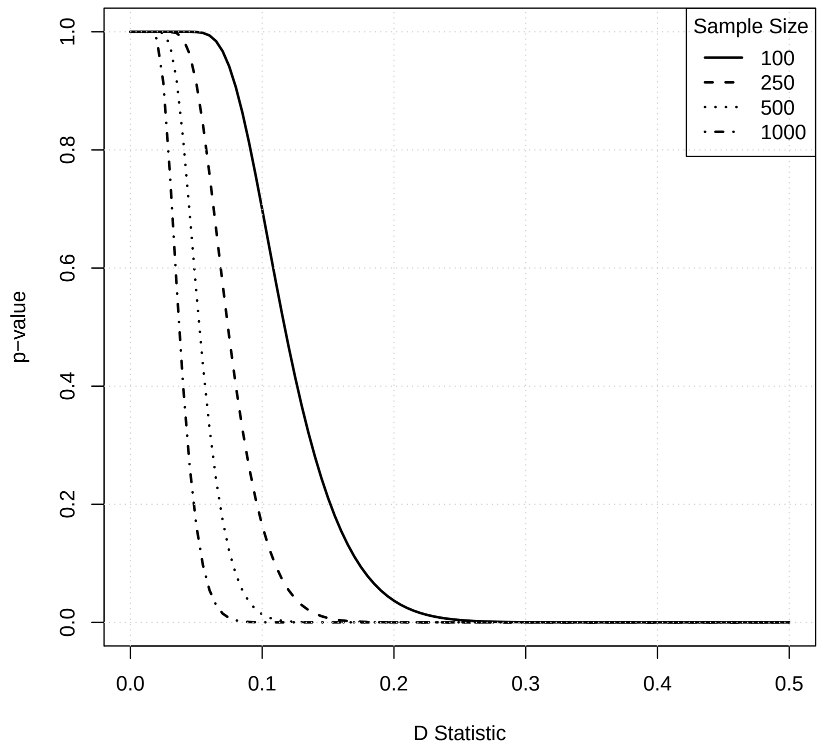


Figure 7.2: Plot of the KS test D statistic vs. the resulting p-value for various sample sizes, assuming $n_1 = n_2$.

7.4 Fitness Evaluation

Figure 7.2 depicts the p-value of a KS test as a function of the D statistic for various sample sizes (assuming $n_1 = n_2$). From this plot, it was clear that using the p-value as a fitness measure can be problematic. While the p-value gives a stronger indication of (dis)similarity, the range of possible values is limited. When such p-values are used as during fitness evaluation, the resulting landscape has very little gradient to guide the search which could cause poor performance. As Figure 7.2 demonstrates, the p-value approaches 0 quite rapidly relative to the D statistic; even on the smallest, 100 vertex networks, the p-value becomes negligible when the D statistic is roughly 0.3 (i.e. a 30% difference between the empirical cumulative distribution functions (ECDFs)). In terms of evolution, this means that a difference between the ECDF of a target network and an evolved network of 30% would be nearly indistinguishable from a difference of 100%.

Using the results from Chapters 5 and 6 alongside the above results, the following three fitness measures were used during the evolutionary process.

f_1 average KS test statistic comparing the degree distributions of the target graph and each of the generated graphs.

f_2 average KS test statistic comparing the betweenness centrality distributions of the target graph and each of the generated graphs.

f_3 average KS test statistic comparing the PageRank of the target graph and each of the generated graphs.

Chapter 8

Evolving Known Graph Models

In general, when inferring a graph model for a complex network, the graph model used to construct the network is not known – if the model was known, why would one attempt to infer it? The answer is to allow the results of the inference methodology to easily validated against a known model. By evolving a graph model for a known algorithm, the evolved model can be easily validated against both the network used as a target, referred to as the *target graph*, as well as other networks generated by the mode. This allows for much stronger conclusions to be made regarding the evolutionary capabilities of the system. As such, this chapter discusses the evolution of four known graph models.

Four of the previously examined graph models were used to generate target graphs for the evolutionary process. The models and reasoning for selection is as follows. The growing random model (GR) was selected due to its trivial construction process, i.e., it was selected as a "toy" problem because of its simplicity. The Barabasi-Albert (BA) model was selected as it exhibits a non-trivial method of vertex selection, namely the preferential attachment mechanism. In contrast to the BA and GR models, the Erdos-Reyni (ER) model does not create a static number of edges each iteration. That is, the number of edges created each iteration is dependent upon the (constant) probability of connection between vertices. This probability constant depicts another motivation for including the ER model. Accurately evolving the ER model would require the evolution of a floating-point constant, a known difficulty for traditional GP systems [32] which has been explicitly addressed by the GP system used in this work. Finally, the Forest-Fire model was selected for evolution as it exhibits a much more complex construction mechanism than the other three models. To correctly model the "burning" phenomena of the FF model, an evolved model would require the use of the *SecondaryActions* method. Furthermore, the FF model is known to

Table 8.1: GP Parameters.

Parameter	Value
Runs	30
Population Size	50
Generations	50
Crossover Rate	80%
Mutation Rate	20%
Elitism	2 Individuals
Selection	Tournament, k=3
Samples for Evaluation	1
Initial Chromosome Length	*SelectVertices* – 1 to 15
	CreateEdges – 1 to 5
	SecondaryActions – 1 to 15

exhibit community structure [36].

For each of the above models, a target graph was generated with 100, 250, and 500 vertices, producing twelve target graphs. The parameters used to generate the target networks were the same as used in Chapter 5. For each target graph, the GP system was run 30 times to produce a set of candidate models. The empirically-determined parameters used during the GP evolution process are given in Table 8.1. Following recent work involving the automatic inference of a graph model for the BA model [38], the highest ranked model from the set of candidate models, selected using sum of ranks, was taken as the final model. Further post-validation was then performed to evaluate the performance of the final model. In all cases where KS test results are presented, bold entries denote a D statistic below the corresponding critical threshold given in Table 5.1, i.e., the KS test determined the distributions were insignificantly different. For the interested reader, Appendix C provides more detailed results from the GP system in the form of convergence plots and the raw GP output for 250 vertex experiments. The remainder of this chapter outlines the results of the above experiments.

8.1 Evolving the Growing Random Model

A target graph was generated by the growing random model with each of 100, 250, and 500 vertices. Table 8.2 presents a summary of the best evolved models from each of the 30 runs. When the evolved solution fitnesses, demonstrated in Table 8.2, were compared to the expected average D statistics, shown in Tables 5.2, 5.4, and 5.6, respectively, the evolved models clearly depicted that the GR model was consistently

Table 8.2: Summary of all 30 models evolved against a Growing Random graph.

Target Size	Measure	Min	μ	Max	σ
100	Degree	**0.000**	**0.012**	**0.030**	0.005
	Betweenness	**0.020**	**0.033**	**0.040**	0.005
	PageRank	**0.040**	**0.054**	**0.070**	0.006
250	Degree	**0.004**	**0.017**	0.156	0.027
	Betweenness	**0.012**	**0.019**	**0.040**	0.005
	PageRank	**0.032**	**0.042**	0.168	0.024
500	Degree	**0.004**	**0.016**	0.114	0.025
	Betweenness	**0.010**	**0.026**	0.292	0.052
	PageRank	**0.020**	**0.037**	0.166	0.031

replicated with a high degree of accuracy. The best evolved models for each network size are presented in Algorithms 10 to 12, respectively. Due to the simplicity of the GR model, it was trivial to see that the evolved model for all 3 network sizes exhibited identical behavior to that of the GR model. However, there were slight variations on the construction process for each network size. Nonetheless, the evolved models each selected a single vertex at random and connected to this vertex – exactly as the GR model is defined.

The evolved GR_{100} model, shown in Algorithm 10, randomly selected a single vertex using a stack and constructed an edge from the selected vertex to the new vertex, which due to the lack of directionality in undirected networks, made no difference whatsoever in terms of the network structure. Table 8.3 presents the post analysis statistics when 30 networks generated by the GR_{100} model were compared to 30 networks generated by the GR model, $n = 100$. Post-analysis results further demonstrated the similarity of the evolved and GR models. For all centrality measures examined, the average KS statistic was well below the critical threshold of 0.19233. Furthermore, the maximum observed D statistic for all measures was also below the critical threshold. Note that while the average D statistic when comparing the evolved and GR models was significantly higher than that of the degree and betweenness measures, the observed average D statistic (0.103) was in line with the expected value of 0.105, as shown in Table 5.6a. Another notable difference between the evolved model and the true model was the largest observed average geodesic path length. The largest path of the evolved model was significantly higher than that of the true model. However, a network with such a high average geodesic path length is clearly anomalous as this value is nearly 3 standard deviations from the mean. To demonstrate the visual similarity of the evolved and target networks, Figure 8.1

Table 8.3: Comparison of 100 vertex graphs generated by the GR_{100} model and the Growing Random model. Values displayed are calculated over 30 generated graphs.

	Measure	Min	μ	Max	σ
GR_{100}	Edges	99	99	99	0
	AGP	5.666	6.569	7.939	0.463
	CC	0	0	0	0
Growing Random	Edges	99	99	99	0
	AGP	5.703	6.454	7.326	0.426
	CC	0	0	0	0
Average D Statistic	KS Deg	**0.010**	**0.004**	**0.080**	0.017
	KS Bet	**0.030**	**0.058**	**0.090**	0.016
	KS PR	**0.060**	**0.103**	**0.170**	0.031

depicts a network generated by the evolved model alongside the target network.

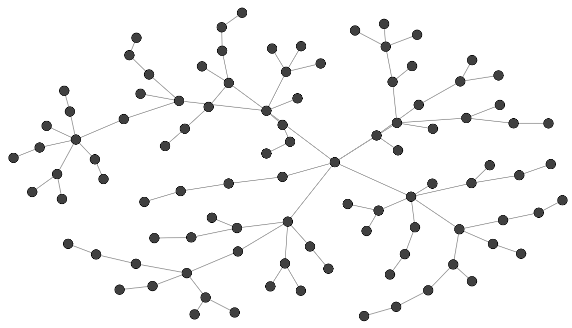
Algorithm 10 Simplified GR_{100} Model

```
function SelectVertices($g$)
    $S \leftarrow$ GetRandomStack($g$)
    return $S$
end function

function CreateEdges($newVertex, v$)
    $E \leftarrow$ AddEdge($v, newVertex$)
    return $E$
end function

function SecondaryActions($v, S$)
        // Nothing is performed in SecondaryActions
end function
```

The evolved GR_{250} model, shown in Algorithm 11, used a stack to select a random vertex and formed an edge from the new vertex to the selected vertex. The post analysis results comparing networks generated by the GR_{250} and GR models, presented in Table 8.4, demonstrated the basic structural properties were insignificantly different. While the average KS statistic for each centrality measure was well below the critical threshold of 0.12164, the maximum observed statistic for the PageRank was above this critical threshold. However, obtaining a D statistic for PageRank which was higher than the threshold would be somewhat rare as the critical threshold was roughly 2.5 standard deviations above the mean D statistic observed for PageRank. A network generated by the evolved model alongside the target, shown in Figure 8.2, provides a visual reinforcement of the similarity already noted analytically.


(a) GR_{100}

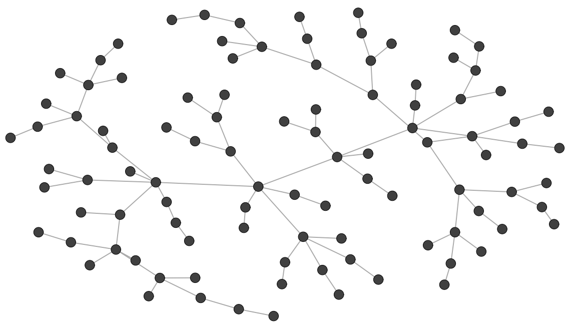

(b) Growing Random target, $n = 100$

Figure 8.1: Network generated by the GR_{100} model and the Growing Random target network, $n = 100$.

Algorithm 11 Simplified GR_{250} Model

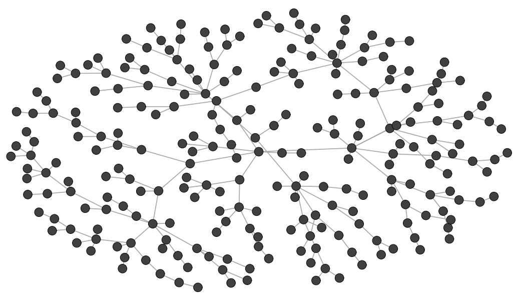
```
function SelectVertices($g$)
    $S \leftarrow$ GetRandomStack($g$)
    return $S$
end function

function CreateEdges($newVertex, v$)
    $E \leftarrow$ AddEdge($newVertex, v$)
    return $E$
end function

function SecondaryActions($v, S$)
            // Nothing is performed in SecondaryActions
end function
```

Table 8.4: Comparison of 250 vertex graphs generated by the GR_{250} model and the Growing Random model. Values displayed are calculated over 30 generated graphs.

	Measure	Min	μ	Max	σ
GR_{250}	Edges	249	249	249	0
	AGP	7.418	8.422	10.064	0.541
	CC	0	0	0	0
Growing Random	Edges	249	249	249	0
	AGP	7.353	8.233	10.103	0.702
	CC	0	0	0	0
Average D Statistic	KS Deg	**0.012**	**0.027**	**0.064**	0.011
	KS Bet	**0.020**	**0.041**	**0.064**	0.011
	KS PR	**0.036**	**0.069**	0.124	0.022


(a) GR_{250}

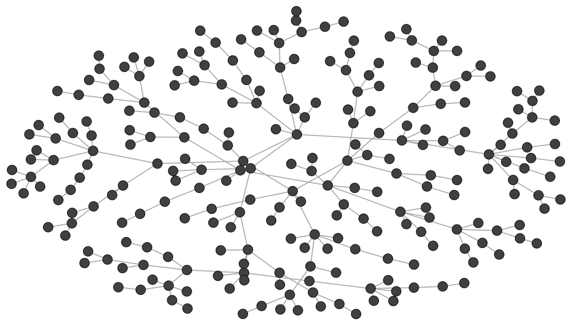

(b) Growing Random target, $n = 250$

Figure 8.2: Network generated by the GR_{250} model and the Growing Random target network, $n = 250$.

Finally, the evolved GR_{500} model, shown in Algorithm 12, made use of a queue structure which explicitly requested for 1 random vertex which was used as a target for an edge from the newly created vertex. Note that the GR_{500} model actually evolved the constant 1 to use when requesting vertices rather than using the implicit single vertex selector. Table 8.5 presents the post-analysis results of this experiment. Networks generated by the GR_{500} model were, on average, insignificantly different that those produced by the Growing Random model when the degree, betweenness, and PageRank measures were compared. In a similar fashion to that of the GR_{250} model, there were instances of the GR_{500} model which would be considered significantly different with respect to the PageRank when compared to true Growing Random networks. Figure 8.3 depicts a network generated by the GR_{500} model alongside the target network.

Algorithm 12 Simplified GR_{500} Model

```
function SelectVertices($g$)
    $S \leftarrow$ GetRandomQueue($g, 1$)
    return $S$
end function

function CreateEdges($newVertex, v$)
    $E \leftarrow$ AddEdge($newVertex, v$)
    return $E$
end function

function SecondaryActions($v, S$)
        // Nothing is performed in SecondaryActions
end function
```

Table 8.5: Comparison of 500 vertex graphs generated by the GR_{500} model and the Growing Random model. Values displayed are calculated over 30 generated graphs.

	Measure	Min	μ	Max	σ
GR_{500}	Edges	499	499	499	0
	AGP	8.831	9.816	11.020	0.513
	CC	0	0	0	0
Growing Random	Edges	499	499	499	0
	AGP	8.572	9.731	10.891	0.549
	CC	0	0	0	0
Average D Statistic	KS Deg	**0.008**	**0.021**	**0.052**	0.011
	KS Bet	**0.016**	**0.030**	**0.054**	0.010
	KS PR	**0.030**	**0.051**	0.094	0.016

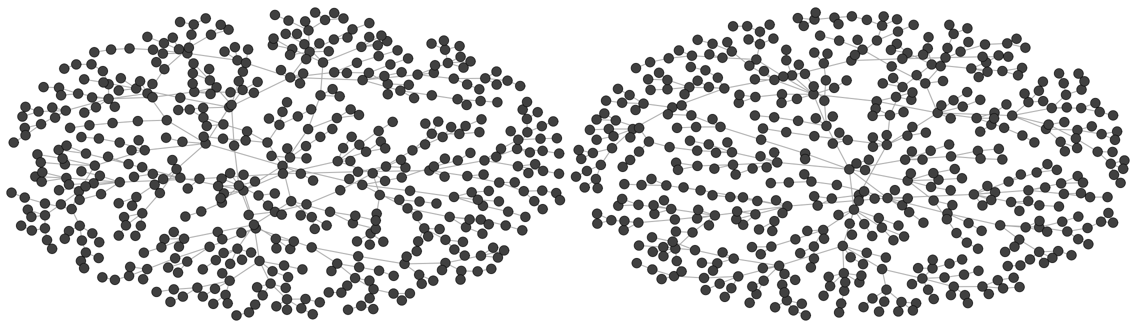

(a) GR_{500} (b) Growing Random target, $n = 500$

Figure 8.3: Network generated by the GR_{500} model and the Growing Random target network, $n = 500$.

The results in this section clearly indicated that the GR model was able to be both consistently and accurately inferred automatically by the GP system, regardless of the target size. While simplistic in principle, this experiment provided experimental verification that a correct model could be inferred using the generalized graph model, GP language, and fitness functions proposed for the experiments in this work. The following sections depict the evolution of graph models which exhibit non-trivial construction mechanisms like that of the GR model.

8.2 Evolving the Barabasi-Albert Model

This section describes the experiments performed using graphs generated by the BA model as the target networks. Experiments were performed using 100, 250, and 500 vertex networks, respectively, as the target for evolution. Table 8.6 gives the summary of the best evolved models from each of the 30 runs while Algorithms 13 to 15 present the best evolved model from each experiment. Manual inspection of the evolved models demonstrated that each evolved model made use of a single vertex selector in conjunction with the **GetDegree** vertex evaluator function, which almost perfectly replicated the preferential attachment behavior of the BA model. In contrast to the results from the GR model, the evolved models using a BA target were not always fit models. More specifically, the average D statistics from these experiments indicated that a fit model was not produced during each run for the 250 and 500 vertex networks. While the fitness values (i.e., the maximum observed values) were always below the critical threshold the 100 vertex experiments, this was not the case for 250 and 500 vertex experiments. Using a 250 vertex network as a target, networks produced by evolved model were, on average, insignificantly different than networks generated by the BA model while this was only the case with betweenness when 500 networks were considered. Furthermore, the maximum D statistic observed on both 250 and 500 vertex experiments was significantly above the critical threshold, indicating that unfit models were produced during evolution. Nonetheless, the following results indicate that a fit model was produced for each of the target networks.

The best evolved when a 100 vertex graph generated by the BA model was used as a target is presented in Algorithm 13. This model exemplified the use of the **GetDegree** vertex evaluator function in conjunction with a single vertex, stack-based, selector and a single edge formed between the new and selected vertices. Table 8.7 shows the results of the post-analysis procedure used to compare networks from

Table 8.6: Summary of all 30 models evolved against a Barabasi-Albert graph.

Target Size	Measure	Min	μ	Max	σ
100	Degree	**0.010**	**0.045**	**0.070**	0.018
	Betweenness	**0.020**	**0.062**	**0.110**	0.027
	PageRank	**0.070**	**0.105**	**0.180**	0.025
250	Degree	**0.008**	**0.037**	0.176	0.045
	Betweenness	**0.012**	**0.035**	0.124	0.032
	PageRank	**0.060**	**0.088**	0.260	0.051
500	Degree	**0.006**	0.126	0.660	0.130
	Betweenness	**0.010**	**0.085**	0.214	0.056
	PageRank	**0.046**	0.171	0.344	0.089

the BA model to networks from the evolved model. The differences between the evolved model and the BA model were minimal. The mean and minimum observed average geodesic path lengths were slightly higher for the evolved model than with the true model, but were within a single standard deviation in both cases. Similarly, the average D statistic for the PageRank measure was seemingly high, however, this value was both below the critical threshold and corresponded precisely with the expected value as seen in Chapter 5. The remaining centrality measures, namely degree and betweenness, demonstrated average D statistics well within the expected values. By definition, the transitivity of both the evolved and BA models was always zero. For a visual comparison, a network generated by the evolved model and the BA model are presented in Figure 8.4. Both networks depict a strongly connected, central hub node along with a tree-like, branching structure.

Algorithm 13 Simplified *BA_{100}* Model

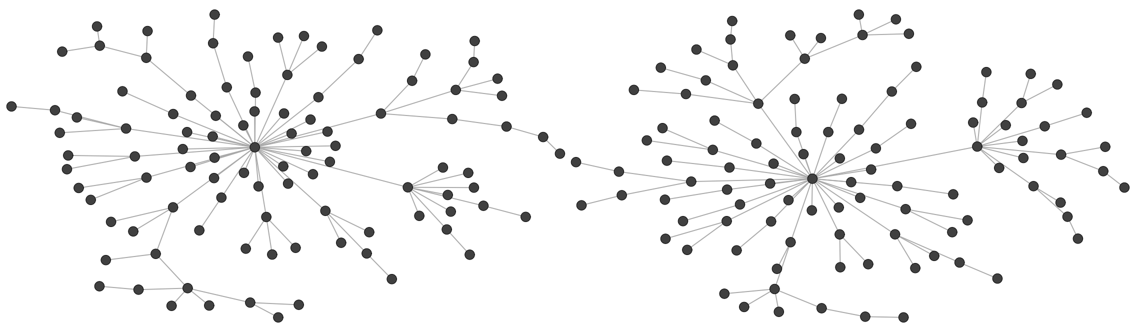
```
function SELECTVERTICES($g$)
    $f \leftarrow$ GetDegree( )
    $S \leftarrow$ GetRouletteStack($g, f$)
    return $S$
end function

function CREATEEDGES($newVertex, v$)
    $E \leftarrow$ AddEdge($newVertex, v$)
    return $E$
end function

function SECONDARYACTIONS($v, S$)
            // Nothing is performed in SecondaryActions
end function
```

Table 8.7: Comparison of 100 vertex graphs generated by the BA_{100} model and the Barabasi-Albert model. Values displayed are calculated over 30 generated graphs.

	Measure	Min	μ	Max	σ
	Edges	99	99	99	0
BA_{100}	AGP	3.947	4.806	5.843	0.481
	CC	0	0	0	0
	Edges	99	99	99	0
Barabasi-Albert	AGP	3.595	4.481	5.842	0.522
	CC	0	0	0	0
	Degree	**0.020**	**0.053**	**0.100**	0.020
Average D Statistic	Betweenness	**0.030**	**0.066**	**0.140**	0.024
	PageRank	**0.090**	**0.159**	0.300	0.043



(a) BA_{100} (b) Barabasi-Albert target, $n = 100$

Figure 8.4: Network generated by the BA_{100} model and the Barabasi-Albert target network, $n = 100$.

Algorithm 14 presents the evolved model when a 250 vertex network generated by the BA model was used as a target. As expected, the BA_{250} model made use of the **GetDegree** evaluator alongside a single vertex selector and a standard edge creator. Despite the small algorithmic differences, namely the selection container and explicit vs. implicit number of nodes to be selected, the BA_{100} and BA_{250} models behaved identically. The post-analysis results comparing the BA_{250} model to the true BA model are presented in Table 8.8. As observed with smaller networks, the average path lengths were slightly higher with the evolved networks than with the BA networks. Each of the centrality measures were, on average, insignificantly different between networks generated by the evolved and true models. Furthermore, the D statistics observed when comparing evolved networks to BA networks closely matched those expected when comparing networks from the BA model. Figure 8.5 depicts a network generated by the evolved model alongside the target network.

Algorithm 14 Simplified BA_{250} Model

```
function SelectVertices($g$)
    $f \leftarrow$ GetDegree( )
    $S \leftarrow$ GetRouletteQueue($g, 1, f$)
    return $S$
end function

function CreateEdges($newVertex, v$)
    $E \leftarrow$ AddEdge($v, newVertex$)
    return $E$
end function

function SecondaryActions($v, S$)
            // Nothing is performed in SecondaryActions
end function
```

The best model evolved when a 500 vertex BA network was used as a target is presented in Algorithm 15. The evolved model, like those evolved for smaller networks, **GetDegree** and single vertex selector to replicate the preferential attachment mechanism from the BA model. Post-analysis results for the BA_{500} model are given in Table 8.9. Due to the functionally identical models, the observations made for the BA_{500} model are in line with the observations for smaller networks. Namely, the average geodesic path length was slightly, and reasonably, higher for the evolved model and the centrality measures were nearly identical to those observed in Chapter 5 when networks from the BA model were compared. However, with 500 vertex networks, the maximum path length for the evolved model was quite a bit higher than observed

Table 8.8: Comparison of 250 vertex graphs generated by the BA_{250} model and the Barabasi-Albert model. Values displayed are calculated over 30 generated graphs.

	Measure	Min	μ	Max	σ
BA_{250}	Edges	249	249	249	0
	AGP	4.715	5.558	6.969	0.488
	CC	0	0	0	0
Barabasi-Albert	Edges	249	249	249	0
	AGP	4.326	5.610	6.627	0.539
	CC	0	0	0	0
Average D Statistic	KS Deg	**0.008**	**0.032**	**0.088**	0.016
	KS Bet	**0.012**	**0.035**	**0.088**	0.017
	KS PR	**0.064**	**0.104**	0.172	0.027

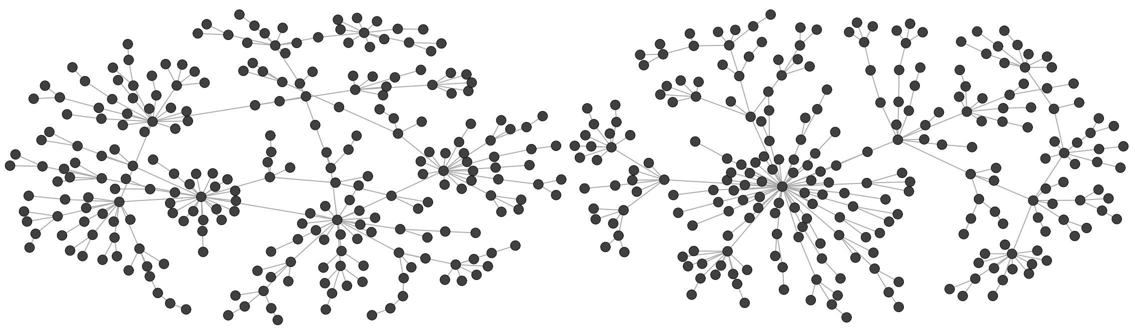

(a) BA_{250} (b) Barabasi-Albert target, $n = 250$

Figure 8.5: Network generated by the BA_{250} model and the Barabasi-Albert target network, $n = 250$.

Table 8.9: Comparison of 500 vertex graphs generated by the BA_{500} model and the Barabasi-Albert model. Values displayed are calculated over 30 generated graphs.

	Measure	Min	μ	Max	σ
BA_{500}	Edges	499	499	499	0
	AGP	5.497	6.353	7.640	0.560
	CC	0	0	0	0
Barabasi-Albert	Edges	499	499	499	0
	AGP	5.261	6.092	6.864	0.434
	CC	0	0	0	0
Average D Statistic	KS Deg	**0.010**	**0.023**	**0.050**	0.011
	KS Bet	**0.012**	**0.026**	**0.050**	0.009
	KS PR	**0.050**	**0.073**	0.114	0.016

with the true model. For a visual comparison, Figure 8.6 presents a network generated by the BA_{500} model along with the target network. Visual comparison depicts an extremely similar structure between the networks.

Algorithm 15 Simplified BA_{500} Model

```
function SelectVertices($g$)
    $f \leftarrow$ GetDegree( )
    $S \leftarrow$ GetRouletteStack($g, f$)
    return $S$
end function

function CreateEdges($newVertex, v$)
    $E \leftarrow$ AddEdge($v, newVertex$)
    return $E$
end function

function SecondaryActions($v, S$)
        // Nothing is performed in SecondaryActions
end function
```

The difference in average geodesic path lengths was noted for both 250 and 500 vertex networks. While individually, these result could most likely be attributed to statistic chance, combined they seemed to suggest there was a difference between the evolved models and the true model. This lead to a more detailed manual inspection of the evolved models which determined an extremely subtle, yet noteworthy difference between the evolved models and the BA model being uncovered. The difference was attributed, somewhat surprisingly, to the fact that none of the evolved models carried the notion of a zero-degree appeal. Intuitively, one would assume that the

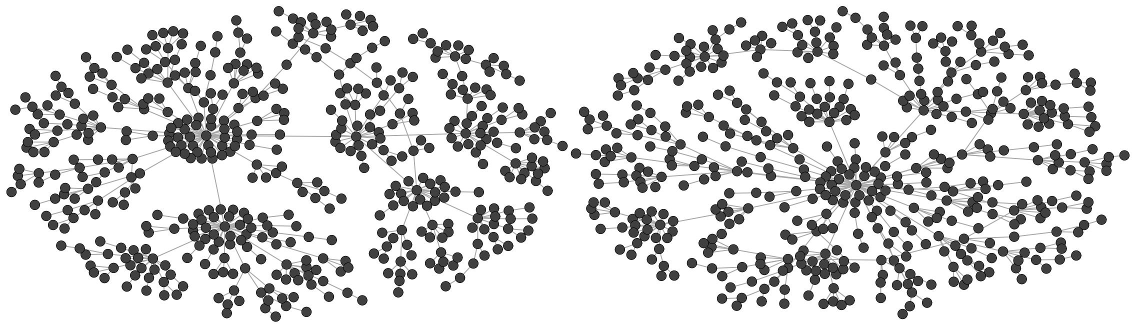

(a) BA_{500} (b) Barabasi-Albert target, $n = 500$

Figure 8.6: Network generated by the BA_{500} model and the Barabasi-Albert target network, $n = 500$.

zero-degree appeal would have no effect on undirected networks as the notion of in-degree would naturally be expected to deteriorate to simply degree. The degree of a vertex in the BA model will always be non-zero as a vertex which is added to the graph is immediately connected to another vertex and, thus, always has a non-zero degree. Believed to be an implementation specific detail, this assumption does not hold for *igraph* which uses the notion of in-degree (i.e., a concept of directed networks) even when constructing undirected networks. While this subtle difference between the target and evolved models did not cause any significant difference in network structure or behavior, the observation was noted as it does depict a known difference in models.

8.3 Evolving the Erdos-Reyni Model

This section describes the automatic inference of the Erdos-Reyni model. This model is of particular difficulty as correctly modeling the Erdos-Reyni model would require the evolution of a floating-point constant, a known difficulty for traditional GP systems [32]. Note that the GP system used in this work directly facilitates the evolution of floating-point constants, not to be confused with the notion of an ephemeral random constant, which is a substantial difference between this study and previous work on the automatic inference of graph models [29]. A summary of the best evolved models is given in Table 8.10, which depicts that the ER model was significantly more difficult to evolve than the previously examined models. When 100 vertex networks were considered, nearly all the evolved models were insignificantly different with respect to the degree, betweenness, and PageRank measures. Note that the maximal D statistic observed in the betweenness distribution was significantly different than

Table 8.10: Summary of all 30 models evolved against an Erdos-Reyni graph.

Target Size	Measure	Min	μ	Max	σ
100	Degree	**0.020**	**0.089**	**0.150**	0.039
	Betweenness	**0.060**	**0.118**	0.200	0.040
	PageRank	**0.050**	**0.113**	**0.180**	0.036
250	Degree	**0.024**	0.146	0.332	0.077
	Betweenness	**0.032**	0.132	0.180	0.049
	PageRank	**0.032**	0.144	0.208	0.058
500	Degree	0.100	0.489	0.986	0.191
	Betweenness	**0.040**	0.221	0.560	0.095
	PageRank	**0.046**	0.264	0.438	0.085

that of a true ER network. However, the experiments with larger networks proved even more difficult, with the average results being slightly above the critical threshold on 250 vertex networks, and significantly above the critical for 500 vertex networks. Due to the entirely random behavior of the ER model, the chaotic behavior of the ER model became much more pronounced with the larger networks, leading to increased difficulty of replicating the construction mechanism. Furthermore, one would expect a much higher statistical variation with the larger networks, again providing more difficulty for the system to model.

Algorithm 16 provides the best model evolved when a 100 vertex ER network was used as the target graph. Immediately, the striking similarity to the true model was noted. The evolved model, via the **GetAllQueue** method, selected all existing vertices to be considered, then connected to each with a probability of 0.05072. Note that the only difference between the evolved model and the ER model used to generate the target was the 0.00072, i.e., 0.072%, difference in the connection probability. This result demonstrated the ability of the GP system to not only infer the structure of ER model, but also to infer the probability of connection between vertices with a high degree of accuracy. The post-validation results, shown in Table 8.11, further exemplified the functional similarity of the ER_{100} model and the true ER model. The average number of edges, average geodesic path length, and clustering coefficient were all extremely close between the evolved model and the true model. However, the average number of edges as well as the maximum observed edges was higher for the evolved model. The increased edges was a direct result of the higher connection probability. Similarly, the slightly lower average geodesic path length was caused by the increased number of edges. Each of the centrality measures demonstrated an average D statistic, when the networks from the evolved model and true model were

Table 8.11: Comparison of 100 vertex graphs generated by the ER_{100} model and the Erdos-Reyni model. Values displayed are calculated over 30 generated graphs.

	Measure	Min	μ	Max	σ
ER_{100}	Edges	214	246.833	280	13.643
	AGP	2.855	3.013	3.237	0.084
	CC	0.026	0.048	0.070	0.012
Erdos-Reyni	Edges	217	245.867	267	14.920
	AGP	2.900	3.033	3.221	0.102
	CC	0.024	0.051	0.067	0.010
Average D Statistic	KS Deg	**0.030**	**0.099**	0.220	0.041
	KS Bet	**0.060**	**0.107**	**0.160**	0.030
	KS PR	**0.060**	**0.106**	**0.140**	0.024

compared, which was extremely close to the expected value when comparing networks from the ER model shown in Chapter 5. To further exemplify the similar structures, Figure 8.7 depicts a network generated by the ER_{100} model alongside the Erdos-Reyni target network with $n = 100$.

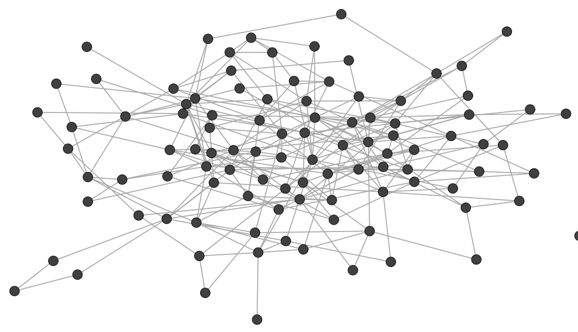
Algorithm 16 Simplified ER_{100} Model

```
function SelectVertices($g$)
    $S \leftarrow$ GetAllQueue($g$)
    return $S$
end function

function CreateEdges($newVertex, v$)
    $E \leftarrow$ AddEdgeWithProbability($v, newVertex, 0.05072$)
    return $E$
end function

function SecondaryActions($v, S$)
        // Nothing is performed in SecondaryActions
end function
```

Next, a 250 vertex network produced by the ER model was used as a target for the GP system. The best inferred model is shown in Algorithm 16. The model inferred with a 250 vertex network behaves nearly identical to the model inferred with a 100 vertex target. The ER_{250} model made use of the *GetAllStack* method, which allowed every existing vertex to be considered for connection. Connections were formed with a probability of 0.05173, slightly higher than with the ER_{100} model. The ER_{250} is thus different from the true ER model only by the additional 0.00173, i.e., 0.173%, in the connection probability between each vertex. However, as Table 8.12 demonstrates, the


(a) *ER*$_{100}$

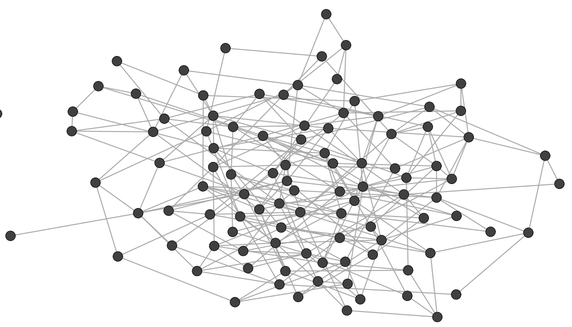

(b) Erdos-Reyni target, $n = 100$

Figure 8.7: Network generated by the *ER*$_{100}$ model and the Erdos-Reyni target network, $n = 100$.

Table 8.12: Comparison of 250 vertex graphs generated by the *ER*$_{250}$ model and the Erdos-Reyni model. Values displayed are calculated over 30 generated graphs.

	Measure	Min	μ	Max	σ
ER$_{250}$	Edges	1561	1607.833	1699	35.639
	AGP	2.396	2.440	2.463	0.017
	CC	0.046	0.051	0.056	0.003
Erdos-Reyni	Edges	1487	1563.600	1620	37.242
	AGP	2.436	2.462	2.504	0.019
	CC	0.045	0.050	0.056	0.003
Average D Statistic	KS Deg	**0.036**	**0.081**	0.192	0.0370
	KS Bet	**0.036**	**0.069**	**0.116**	0.020
	KS PR	**0.048**	**0.079**	0.124	0.015

evolved model generated networks which functioned extremely similar to their true counterparts from the ER model. The evolved model would generate, on average, roughly 44 more edges per network than the true model. This differences in edges caused the average D statistic for the degree distribution to be slightly higher than the expected value when networks from the ER were compared, but still well within the critical region to be considered insignificantly different. Furthermore, the average geodesic path length was again slightly smaller among networks from the evolved model due to the increased edge density. Even with these slight differences, the evolved model was an extremely good fit. For visual comparison, Figure8.8 presents a network produced by the evolved model alongside the target network. Although, due to the density of edges, not much useful information can be discerned visually for these networks, it was still apparent that the networks had a visible structural similarity.

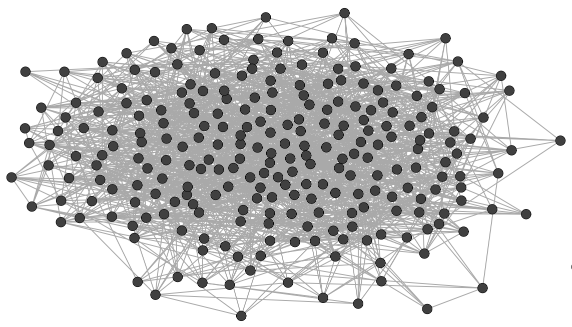
A final experiment with the Erdos-Reyni model was performed by using a 500

Algorithm 17 Simplified ER_{250} Model

```
function SelectVertices($g$)
    $S \leftarrow$ GetAllStack($g$)
    return $S$
end function

function CreateEdges($newVertex, v$)
    $E \leftarrow$ AddEdgeWithProbability($v, newVertex, 0.05173$)
    return $E$
end function

function SecondaryActions($v, S$)
        // Nothing is performed in SecondaryActions
end function
```


(a) ER_{250}

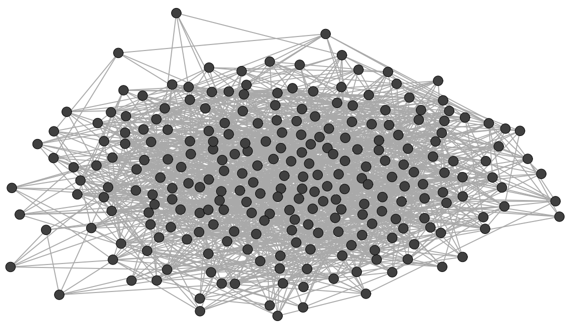

(b) Erdos-Reyni target, $n = 250$

Figure 8.8: Network generated by the ER_{250} model and the Erdos-Reyni target network, $n = 250$.

vertex network as the target. The best evolved model, ER_{500}, is presented in Algorithm 18. Again, the evolved model was extremely similar in functionality to the true model, with the only difference as a result of the connection probability. The probability of the connection for the evolved 500 vertex model was 0.05350 – even higher than that produced for the 250 vertex model. This is an effective difference of 0.00350, or 0.350%, for the connection probability. Although this difference would appear negligible, with such large networks the effect is much more pronounced. As the post-analysis results in Table 8.13 demonstrated, the evolved model would produce roughly 422 more edges, on average, then the Erdos-Reyni model. Furthermore, the minimum number of edges observed in a network generated by the evolved model was 6529, while the maximum number produced by the ER model was only 6366. This suggested that the ER model would need to produce a number of edges which was approximately 4.63 standard deviations away from the mean to produce a network with the minimum number of edges observed with the evolved model. Thus, the 0.350% difference has a significant impact on networks of larger scale. The average geodesic path lengths, while slightly lower for the evolved model, are arguably not significantly different. On average, the mean AGP was only 0.048 larger in the true model than the evolved model. The transitivity was only slightly higher in the evolved model, again as a result of the increased probability of connection. The betweenness and PageRank measures were both, on average, well within the range to be considered insignificantly different. The difference in degree distribution, on the other hand, was nearly always above the critical threshold when networks were compared. The increased number of edges would, in theory, cause the degree of each vertex to increase, thus making the degree distribution significantly different from that of the true model. Figure 8.9 depicts a network generated by the ER_{500} model along with the target network produced by the Erdos-Reyni model with $n = 500$. Visibly, the networks appear similar, however, this is due to the fact that no discernible structure can be seen with such a high density of edges – each network shown has over 6000 edges.

8.3.1 Effective Difference in Connection Probabilities

While the connection probabilities for the Erdos-Reyni model evolved by the GP system were quite close to the target value of 0.05, seemingly insignificant differences in the probability can have large effects on the degree distribution of the networks, especially as the network size grows. Note that the average D statistic for degree

Algorithm 18 Simplified ER_{500} Model

```
function SelectVertices($g$)
    $S \leftarrow$ GetAllStack($g$)
    return $S$
end function

function CreateEdges($newVertex, v$)
    $E \leftarrow$ AddEdgeWithProbability($newVertex, v, 0.05350$)
    return $E$
end function

function SecondaryActions($v, S$)
        // Nothing is performed in SecondaryActions
end function
```

Table 8.13: Comparison of 500 vertex graphs generated by the ER_{500} model and the Erdos-Reyni model. Values displayed are calculated over 30 generated graphs.

	Measure	Min	μ	Max	σ
ER_{500}	Edges	6529	6660.767	6761	58.078
	AGP	2.164	2.175	2.189	0.006
	CC	0.051	0.053	0.054	0.001
Erdos-Reyni	Edges	6094	6238.300	6366	62.745
	AGP	2.209	2.223	2.240	0.007
	CC	0.048	0.050	0.052	0.001
Average D Statistic	KS Deg	**0.080**	0.145	0.224	0.033
	KS Bet	**0.044**	**0.069**	0.090	0.013
	KS PR	**0.036**	**0.056**	0.088	0.011

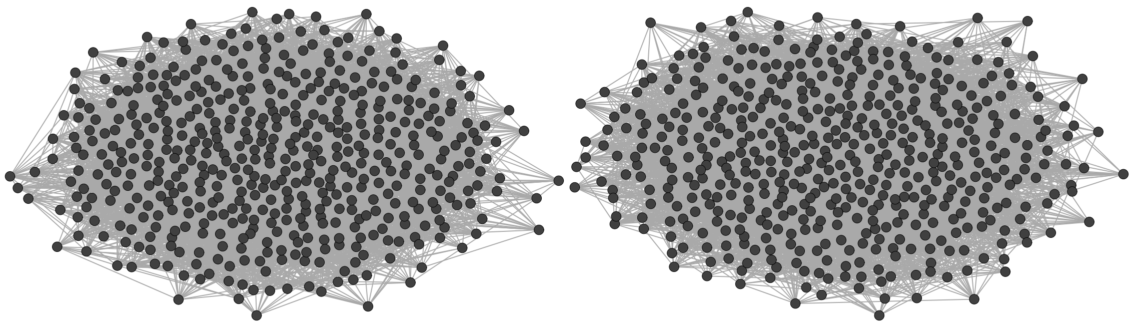

(a) ER_{500} (b) Erdos-Reyni target, $n = 500$

Figure 8.9: Network generated by the ER_{500} model and the Erdos-Reyni target network, $n = 500$.

Table 8.14: Expected degree and number of edges for the Erdos-Reyni and evolved models on various network sizes.

	100 Vertices		250 Vertices		500 Vertices	
Model	Degree	Edges	Degree	Edges	Degree	Edges
Erdos-Reyni	4.950	247.500	12.450	1556.250	24.950	6237.500
ER_{100}	5.021	251.064	12.629	1578.660	25.309	6327.320
ER_{250}	5.121	256.063	12.881	1610.096	25.813	6453.318
ER_{500}	5.296	264.825	13.322	1665.188	26.697	6674.125

distribution between 500 vertex networks generated by the ER_{500} model and the true model was 0.145 – significantly above the critical threshold of 0.08601 for samples of this size. The largest difference in the connection probability, demonstrated by the ER_{500} model, was 107% of the target probability. Thus, the expected degree and number of edges generated by the ER_{500} will be 1.07 times larger than the true model. Similarly, the probability evolved by the ER_{100} model, and thus the expected degree and edges, was 1.014 times larger than the target, while the probability of the ER_{250} was 1.034 times larger.

To demonstrate the emergent differences in network structure of as a result of these connection probabilities, Table 8.14 presents the expected degree of each vertex alongside the expected number of edges in the network when the true Erdos-Reyni (i.e., $p = 0.05$) and evolved models are used. Because the Erdos-Reyni model constructs networks in such a simple fashion and has been studied for a long period of time, analytical results demonstrating the expected degree of a vertex and number of edges can be readily calculated. The expected degree of a vertex, denoted by c, is simply $c = p(n-1)$ while the expected number of edges, $\langle m \rangle$, is given by $\langle m \rangle = p\binom{n}{2}$ [1]. When 100 vertex networks were considered, the expected degree of each vertex was relatively close; the worst evolved model (ER_{500}) demonstrated an expected difference in degree of only 0.346 when compared to the true model. However, this small difference in degree lead to an expected difference of 17.325 edges on 100 vertex networks. When examined on 500 vertex networks, the same two models had an expected degree which differed by 1.747 for each vertex and an expected difference of 436.625 edges. In contrast, the best evolved model (ER_{500}) had an expected difference of only 0.0359 for degree and 89.820 edges when the 500 vertex networks were considered.

Regardless of the differences in connection probabilities and, as a result, the degree distributions, the GP system was able to automatically infer the Erdos-Reyni model at three different network sizes. While the evolved model using a 500 vertex target was significantly different with regards to the degree distribution, this problem could be somewhat mitigated by the use of the *ER_{100}* model to generate larger networks. However, doing so would only postpone the point in which the network size was large enough for a significant difference in degree distribution to be apparent. Nonetheless, these results from inferring the ER model demonstrated that connection probabilities could be evolved, at minimum, to within 2 decimal places of their respective target. For a GP system, this is quite an impressive feat; this alone provides a justification of the merit of this approach to the automatic inference of graph models for complex networks.

8.4 Evolving the Forest Fire Model

The final known model used as a target for automatic inference was the Forest Fire model. This model was the most difficult known model used as a target for inference and exhibits relatively complex construction mechanisms and emergent behaviors. For a GP system, this model presents a number of challenges for automatic inference. The Forest Fire models makes use a queue whereby vertices are enqueued for processing based on a recursive "burning" procedure. This "burning" behavior is parametric in the probability of spreading to adjacent vertices. Thus, a model evolved by the GP system would be responsible for not only reproducing a recursive procedure, but also discovering the probability of spread. Furthermore, the FF model exhibits community structure, a known difficulty with the automatic inference of graph models for complex networks [98].

A summary of the best evolved models from each run is given in Table 8.15. When 100 vertex networks were considered, the fitnesses of the evolved models were nearly all insignificantly different when their networks were compared to the target network. This demonstrated a consistency among the evolved models at this size. With the larger networks, the average fitness of the evolved models degraded quite significantly. The average fitness for the 250 vertex target network experiment was only below the critical threshold for the betweenness while there was no fitness measure that was below the average for the 500 vertex target experiment. However, with both the 250 and 500 vertex experiments, the minimum observed D statistic for each fitness measure was below the critical threshold, which demonstrated that models with good fitnesses were achieved, albeit not consistently. It should be noted here that due to the multi-objective nature of the GP system, the minimum values presented in Table 8.15 were not necessarily observed by the same model. A more detailed look at the

Table 8.15: Summary of all 30 models evolved against a Forest Fire graph.

Target Size	Measure	Min	μ	Max	σ
100	Degree	**0.040**	**0.131**	**0.190**	0.055
	Betweenness	**0.050**	**0.072**	**0.140**	0.017
	PageRank	**0.060**	**0.130**	0.210	0.037
250	Degree	**0.036**	0.152	0.316	0.083
	Betweenness	**0.048**	**0.103**	0.340	0.056
	PageRank	**0.044**	0.138	0.296	0.081
500	Degree	**0.064**	0.179	0.334	0.065
	Betweenness	**0.058**	0.116	0.296	0.052
	PageRank	**0.074**	0.203	0.302	0.089

best model evolved at each network size is provided below.

First, a 100 vertex network generated by the Forest Fire model was used as a target for inference. The best evolved model, shown in Algorithm 19, has some similar characteristics to the FF model. In the evolved model, a single random vertex was selected via the **GetRandomStack** method, analogous to selection of an ambassador node, while the *SecondaryActions* procedure depicted the use of **AddNeighbours**, which effectively produced a recursive construction process. The FF model uses two recursive burning mechanisms, to represent forward and backward burning, respectively, whereas the evolved model did not. However, the evolved model compensated for the reduction of vertices added to the "burning" list using the **AddTriangle-WithProbability** edge creation function. The post-analysis results, presented in Table 8.16, indicated that the FF_{100} produced networks which were functionally similar to those produced by the FF model. The average number of edges differed by less than 7 while the mean average geodesic path length was only different by 0.035 when networks produced by the evolved model and the target model were compared. The average transitivity was nearly identical between the evolved model and the target model. Furthermore, each of the 3 centrality measures were, on average, insignificantly different among the evolved and FF models. Interestingly, the average D statistic for both the degree and PageRank measures was noticeably lower than would be expected for networks from the FF model, as demonstrated in Chapter 5.

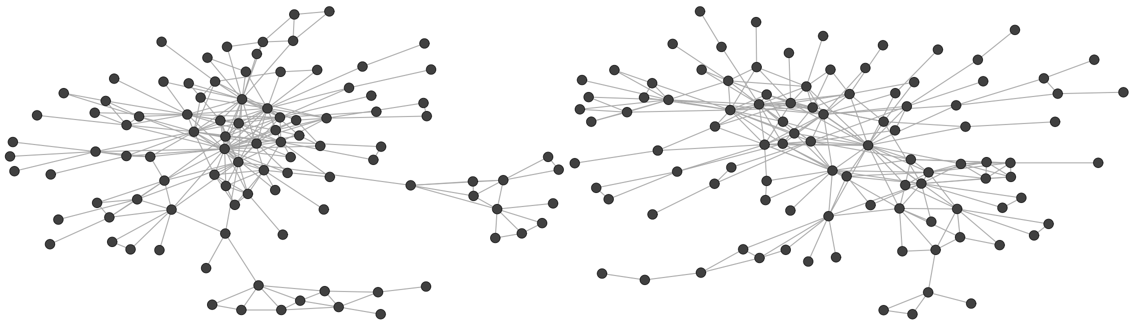
The Forest Fire model is known to exhibit community structure, thus, a model which creates functionally similar networks should also exhibit such community structure. As such, the evolved model was also compared to the target model by comparing the number of communities detected in the generated networks by the leading eigenvector community detection algorithm [72]. The number of communities detected is presented alongside the structural properties, denoted *Comm*, in Table 8.16. The results indicated that the evolved model was creating community structures which closely resembled their target counterparts. The mean number of communities differed by an average of 0.500, well within a third of a standard deviation from the respective means. Furthermore, the observed range for the number of communities was nearly identical between the evolved and true models, with the evolved model demonstrating a slightly larger range. The similarity between a network generated by the evolved model and the target network is visualized in Figure 8.10. It can be seen from these graphs that the networks exhibit similar structural and community behaviors.

Algorithm 19 Simplified *FF₁₀₀* Model

```
function SelectVertices($g$)
    $S \leftarrow$ GetRandomStack($g$)
    return $S$
end function

function CreateEdges($newVertex, v$)
    $E \leftarrow$ AddTriangleWithProbability($v, newVertex, 0.46176$)
    return $E$
end function

function SecondaryActions($v, S$)
    $a \leftarrow$ GetGeometricValue(0.38883)
    $b \leftarrow$ GetRandomValue($0, a$)
    AddNeighbours($S, b, v$)
end function
```



(a) *FF*$_{100}$ (b) Forest Fire target, $n = 100$

Figure 8.10: Network generated by the *FF*$_{100}$ model and the Forest Fire target network, $n = 100$.

Next, the GP system was used to infer a model against a 250 vertex network

Table 8.16: Comparison of 100 vertex graphs generated by the FF_{100} model and the Forest-Fire model. Values displayed are calculated over 30 generated graphs.

	Measure	Min	μ	Max	σ
FF_{100}	Edges	200	260.567	334	30.658
	AGP	3.101	3.665	4.575	0.326
	CC	0.340	0.388	0.467	0.030
	Comm	5	9.200	16	2.644
Forest Fire	Edges	200	267	366	41.745
	AGP	3.03	3.700	4.661	0.352
	CC	0.328	0.389	0.463	0.032
	Comm	6	9.700	15	1.822
Average D Statistic	KS Deg	**0.040**	**0.102**	0.240	0.043
	KS Bet	**0.050**	**0.119**	0.210	0.043
	KS PR	**0.060**	**0.098**	**0.160**	0.028

generated by the FF model. Algorithm 20 depicts the best evolved model from this experiment. Manual inspection of the evolved model indicated that the recursive procedure was not replicated. The evolved model did, interestingly, make use of the **Duplicate** method of edge creation which allowed for connections to be formed to neighbors of the 3 randomly selected vertices. This construction mechanism, as seen in Table 8.17, lead to some relatively large structural dissimilarities between networks generated by the FF_{250} model and the target FF model. The networks produced by the FF model contained, on average, more than 219 additional edges when compared to the networks produced by the evolved model. The evolved model also created a more regular distribution of edges, showing a standard deviation of only 37.245 while the FF model had a standard deviation of 160.934 when the number of edges were measured. The evolved model generated networks which, on average, had less than half of the transitivity of the FF networks. This experiment does demonstrate an interesting result, however, in that the average D statistic for each of the centrality measures were, on average, within the critical region when comparing the evolved model to the FF model. That is, the evolved model was able to produce insignificantly different distributions for the degree, betweenness, and PageRank of vertices, but the evolved networks were constructed, and structured, quite different. Figure 8.11 presents a network generated by the FF_{250} model alongside the target network for this experiment. Immediately the number of isolate vertices (i.e., 15) was noted in the evolved network. This phenomena also lead to the number of communities being, arguably, misrepresented. The leading eigenvector community detection algorithm labeled each of the isolated vertices as its own community, thereby

Table 8.17: Comparison of 250 vertex graphs generated by the FF_{250} model and the Forest Fire model. Values displayed are calculated over 30 generated graphs.

	Measure	Min	μ	Max	σ
FF_{250}	Edges	522	603.133	689	37.245
	AGP	3.397	3.592	3.831	0.093
	CC	0.140	0.166	0.181	0.010
	Comm	18	32.667	51	6.530
Forest Fire	Edges	629	822.300	1384	160.934
	AGP	3.243	4.071	4.84	0.406
	CC	0.303	0.343	0.407	0.027
	Comm	7	14.100	21	3.428
Average D Statistic	KS Deg	**0.060**	**0.107**	0.188	0.032
	KS Bet	**0.056**	**0.093**	0.124	0.020
	KS PR	**0.068**	**0.100**	0.172	0.024

drastically increasing the number of communities detected. While even the connected component of the evolved network did not visually replicate the community structure of the target network, the statistics regarding the number of communities should be considered along with this observation. In fact, when isolate vertices were disregarded for the calculation of communities, the average number of communities was 15.200 – significantly lower than the 32.667 when the isolate vertices were included.

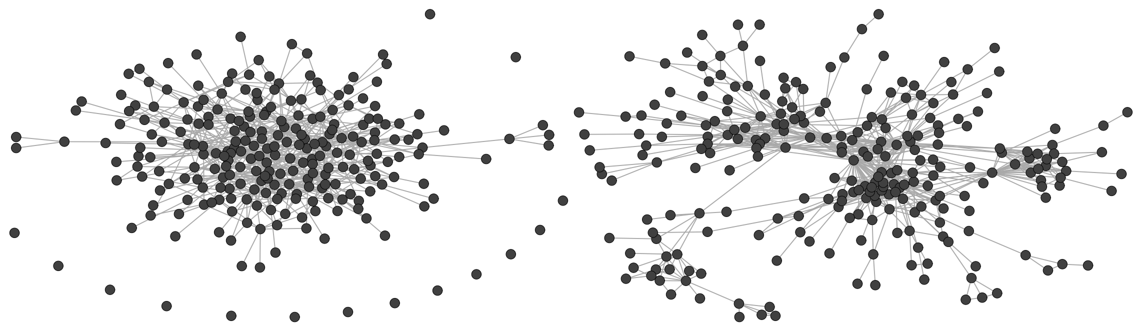
Algorithm 20 Simplified FF_{250} Model

```
function SelectVertices($g$)
    $S \leftarrow$ GetRandomStack($g, 3$)
    return $S$
end function

function CreateEdges($newVertex, v$)
    $E \leftarrow$ Duplicate($v, newVertex, 0.41684$)
    return $E$
end function

function SecondaryActions($v, S$)
            // Nothing is performed in SecondaryActions
end function
```

The final automatic inference experiment using the Forest Fire model as a target was performed with a 500 vertex target. As algorithm 21 depicts, the FF_{500} model differed from the FF_{250} model only in the probability of connecting to the vertex which was being duplicated. The evolved model made use of the **GetRandomStack**



(a) FF_{250} (b) Forest Fire target, $n = 250$

Figure 8.11: Networks generated by the FF_{250} model and the Forest Fire target network, $n = 250$.

to select three vertices whose edges were to be duplicated by the use of the **Duplicate** method. Each of the three vertices which were selected had a probability of being connected to the new vertex of 0.42358, slightly higher than the 0.41684 observed with the FF_{250} model. As such, the observations made when the post-analysis results, presented in Table 8.18, were considered are similar to those in the previous experiment. Namely, the number of edges was significantly lower in the evolved model, the evolved model demonstrated an average of 624.566 less edges per network than the FF model. The mean average geodesic path length difference was relatively high at 0.383, with the evolved model having the lower AGP values, while the transitivity of the evolved networks was again less than half of the networks produced by the FF model. None of the three centrality measures obtained an average D statistic below the critical threshold, signifying that the networks exhibited significantly different distributions for degree, betweenness, and PageRank. Figure 8.12 presents a network generated by the evolved model as well as the target network for this experiment. Visual inspection indicated that isolate vertices, expectedly so, were an issue with the evolved model. The community structure was visibly different among the networks, even when only the connected component of the evolved network was considered. The evolved model would produce nearly three times the number of communities produced by the FF model. However, when isolate vertices were disregarded, the FF_{500} model produced an average of 19.900 communities per network.

8.4.1 Scalability of the FF_{100} Model

The previous results indicated that the FF_{100} model did replicate the recursive nature of the FF model while the FF_{250} and FF_{500} models did not. Thus, a natural question

Algorithm 21 Simplified *FF_{500}* Model

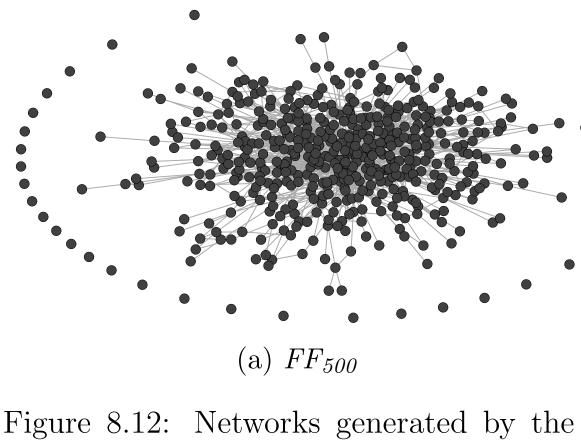
```
function SelectVertices($g$)
    $S \leftarrow$ GetRandomStack($g, 3$)
    return $S$
end function

function CreateEdges($newVertex, v$)
    $E \leftarrow$ Duplicate($v, newVertex, 0.42358$)
    return $E$
end function

function SecondaryActions($v, S$)
        // Nothing is performed in SecondaryActions
end function
```

Table 8.18: Comparison of 500 vertex graphs generated by the FF_{500} model and the Forest Fire model. Values displayed are calculated over 30 generated graphs.

	Measure	Min	μ	Max	σ
FF_{500}	Edges	1190	1242.067	1315	34.690
	AGP	3.805	3.929	4.018	0.051
	CC	0.126	0.138	0.146	0.006
	Comm	40	52.967	69	7.495
Forest Fire	Edges	1465	1866.633	2354	257.550
	AGP	3.632	4.312	4.977	0.325
	CC	0.269	0.299	0.333	0.015
	Comm	5	18.033	29	6.083
Average D Statistic	KS Deg	**0.062**	0.110	0.174	0.030
	KS Bet	**0.052**	0.090	0.144	0.024
	KS PR	**0.062**	0.098	0.136	0.021


(a) FF_{500}

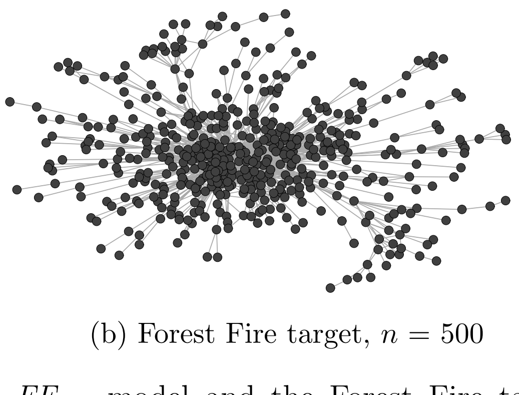

(b) Forest Fire target, $n = 500$

Figure 8.12: Networks generated by the FF_{500} model and the Forest Fire target network, $n = 500$.

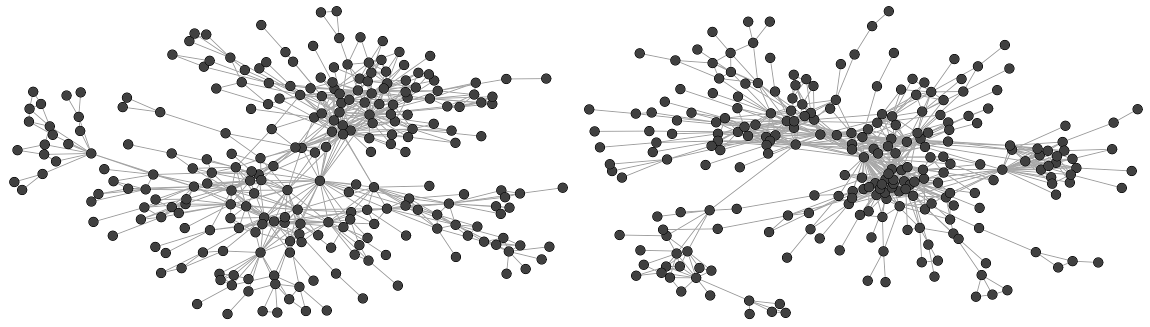
Table 8.19: Comparison of 250 vertex graphs generated by the FF_{100} model and the Forest Fire model. Values displayed are calculated over 30 generated graphs.

	Measure	Min	μ	Max	σ
FF_{100}	Edges	592	712.033	865	58.801
	AGP	3.599	4.148	4.782	0.300
	CC	0.294	0.336	0.382	0.020
	Comm	9	14.333	22	3.111
Forest Fire	Edges	629	822.300	1384	160.934
	AGP	3.243	4.071	4.840	0.406
	CC	0.303	0.343	0.407	0.027
	Comm	7	14.100	21	3.428
Average D Statistic	KS Deg	**0.052**	**0.096**	0.192	0.035
	KS Bet	**0.040**	**0.091**	0.172	0.030
	KS PR	**0.052**	**0.081**	0.136	0.023

to ask was how well the FF_{100} model replicated larger networks generated by the FF model. As such, the FF_{100} was used to generate 250 and 500 vertex networks which were then compared to networks generated by the FF model of the same respective size. The results given below indicated that the FF_{100} model, while improvements could be made, was able to reasonably replicate the behaviors of larger networks.

Table 8.19 provides the comparison of 250 vertex networks generated by the FF_{100} and FF models, respectively. While the average number of edges was significantly lower, roughly 110, for the FF_{100} model, this had minimal impact on the other properties. The average geodesic path length differed by only 0.077 while the clustering coefficient was, on average, only different by 0.007. The average and range of communities was comparable between the evolved networks and the FF networks. Each of the centrality measures, while above what would be expected for comparing networks from the FF model, observed an average statistic well below the critical threshold for 250 vertices. Thus, with the exception of the number of edges, the FF_{100} model produced networks which were extremely close in both structure and behavior to those produced by the FF model when 250 vertex networks were considered. Examining a network generated by the evolved model along with the target from the 250 vertex inference experiments, presented in Figure 8.13, indicated that the FF_{100} model was able to visibly replicate the target significantly better than the FF_{250} model (see Figure 8.11).

Similarly, the FF_{100} model was used to generated 500 vertex networks which were compared to networks generated by the FF model, the results of which are given in Table 8.20. Note that the number of edges for the evolved model were significantly



(a) *FF₁₀₀* (b) Forest Fire target, $n = 250$

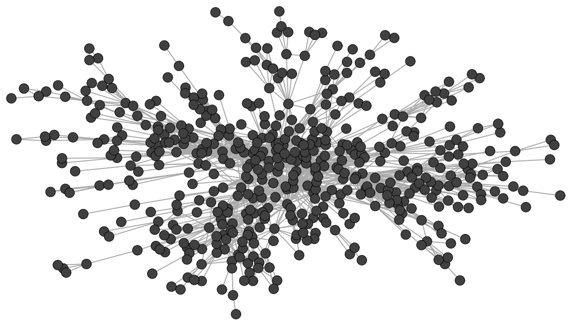
Figure 8.13: Network generated by the FF_{100} model and the Forest Fire target network, $n = 250$.

lower than the true FF model, with an average of over 402 less edges per network. This reduction of edges lead to a relatively large increase in the AGP over the true model, with the FF_{100} model producing networks which exhibited a mean AGP of 0.403 longer than the FF model. Interestingly, while there were less edges produced by the evolved model, there was a somewhat higher clustering coefficient which resulted from the **AddTriangleWithProbability** used to construct the edges. When the number of communities detected were examined, the FF_{100} produced nearly 3 additional communities per network when compared to the FF model. While these observations demonstrated that the basic structure of the networks was different, the centrality measures were, on average, insignificantly different between networks produced by the models. This indicated that while the macro-properties of the networks were different, the micro-properties visible at the vertex level were not significantly different between the networks generated by the models. For visual comparison, a 500 vertex network generated by the FF_{100} model along with the 500 vertex target network are presented in Figure 8.14.

In summary, this section presented the experiments performed to infer a graph model when a network generated by the Forest Fire model was as the target network. Results indicated that the model evolved against the 100 vertex target was a very fit model while the results using the 250 and 500 vertex target networks, respectively, lacked the recursive construction mechanism of the true FF model. The FF_{100} model was demonstrated to accurately replicate networks with 250 vertices generated by the FF model, with the exception of the number of edges. Furthermore, the FF_{100} model replicated the centrality distributions of 500 vertex networks, but showed noticeable differences in the macro, structural properties of the network.

Table 8.20: Comparison of 500 vertex graphs generated by the FF_{100} model and the Forest Fire model. Values displayed are calculated over 30 generated graphs.

	Measure	Min	μ	Max	σ
FF_{100}	Edges	1258	1464.067	1645	87.45
	AGP	4.201	4.715	5.543	0.288
	CC	0.289	0.314	0.343	0.014
	Comm	1	20.967	31	6.856
Forest Fire	Edges	1465	1866.633	2354	257.550
	AGP	3.632	4.312	4.977	0.325
	CC	0.269	0.299	0.333	0.015
	Comm	5	18.033	29	6.083
Average D Statistic	KS Deg	**0.034**	**0.074**	0.118	0.020
	KS Bet	**0.036**	**0.075**	0.120	0.019
	KS PR	**0.030**	**0.073**	0.110	0.023


(a) FF_{100}

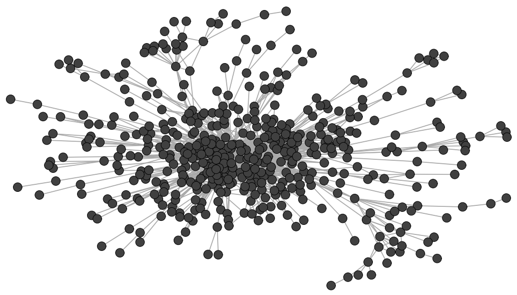

(b) Forest Fire target, $n = 500$

Figure 8.14: Network generated by the FF_{100} model and the Forest Fire target network, $n = 500$.

8.5 Summary

These experiments in this chapter demonstrated the proposed GP system for the automatic inference of graph models for complex networks was effective at replicating known graph models. Evolved models were analyzed both manually, by inspecting the models construction algorithms, and empirically by comparing networks generated by the evolved models with their respective target models. Results indicated that the evolved models were able to accurately replicate their respective targets, wit the exception of the Forest Fire experiments with 250 and 500 vertex networks. However, when the model evolved against a 100 vertex FF target was used to generate larger networks, the resulting networks were significantly better at replicating the behaviors of the FF model than the respective evolved models.

While this chapter demonstrated that known models could be replicated with high degree of accuracy, the GP system should also be able to evolve a graph model for a real-world complex network. As such, the following chapter presents the use of the proposed GP system for the automatic inference of a graph model for the brain network of a cat [12].

Chapter 9

Evolving a Model for a Cat Brain

This chapter briefly introduces the study of cortical networks as complex networks and presents the results of the automatic inference of the brain network of a cat. While Bailey *et al.* [30] have previously inferred a graph model for the cortical network of a cat, this work differs from their study as the data set used in this study is considerably larger; the data set used in Bailey's work contained 52 vertices, representing cortical areas, and 515 edges while the data set used in this work contains 95 vertices, representing both cortical and thalamic areas, and 1170 edges. The work of Bailey *et al.* also made use of an external community detection mechanism and simultaneously evolved two models, one for the community model and another to reproduce the inter-community structure. The two models were then combined to form a final, hierarchical model. In contrast to the work of Bailey *et al.*, this chapter outlines an initial study of using a single evolved model to implicitly generate networks which exhibit community structure. The results of this chapter are significant in that they depict the first attempt, to the best of the author's knowledge, at automatically inferring a graph model for a real-world complex network exhibiting community structure without the explicit, *a priori* detection of communities. Furthermore, the results indicated that the functional structure of the cortical network of a cat can be accurately replicated with a simple construction mechanism, as determined by the GP system.

When inferring a graph model for a complex network where the target model is unknown, as demonstrated in Figure 5.1, the process of model verification becomes much more challenging. Verification of the model can no longer be done by manual inspection, as the models inferred in the previous chapter allowed, and must rely solely on the intrinsic properties of the network. Thus, the evolved model must be used to generate networks which can be compared against the target network on features deemed important to the functionality of the target network. For example, in the case of cortical networks, path length and community structure are believed to be crucial to the functionality the network [99, 100], and thus, must be considered when analyzing the evolved model.

9.1 Cortical Networks

Cortical networks are often considered to be small-world networks due to their low average geodesic path lengths and high transitivity [63, 100, 99]. While the average geodesic path length of cortical networks have been found to be comparable to random networks, the transitivity is much too high to be formed by a random process [63]. Stephan *et al.* [63] provided functional evidence of small-world behavior in mammalian cortical networks and demonstrated that there existed multiple, functionally distinct, segregated regions within the cortex. The aforementioned results, among others, have lead to cortical networks commonly being considered to be small-world networks. Although Stam and Reijneveld [13], among others, have concluded that the small-world architecture of the cortex plays a crucial role for information processing, alternative observations have been made suggesting scale-free distributions, and their emergent hubs, better model the robustness of cortical networks [101, 102]. Irrespective of the debate on which of these two model types best describe cortical data, neither model has been able to accurately replicate both the path length distributions as well as the community structure crucial to the functionality of cortical networks [99].

The cortical network of a cat was selected for the automatic inference of a model due to its large presence in the literature as well as the previously mentioned inability for existing models to replicate its behaviors. Furthermore, the cortical data of the cat is regarded as one of the most complete data sets of its kind [99]. The data set used in the following experiments was an implicitly directed data set, however, a relaxed version was used whereby the directionality of edges was disregarded. The relaxation of the edge direction within the data set was considered reasonable as a majority of the connections were reciprocal, namely 88.74% with this particular data set, which is a common property of cortical networks [103]. In the undirected form, the data set used in this experiment contained 95 vertices and 1170 edges. Note that both multi-edges and loops were removed from the data set. The following section describes the results of the automatic inference of this cortical network.

Table 9.1: Summary of all 30 models evolved against the cat brain network.

Measure	Min	μ	Max	σ
Degree	**0.060**	**0.102**	0.404	0.065
Betweenness	**0.067**	**0.105**	**0.182**	0.029
PageRank	**0.060**	**0.081**	**0.144**	0.018

9.2 The Evolved Cortical Model

The cat brain network was used as the target network for the automatic inference system. The parameters for this experiment were the same as listed in Table 8.1, with the exception of the samples for evolution was 3, i.e., during the fitness evaluation, each candidate solution produced 3 networks which were compared to the target. A summary of the best models produced during each run is given in Table 9.1. The results from this table indicated that in a majority of the runs, an extremely fit model was produced. More concretely, the difference in the betweenness and PageRank measures was always below the KS critical threshold, at a 95% confidence level, of 0.19733. While the average of the D statistics produced when the degree distributions were considered was well below the critical threshold, the maximum difference in degree was quite large. Thus, there were evolved models which produced drastically different degree distributions than that of the brain network. Note that the minimum observed value was within one standard deviation of the mean, while the maximum was nearly three standard deviations above the mean. This would suggest that the large maximum was somewhat anomalous, and as such it can be safely concluded that, in general, the evolved models were quite fit with respect to the fitness measures used.

While the model was fit with respect to the fitness measures used, a post-analysis procedure was performed to determine the model's ability to recreate the structure of the cortical network. Algorithm 22 presents the best evolved model, selected via sum of ranks on the evolution fitnesses [38]. The evolved model was surprisingly simple – it selected 7 random vertices and used the **Duplicate** edge creation method to connect to each neighbor of the selected vertices. The probability of connecting the newly created vertex to the selected vertices was 0.73235, i.e., 73.235%. The *SecondaryActions* method had no effect whatsoever on the construction process. Initially it was thought that such a simple model could not be a fit model. However, a post-analysis procedure demonstrated that this model was strikingly similar to the target network.

Table 9.2 presents the post-analysis results. For reference, the structural properties of the cortical network are presented in Table 9.3. Comparing the properties of the evolved model to those of the cortical network, the number of edges in the evolved model was on average 22.467 higher than the cortical network, however, this was well within a single standard deviation. The average geodesic path lengths produced by the evolved model were slightly lower than the cortical network, with an average difference of 0.062. Note that while the average geodesic path lengths were, arguably, insignificantly different, in no instance did the evolved model generate an AGP which was as high as that of the cortical network. The average clustering coefficient was slightly higher, however, the value of 0.489 observed in the cortical network was still a statistically likely value for the evolved model, being slightly over one standard deviation away from the mean.

The community structure, known to be an important feature which represents different modalities [12, 99, 100], was also replicated quite well by the evolved model. The average number of communities which emerged in the evolved networks, as determined by the leading eigenvector community detection algorithm [72], was 3.767 while all the networks had between 2 and 6 communities. Furthermore, the community structure is a feature which is missing in previous attempts to model cortical networks [99]. The examined vertex-level centrality measures have also been replicated extremely well, exemplified by the maximal D statistic observed for any centrality measure, presented in Table 9.2, being well under the critical threshold of 0.19733. Thus, the centrality measures were insignificantly different among all networks generated by the evolved model when compared to the target network.

The number of pathways between cortical areas is believed to have an influence on the processing behavior of the network, with longer pathways generally traveling between different cortical areas [99]. Furthermore, the mixture of path lengths is thought to produce both complex, fast information processing capabilities as well as robustness in communication pathways [99]. Thus, a graph model which effectively models a cortical network should posses similar path length distributions as the cortical network. To compare the distribution of path lengths, Figure 9.1 presents a density plot where the shortest paths of 30 networks generated networks are compared to the target network. This figure demonstrated that the evolved model was able to reproduce the path length distribution relatively well. The evolved model produced, on average, slightly more paths of length 2 while producing slightly less paths of length 3.

A single network generated by the evolved model, determined to be the most similar to the target using sum of ranks on the measured properties, was selected for closer inspection. Table 9.3 presents the structural properties of the best network along with the properties of the target network. These structural properties demonstrated that

Algorithm 22 Simplified Evolved Model for the Cat Brain Network

```
function SelectVertices($g$)
    $S \leftarrow$ GetRandomStack($g, 7$)
    return $S$
end function

function CreateEdges($newVertex, v$)
    $E \leftarrow$ Duplicate($v, newVertex, 0.73235$)
    return $E$
end function

function SecondaryActions($v, S$)
            // Nothing is performed in SecondaryActions
end function
```

Table 9.2: Comparison of graphs generated by the evolved model and the cat brain network. Values displayed are calculated over 30 generated graphs.

	Measure	Min	μ	Max	σ
Evolved Model	Edges	1123	1192.467	1272	31.768
	AGP	1.768	1.803	1.825	0.014
	CC	0.477	0.504	0.520	0.011
	Comm	2	3.767	6	0.971
Average D Statistic	KS Deg	**0.063**	**0.110**	**0.158**	0.021
	KS Bet	**0.063**	**0.109**	**0.158**	0.024
	KS PR	**0.074**	**0.103**	**0.137**	0.017

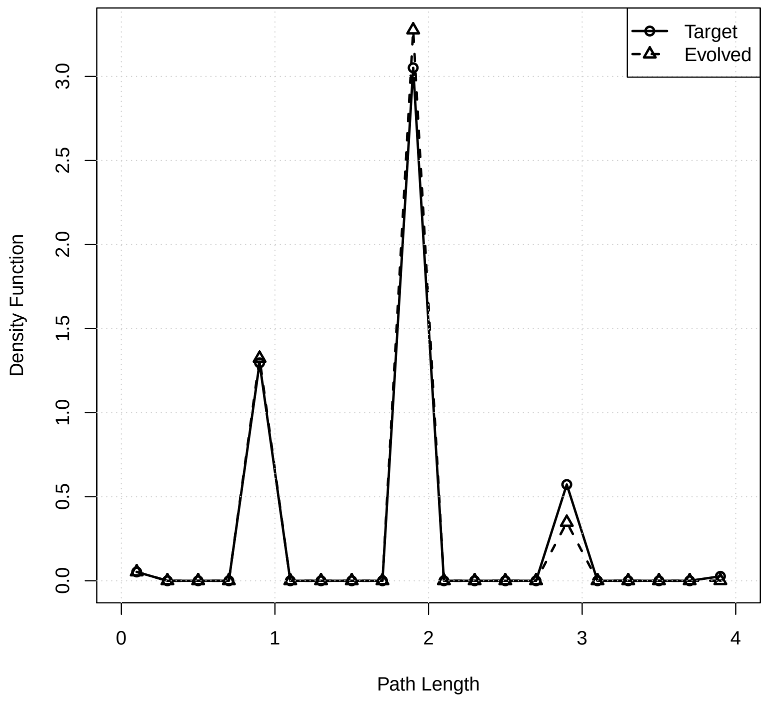
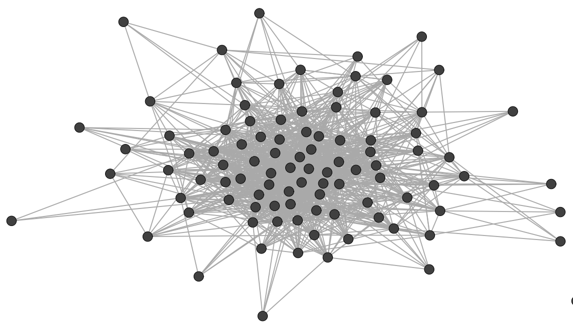


Figure 9.1: Density of the shortest paths of the networks generated by the evolved model and the cat brain network.

Table 9.3: Basic properties of the best graph generated by the evolved model and the cat brain network.

Measure	Cat Brain	Evolved
Edges	1170	1167
AGP	1.865	1.806
CC	0.489	0.491
Comm	4	4

the evolved model was capable of generating an extremely similar network to the target. Namely, the best network had only 3 less edges than the target and had an AGP which was only 0.059 less than the cortical network. The clustering coefficient was only 0.002 higher than the target, while the number of detected communities detected was 4 for both networks. Visual inspection of the networks, presented in Figure 9.2, demonstrated that both possessed a central region with a dense connection pattern. Figure 9.3 and 9.3 presents the degree distributions of the target and evolved network, respectively, while Figure 9.4 shows their respective ECDF plots. The bizarre degree distribution of the cortical network was visibly well modeled, reinforced by a KS test p-value of 0.991.


(a) Evolved

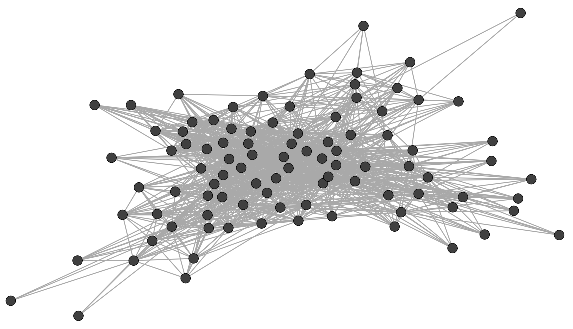

(b) Brain network

Figure 9.2: Network generated by the evolved model and the cat brain network.

The betweenness centrality has been observed to be an important measure of dynamics in cortical networks [13], and is closely related to community structure of complex networks in general [64]. The empirical cumulative distribution plot depicting the betweenness centrality of the evolved and target networks can be seen in Figure 9.5. Similarly, the ECDF plots of the PageRank measure can be found in Figure 9.6. KS tests for these distributions resulted in p-values of 0.788 and 0.959, respectively which demonstrated that the distributions of both centrality measures were insignificantly different among the evolved and target networks.

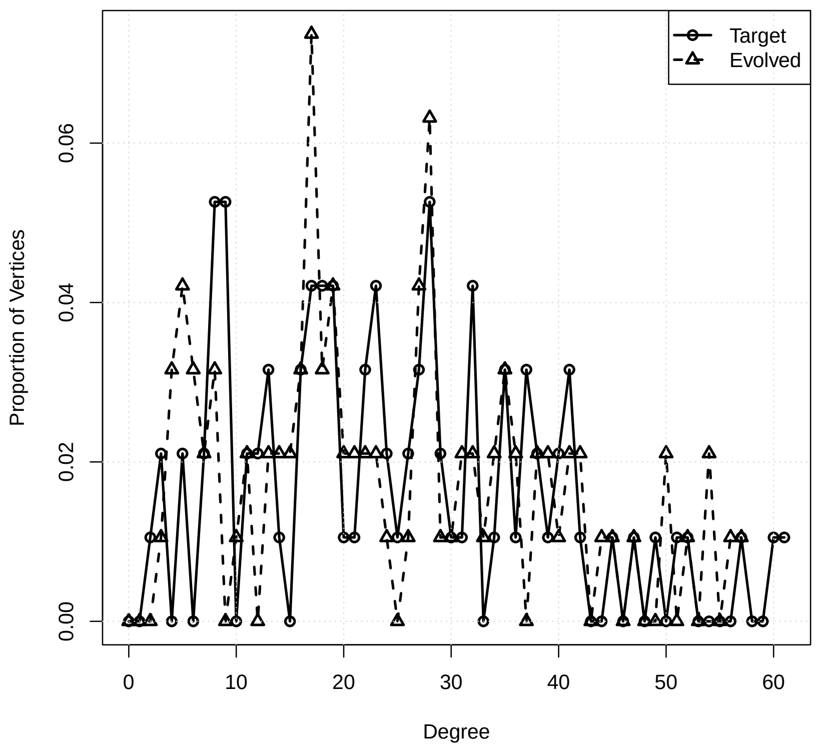


Figure 9.3: Plot of the degree distributions of an evolved graph and the cat brain network.

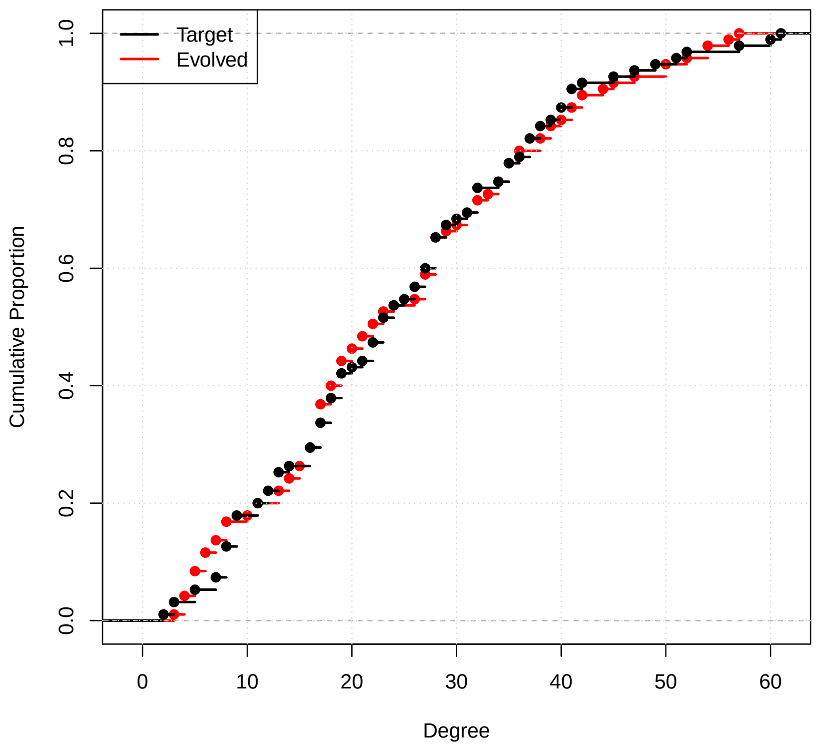


Figure 9.4: ECDF plot of the degree distributions of an evolved graph and the cat brain network. KS test: D = 0.063, p-value = 0.991.

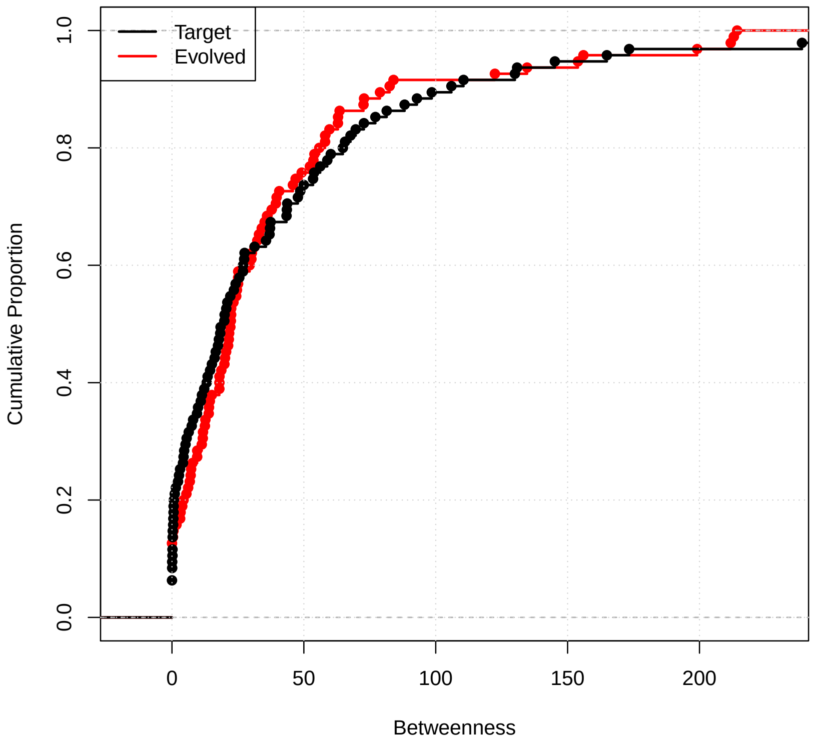


Figure 9.5: ECDF plot of the betweenness centrality of an evolved graph and the cat brain network. KS test: D = 0.095, p-value = 0.788.

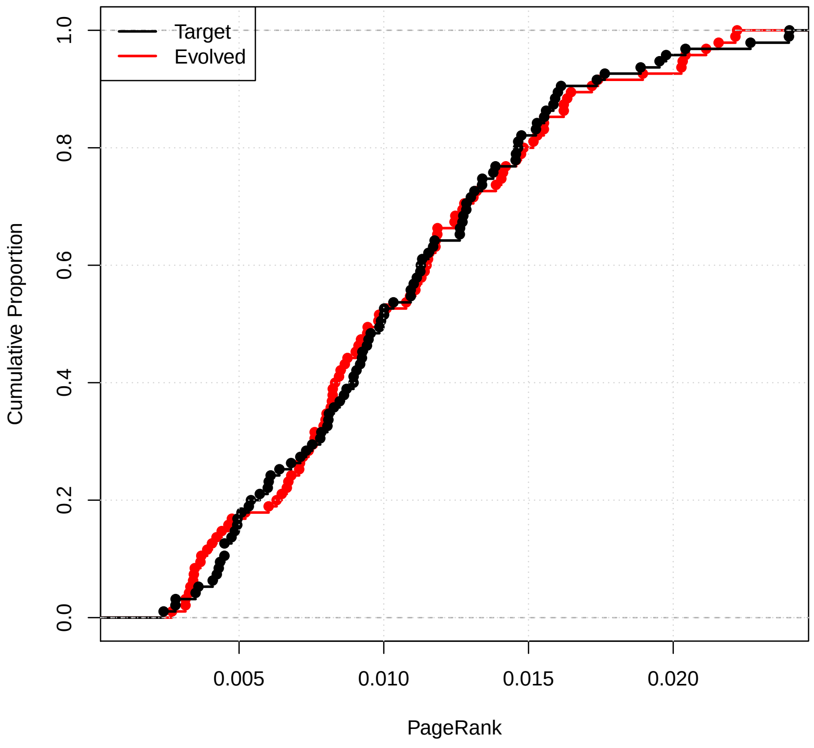


Figure 9.6: ECDF plot of the PageRank of an evolved graph and the cat brain network. KS test: D = 0.074, p-value = 0.959.

This chapter outlined the experimental procedures used to automatically infer a graph model for a real-world, cortical network of a cat. The evolved model was able to replicate the structural and emergent behaviors of the target network. While not the first real-world network to have a model automatically inferred for it (see Bailey *et al.* [30]), this chapter does detail, to the best of this author's knowledge, the first attempt at generating a model for a real-world network which exhibits strong community structure without the use of an external community detection mechanism.

Chapter 10

Conclusion

The major contributions of this thesis were threefold. First, this work provided an in-depth study of the behaviors of network measures when used to quantify the (dis)similarity of networks. The analytical results were formed using a combination of multi-dimensional scaling to visualize the spatial distributions of each model as well as through the use of a meta-analysis framework whereby the interactions of network measures, when used in conjunction with one another, was examined. The analytical results allowed insight to be made regarding evaluation criteria that demonstrates the ability of a graph model to construct functionally similar networks to that of its target.

The second major contribution of this work was the genetic programming (GP) methodology applied to the automated construction of graph model algorithms. A recently proposed GP methodology, namely linear object-oriented GP, was used in this work as it was shown to be promising in the area of automated construction of graph models for complex networks. The GP made use of a generalized graph model algorithm to create a logical representation of the problem to guide the search process.

Finally, the GP system facilitated the direct construction of community structure within the evolved graph models. Previous research on the construction of graph models, as seen in Chapter 9, made use of an external mechanism to detect, and thus model, community structure. While not the first work to evolve community structure, this work presented, to best of the author's knowledge, an initial study of the first GP system capable of directly constructing graph models which exhibit community structure without the use of an external community detection mechanism.

Six well-known graph models were used to evaluate the discriminatory power of ten network measures (four network-level and six vertex-level). The study of network measures indicated that network-level measures, i.e., those that assign a single value to the entire network, did not convey enough model-specific information to effectively discriminate between networks generated by different models. While useful for characterizing the behavior of the network, these measures are too relaxed to be useful when distinguishing networks generated by different models. Measures of vertex importance, i.e., centrality measures, were much more important to the classification of network behavior as they provided a great deal of information regarding the intrinsic properties of the network. Results indicated that of the six examined centrality measures, the degree distribution, betweenness centrality, and PageRank were the most effective for the context of quantifying the (dis)similarity of networks generated by different graph models.

Using the degree, betweenness, and PageRank as measures of evolutionary fitness, a GP system for the automatic construction of graph models was proposed. The GP system was provided a single target network with the goal of evolving a graph model algorithm that could construct networks which functioned in a similar fashion to the provided target network. The GP system was used to automatically infer four well-known graph models, namely the Growing Random, Barabasi-Albert, Erdos-Reyni, and Forest-Fire models. Target networks from the models were generated with 100, 250, and 500 vertices, respectively, and used as the target networks within the GP system. Evolved models were verified both by manual inspection of the algorithm and by analytical verification of their functional similarity. Results indicated that these well-known graph models could be evolved with striking accuracy. Finally, using the aforementioned approach, a graph model was automatically inferred for a mammalian cortical network, the results of which showed great promise. A noteworthy result was that the community structure, known to be an important factor for the functionality of cortical networks, was constructed effectively by the automatically inferred model.

While inspired by the work of Bailey *et al.* [29, 30, 31], this thesis differed from the previous work in a number of ways. First and foremost, this thesis made use of a drastically different GP paradigm, namely linear object-oriented GP, whereas the work of Bailey *et al.* made use of traditional, tree-based GP. Due to the various differences between this work and that of Bailey *et al.* (e.g., the fitness measures and the GP language), no conclusions can readily be made regarding which GP methodology is more well-suited for the automatic construction of graph models. Two of the known graph models used in this thesis, namely the Barabasi-Albert and Erdos-Reyni models, were also examined by Bailey *et al.*, while the Growing Random and Forest Fire models were not. The fitness measures used in this thesis differed significantly from those seen in Bailey *et al.*'s work as their measures were based on literary presence while the fitness measures used in this thesis were based on the observed ability of each measure to discriminate between networks generated by different models. The results of the empirical study on network measures suggests that the fitness evaluation scheme used in this thesis was an improvement over that of the previous work. However, this cannot be conclusively stated without a direct comparison between the two fitness evaluation schemes. Finally, this thesis presented an initial study of directly evolving graph models which were capable of constructing community structure. The emergent community structure resulted from the ability to build recursively defined construction mechanisms, facilitated by the new GP language in this work.

10.1 Future Work

Many avenues of future study have become apparent throughout the course of this work. First and foremost, this thesis only addresses undirected, unweighted networks – the simplest types of networks. Undirected networks are a special case of directed network whereby all edges are taken to be bidirectional. This is a reasonable relaxation in some cases, particularly cortical networks where a large majority of the edges are reciprocal [103]. Similarly, unweighted networks are a special case of weighted networks where all weights are considered equal. While many of the elements of this work can be applied to directed and weighted networks by taking the respective analogues into account, it is unknown whether such an approach will lead to reasonable results. Thus considering each of these network types will lead to much more generalized insight on the automatic construction of graph models for complex networks.

An immediate future study is to compare the results using the system proposed in this work against that of the previous work [29, 30, 31]. As both the fitness evaluation and the GP system used in this work differed from those used in previous work, the effects of each the fitness evaluation and GP system independently should be an avenue of future consideration. Namely, using the proposed fitness evaluation in the tree-style system and, conversely, using the fitness evaluation of Bailey's work within the proposed GP system should be considered so the benefits of each can be properly attributed.

Another topic of future interest lies in the way in which fitness is assigned. For larger networks, network measures become quite expensive to calculate. Although this can be alleviated by increasing the number of CPU processors available, such a solution is only scalable to a certain extent. An alternative approach may be to sample the target network forming random walks which can then be used for comparison. Research has shown that some centrality measures are stable when a network is sampled [104], thus an investigation into this technique may be warranted if larger networks are to be considered.

A goal of this work was to create graph models which were capable of capturing the growth mechanics of a particular network. With this in mind, it may be beneficial to use multiple snapshots of a target network, if possible, to have an idea of how the structure evolved to its current state. This would allow the GP to evaluate not only the ability to reconstruct a network similar to the current target, but also that validate that a candidate model captures the target at previous snapshots.

The final area of future work to be discussed lies in the notion of community structure. While this study provides an initial study of community structure directly evolved by the GP system, many areas of improvement with regards to this exist. First, the fitness evaluation of a candidate model did not explicitly account for community structure. Thus, the ability for the GP to evolve such community structures was a byproduct, and not an intended feature. This leads to a further improvement relating to community structure – the GP language. As community structure was not a large focus of this thesis, little effort was put forth to create a language which addressed the construction of communities. Although the language does facilitate a recursive construction mechanism whereby communities can be formed, more focus on the language elements with regards to communities will be highly beneficial.

Bibliography

[1] M. Newman, *Networks: an introduction.* Oxford University Press, 2010.

[2] W. Zachary, "An information flow modelfor conflict and fission in small groups," *Journal of anthropological research*, vol. 33, no. 4, pp. 452–473, 1977.

[3] T. Nepusz, A. Petróczi, L. Négyessy, and F. Bazsó, "Fuzzy communities and the concept of bridgeness in complex networks," *Physical Review E*, vol. 77, no. 1, p. 016107, 2008.

[4] J. Scott and P. J. Carrington, *The SAGE handbook of social network analysis.* SAGE publications, 2011.

[5] S. Wasserman, *Social network analysis: Methods and applications*, vol. 8. Cambridge university press, 1994.

[6] J. L. Moreno and H. H. Jennings, *Who shall survive?* Nervous and Mental Disease Publishing Co., 1934.

[7] M. Ventresca and D. Aleman, "Evaluation of strategies to mitigate contagion spread using social network characteristics," *Social Networks*, vol. 35, no. 1, pp. 75–88, 2013.

[8] J. Berg, M. Lässig, and A. Wagner, "Structure and evolution of protein interaction networks: a statistical model for link dynamics and gene duplications," *BMC evolutionary biology*, vol. 4, no. 1, p. 51, 2004.

[9] J.-F. Rual, K. Venkatesan, T. Hao, T. Hirozane-Kishikawa, A. Dricot, N. Li, G. F. Berriz, F. D. Gibbons, M. Dreze, N. Ayivi-Guedehoussou, *et al.*, "Towards a proteome-scale map of the human protein–protein interaction network," *Nature*, vol. 437, no. 7062, pp. 1173–1178, 2005.

[10] R. J. Williams and N. D. Martinez, "Simple rules yield complex food webs," *Nature*, vol. 404, no. 6774, pp. 180–183, 2000.

[11] T. L. Hopkins and J. J. Torres, “Midwater food web in the vicinity of a marginal ice zone in the western weddell sea,” *Deep Sea Research Part A. Oceanographic Research Papers*, vol. 36, no. 4, pp. 543–560, 1989.

[12] J. W. Scannell, C. Blakemore, and M. P. Young, “Analysis of connectivity in the cat cerebral cortex,” *The Journal of Neuroscience*, vol. 15, no. 2, pp. 1463–1483, 1995.

[13] C. J. Stam and J. C. Reijneveld, “Graph theoretical analysis of complex networks in the brain,” *Nonlinear biomedical physics*, vol. 1, no. 1, p. 3, 2007.

[14] S.-H. Yook, H. Jeong, and A.-L. Barabási, “Modeling the internet’s large-scale topology,” *Proceedings of the National Academy of Sciences*, vol. 99, no. 21, pp. 13382–13386, 2002.

[15] S. L. Tauro, C. Palmer, G. Siganos, and M. Faloutsos, “A simple conceptual model for the internet topology,” in *Global Telecommunications Conference, 2001. GLOBECOM’01. IEEE*, vol. 3, pp. 1667–1671, IEEE, 2001.

[16] S. Arianos, E. Bompard, A. Carbone, and F. Xue, “Power grid vulnerability: A complex network approach,” *Chaos: An Interdisciplinary Journal of Nonlinear Science*, vol. 19, no. 1, p. 013119, 2009.

[17] J. Balthrop, S. Forrest, M. E. Newman, and M. M. Williamson, “Technological networks and the spread of computer viruses,” *arXiv preprint cs/0407048*, 2004.

[18] L. Euler, “Solutio problematis ad geometriam situs pertinentis,” *Commentarii academiae scientiarum Petropolitanae*, vol. 8, pp. 128–140, 1741.

[19] M. E. Newman, “The structure and function of complex networks,” *SIAM review*, vol. 45, no. 2, pp. 167–256, 2003.

[20] F. A. Azevedo, L. R. Carvalho, L. T. Grinberg, J. M. Farfel, R. E. Ferretti, R. E. Leite, R. Lent, S. Herculano-Houzel, *et al.*, “Equal numbers of neuronal and nonneuronal cells make the human brain an isometrically scaled-up primate brain,” *Journal of Comparative Neurology*, vol. 513, no. 5, pp. 532–541, 2009.

[21] Z. Fan, G. Chen, and Y. Zhang, “Using topological characteristics to evaluate complex network models can be misleading,” *arXiv preprint arXiv:1011.0126*, 2010.

[22] P. Erdös and A. Rényi, "On random graphs," *Publicationes Mathematicae*, vol. 6, pp. 290–297, 1959.

[23] J. Leskovec, D. Chakrabarti, J. Kleinberg, C. Faloutsos, and Z. Ghahramani, "Kronecker graphs: An approach to modeling networks," *The Journal of Machine Learning Research*, vol. 11, pp. 985–1042, 2010.

[24] R. Clegg, R. Landa, H. Haddadi, and M. Rio, "Measuring the likelihood of models for network evolution," in *INFOCOM Workshops 2009, IEEE*, pp. 1–6, IEEE, 2009.

[25] F. Chung and L. Lu, "The average distance in a random graph with given expected degrees," *Internet Mathematics*, vol. 1, no. 1, pp. 91–113, 2004.

[26] J. Leskovec and C. Faloutsos, "Scalable modeling of real graphs using kronecker multiplication," in *Proceedings of the 24th international conference on Machine learning*, pp. 497–504, ACM, 2007.

[27] S. Wasserman and P. Pattison, "Logit models and logistic regressions for social networks: I. an introduction to markov graphs andp," *Psychometrika*, vol. 61, no. 3, pp. 401–425, 1996.

[28] D. Fay, A. W. Moore, K. Brown, M. Filosi, and G. Jurman, "Graph metrics as summary statistics for approximate bayesian computation with application to network model parameter estimation," *Journal of Complex Networks*, p. cnu009, 2014.

[29] A. Bailey, M. Ventresca, and B. Ombuki-Berman, "Automatic generation of graph models for complex networks by genetic programming," in *Proceedings of the Fourteenth International Conference on Genetic and Evolutionary Computation Conference*, GECCO '12, (New York, NY, USA), pp. 711–718, ACM, 2012.

[30] A. Bailey, B. Ombuki-Berman, and M. Ventresca, "Automatic inference of hierarchical graph models using genetic programming with an application to cortical networks," in *Proceeding of the Fifteenth Annual Conference on Genetic and Evolutionary Computation Conference*, GECCO '13, (New York, NY, USA), pp. 893–900, ACM, 2013.

[31] A. Bailey, M. Ventresca, and B. Ombuki-Berman, “Genetic programming for the automatic inference of graph models for complex networks,” *IEEE Transactions on Evolutionary Computation*, 2013.

[32] J. R. Koza, *Genetic Programming: vol. 1, On the programming of computers by means of natural selection*, vol. 1. MIT press, 1992.

[33] R. Poli, W. W. B. Langdon, N. F. McPhee, and J. R. Koza, *A field guide to genetic programming.* Lulu. com, 2008.

[34] C. Darwin, “On the origins of species by means of natural selection,” *London: Murray*, 1859.

[35] A.-L. Barabási and R. Albert, “Emergence of scaling in random networks,” *science*, vol. 286, no. 5439, pp. 509–512, 1999.

[36] J. Leskovec, J. Kleinberg, and C. Faloutsos, “Graphs over time: densification laws, shrinking diameters and possible explanations,” in *Proceedings of the eleventh ACM SIGKDD international conference on Knowledge discovery in data mining*, pp. 177–187, ACM, 2005.

[37] M. R. Medland, K. R. Harrison, and B. Ombuki-Berman, “Incorporating expert knowledge in object-oriented genetic programming,” in *Proceedings of the 2014 Conference on Genetic and Evolutionary Computation Companion*, GECCO Comp ’14, (New York, NY, USA), pp. 145–146, ACM, 2014.

[38] M. R. Medland, K. R. Harrison, and B. Ombuki-Berman, “Demonstrating the power of object-oriented genetic programming via the inference of graph models for complex networks,” in *Nature and Biologically Inspired Computing (NaBIC), 2014 Sixth World Congress on*, NaBIC ’14, pp. 305–311, IEEE, 2014.

[39] R. Pastor-Satorras and A. Vespignani, *Evolution and structure of the Internet: A statistical physics approach.* Cambridge University Press, 2007.

[40] R. Albert and A.-L. Barabási, “Statistical mechanics of complex networks,” *Reviews of modern physics*, vol. 74, no. 1, p. 47, 2002.

[41] V. D. Blondel, J.-L. Guillaume, R. Lambiotte, and E. Lefebvre, “Fast unfolding of communities in large networks,” *Journal of Statistical Mechanics: Theory and Experiment*, vol. 2008, no. 10, p. P10008, 2008.

[42] J. C. Almack, “The influence of intelligence on the selection of associates,” *School and Society*, vol. 16, pp. 529–530, 1922.

[43] B. Wellman, “The school child’s choice of companions,” *The Journal of Educational Research*, pp. 126–132, 1926.

[44] H. Russell Bernard, E. C. Johnsen, P. D. Killworth, and S. Robinson, “Estimating the size of an average personal network and of an event subpopulation: Some empirical results,” *Social science research*, vol. 20, no. 2, pp. 109–121, 1991.

[45] S. Milgram, “The small world problem,” *Psychology today*, vol. 2, no. 1, pp. 60–67, 1967.

[46] J. Moody, “Race, school integration, and friendship segregation in america1,” *American Journal of Sociology*, vol. 107, no. 3, pp. 679–716, 2001.

[47] J. W. Grossman, “The evolution of the mathematical research collaboration graph,” *Congressus Numerantium*, pp. 201–212, 2002.

[48] M. E. Newman, “The structure of scientific collaboration networks,” *Proceedings of the National Academy of Sciences*, vol. 98, no. 2, pp. 404–409, 2001.

[49] A. S. Klovdahl, J. J. Potterat, D. E. Woodhouse, J. B. Muth, S. Q. Muth, and W. W. Darrow, “Social networks and infectious disease: The colorado springs study,” *Social science & medicine*, vol. 38, no. 1, pp. 79–88, 1994.

[50] F. Liljeros, C. R. Edling, L. A. N. Amaral, H. E. Stanley, and Y. Åberg, “The web of human sexual contacts,” *Nature*, vol. 411, no. 6840, pp. 907–908, 2001.

[51] M. J. Salganik and D. D. Heckathorn, “Sampling and estimation in hidden populations using respondent-driven sampling,” *Sociological methodology*, vol. 34, no. 1, pp. 193–240, 2004.

[52] K. Lewis, J. Kaufman, M. Gonzalez, A. Wimmer, and N. Christakis, “Tastes, ties, and time: A new social network dataset using facebook. com,” *Social networks*, vol. 30, no. 4, pp. 330–342, 2008.

[53] J. Travers and S. Milgram, “An experimental study of the small world problem,” *Sociometry*, pp. 425–443, 1969.

[54] A. Rapoport and W. J. Horvath, “A study of a large sociogram,” *Behavioral Science*, vol. 6, no. 4, pp. 279–291, 1961.

[55] A. Davis, B. B. Gardner, and M. R. Gardner, *Deep South.* University of Chicago Press, Chicago, 1941.

[56] C. Schulz, A. Mazloumian, A. M. Petersen, O. Penner, and D. Helbing, “Exploiting citation networks for large-scale author name disambiguation,” *arXiv preprint arXiv:1401.6157*, 2014.

[57] E. Mones, P. Pollner, and T. Vicsek, “Universal hierarchical behavior of citation networks,” *Journal of Statistical Mechanics: Theory and Experiment*, vol. 2014, no. 5, p. P05023, 2014.

[58] F. Radicchi, S. Fortunato, and A. Vespignani, “Citation networks,” in *Models of Science Dynamics* (A. Scharnhorst, K. Brner, and P. van den Besselaar, eds.), Understanding Complex Systems, pp. 233–257, Springer Berlin Heidelberg, 2012.

[59] M. E. Newman, “Finding community structure in networks using the eigenvectors of matrices,” *Physical review E*, vol. 74, no. 3, p. 036104, 2006.

[60] D. J. de Solla Price, “Networks of scientific papers,” *Science*, vol. 149, no. 3683, pp. 510–515, 1965.

[61] O. H. Lowry, N. J. Rosebrough, A. L. Farr, R. J. Randall, *et al.*, “Protein measurement with the folin phenol reagent,” *J biol Chem*, vol. 193, no. 1, pp. 265–275, 1951.

[62] L. Négyessy, T. Nepusz, L. Kocsis, and F. Bazsó, “Prediction of the main cortical areas and connections involved in the tactile function of the visual cortex by network analysis,” *European Journal of Neuroscience*, vol. 23, no. 7, pp. 1919–1930, 2006.

[63] K. E. Stephan, C.-C. Hilgetag, G. A. Burns, M. A. O’Neill, M. P. Young, and R. Kotter, “Computational analysis of functional connectivity between areas of primate cerebral cortex,” *Philosophical Transactions of the Royal Society of London. Series B: Biological Sciences*, vol. 355, no. 1393, pp. 111–126, 2000.

[64] M. Girvan and M. E. Newman, “Community structure in social and biological networks,” *Proceedings of the National Academy of Sciences*, vol. 99, no. 12, pp. 7821–7826, 2002.

[65] A. Clauset, C. Moore, and M. E. Newman, "Hierarchical structure and the prediction of missing links in networks," *Nature*, vol. 453, no. 7191, pp. 98–101, 2008.

[66] L. C. Freeman, "A set of measures of centrality based on betweenness," *Sociometry*, pp. 35–41, 1977.

[67] L. C. Freeman, "Centrality in social networks: conceptual clarification," *Social networks*, vol. 1, no. 3, pp. 215–239, 1979.

[68] R. S. Burt, *Structural holes: The social structure of competition.* Harvard university press, 2009.

[69] P. Bonacich, "Power and centrality: A family of measures," *American journal of sociology*, pp. 1170–1182, 1987.

[70] S. Brin and L. Page, "The anatomy of a large-scale hypertextual web search engine," *Computer networks and ISDN systems*, vol. 30, no. 1, pp. 107–117, 1998.

[71] N. Perra and S. Fortunato, "Spectral centrality measures in complex networks," *Physical Review E*, vol. 78, no. 3, p. 036107, 2008.

[72] M. E. Newman, "Modularity and community structure in networks," *Proceedings of the National Academy of Sciences*, vol. 103, no. 23, pp. 8577–8582, 2006.

[73] D. J. Watts and S. H. Strogatz, "Collective dynamics of small-worldnetworks," *nature*, vol. 393, no. 6684, pp. 440–442, 1998.

[74] S. N. Dorogovtsev and J. F. F. Mendes, "Evolution of networks with aging of sites," *Physical Review E*, vol. 62, no. 2, p. 1842, 2000.

[75] A. Goldenberg, A. X. Zheng, S. E. Fienberg, and E. M. Airoldi, "A survey of statistical network models," *Foundations and Trends® in Machine Learning*, vol. 2, no. 2, pp. 129–233, 2010.

[76] S. Boccaletti, V. Latora, Y. Moreno, M. Chavez, and D.-U. Hwang, "Complex networks: Structure and dynamics," *Physics reports*, vol. 424, no. 4, pp. 175–308, 2006.

[77] R. Solomonoff and A. Rapoport, "Connectivity of random nets," *The bulletin of mathematical biophysics*, vol. 13, no. 2, pp. 107–117, 1951.

[78] D. J. Montana, "Strongly typed genetic programming," *Evolutionary computation*, vol. 3, no. 2, pp. 199–230, 1995.

[79] D. Augusto and H. J. C. Barbosa, "Symbolic regression via genetic programming," in *Neural Networks, 2000. Proceedings. Sixth Brazilian Symposium on*, pp. 173–178, 2000.

[80] E. Vladislavleva, G. Smits, and D. den Hertog, "Order of nonlinearity as a complexity measure for models generated by symbolic regression via pareto genetic programming," *Evolutionary Computation, IEEE Transactions on*, vol. 13, pp. 333–349, April 2009.

[81] J. R. Koza, I. Bennett, Forrest H., and O. Stiffelman, "Genetic programming as a darwinian invention machine," in *Genetic Programming* (R. Poli, P. Nordin, W. B. Langdon, and T. C. Fogarty, eds.), vol. 1598 of *Lecture Notes in Computer Science*, pp. 93–108, Springer Berlin Heidelberg, 1999.

[82] S. Bergen and B. Ross, "Evolutionary art using summed multi-objective ranks," in *Genetic Programming Theory and Practice VIII* (R. Riolo, T. McConaghy, and E. Vladislavleva, eds.), vol. 8 of *Genetic and Evolutionary Computation*, pp. 227–244, Springer New York, 2011.

[83] K. Sastry and D. Goldberg, "Probabilistic model building and competent genetic programming," in *Genetic Programming Theory and Practice* (R. Riolo and B. Worzel, eds.), vol. 6 of *Genetic Programming Series*, pp. 205–220, Springer US, 2003.

[84] G. M. Khan, J. F. Miller, and D. M. Halliday, "Coevolution of intelligent agents using cartesian genetic programming," in *Proceedings of the 9^{th} Annual Conference on Genetic and Evolutionary Computation*, GECCO '07, (New York, NY, USA), pp. 269–276, ACM, 2007.

[85] A. Runka, "Genetic programming for the robocup rescue simulation system," Master's thesis, Brock University, St. Catharines, ON, Canada, 2011.

[86] R. Abbott, "Object-oriented genetic programming, an initial implementation," in *International Conference on Machine Learning: Models, Technologies and Applications*, pp. 26–30, 2003.

[87] M. Brameier and W. Banzhaf, “Explicit control of diversity and effective variation distance in linear genetic programming,” in *Genetic Programming* (J. Foster, E. Lutton, J. Miller, C. Ryan, and A. Tettamanzi, eds.), vol. 2278 of *Lecture Notes in Computer Science*, pp. 37–49, Springer Berlin Heidelberg, 2002.

[88] W. Banzhaf, “Genetic programming for pedestrians,” in *Proceedings of the 5th International Conference on Genetic Algorithms*, p. 628, Morgan Kaufmann Publishers Inc., 1993.

[89] W. Banzhaf, “Genotype-phenotype-mapping and neutral variationa case study in genetic programming,” in *Parallel problem solving from naturePPSN III*, pp. 322–332, Springer, 1994.

[90] M. Oltean and C. Grosan, “A comparison of several linear genetic programming techniques,” *Complex Systems*, vol. 14, no. 4, pp. 285–314, 2003.

[91] H. G. Rice, “Classes of recursively enumerable sets and their decision problems,” *Transactions of the American Mathematical Society*, vol. 74, no. 2, pp. 358–366, 1953.

[92] R. Fisher, *Statistical Methods for Research Workers.* Oliver and Boyd, 1925.

[93] T. Fawcett, “An introduction to roc analysis,” *Pattern recognition letters*, vol. 27, no. 8, pp. 861–874, 2006.

[94] J. A. Hanley and B. J. McNeil, “The meaning and use of the area under a receiver operating characteristic (roc) curve.,” *Radiology*, vol. 143, no. 1, pp. 29–36, 1982.

[95] S. J. Mason and N. E. Graham, “Areas beneath the relative operating characteristics (roc) and relative operating levels (rol) curves: Statistical significance and interpretation,” *Quarterly Journal of the Royal Meteorological Society*, vol. 128, no. 584, pp. 2145–2166, 2002.

[96] G. Alves and Y.-K. Yu, “Accuracy evaluation of the unified p-value from combining correlated p-values,” *PloS one*, vol. 9, no. 3, p. e91225, 2014.

[97] G. Csardi and T. Nepusz, “The igraph software package for complex network research,” *InterJournal, Complex Systems*, vol. 1695, no. 5, 2006.

[98] A. Bailey, “Automatic inference of graph models for complex networks with genetic programming,” Master’s thesis, Brock University, St. Catharines, Ontario, Canada, 2013.

[99] G. Zamora-López, C. Zhou, and J. Kurths, “Graph analysis of cortical networks reveals complex anatomical communication substrate,” *Chaos: An Interdisciplinary Journal of Nonlinear Science*, vol. 19, no. 1, p. 015117, 2009.

[100] O. Sporns and J. D. Zwi, “The small world of the cerebral cortex,” *Neuroinformatics*, vol. 2, no. 2, pp. 145–162, 2004.

[101] V. M. Eguiluz, D. R. Chialvo, G. A. Cecchi, M. Baliki, and A. V. Apkarian, “Scale-free brain functional networks,” *Physical review letters*, vol. 94, no. 1, p. 018102, 2005.

[102] M. Kaiser, R. Martin, P. Andras, and M. P. Young, “Simulation of robustness against lesions of cortical networks,” *European Journal of Neuroscience*, vol. 25, no. 10, pp. 3185–3192, 2007.

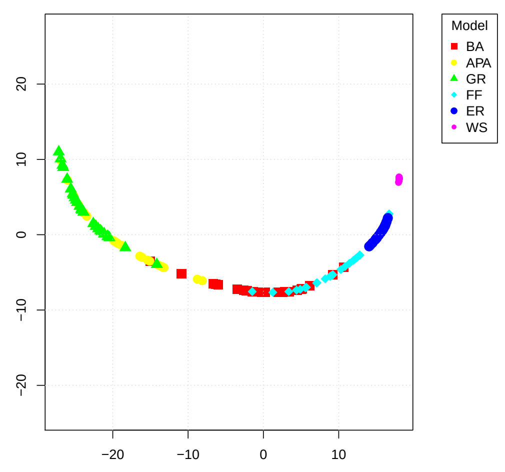
[103] T. Nepusz, L. Négyessy, G. Tusnády, and F. Bazsó, “Reconstructing cortical networks: Case of directed graphs with high level of reciprocity,” in *Handbook of Large-Scale Random Networks*, pp. 325–368, Springer, 2008.

[104] E. Costenbader and T. W. Valente, “The stability of centrality measures when networks are sampled,” *Social networks*, vol. 25, no. 4, pp. 283–307, 2003.

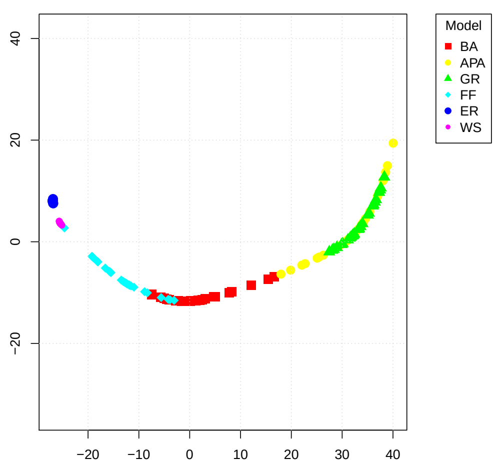
Appendix A

Supplementary Results for Chapter 5

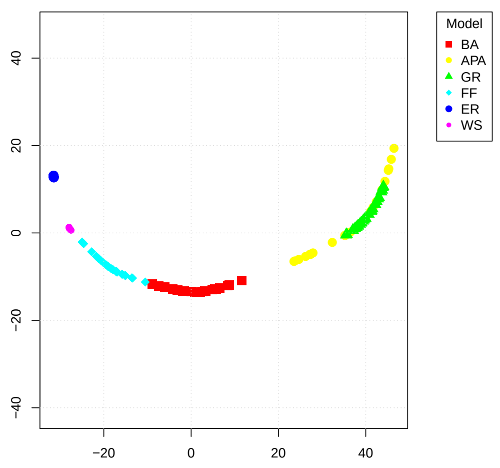
This Appendix presents MDS plots not included in Chapter 5. Figures A.1 through A.4 present MDS applied to the global network measures while Figures A.5 through A.9 depict the MDS procedure applied to all centrality measures, apart from PageRank which was included in the thesis body. Discussions regarding the plots can be found in Chapter 5. However, the plots are presented here for brevity within the thesis itself. Each of the plots indicate the variance accounted for by each axis as a measure of their goodness of fit.



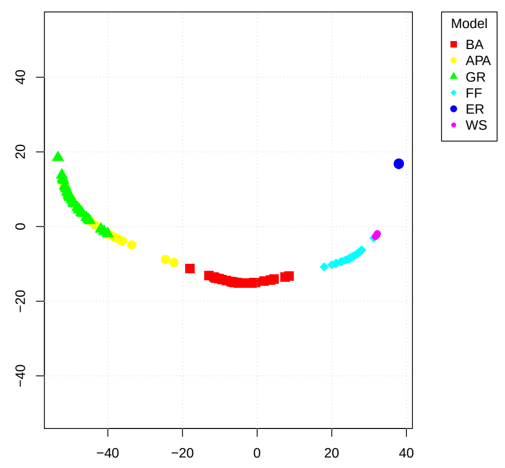
(a) 100 vertex networks. 98.63% variance accounted for on each axis.



(b) 250 vertex networks. 98.85% variance accounted for on each axis.

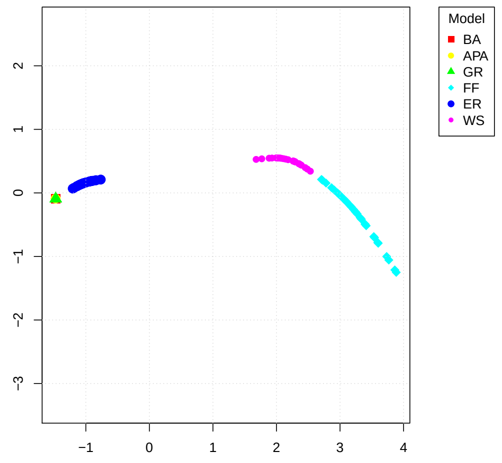


(c) 500 vertex networks. 98.59% variance accounted for on each axis.

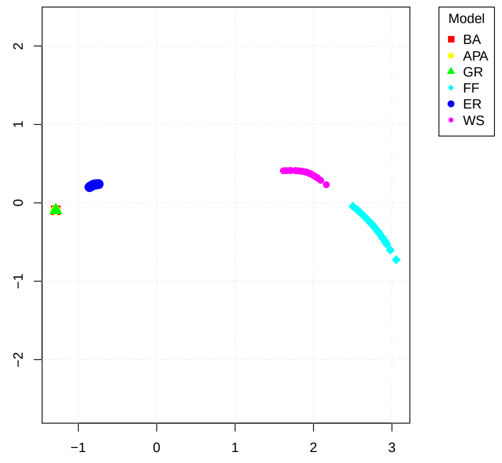


(d) 1000 vertex networks. 97.78% variance accounted for on each axis.

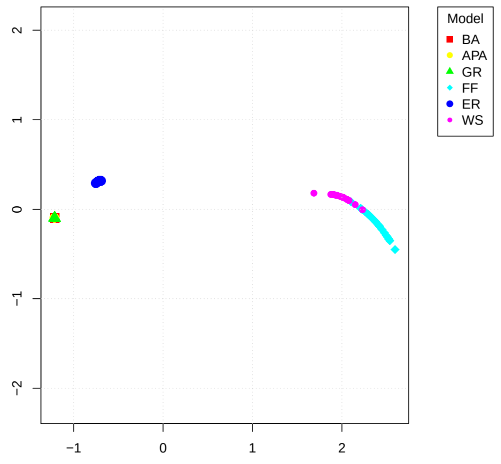
Figure A.1: MDS applied to average geodesic path length for various network sizes.



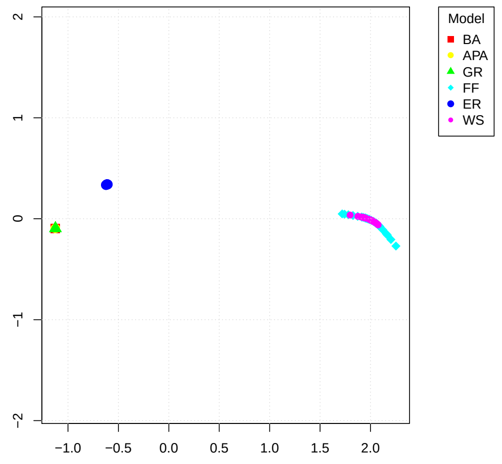
(a) 100 vertex networks. 99.50% variance accounted for on each axis.



(b) 250 vertex networks. 99.37% variance accounted for on each axis.

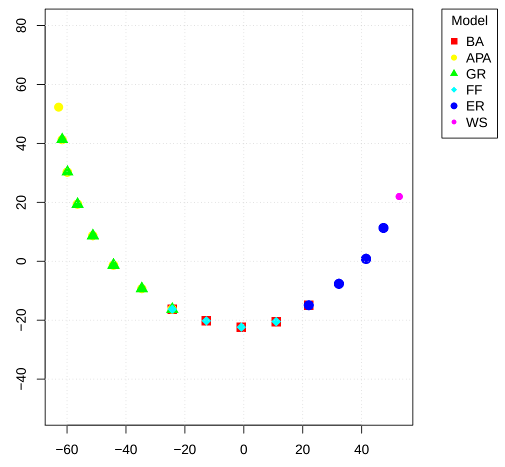


(c) 500 vertex networks. 99.50% variance accounted for on each axis.

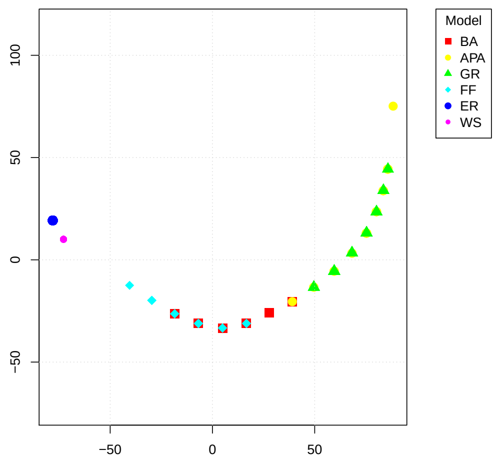


(d) 1000 vertex networks. 99.80% variance accounted for on each axis.

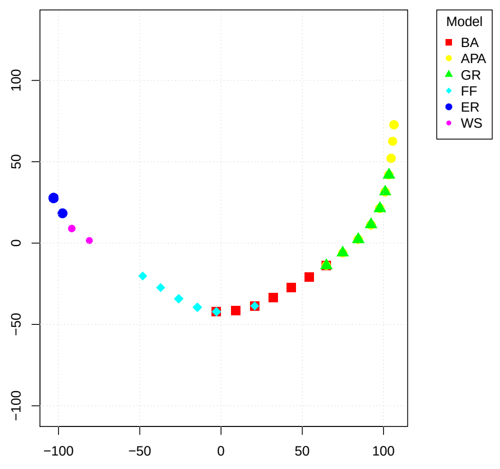
Figure A.2: MDS applied to network transitivity for various network sizes.



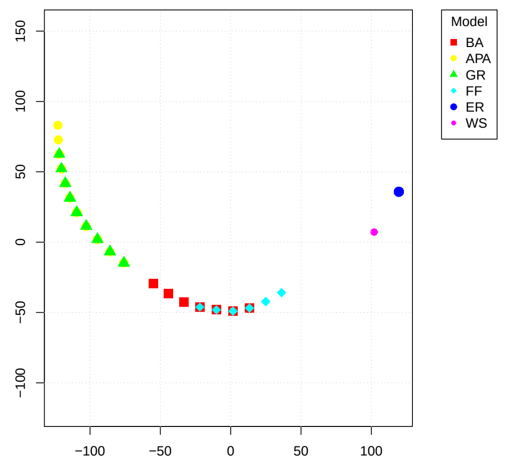
(a) 100 vertex networks. 97.78% variance accounted for on each axis.



(b) 250 vertex networks. 98.63% variance accounted for on each axis.

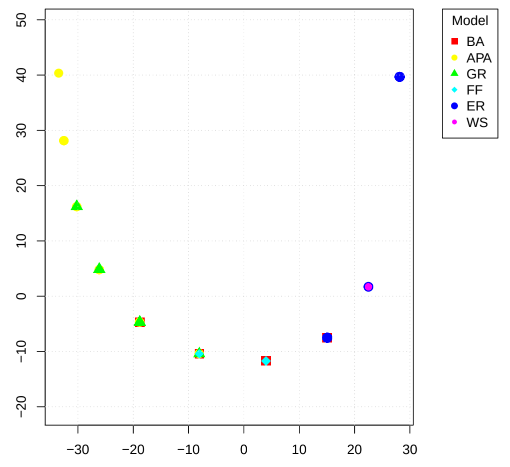


(c) 500 vertex networks. 98.57% variance accounted for on each axis.

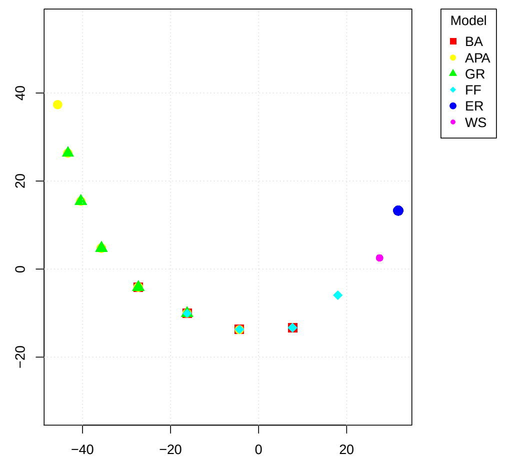


(d) 1000 vertex networks. 98.51% variance accounted for on each axis.

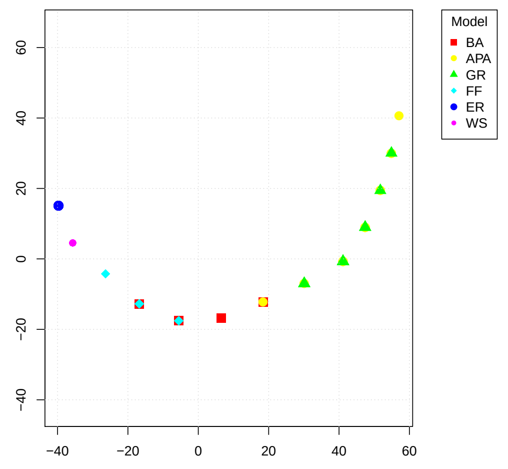
Figure A.3: MDS applied to network diameter for various network sizes.



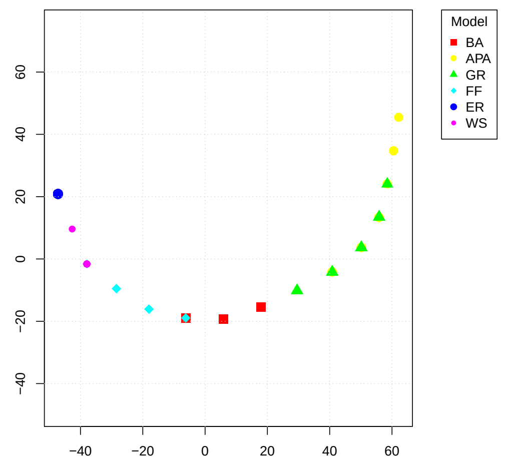
(a) 100 vertex networks. 95.74% variance accounted for on each axis.



(b) 250 vertex networks. 97.52% variance accounted for on each axis.

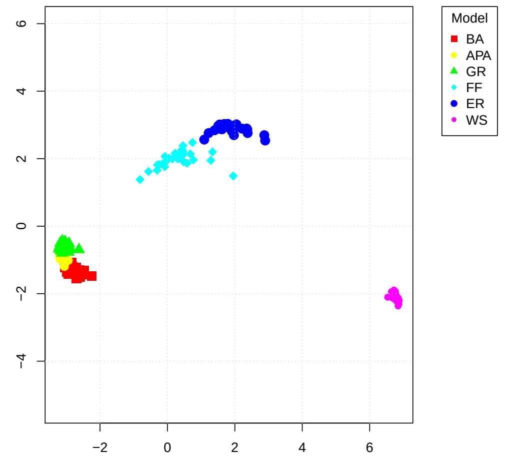


(c) 500 vertex networks. 97.87% variance accounted for on each axis.

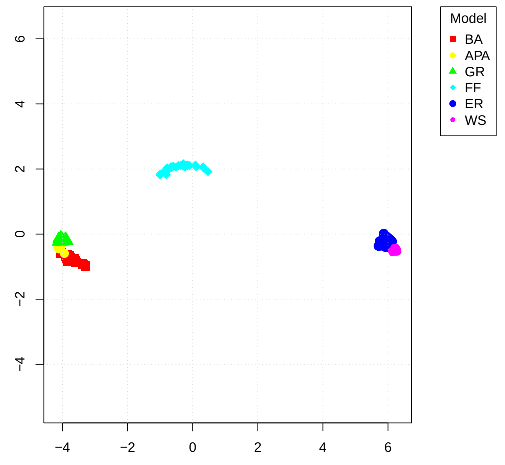


(d) 1000 vertex networks. 97.91% variance accounted for on each axis.

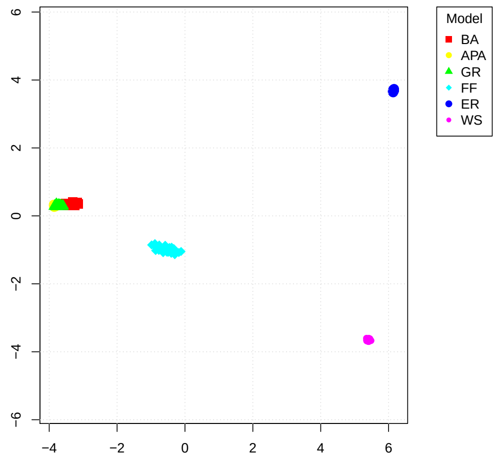
Figure A.4: MDS applied to network radius for various network sizes.



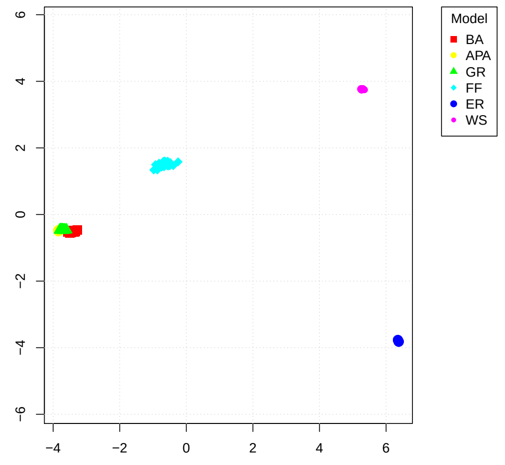
(a) 100 vertex networks. 97.57% variance accounted for on each axis.



(b) 250 vertex networks. 96.32% variance accounted for on each axis.

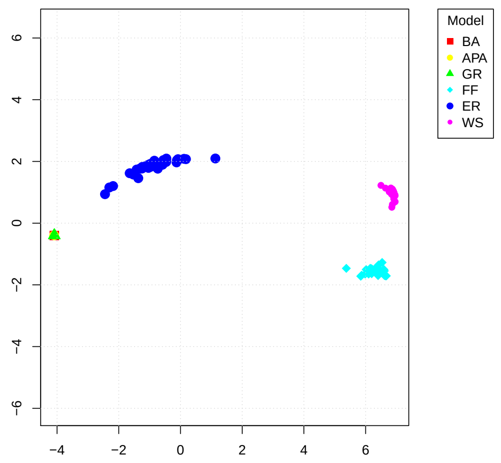


(c) 500 vertex networks. 95.42% variance accounted for on each axis.

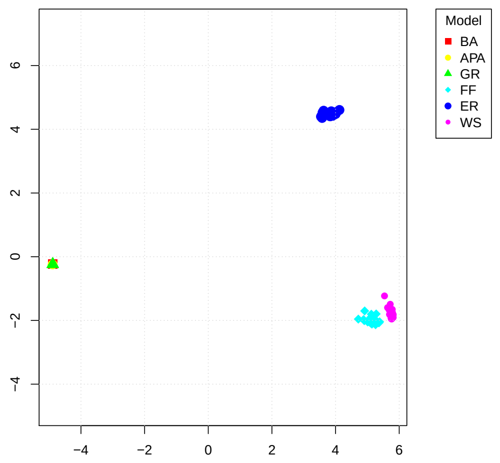


(d) 1000 vertex networks. 95.65% variance accounted for on each axis.

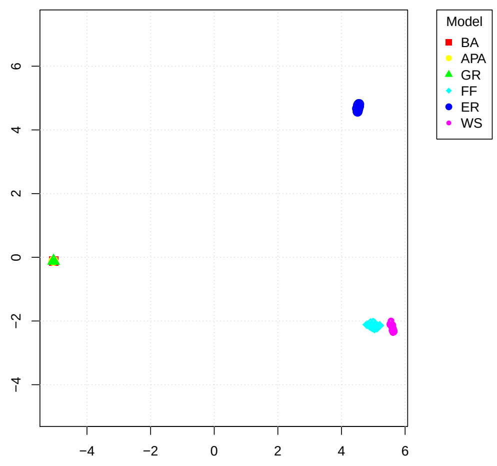
Figure A.5: MDS applied to degree distribution for various network sizes.



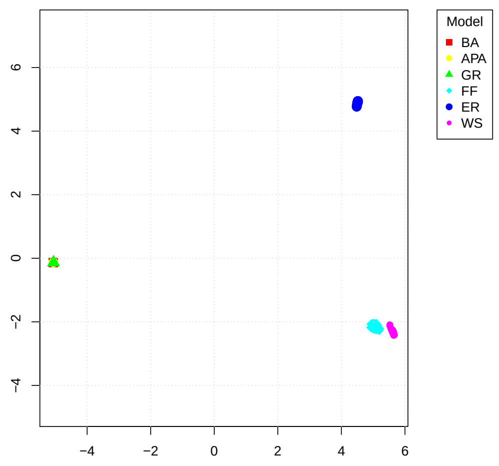
(a) 100 vertex networks. 96.43% variance accounted for on each axis.



(b) 250 vertex networks. 96.39% variance accounted for on each axis.

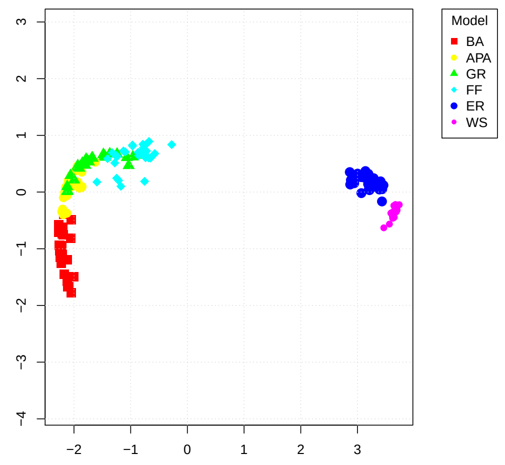


(c) 500 vertex networks. 96.56% variance accounted for on each axis.

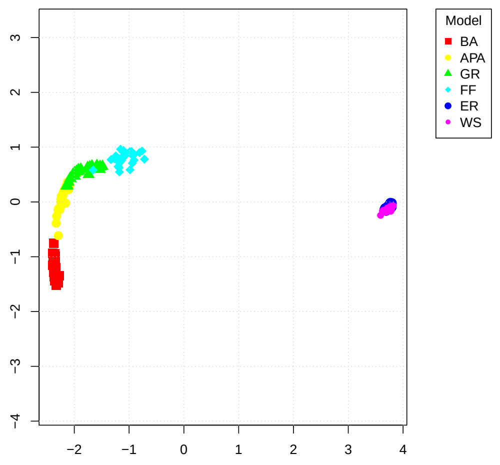


(d) 1000 vertex networks. 96.67% variance accounted for on each axis.

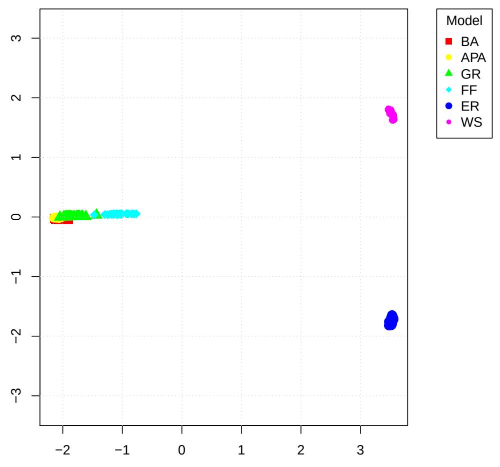
Figure A.6: MDS applied to local clustering coefficient for various network sizes.



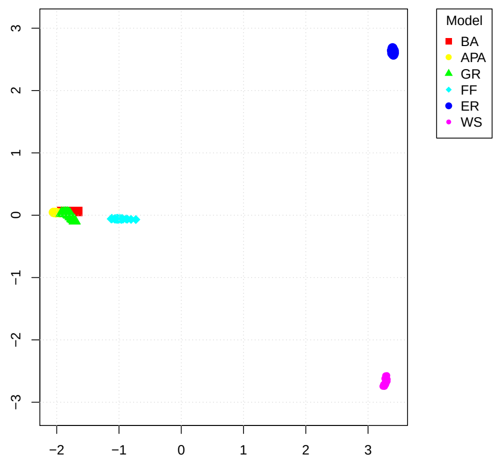
(a) 100 vertex networks. 96.74% variance accounted for on each axis.



(b) 250 vertex networks. 96.91% variance accounted for on each axis.

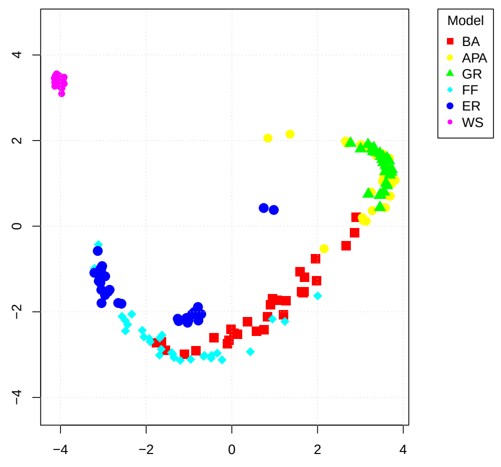


(c) 500 vertex networks. 93.01% variance accounted for on each axis.

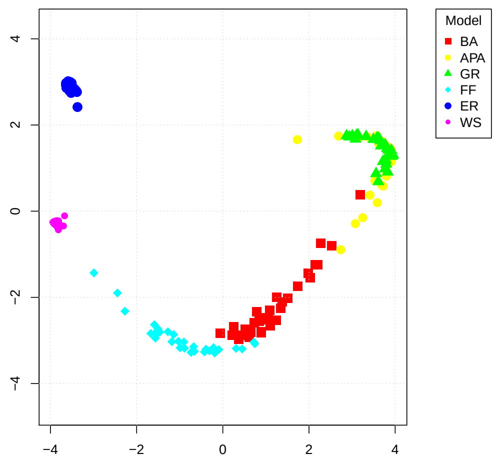


(d) 1000 vertex networks. 92.84% variance accounted for on each axis.

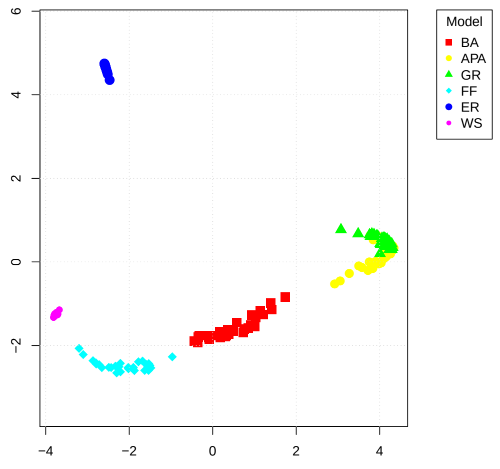
Figure A.7: MDS applied to betweenness centrality for various network sizes.



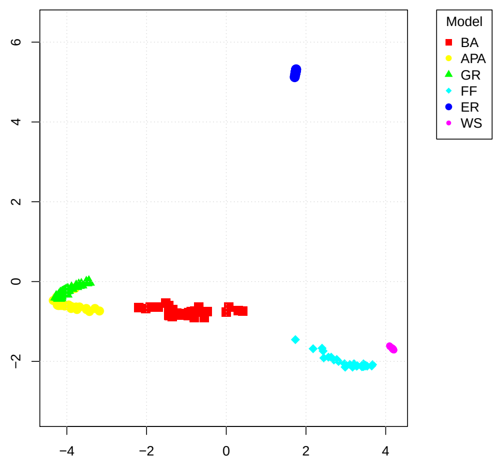
(a) 100 vertex networks. 86.88% variance accounted for on each axis.



(b) 250 vertex networks. 77.95% variance accounted for on each axis.

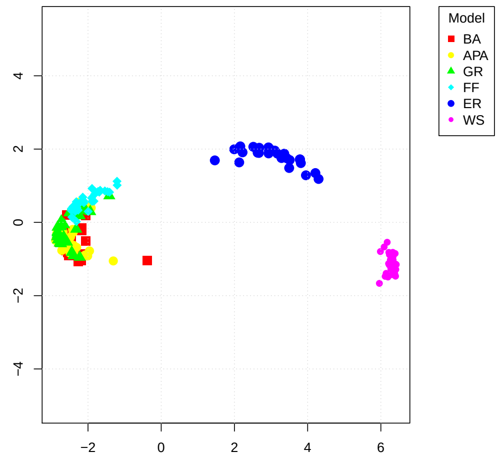


(c) 500 vertex networks. 77.94% variance accounted for on each axis.

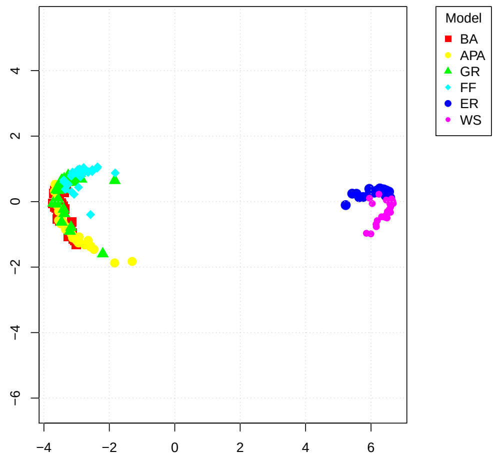


(d) 1000 vertex networks. 81.83% variance accounted for on each axis.

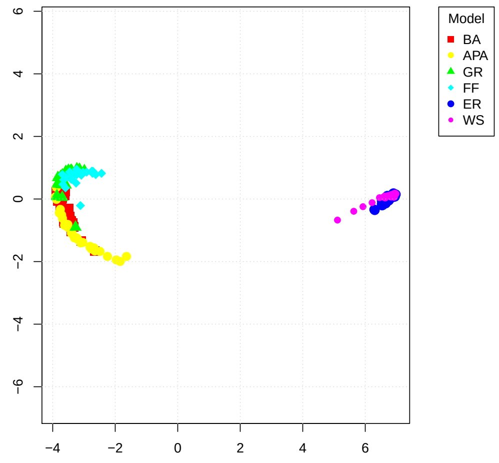
Figure A.8: MDS applied to closeness centrality for various network sizes.



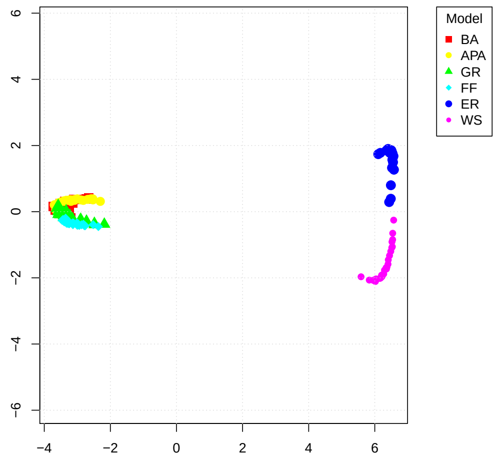
(a) 100 vertex networks. 96.23% variance accounted for on each axis.



(b) 250 vertex networks. 97.91% variance accounted for on each axis.



(c) 500 vertex networks. 98.31% variance accounted for on each axis.



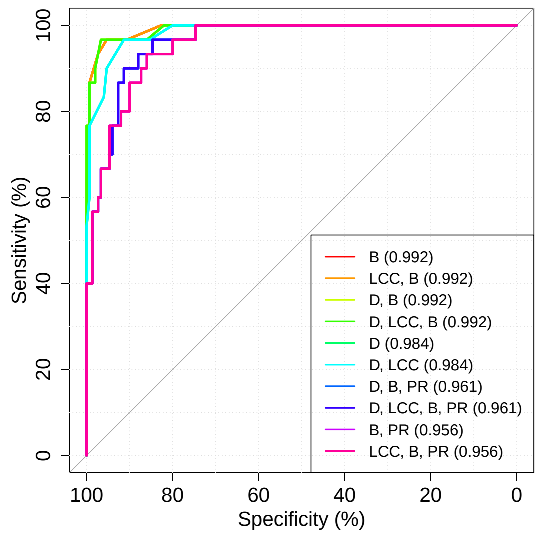
(d) 1000 vertex networks. 95.78% variance accounted for on each axis.

Figure A.9: MDS applied to eigenvector centrality for various network sizes.

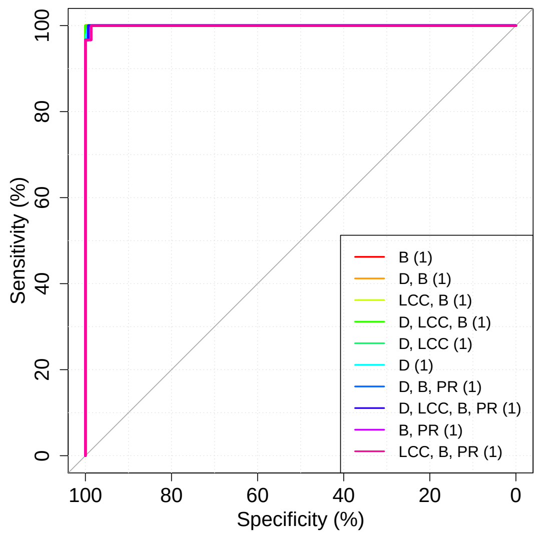
Appendix B

Supplementary Results for Chapter 6

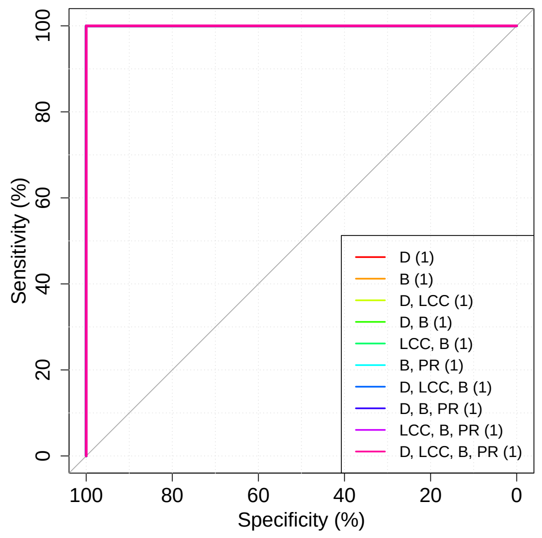
This appendix gives supplementary results to those in Chapter 6. Figures B.1 to B.6 present the results of the meta-analysis procedure applied against a target generated by each of the six models model. Many of the plots, especially on the larger networks, exemplified that perfect classifiers could be formed by some set of measures for most models. In fact, the only model in which this was not true for 1000 vertex target networks was the APA model, where the best classifier obtained an AUC of 0.999. This is arguably an insignificantly difference, but nonetheless corresponds to a non-perfect classifier. No further discussion of these results is given as they are only provided for the interested reader.



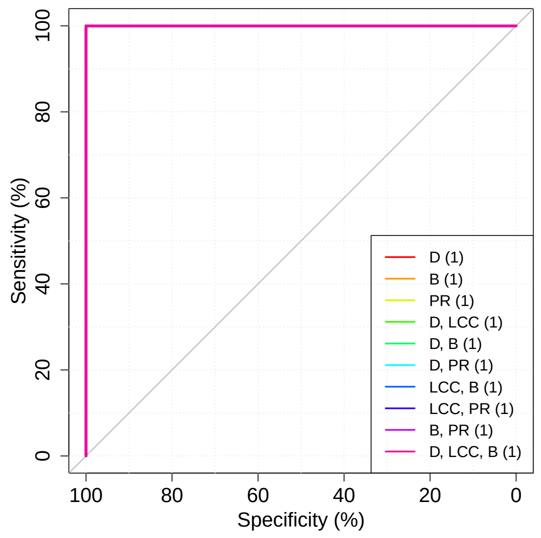
(a) 100 vertex networks.



(b) 250 vertex networks.

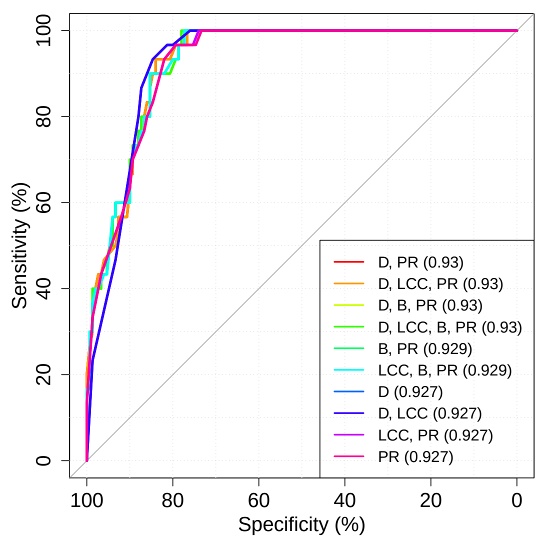


(c) 500 vertex networks.

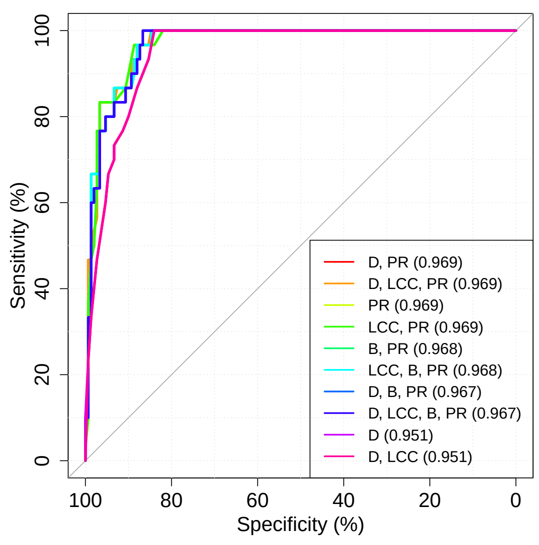


(d) 1000 vertex networks.

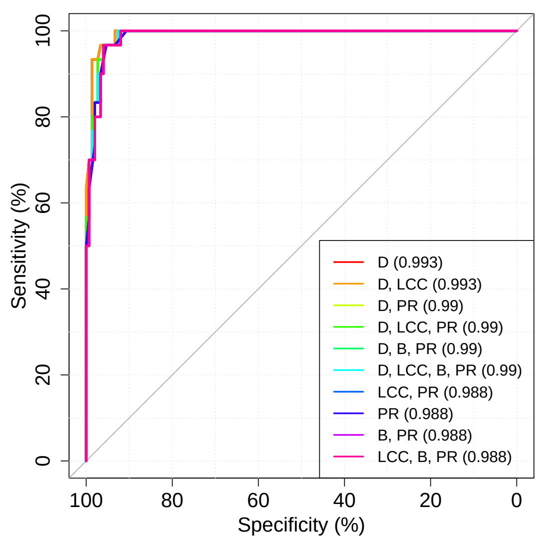
Figure B.1: ROC curves depicting the ten measure sets with the highest area under the curve (AUC) values, shown in the legend, for various network sizes using a target graph generated by the BA model.



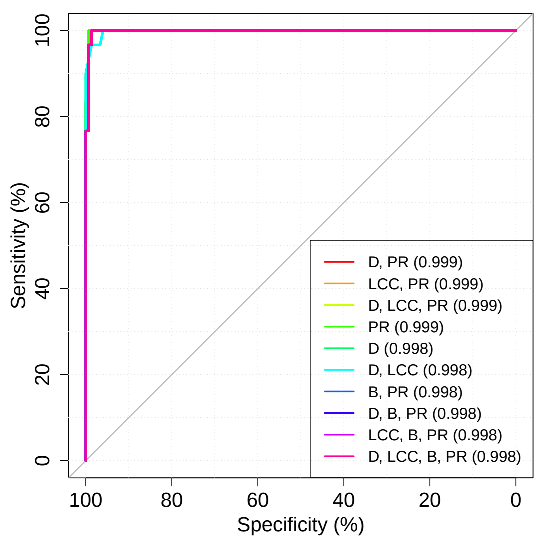
(a) 100 vertex networks.



(b) 250 vertex networks.

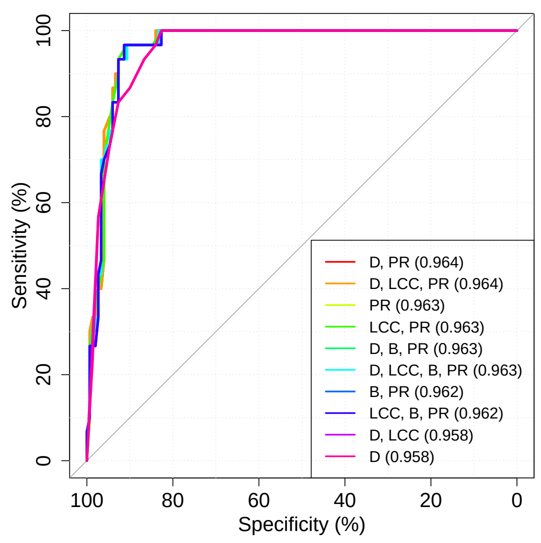


(c) 500 vertex networks.

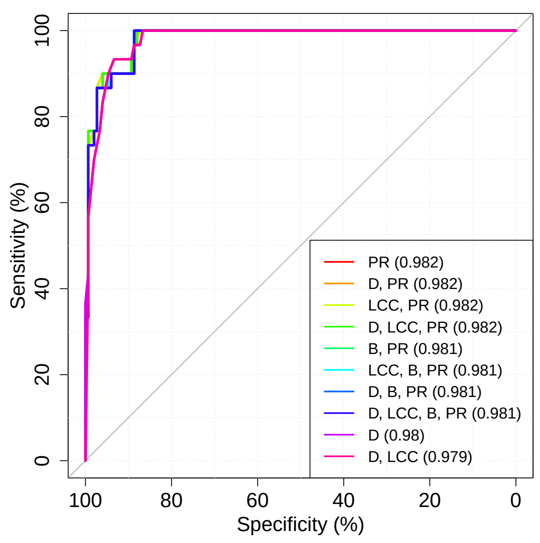


(d) 1000 vertex networks.

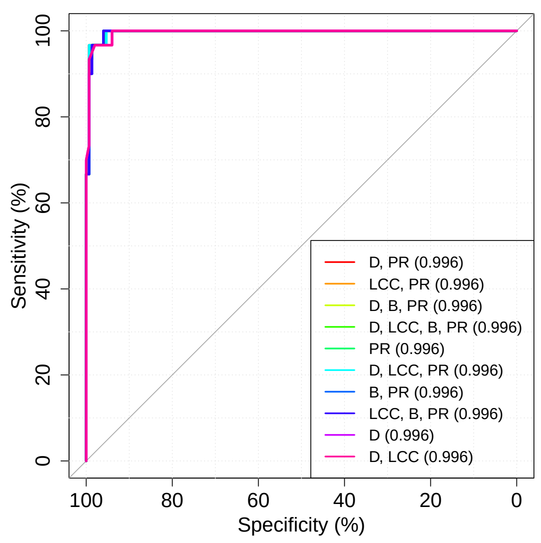
Figure B.2: ROC curves depicting the ten measure sets with the highest area under the curve (AUC) values, shown in the legend, for various network sizes using a target graph generated by the APA model.



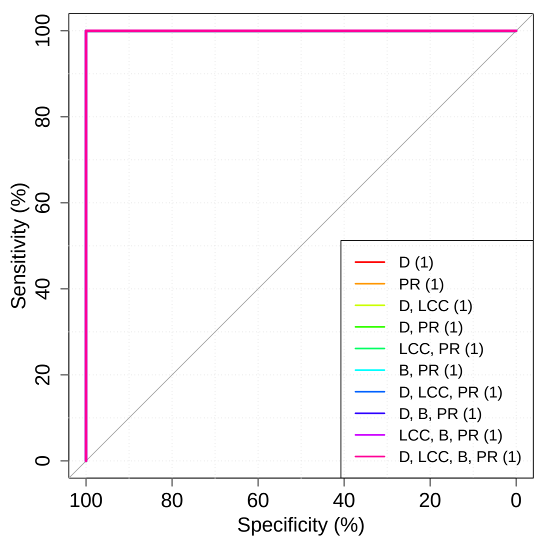
(a) 100 vertex networks.



(b) 250 vertex networks.

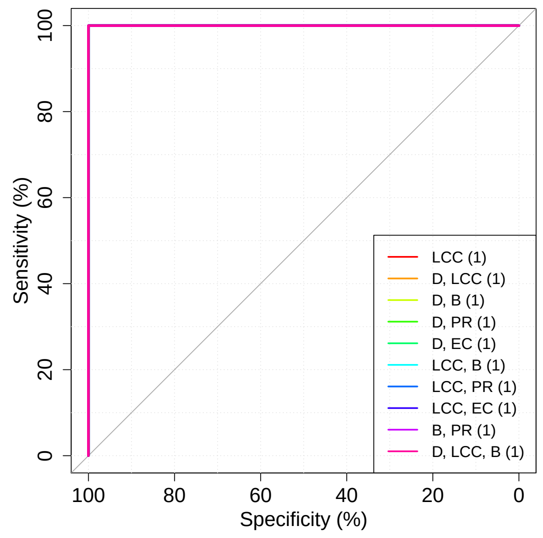


(c) 500 vertex networks.

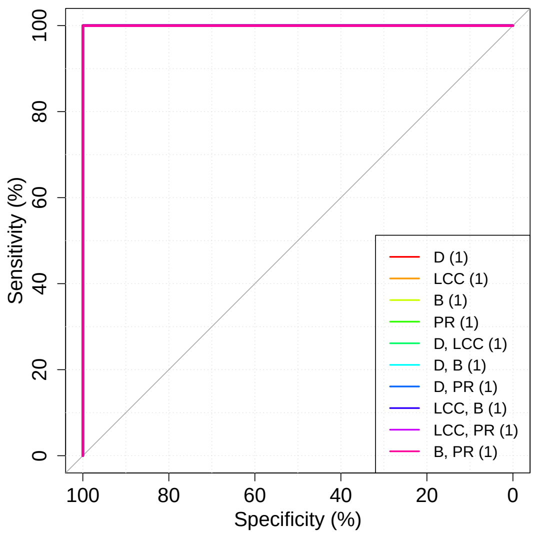


(d) 1000 vertex networks.

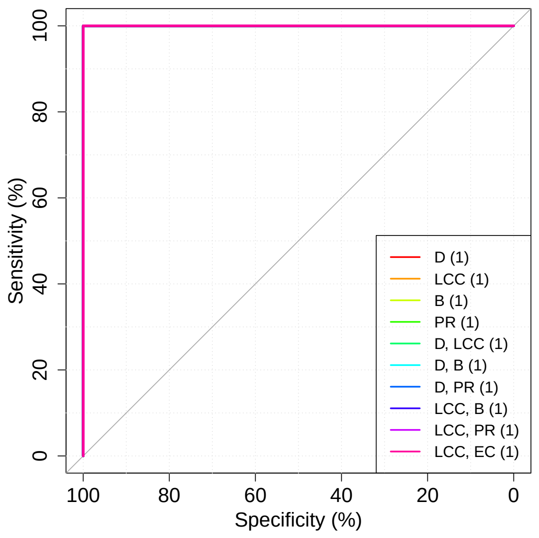
Figure B.3: ROC curves depicting the ten measure sets with the highest area under the curve (AUC) values, shown in the legend, for various network sizes using a target graph generated by the GR model.



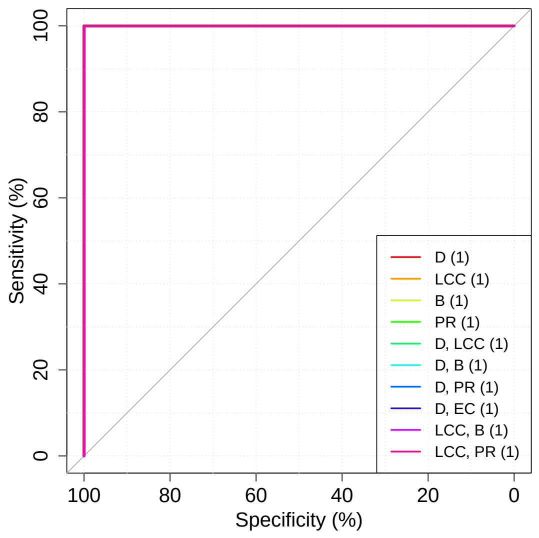
(a) 100 vertex networks.



(b) 250 vertex networks.

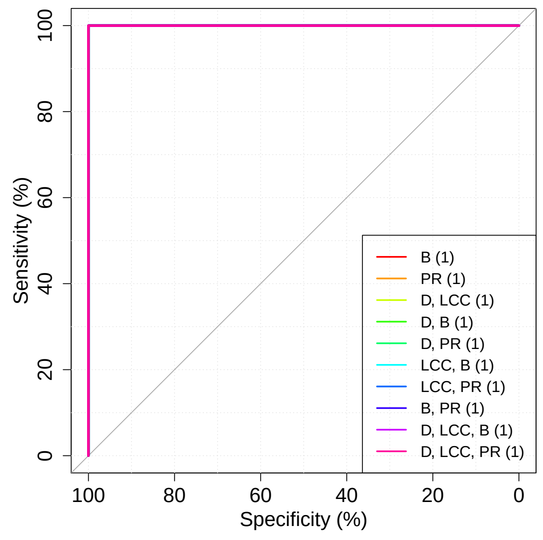


(c) 500 vertex networks.

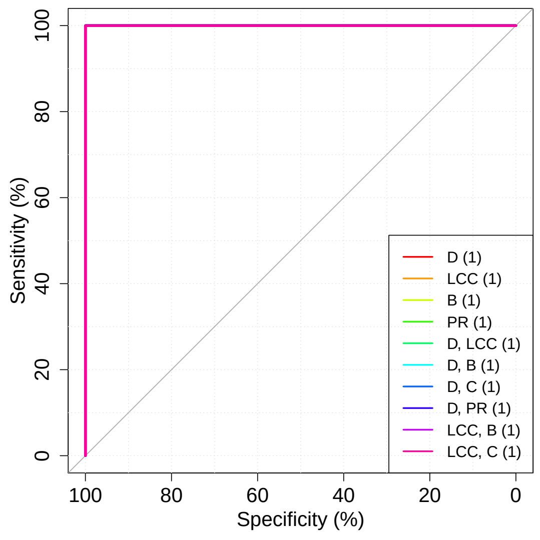


(d) 1000 vertex networks.

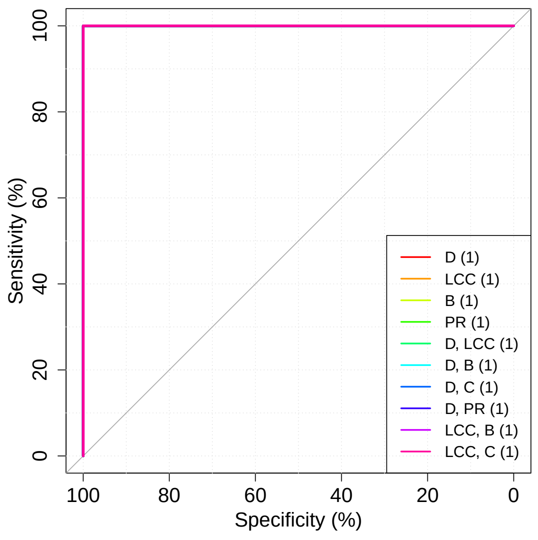
Figure B.4: ROC curves depicting the ten measure sets with the highest area under the curve (AUC) values, shown in the legend, for various network sizes using a target graph generated by the FF model.



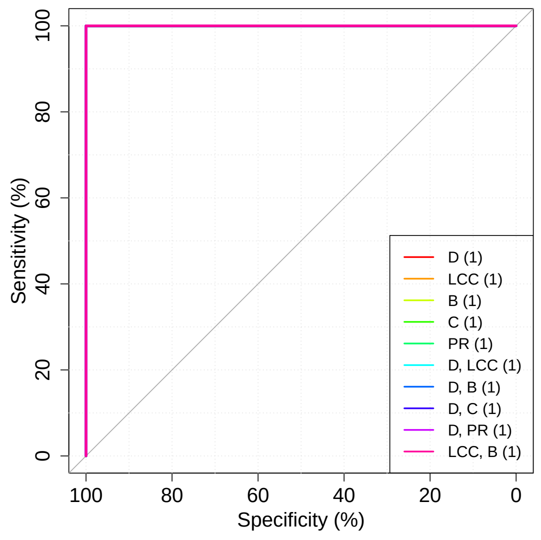
(a) 100 vertex networks.



(b) 250 vertex networks.

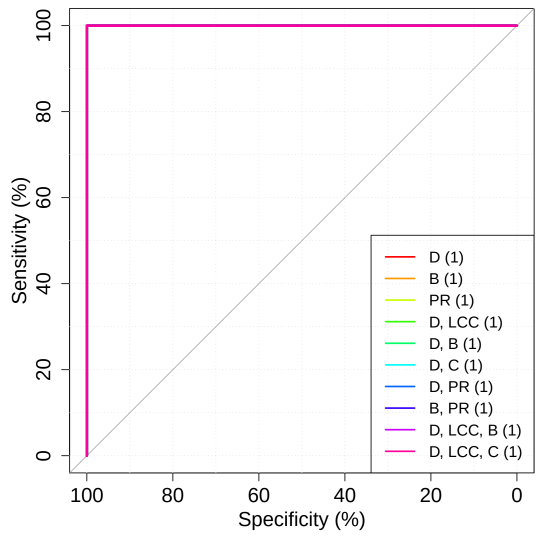


(c) 500 vertex networks.

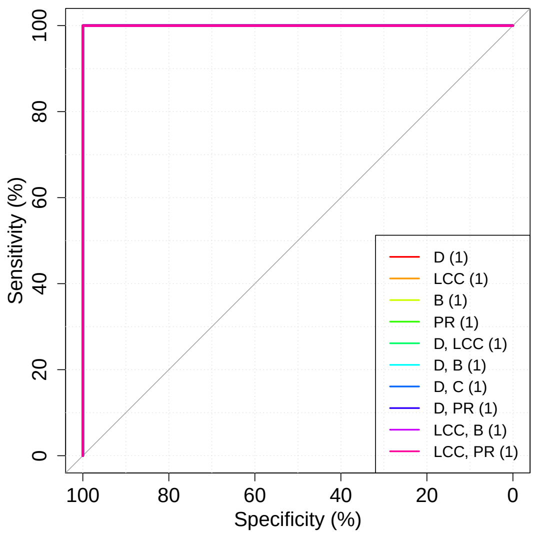


(d) 1000 vertex networks.

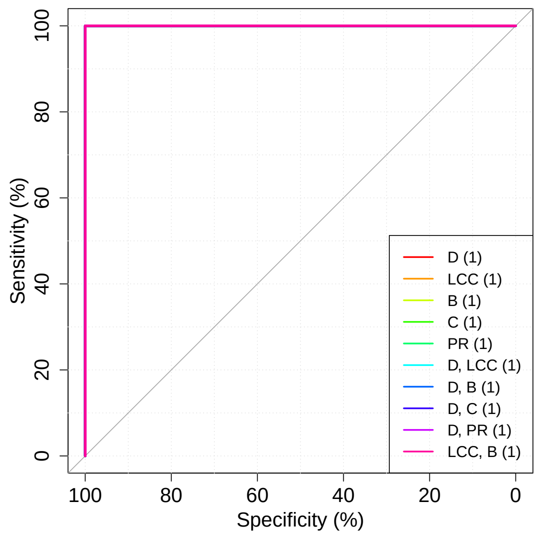
Figure B.5: ROC curves depicting the ten measure sets with the highest area under the curve (AUC) values, shown in the legend, for various network sizes using a target graph generated by the ER model.



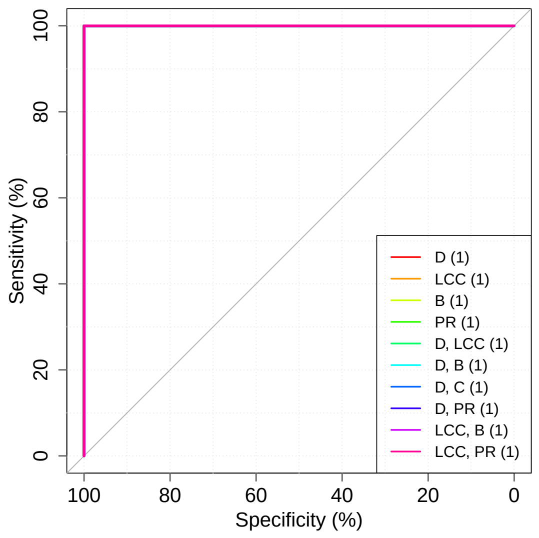
(a) 100 vertex networks.



(b) 250 vertex networks.



(c) 500 vertex networks.



(d) 1000 vertex networks.

Figure B.6: ROC curves depicting the ten measure sets with the highest area under the curve (AUC) values, shown in the legend, for various network sizes using a target graph generated by the WS model.

Appendix C

Supplementary Results for Chapter 8

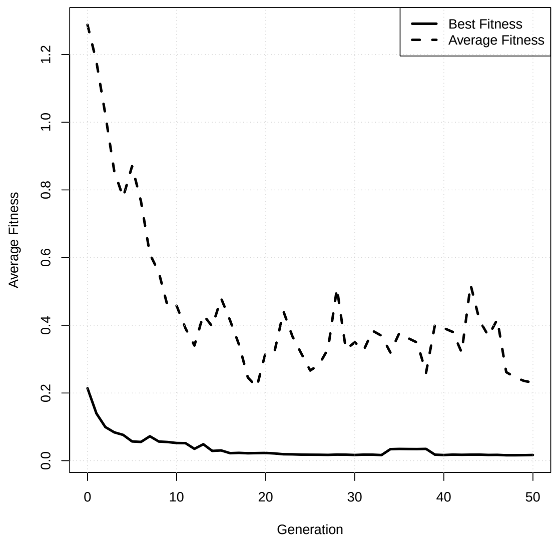
This appendix presents supplementary results to those presented in Chapter 8. Section C.1 provides convergence curves for 250 vertex experiments to give an overview of the evolutionary behavior while Section C.2 shows the unsimplified results from the GP system corresponding to the best evolved models for 250 vertex target networks.

C.1 Convergence Plots

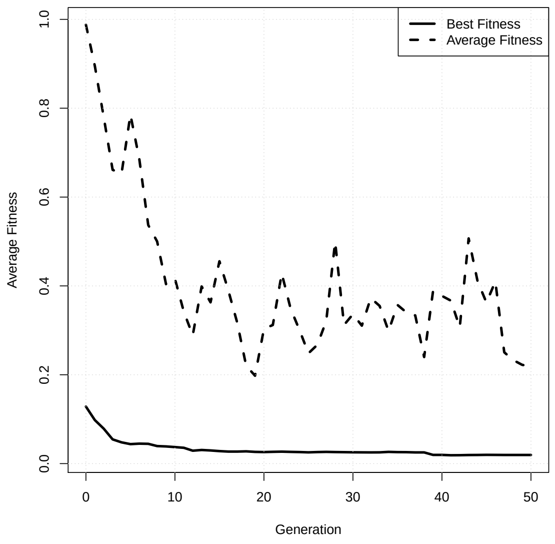
Figures C.1 through C.4 provide convergence plots for the 250 vertex experiments with the Growing Random (GR), Barabasi-Albert (BA), Erdos-Reyni (ER), and Forest Fire (FF) models, respectively. Each plot shows the average fitness of the best solution as well as the population average over 30 runs. Each fitness objective was a minimization objective, with 0 being the ideal fitness. In general, these plots showed that the best solutions converged after roughly 30 to 40 generations.

C.2 Raw Output from LinkableGP

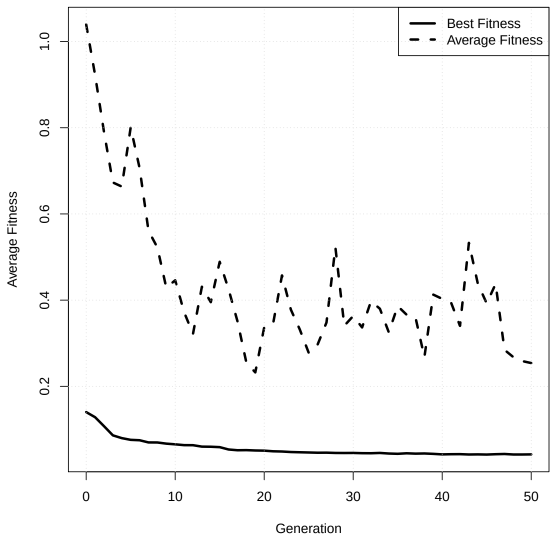
This section provides the raw output, with spacing added for readability, from LinkableGP for the best evolved models using 250 vertex target networks. Note that each of the GP results were a fully compilable (C#.NET v4.5) class, provided that the language elements are present, which implemented an abstract class. The abstract class, *InferredUndirectedGraphModel*, corresponded to the generalized graph model given in Algorithm 9.



(a) f_1: KS test statistic – Degree

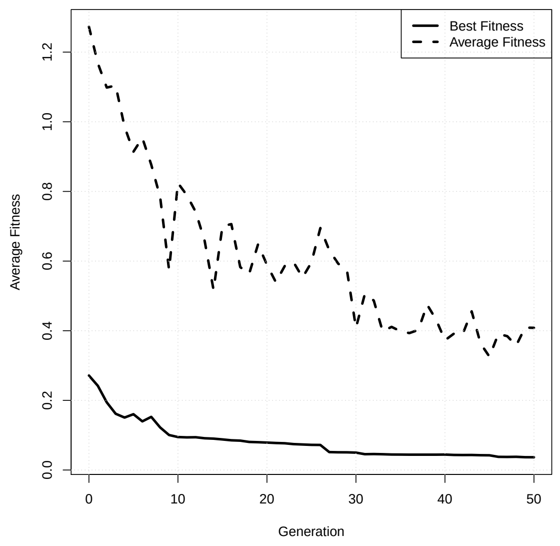


(b) f_2: KS test statistic – Betweenness

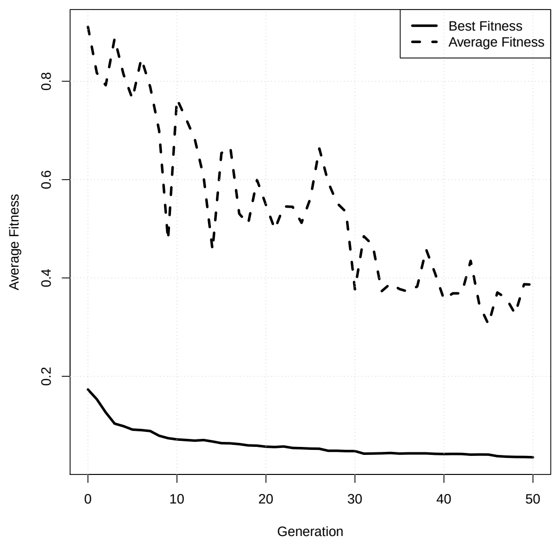


(c) f_3: KS test statistic – PageRank

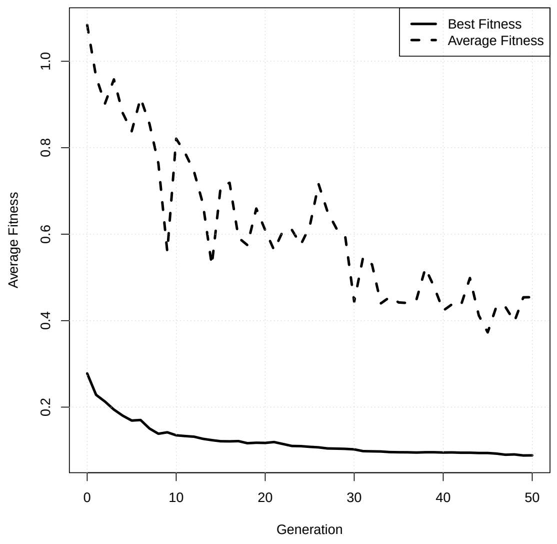
Figure C.1: Convergence plots showing the average of the best and average fitnesses over 30 runs using a 250 vertex GR target network.



(a) f_1: KS test statistic – Degree

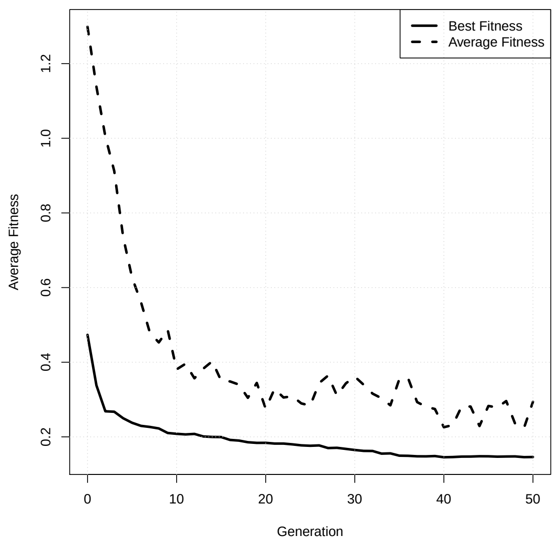


(b) f_2: KS test statistic – Betweenness

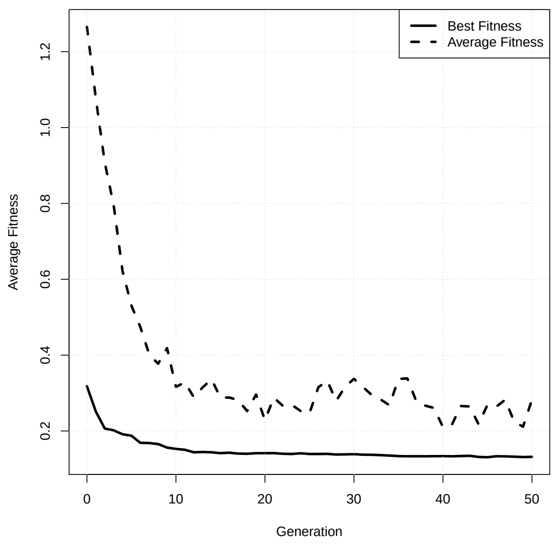


(c) f_3: KS test statistic – PageRank

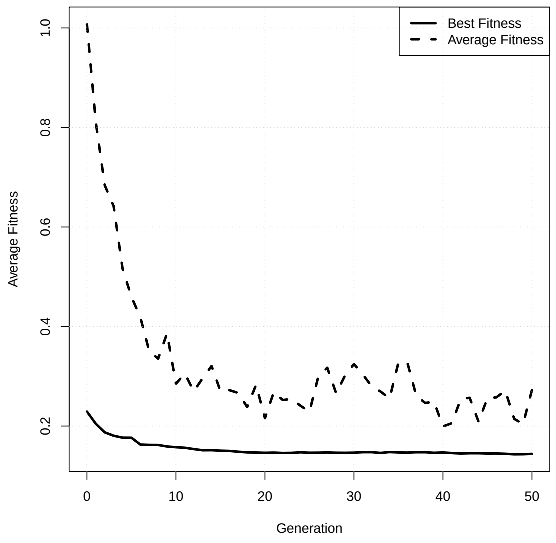
Figure C.2: Convergence plots showing the average of the best and average fitnesses over 30 runs using a 250 vertex BA target network.



(a) f_1: KS test statistic – Degree

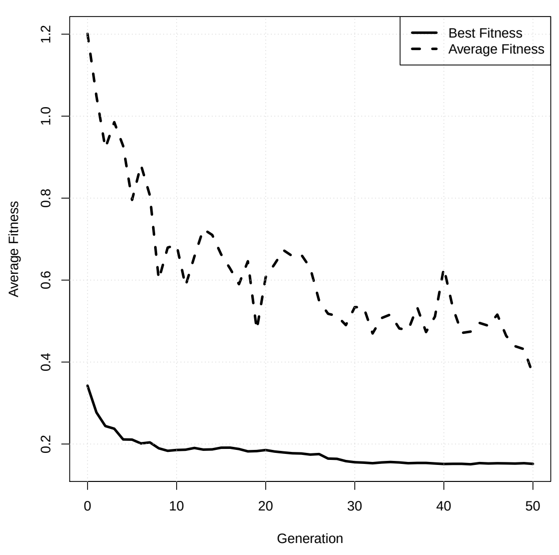


(b) f_2: KS test statistic – Betweenness

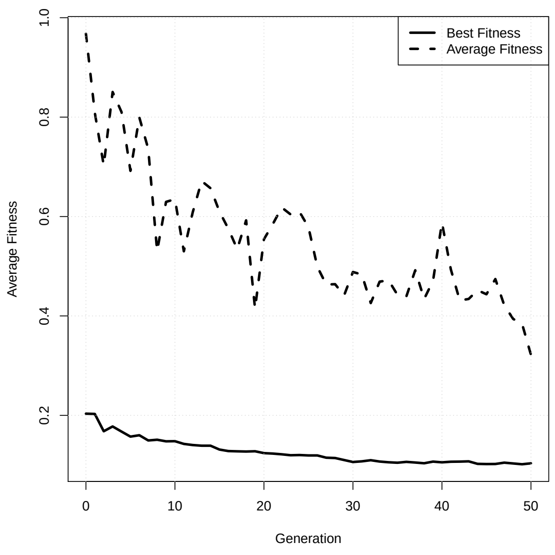


(c) f_3: KS test statistic – PageRank

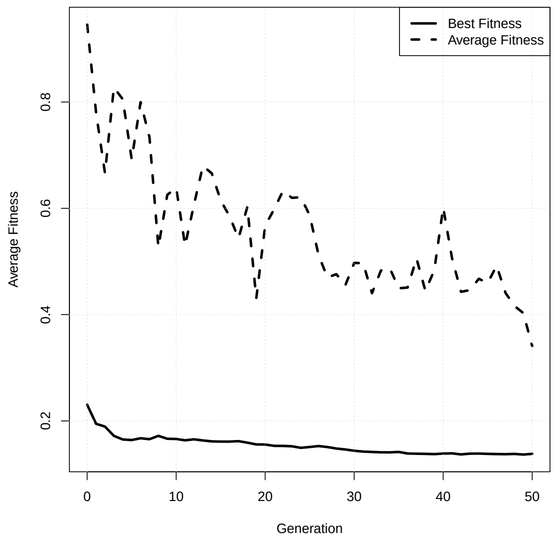
Figure C.3: Convergence plots showing the average of the best and average fitnesses over 30 runs using a 250 vertex ER target network.



(a) f_1: KS test statistic – Degree



(b) f_2: KS test statistic – Betweenness



(c) f_3: KS test statistic – PageRank

Figure C.4: Convergence plots showing the average of the best and average fitnesses over 30 runs using a 250 vertex FF target network.

C.2.1 Raw Output for GR_{250}

```
public class AutoGeneratedClass : InferredUndirectedGraphModel
{
    protected override TabooVertexCollection SelectVertices(IGraph g)
    {
        TabooVertexCollection a;
        TabooVertexCollectionProvider TabooVertexCollectionProvider
            = new TabooVertexCollectionProvider();
        a = TabooVertexCollectionProvider.GetRandomStack(g);
        return a;
    }

    protected override IEnumerable<IEdge> CreateEdges(IVertex newVertex,
                                                      IVertex v)
    {
        IEnumerable<IEdge> a;
        EdgeCreator EdgeCreator = new EdgeCreator();
        a = EdgeCreator.AddEdge(newVertex, v);
        return a;
    }

    protected override void SecondaryActions(IVertex vertex,
                                             TabooVertexCollection vertices)
    {
        int a;
        ValueProvider ValueProvider = new ValueProvider();
        a = ValueProvider.GetRandomValue(6, 7);
        a = ValueProvider.GetRandomValue(5, 2);
    }
}
```

C.2.2 Raw Output for BA_{250}

```
public class AutoGeneratedClass : ICoN.Modelling.InferredUndirectedGraphModel
{
    protected override TabooVertexCollection SelectVertices(IGraph g)
    {
        Func<IVertex, double> b;
        VertexPropertyProvider VertexPropertyProvider
            = new VertexPropertyProvider();
        TabooVertexCollection a;
        TabooVertexCollectionProvider TabooVertexCollectionProvider
            = new TabooVertexCollectionProvider();
        a = TabooVertexCollectionProvider.GetRandomQueue(g, 4);
        b = VertexPropertyProvider.GetDegree();
        a = TabooVertexCollectionProvider.GetRouletteQueue(g, 1, b);
        return a;
    }

    protected override IEnumerable<IEdge> CreateEdges(IVertex newVertex,
                                                      IVertex v)
    {
        IEnumerable<IEdge> a;
        EdgeCreator EdgeCreator = new EdgeCreator();
        a = EdgeCreator.AddEdge(v, newVertex);
        return a;
    }

    protected override void SecondaryActions(IVertex vertex,
                                             TabooVertexCollection vertices)
    {
        int b;
        int a;
        ValueProvider ValueProvider = new ValueProvider();
        a = ValueProvider.GetRandomValue(6, 6);
        b = ValueProvider.GetGeometricValue(0.794519253910761D);
    }
}
```

C.2.3 Raw Output for ER_{250}

```
public class AutoGeneratedClass : InferredUndirectedGraphModel
{
    protected override TabooVertexCollection SelectVertices(IGraph g)
    {
        TabooVertexCollection b;
        TabooVertexCollection a;
        TabooVertexCollectionProvider TabooVertexCollectionProvider
            = new TabooVertexCollectionProvider();
        a = TabooVertexCollectionProvider.GetRandomQueue(g);
        b = TabooVertexCollectionProvider.GetAllStack(g);
        a = TabooVertexCollectionProvider.GetAllQueue(g);
        return b;
    }

    protected override IEnumerable<IEdge> CreateEdges(IVertex newVertex,
                                                       IVertex v)
    {
        IEnumerable<IEdge> a;
        EdgeCreator EdgeCreator = new EdgeCreator();
        a = EdgeCreator.AddEdgeWithProbability(v, newVertex,
                                               0.051730603003748973D);
        return a;
    }

    protected override void SecondaryActions(IVertex vertex,
                                             TabooVertexCollection vertices)
    {
        SecondaryVertexProvider SecondaryVertexProvider
            = new SecondaryVertexProvider();
        int a;
        ValueProvider ValueProvider = new ValueProvider();
        a = ValueProvider.GetGeometricValue(0.2040185081791219D);
        SecondaryVertexProvider.AddNeighbours(vertices, a, vertex);
        SecondaryVertexProvider.AddNeighbours(vertices, 4, vertex);
    }
}
```

C.2.4 Raw Output for FF_{250}

```
public class AutoGeneratedClass : InferredUndirectedGraphModel
{
    protected override TabooVertexCollection SelectVertices(IGraph g)
    {
        TabooVertexCollection a;
        TabooVertexCollectionProvider TabooVertexCollectionProvider
            = new TabooVertexCollectionProvider();
        a = TabooVertexCollectionProvider.GetRandomStack(g, 3);
        return a;
    }

    protected override IEnumerable<IEdge> CreateEdges(IVertex newVertex,
                                                      IVertex v)
    {
        IEnumerable<IEdge> a;
        EdgeCreator EdgeCreator = new EdgeCreator();
        a = EdgeCreator.Duplicate(v, newVertex, 0.41683564494216613D);
        return a;
    }

    protected override void SecondaryActions(IVertex vertex,
                                             TabooVertexCollection vertices)
    {
        int d;
        int c;
        int b;
        int a;
        ValueProvider ValueProvider = new ValueProvider();
        a = ValueProvider.GetGeometricValue(0.16351415317669238D);
        a = ValueProvider.GetRandomValue(a, a);
        b = ValueProvider.GetGeometricValue(0.32204452078884677D);
        c = ValueProvider.GetGeometricValue(0.10324204019421807D);
        c = ValueProvider.GetRandomValue(b, a);
        d = ValueProvider.GetGeometricValue(0.097357993059492665D);
    }
}
```