Object-Oriented Genetic Programming for the Automatic Inference of Graph Models for Complex Networks

Michael Richard Medland

Department of Computer Science

Submitted in partial fulfillment
of the requirements for the degree of

Master of Science

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

To my children,
You have are the reason for my perseverance,

And

To my extraordinary wife,
You have been my rock, my sounding board, and saving grace.

Abstract

Complex networks are systems of entities that are interconnected through meaningful relationships. The result of the relations between entities forms a structure that has a statistical complexity that is not formed by random chance. In the study of complex networks, many graph models have been proposed to model the behaviours observed. However, constructing graph models manually is tedious and problematic. Many of the models proposed in the literature have been cited as having inaccuracies with respect to the complex networks they represent. However, recently, an approach that automates the inference of graph models was proposed by Bailey [10] The proposed methodology employs genetic programming (GP) to produce graph models that approximate various properties of an exemplary graph of a targeted complex network. However, there is a great deal already known about complex networks, in general, and often specific knowledge is held about the network being modelled. The knowledge, albeit incomplete, is important in constructing a graph model. However it is difficult to incorporate such knowledge using existing GP techniques. Thus, this thesis proposes a novel GP system which can incorporate incomplete expert knowledge that assists in the evolution of a graph model. Inspired by existing graph models, an abstract graph model was developed to serve as an embryo for inferring graph models of some complex networks. The GP system and abstract model were used to reproduce well-known graph models. The results indicated that the system was able to evolve models that produced networks that had structural similarities to the networks generated by the respective target models.

Acknowledgements

Thank you to my supervisor Dr. Beatrice Ombuki-Berman, you have been there for me for a number of years now and have been the compass of my work. Thank you to my peers, the faculty of the department of Computer Science, and other mentors. You have been a wealth of knowledge from which to draw inspiration and have been pivotal in my successes. I especially would like to acknowledge the efforts of Dr. Mario Ventresca, Alex Bailey, and Kyle Harrison. Thank you, finally, to my supervisory committee for your efforts in the process of constructing this thesis, it is undoubtedly better as a result.

M.R.M

Contents

List of Tables

List of Figures

Chapter 1

Introduction

The main contribution of this work is the introduction of a novel genetic programming (GP) technique which is useful for the automation of the inference of complex networks. GP is an evolutionary computational technique based on the ideas of Darwinian evolution. It begins with a population of randomly created programs and utilizes recombination and mutation operators to evolve fit programs. It has been shown to be useful in a number of applications such as regression [9, 74, 87], the development of circuits [45], evolutionary art [16], and the construction of artificial agents [41], and even the automatic inference of complex networks [11].

A complex network is a system of entities that are interrelated via connections in meaningful ways [69]. The study of these systems reveals a great deal about the systems, both natural and artificial, which they represent. For example, Facebook is a kind of a complex network where the members are entities, and the connections are friendships [86]. Food webs are examples of natural complex networks wherein the animals, both prey and predator, are the entities and the transfer of energy from prey to predator are the connections [69]. Other complex networks might include the Internet [70], protein interactions [69], citation networks [5, 13, 29], neural pathways of the brain [12], and even sexual relations between high school students [15].

Understanding complex networks requires that we understand the structure and dynamics of the networks being investigated. It is only then that the behaviour of the entities and the processes acting on them can be understood properly. One way in which complex networks are studied is via graph models [69]. Graph models are algorithms which, put simply, produce graphs. In the case of complex networks, they model the behaviours of the network to approximate the structure and dynamics of a complex network.

There are a number of examples of graph models produced for complex networks

found in literature [68]. There exist graph models which attempt to identify prevalent structural properties found in many complex networks such as power-law degree distributions [13], small world effect [90], and community structure emergence [54]. These networks properties have been demonstrated in a great number of real-world examples such as biological networks [84, 88], social networks [86], and technological networks [37]. However, constructing a graph model requires large amounts of data, in-depth study into the mechanisms of the network, and a great deal of time. Furthermore, constructed models are often cited for inaccuracies [13, 27, 54, 90] and in order to construct them one must have a deep understanding of the processes and behaviours of the network.

In 2012, Bailey *et al.* [11] proposed a method for the automatic inference of graph models for complex networks. The method was an important step in the construction of graph models for complex networks as it helped address the difficulties in the development of graph models for complex networks. In other words, there is a circular nature in constructing graph models whereby it is necessary to understand the complex network at hand to develop a graph model for understanding the network [10].

In Bailey's work [10], he also identified that some researchers reuse existing graph models to produce new models from other data that can lead to inaccuracies of the models. Nevertheless, this suggests that researchers have some understanding of how the graph model of the network at hand should behave. As such, work has gone into identifying classes of complex networks which helps in the manual construction of models (for example, see [8, 46, 47, 52]). The methods proposed by Bailey *et al.* [12] employed additional algorithms, outside of GP, such as community detection algorithms to assist in producing community structures. These observations suggest that there is some merit in utilizing the knowledge of the researcher to assist the GP system in producing models.

This work is motivated primarily by this final point. Researchers utilizing a tool for the automation of the inference of graph models do have valuable knowledge to contribute about the complex network at hand. As such, this thesis is aims to produce a methodology for evolving graph models using a GP system which

- Allows the incorporation of expert knowledge
- Provides a means of expressing incomplete knowledge that does not hinder the GP performance

Expert knowledge, in the context of modelling complex networks, is any knowledge

that is certain about the processes, structure, and properties of the complex network at hand or in general. As will be shown in Chapter 4, there is much that can be known about the structure of a complex networks in general. Moreover, sometimes there are insights based on the context of the network at hand that introduce new properties of the network. For example, Dorogovtsev and Mendes [27] identifies age of research papers as an important property in citation networks. No matter the case, expert knowledge is incomplete about a network being modelled. Thus, a framework should allow incomplete knowledge to be placed as easily as creating a filling the blanks puzzle so that a GP system can fill in the blanks without making creating huge jumps in its problem space.

1.1 Main Goal

This thesis, inspired by the previous work in the automatic inference of graph models, contends that a new methodology which affords the opportunity of the researcher to provide expert knowledge about the network being modelled is warranted. Compared to manual efforts in graph model construction, this reduces the time and effort required to construct models, and also draws on the knowledge already known about the network at hand. Incorporating knowledge into automation can assist in producing not just reasonable models that replication the structure and dynamics of the network, but also models that have some essence of semantic encoded by the research.

As a result of this goal, it is necessary to redesign traditional approaches to GP in order to allow the incorporation of such a system. This thesis proposes a novel GP system using the object-oriented programming paradigm to facilitate the incorporation of a partial program. Using an Object-Oriented GP (OOGP) system which involves partial implementation of a program facilitates the incorporation of expert knowledge. Such a facility has benefits that extend beyond this work, provides further motivation for its development (see [60] for more details).

1.2 Challenges and Contributions

In designing a graph model, there is a great deal of knowledge applied both specifically about the complex network being modelled and complex networks in general. Incorporating this knowledge into a GP system is a non-trivial task, especially as much is still unknown about complex networks. Each network type has its a distinctiveness and caveats that researchers who study them are the best aware. Therefore,

this thesis only begins to identify the utility of the proposed approach.

While this thesis does benefit from previous work in automatic inference of graph models (see [10, 11, 12]), here directed complex networks are explored and therefore a new set of challenges encountered. Directed complex networks provide additional information through the directionality of relations. They provide a sense of flow in time or in information that is not captured completely by undirected networks. As a result of the directionality and the information it conveys, this provides a unique set of challenges. In order to produce a methodology that is capable of handling the additional challenges of directed complex networks modifications to the previous approaches.

1.2.1 Abstracting a Graph Model

The OOGP methodology, proposed in this thesis, requires the implementation of an abstract class as a template for programs produced. In the case of graph models, a model is required which encapsulates general behaviour of a complex network. If a model existed that described the general behaviour of all complex networks, then OOGP would not be beneficial as this would simply be a question of parameter tuning. Instead, some reliance on the network's known behaviours must be encoded into the algorithm. This thesis proposes the first such abstraction for OOGP.

1.2.2 Function Sets

A function set provides the building blocks for the development of a program in GP. In abstracting the graph model for OOGP, the function set proposed in Bailey [10] is no longer usable because an abstract class (OOGP's program) consist of several functions each with different purposes. Therefore, each function within the abstract class requires a function set. Each function set relates only to relevant tasks of the function it is mapped to. The complete set of functions utilized to construct the program needs be sufficient to express a large variety of models, but not so much that the OOGP becomes bogged-down finding useful configurations. This thesis proposes a collection of functions sets specific to the abstract class proposed.

1.2.3 Contributions

In summary, the contributions made by this thesis are

1. Proposes a new methodology for the automatic inference of graph models for complex networks that facilitates the incorporation of known behaviours.

2. Proposes a novel OOGP methodology which facilitates the incorporation of expert knowledge.

3. Proposes an abstraction of graph models for the automatic inference of complex networks.

4. Proposes an OOGP function set for the construction of graph models.

1.3 Thesis Structure

The remainder of this thesis is organized as follows. Chapter 2 provides a brief introduction to complex networks in order to provide the reader with the requisite knowledge for this thesis. In Chapter 3, graph models will be discussed, and an introduction to some well known graph models of complex networks is provided. Then in Chapter 4, a review of some of the notable techniques model construction is provided. Chapter 5 provides a background in GP sufficient to understand the proposed GP system. Chapter 6 presents the proposed OOGP system in detail including the structure and representation of the population of candidate programs. As well, the chapter will also provide details about the implementation of genetic operators for the OOGP. Chapter 7 will describe the exact methods used by the proposed methodology. Chapter 8 will show how the system was used to reproduce some known graph models. Finally, Chapter 10 will provide some conclusions and other inspiration for future works.

Chapter 2

Complex Networks

A complex network is a system of entities joined together through relations or connections. Governed by underlying processes, complex networks form a structure that is neither regular nor completely random. There is no definition of complex networks universally accepted. However, one definition that might be adopted as characteristic of complex networks is that they have a statistical complexity [48]. That is that there are patterns within the structure that have observable properties that can be used to construct probabilistic descriptions of the structure or behaviour of the network.

Complex networks are found everywhere. They are widely studied in many fields such as social sciences [15, 22, 52], biology [57, 75, 84], computer science [76, 77], and others [69]. In biology, complex network are often investigated as molecular systems where protein structure or gene relationships are investigated. In social sciences, complex networks, often called social network, describe human interactions, like friendships in Facebook.

The study of complex networks reveals a great deal about the world [69]. Study of individual classes of network often leads to discoveries outside of the domain of the network and into general theories of complex networks [5, 68]. Thus, a common form of representation is useful. A popular representation of complex networks is a graph [69]. This representation utilizes vertices connect via edges to represent the entities (vertices) and relations (edges) in complex networks. This representation provides a means of mathematically analysing the structure of complex networks [30], and thus facilitates quantitative study of complex networks.

Since complex networks are dynamic systems [70], it is useful to be able to study their behaviour. A means of representing the dynamic behaviours of complex networks is through the graph model. In terms of modelling complex networks, graph models provide an algorithmic means of representing the processes that construct complex

networks. As a result, graph models can be employed to investigate properties of various complex networks [69]. Many graph models have been proposed for the study of complex networks [27, 54, 90].

In this chapter, an introduction to some types of complex networks is provided in Sections 2.1, 2.2, and 2.3. Then, in Section 2.4, a brief introduction to representing complex networks will be provided. This section will highlight two representations of complex networks, namely graphs and graph models. With an understanding of these representations, an introduction to measuring the structural properties of complex networks will be provided in Section 2.5. For those readers interested, a much more comprehensive introduction to complex networks can be found in [69].

2.1 Social Networks

Social Networks are networks that are the product of social interactions of people. The study of social networks can be dated back to the 1920s with works by Almack [7] and Wellman [91]. However, Moreno and Jennings [66] are often credited as pioneers in the study of social networks. Moreno and Jennings's work with sociograms provided a framework for mapping group interactions. Sociograms are often used to limit misbehaviour in the classroom [93].

Milgram [63] is another important name in the study of social networks. His "small world" experiment [85] explored the notion of "six degrees of separation" that states that any given person is separated by no more than five other people for any given person. The property was demonstrated using a letter passing exercise. The exercise had several individuals from Nebraska passing a letter to other people they knew on a first name basis in another state. The recipients then passed the letter on to people in yet another state. The data collected from the experiment formed acquaintance chains showed an average number of intermediate steps of 5.2, and a great deal of the chains overlapped by the same three people.

Social networks often investigate the relationship of friendships [8, 22, 63] but are certainly not limited to those relations. Other examples of social networks include citation networks [25, 90], sexual relations [15], race relations [81], and on-line social networks [52], i.e. Facebook , Google+, Twitter.

2.2 Biological Networks

Biological networks are networks that are biological systems. This includes systems such as protein-protein interactions [57], neural networks [84], and food webs [56]. With increased access to biological information, the study of biological systems as complex networks is becoming more prevalent [75].

The first molecular networks were characterized by Dagley *et al.* [24] and with newer research tools, researchers can characterize protein-protein interactions and gene regulatory systems with increasing accuracy [75]. These networks are observed to have many of the same structural properties also shared outside of biological networks, such as the power-law degree distribution [88].

The study of molecular networks reveals information about the structure of the networks. The study provides insight into understanding biological processes [75]. Proulx *et al.* [75] also remarks that while empirical studies of biological networks provides insight into the theory, application of theory must provide a predictive framework for testing the hypotheses. Such a practice is an important insight about biological networks that extends into complex networks in general. It provides a strong motivation for the efforts of this thesis and its related works [10, 11, 12].

2.3 Information Networks

Information networks are complex systems of data linked together in some fashion. It is thought that all information networks are man-made [69]. Furthermore, networks of information often have social component [69]. Consider, for example, the social media site LinkedIn, a social network where members share affiliations together. However, there is also a flow of information about people observed through the interactions of endorsement of skills between members.

Arguably, one of the most well-known examples of an information network is the World Wide Web (WWW). In this network of information, web pages are the linked together through hyper-links. While the WWW is a man-made collection of web pages, its growth is largely unregulated and leads to an enormous directed graph which makes its study challenging [6]. Nevertheless, the WWW has transformed the way knowledge is transmitted and has an underlying structure [37].

There are a great deal of other types of information networks which have drawn the attention of researchers such as peer-to-peer networks [77], recommender networks [40], keyword indexes [76], and citation networks [5, 13, 25, 38]. Of these, the citation

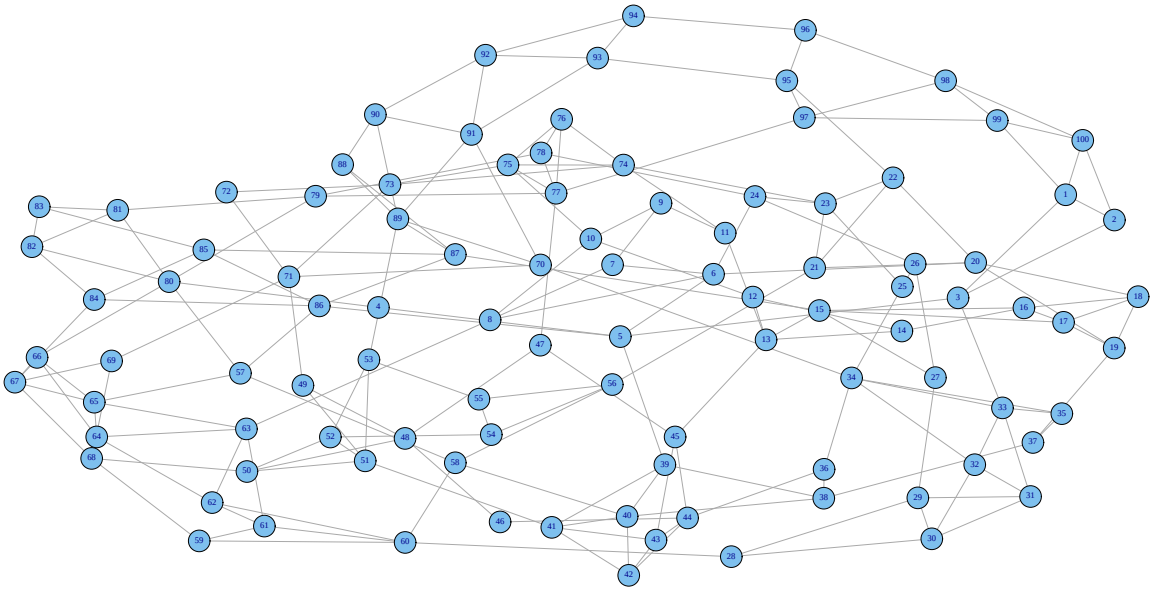


Figure 2.1: Small-World Complex Network

network has been studied the longest [69]. Citation networks have been well studied and have informed theories of complex networks (not just information networks) on structural dynamics of complex networks. Examples include the power-law distribution of degrees [13], community structure emergence [34], and the dynamics of ageing cites [27].

2.4 Representing Complex Networks

Having a good way to represent complex networks makes studying them much easier. One method of representation is a graph. A graph consists of points and links, referred to, respectively, as vertices and edges. Mathematically, a graph, denoted $G = (V, E)$, is a set of vertices or nodes, V, and a set of edges, E. Figure 2.1 provides an example of a complex network in graph form. This graph represents a small-world network. An interpretation of the graph, used by Watts and Strogatz [90], is that the vertices are actors and the edges represent two actors having worked together on a movie [90]. Investigation of graphs of small-world networks shows that the average shortest distance to any two actors is proportional to $\log N$, where N is the number of vertices in the network [90].

Investigating complex network through graphs is somewhat limiting. Without a great deal of observable data, the statistical complexity of a network's structure and behaviour might not be revealed. Instead, attention can be turned to graph models.

Graph models are algorithms that produce graphs. Graph models might produce regular graphs such as the K-regular graph, or they might produce random graphs, for example, the Erdos-Renyi random graph [28].

With a graph model, the processes that act on the network can be approximated and used to gather a large amount of data from the graphs it produces. For example, the network described in Figure 2.1 is produced by Algorithm 1, the Watts-Strogatz Small World model. This algorithm produces a graph, $G = (V, E)$, which an average path length proportional to $\log N$. In producing path lengths of approximately $\log N$, the Watts-Strogatz Small-World model approximates the observed properties of the small-world network.

Algorithm 1: Watts-Strogatz Small World Model

Input: $n > 0$, $k > 0$, $d > 0$, $1 \geq p \geq 0$
Data: Watts-Strogatz Graph, $G = (V, E)$
begin
  $G \leftarrow lattice(n, k, d)$;
  $E_2 \leftarrow \emptyset$;
  foreach $(i, j) \in E$ **do**
    if $random() < p$ **then**
      $E_2 \leftarrow E_2 \cup (i, random(|V|))$;
    else
      $E_2 \leftarrow E_2 \cup (i, j)$;
    end if
  end foreach
  $E \leftarrow E_2$;
  return G
end

2.5 Measuring Complex Networks

The title of this section, *Measuring Complex Networks*, is something of a misnomer. The reality is that complex networks cannot be measured directly. Instead, the structure of the network is analysed. Structural analysis is accomplished using statistical measures that quantify various aspects of the network. There exist a great number of measures for the structure of complex networks, many of which are identified by Newman [69]. However, for the purposes of this thesis, a few statistical measures are selected which identify key characteristics of the structure of networks. These are statistical measures commonly used in the study of complex networks found in

literature [5, 8, 13, 14, 17, 25, 27, 63, 90].

Starting from the methods proposed by Bailey [10], immediately three common structural properties measured in complex networks are identified.

- Geodesic Path Length
- Transitivity
- Degree Distribution

These three measures formed the basis of fitness evaluation of the generated models in the work done on automating the inference of graph models [10]. In the work of [10] these measures were used for fitness evaluation to construct graph models with a high degree of accuracy. Thus, this thesis continue this approach and leave evaluating the sufficiency of these measures to others [36]. Additionally, the aim was to propose a methodology for the inference of graph models for *directed and unweighted* complex networks where Bailey [10] is concerned with *undirected and unweighted* complex networks. Directed networks add new information not found in undirected networks as a result of the directionality of edges. In order to capture this new information, the procedures used in measuring the above structural properties. For computing the geodesic path lengths and transitivity of the network, this means that the computational method should take into account the directionality of the edges. For the degree distribution, two measures are now required: in- and out-degree.

2.5.1 Average Geodesic Path Length

The average geodesic path length refers to the average length of the shortest path, $d_{i,j}$, in a graph, $G = (V, E)$, between all vertices $i, j \in V$ [26]. Let l be the average of all geodesic path lengths of a directed graph such that (i, j) are vertex pairs in G and n is the number of vertices in the graph:

$$l = \frac{1}{n(n+1)} \sum_{i,j \in V | i \neq j} d_{i,j} \tag{2.1}$$

In social networks, persons with a lower mean distance between others might reveal a high influence of their opinions [69]. This measure is often found in studies related to types of social networks to illustrate the small path lengths of these networks (see [13, 54, 90]).

2.5.2 Transitivity

In mathematics, given some relation $\circ$, transitivity is the implication, $a \circ b \wedge b \circ c \rightarrow a \circ c$. In terms of networks transitivity means that if there exists an edge, (a, b), and an edge, (b, c) that implies that there is an edge (a, c). Transitivity is an important property of social networks [69].

A graph that is transitive is said to have perfect transitivity [69]. Perfect transitivity, however, is not a useful metric as only fully connected graphs have such a perfect transitivity. Instead, it is interesting to determine the amount of transitivity which occurs in a network, i.e. partial transitivity. One method for computing the partial transitivity of a network is to compute its clustering coefficient. The clustering coefficient refers to the probability that a vertex, and its neighbour are connected to a common vertex.

There exist a number of measures of the clustering coefficient such as the global clustering coefficient, the local clustering coefficient, and the network average clustering coefficient. This thesis uses the network average clustering coefficient when considering clustering coefficients as it provides a single value and utilizes to the local clustering coefficient. The network average clustering coefficient is the average of the local clustering coefficient over all n vertices in the network (see Equation 2.2). To compute the local clustering coefficient for each vertex, i, Equation 2.3 is employed, where N_i is the set of vertices immediately connected vertex v_i and $k_i = |N_i|$.

$$\bar{C} = \frac{\sum_{i=1}^{n} C_i}{n} \tag{2.2}$$

$$C_i = \frac{|\{e_{jk} : v_j, v_k \in N_i, e_{jk} \in E\}|}{k_i(k_i - 1)} \tag{2.3}$$

This measure can be used to detect the presence of a small-world network [90]. If $\bar{C}$ is higher in a network than the observed $\bar{C}$ of a random network with a similar average geodesic path length than this indicates that the network exhibits characteristics of a small-world network.

2.5.3 Degree Distribution

Degree centrality is a prevalent in research into complex networks [69]. Degree centrality addresses the importance of a vertex within its network. There exist a number of measures that offer indications of the importance of a vertex in the network. The simplest of these is the degree of a vertex, that is, the number of edges connected

to it. In a directed graph, there is also the in-degree and out-degree of a vertex. This is the number of in-bound edges (in-degree) and the number of out-bound edges (out-degree) of a vertex.

The distribution of the degrees of vertices in a complex network can reveal important information about the behaviour of a complex network. In preferential-attachment networks with a growing number of vertices (i.e., new vertices are added to the network over time), like the Barabasi-Albert model [13], the distribution of the degrees follows a power-law distribution. The power-law degree distribution referred to as a scale-free distribution, describes a distribution where the number of vertices with small degrees is large, and the distribution is heavily-tailed with only a few vertices having a high degree. More specifically, the probability, $P(X)$ that a vertex will have a degree, $X = d$, where α and k are some constants, is

$$P(X) = \alpha X^k \tag{2.4}$$

In order to compare the degree distributions of two graphs, a statistical method is required. In the case of this work, the Kolmogorov-Smirnoff test (KS test) [42] is used. In the two sample form, the KS test is a non-parametric hypothesis test such that the null hypothesis is that two samples originate from the same distribution function. The null hypothesis of a two-sample test is rejected at level α if Equation 2.5 is satisfied where $D_{n,n'}$ is computed as in Equation 2.6 where is $c(\alpha)$ is a value determined by the significance level of the test and the sample sizes of the two samples are n and n'. In Equation 2.6, $F_{1,n}$ and $F_{2,n'}$ are the empirical cumulative distributions of sample 1 and 2, respectively and x is common observation point, an arbitrary point in the range of both functions in the distributions.

$$D_{n,n'} > c(\alpha)\sqrt{\frac{n+n'}{nn'}} \tag{2.5}$$

$$D_{n,n'} = \max_{x \in [0,\min(n,n')]} |F_{1,n}(x) - F_{2,n'}(x)| \tag{2.6}$$

For computing the empirical cumulative distribution of the degrees in a network, $F_n(x)$ is the proportion of vertices with degree d_i that satisfy $d_i \leq x$ (see Equation 2.7).

$$F_n(x) = \frac{1}{n}\sum_{i=1}^{n} d_i \leq x \tag{2.7}$$

Chapter 3

Graph Models

As briefly introduced in Section 2.4 a graph model is an algorithm that produces a graph that holds certain properties of interest. Graph models help to understand complex network behaviours as they model the processes that form the network. Through repeated execution, graph models produce a set of graphs that have some form of commonality. The commonality of the graphs is dependent on the design of the graph model. For example, a graph designed to replicate transitivity will produce graphs that have similar clustering coefficients. However, the clustering coefficients of two graphs produced by such a model will differ somewhat. In other words, the graphs will not be isomorphisms, but rather will exhibit certain predictable properties.

A number of graph models have been proposed which are designed to model various phenomena Barabási and Albert [13], Dorogovtsev and Mendes [27], Leskovec *et al.* [54], Watts and Strogatz [90]. Furthermore, the phenomena are not attributable to random chance and the properties observed, with respect to a phenomenon, are not found in random graphs. In this chapter, the random graph model will be introduced. This model serves as a starting point for the study of graph models for complex networks and is important to the comparison of other graph models. Also in this chapter, a variety of graph models are introduced which model some structural properties found in real-world networks. Further information about graph models can be found in Amaral *et al.* [8], Krapivsky *et al.* [47], Newman [68, 69], Newman *et al.* [70], Watts and Strogatz [90].

3.1 Erdos-Renyi Model

The Erdos-Renyi model is a random graph model that can be represented two ways [33, 82]: $G(n, p)$ and $G(n, m)$ (illustrated later). The Erdos-Renyi model, specifically

the one introduced by Solomonoff and Rapoport [82], is a well studied model (See, for example, Boccaletti *et al.* [17], Diestel [26], Newman [68, 69], Newman *et al.* [70]). Popularized by Erdos and Renyi [28], the network is a random graph model containing n vertices are connected with an independent probability of p.

Algorithm 2: Erdos-Renyi $G(n, p)$ Graph Model

```
Input: n > 0 0 ≤ p ≤ 1
Data: Erdos-Renyi Graph, G = (V, E)
begin
    for i ← 1 to n do
        V ← V ∪ v_i;
    end for
    for i ← 1 to n do
        for j ← 1 to n do
            if random() < p then
                E ← E ∪ (v_i, v_j);
            end if
        end for
    end for
    return G
end
```

The model introduce by Solomonoff and Rapoport [82], denoted $G(n, p)$, is defined in Algorithm 2. This is the original random graph model popularized by Erdos-Renyi [28]. In this model, n vertices are added to a graph. Afterwards, for all potential edges (i, j), the edge is added with a probability of p. The graphs produced by this model follow a binomial distribution of number of edges in the graph [29], where it is expected $|E| = p\binom{n}{2}$ in a produced graph, $G = (V, E)$. An example graph produced by the $G(n.p)$ model is provided in Figure 3.1.

An alternative representation of the Erdos-Renyi model provided by Gilbert [33] differs from $G(n, p)$ model as the edges are added randomly using an uniform distribution. Furthermore, this representation only M edges are added to the graph. The work of Gilbert [33] and application of the alternative representation, denoted $G(n, M)$, explores the probability that a path exists between two edges. In the $G(n, M)$ model, given probability, p, that an edge between two vertices exists, the probability that two vertices are connected is $P_n \sim 1-n(p-1)^{n-1}$ and the probability that a path exists to all other vertices from a vertex is $R_n \sim 1-2(p-1)^{n-1}$. Notably, the $G(n, M)$ model tends to behave similarly to the $G(n, p)$ when $M = p\binom{n}{2}$.

In Algorithm 3, the $G(n, M)$ model begins by adding n vertices to the graph.

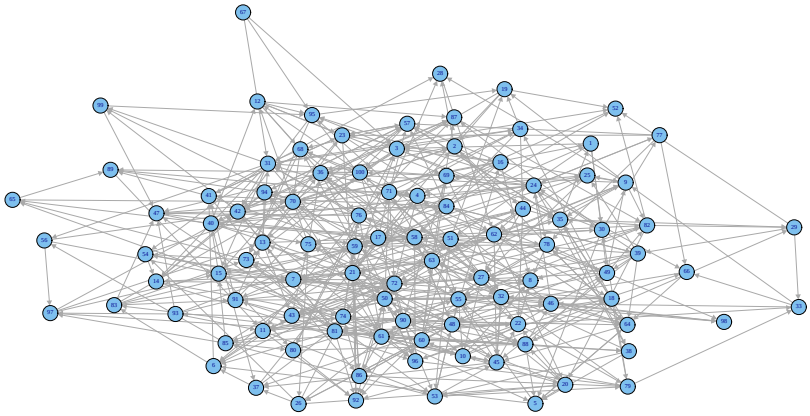


Figure 3.1: Erdos-Renyi $G(n, p)$ Graph

Constructed where $n = 100$, $p = 0.05$, and self-edges where restricted

Then, m edges are constructed using two vertices randomly selected from the graph. An example of the output from the $G(n, M)$ model is provided in Figure 3.2.

Algorithm 3: Erdos-Renyi $G(n, M)$ Graph Model

Input: $n > 0$, $M > 0$
Data: Erdos-Renyi Graph, $G = (V, E)$
begin
  for $i \leftarrow 1$ **to** n **do**
    $V \leftarrow V \cup v_i$;
  end for
  for $i \leftarrow 1$ **to** M **do**
    $v_1 \leftarrow select(V)$;
    $v_2 \leftarrow select(V)$;
    $E \leftarrow E \cup (v_1, v_2)$;
  end for
  return G
end

3.2 Watts-Strogatz Small World Model

The development of the Small World model proposed by Watts and Strogatz [90], attempted to address some short-comings of the Erdos-Renyi model in capturing real-world network properties. The model illustrates a "small world" effect [63] such that path-lengths to other vertices in the graph are small. [90] proposed that naturally occurring networks were neither completely random nor completely regular.

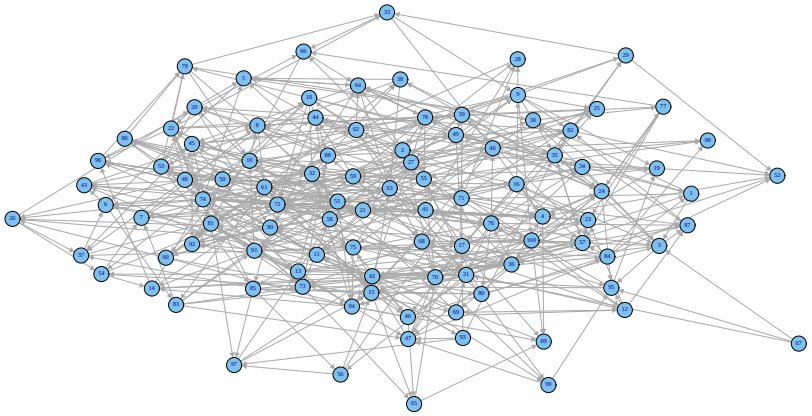


Figure 3.2: Erdos-Renyi $G(n, M)$ Graph

Constructed where $n = 100$, $m = 300$, and self-edges where restricted

Networks are considered small world when the average local clustering efficient is significantly higher than that of a random graph with the same number of vertices and edges [90]. Many networks have been shown to exhibit the small-world property Watts and Strogatz [90] such as power grids, collaborative networks, and C. elegans worms. However, the model proposed by Watts and Strogatz does not generate the same degree distributions as those of real-world networks Newman [69].

Algorithm 4: Watts-Strogatz Small World Model

Input: $n > 0$, $k > 0$, $d > 0$, $0 \leq p \leq 1$
Data: Watts-Strogatz Graph, $G = (V, E)$
begin
  $G \leftarrow lattice(n, k, d)$;
  $E_2 \leftarrow \emptyset$;
  foreach $(i, j) \in E$ **do**
    if $random() < p$ **then**
      $E_2 \leftarrow E_2 \cup (i, random(|V|))$;
    else
      $E_2 \leftarrow E_2 \cup (i, j)$;
    end if
  end foreach
  $E \leftarrow E_2$;
  return G
end

The Watts-Strogatz model recreates the local clustering and hub formation found in real-world networks [8]. As described in Algorithm 4, the network begins as lattice. The lattice has k dimensions, each with n vertices, and each vertex has d neighbours

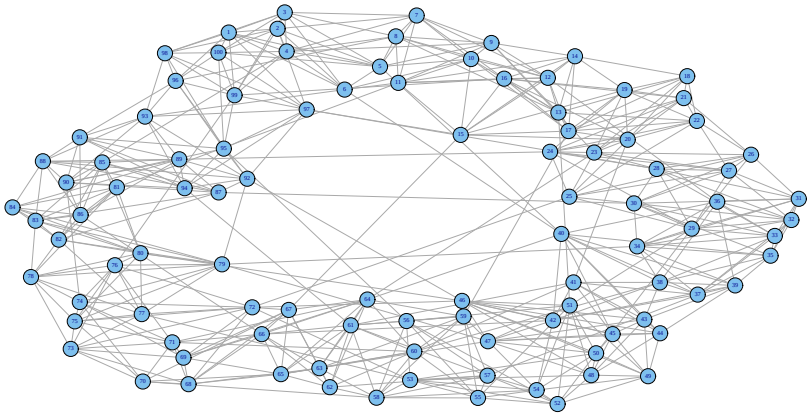


Figure 3.3: Watts-Strogatz Small World Graph

Generated where $n = 100$, $k = 5$, $d = 1$, and $p = 0.05$

to either side. The algorithm then proceeds to rewire each edge with probability p. Figure 3.3 provides an example of an output graph from the Watts-Strogatz model.

3.3 Barabasi-Albert Model

The Barabasi-Albert (BA) model [13] is a well known model that demonstrates preferential attachment whereby new entities are more likely to attach to well established entities within the network [13]. Furthermore, it exhibits scale-free growth, that is, a network with $t + m$ vertices will have mt edges [68]. In this class of networks, it is observed that the probability a vertex will have degree k is $P_k \approx ak^{\alpha} + c$, where α, a, c are some constants [5].

[13] found that real-world networks with scale-free distributions required both growth and preferential attachment. Their experiments with the BA model and a random network using only the growing degrees showed that the degree distributions where divergent.

Algorithm 5 illustrates the BA model. As new vertices are added to the graph, existing vertices are attached to them. The probability of attachment is determined by $k_j^{\alpha} + a$ as found in Algorithm 5. Vertices with more existing connections have a higher probability of attaching to the new vertices.

Figure 3.4 provides an output graph of the BA model. In the graph, it can be observed that there exist hubs, central vertices with a higher in-degree than other vertices in the graph. These hubs are the characteristic of the effect from preferential-attachment.

Algorithm 5: Barabasi-Albert Graph Model

Input: $n > 0$, $m > 0$, $\alpha > 0$, a, c
Output: A completed Barabasi-Albert Network, $G = (V, E)$
begin
  for $i \leftarrow 1$ **to** n **do**
    $V \leftarrow V \cup v_i$;
    for $j \leftarrow 1$ **to** $i - 1$ **do**
      $P_j \leftarrow ak_j^{-\alpha} + c$;
    end for
    for $j \leftarrow 1$ **to** m **do**
      $s \leftarrow select(P)$;
      $E \leftarrow E \cup (v_i, v_s)$;
    end for
  end for
  return G
end

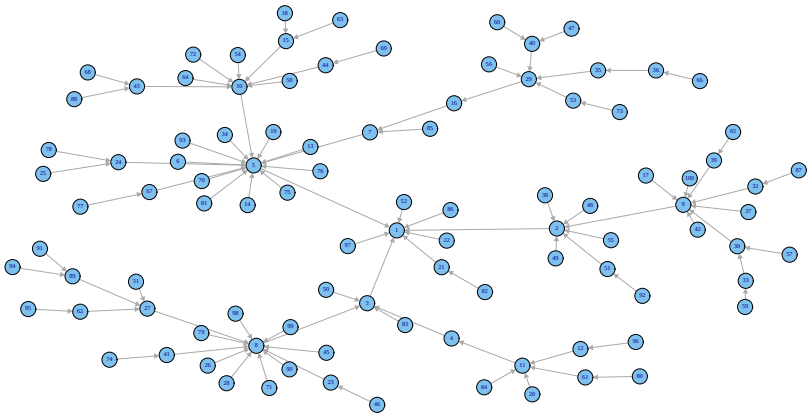


Figure 3.4: Barabasi-Albert Preferential-Attachment Graph

Constructed where $n = 100$, $m = 1$, $alpha = 1$, $a = 1$, $c = 1$

3.4 Growing Random Graph Model

The Growing Random (GR) model [13] is a model similar to the Barabasi-Albert model except that it eliminates preferential attachment. This model, as described in Algorithm 6, was used to demonstrate that the scale-free property does not occur in growing networks without the presence of preferential attachment. Instead of a power-law distribution of degrees found in preferential-attachment networks, it is observed that the probability of degree, k, is exponential [13] as in Equation 3.1.

$$P(k) \sim \exp(-\beta k) \tag{3.1}$$

Algorithm 6: Growing Random Graph Model

Input: $n > 0$, $m > 0$
Output: Growing Random Graph, $G = (V, E)$
begin
  for $i \leftarrow 1$ **to** n **do**
    $V \leftarrow V \cup v_i$;
    foreach $a \in RandomVertices(V - v_i, m)$ **do**
      $E \leftarrow E \cup (v_i, a)$;
    end foreach
  end for
  return G
end

Examining Figure 3.5, an output graph from the GR model, the same spanning trees observed in BA graphs (See Figure 3.4) are observed. However, the high degrees hubs found in BA graphs are missing in the GR graphs. Instead, several smaller degree hubs are found scattered throughout the graph.

3.5 Ageing Preferential Attachment Model

The Ageing Preferential Attachment (APA) model [27] is similar in many respects to the BA model, but it introduces some important differences. Like the BA model, creation of edges is guided by preferential attachment. The APA model introduces vertex age to preferential attachment. The idea is that as vertices age there is a change in their probability of new attachments. de Solla Price [25] remarked that in scientific citation networks there existed a phenomenon he called the "immediacy factor.' That is, as papers age, the number of new citations decreases because the paper is considered to be obsolete.

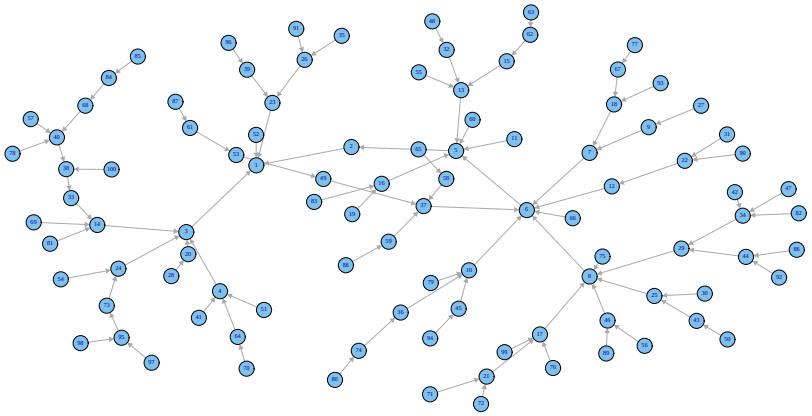


Figure 3.5: Growing Random Graph

Constructed where $n = 100$, $m = 1$

This attachment rule, however, has the ability to break the scale-free behaviour of the BA model when $\beta \geq 1$ in Equation 3.2, where β, b, d are constants acting on t_i, the age of the i^{th} vertex [27]. The Ageing Preferential Attachment model, modified from the BA model, is presented in Algorithm 7. The algorithm differs from Algorithm 5 in the probability calculation and the introduction of the new variables: β (the exponent for age), b (a constant factor of age preference), and d (the zero age appeal).

$$P_i = ak_i^{\alpha} + c + bl_i^{\beta} + d \tag{3.2}$$

Algorithm 7: Ageing Preferential Attachment Graph Model

Input: $n > 0$, $m > 0$, $\alpha > 0$, a, c, β, b, d
Output: A completed Barabasi-Albert Network, $G = (V, E)$
begin
  for $i \leftarrow 1$ **to** n **do**
    $V \leftarrow V \cup v_i$;
    for $j \leftarrow 1$ **to** $i - 1$ **do**
      $P_j \leftarrow ak_j^{\alpha} + c + bl_j^{\beta} + d$;
    end for
    for $j \leftarrow 1$ **to** m **do**
      $s \leftarrow select(P)$;
      $E \leftarrow E \cup (v_i, v_s)$;
    end for
  end for
  return G
end

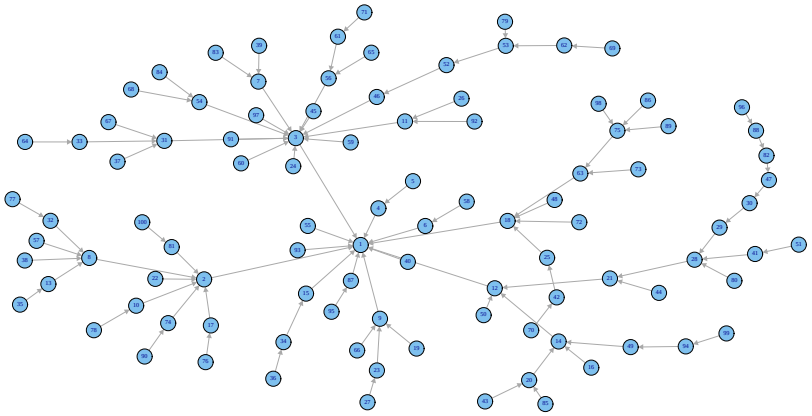


Figure 3.6: Ageing Preferential-Attachment Graph

Constructed where $n = 100$, $m = 1$, $alpha = 1$, $a = 1$, $c = 1$, $beta = -1$, $b = 1$, $d = 0$

Output graphs of the Ageing Preferential Attachment model, like found in Figure 3.6, generate hubs similar to those found in BA graphs (See 3.4). The spanning property observed in the growing networks exemplified in Figures 3.5 and 3.4 is also present in the Ageing Preferential Attachment. However, it is important to note that the model presented by [27] only exhibits the power-law degree distributions when the age exponent, β is less than 1.

3.6 Forest Fire Model

Leskovec *et al.* [54] observed that many real-world complex networks exhibit properties of heavily-tailed in and out degrees, formation of communities, and increase denseness due to the power law distribution of links and a shrinking diameter. As such, they proposed a model called the Forest Fire Model, which purports to encompass all of these properties.

Algorithm 8 illustrates the Forest Fire model algorithmically. The model begins each iteration of graph construction with the addition of a new vertex, v_i. Then, m ambassadors are selected and put into A. For each ambassador, a, an edge is created from v_i to a. Also, two numbers, x and y, are chosen randomly from a geometric distribution with a means of $\frac{p}{1-p}$, and $\frac{rp}{1-rp}$, respectively. Using x, and y, x successors and y predecessors of a are uniformly selected which are not already attached to v_i. The selected vertices are appended to A to act later as ambassadors to v_i.

One characteristic important to Forest Fire model is the formation of community structures, groups of vertices easily formed through dense connections. In Figure

Algorithm 8: Forest Fire Model

Input: $n > 0$, $m > 0$, $1 \geq p \geq 0$, $1 \geq r \geq 0$
Output: A completed forest fire graph, $G = (V, E)$
begin
  for $i \leftarrow 1$ **to** n **do**
    $V \leftarrow V \cup v_i$;
    $A \leftarrow RandomVertices(V - v_i, m)$;
    foreach $a \in A$ **do**
      $E \leftarrow E \cup (v_i, a)$;
      $x \leftarrow geoProb(\frac{p}{1-p})$;
      $y \leftarrow geoProb(\frac{rp}{1-rp})$;
      $W \leftarrow RandomVertices(b \in V | b \notin A \wedge (a, b) \in E, x) \cup$
      $RandomVertices(b \in V | b \notin A \wedge (b, a) \in E, y)$;
      $A \leftarrow append(A, W)$;
    end foreach
  end for
  return G
end

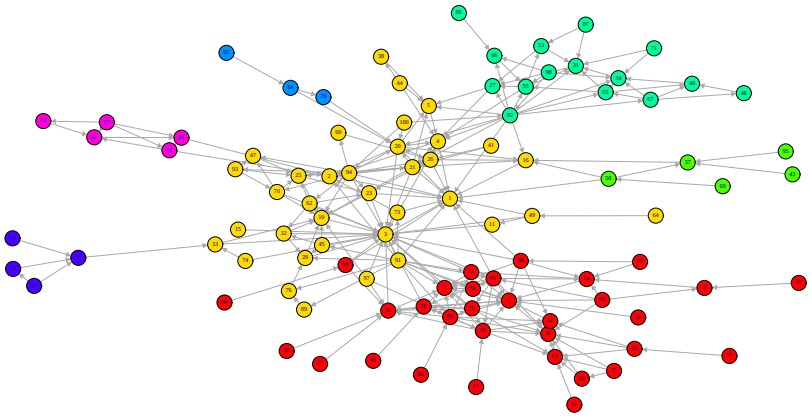


Figure 3.7: Forest Fire Graph

Constructed where $n = 100$, $m = 1$, $p = 0.37$, $r = \frac{0.32}{0.37}$.
note: Colours represent communities.

3.7, the communities of a graph derived from the Forest Fire model is illustrated by assigning a colour to each.

Chapter 4

Modelling Techniques for Graph Models of Complex Networks

A graph model provides a means of understanding the processes and behaviours that act on a specific network. It allows the exploration of the structural properties of the network and the development of predictors for the network. However, in order to construct a graph model, it is often necessary to have an intimate understanding of the structural properties and the behaviours of the network. Such a requirement means that constructing a graph model can present a significant challenge.

Despite challenges in the formation of graph models for complex networks, there have been approaches proposed to construct graph models. Some of these methods exploit properties commonly found in graph models, other make assumptions about the processes that form edges within the network. Moreover, some automate the entire model construction making few assumptions about the network at all.

In this chapter, a brief overview of works relating to the construction of graph models is provided. Moreover, the strengths and weaknesses of the techniques proposed by the works review will be presented. Finally, Section 4.5 conducts a discussion about previous methodologies in contrast to the proposed methodology of this thesis.

4.1 P^* Graphs

Exponential random graphs, called p^* graphs [78], are random graphs consisting of n nodes and m dyads that can be explained by some statistics $s(y)$. The set of dyads (edges) are defined in a set $Y = \{Y_{i,j} : i = 1 \dots n, j = 1 \dots n\}$ where $Y_{i,j} = 1$ if vertices i and j are connected via an edge otherwise $Y_{i,j} = 0$.

The main idea of modelling using the p^* technique is that the observed values

of structural measures on some given network only captures one of a large number of possible networks. In order to determine the dyads given n vertices, a likelihood algorithm is employed to determine the $s(y)$ and construct the dyads sets.

There exist a number of proposed methods for constructing the graphs employing various likelihood algorithms. Some of these methods include loglikelihood [39], pseudolikelihood [39], and Markov chains [79]. Exponential random graphs have been demonstrated to be successful at modelling complex networks [78]. However, p^* graphs rely heavily on local structures in networks and are therefore very slow and do not capture the network as a whole [55].

4.2 Kronecker Graphs

The Kronecker graph was proposed by Leskovec *et al.* [55]. The graphs produced are said to exhibit properties of real-world networks such as heavily tailed in- and out-degrees, and small diameters that shrink over time. The model employs a matrix operation called the Kronecker product.

Kronecker graphs are used to model networks by constructing a Kronecker matrix containing parameters for the graph model constructed. The parameters are refined using a maximum likelihood algorithm which runs in linear time. Such a runtime is considerably quick considering prior efforts at constructing maximum likelihood algorithms produced asymptotic times that were quadratic [55].

The modelling of a network using the Kronecker graph model begins with an initial graph K_1 containing N_1 vertices and E_1 edges. Recursively, larger graphs $K_2, K_3, \ldots$ are produced such that K_k has N_1^k vertices and E_1^k edges. This process creates the Densification power-law observed in real-networks [53] and is accomplished using the Kronecker product.

There are two types of Kronecker graph models, the first constructs a graph by successively applying the Kronecker product to the adjacency matrix of the graph N_1. This model is known as the deterministic Kronecker graph model. The other, the stochastic Kronecker graph model, applies the Kronecker product recursively to a probability matrix N_1. The matrix contains cells such that a cell i, j is the probability of forming an edge between vertices i and j.

The stochastic version of the Kronecker graph models has been applied to create supercomputer benchmarks for graph algorithms [72]. The model has also been shown to replicate real-world networks [51]. Furthermore, the model is simple and fast. However, it has been suggested that this model has a limited range degree distributions

[72]. Moreover, the model relies on the assumptions of power-law distributions and the Densification property.

4.3 Chung-Lu graphs

The Chung-Lu graph model [2] constructs graph with a degree distribution given by α and β such that y vertices of degree x satisfy

$$\log y = \alpha - \beta \log x \tag{4.1}$$

In general, to construct a graph the model uses the sequence [3]

1. Form a set L of x distinct copies of vertex v
2. Choose, randomly, a matching of elements from L
3. For vertices u and v, the number of edges formed between u and v is equal to the number of edges in the matching of L

Aiello *et al.* [4] present four variations of the Chung-Lu graph model. The first model(Model A), is considered the basic model and forms the basis of the other three models. At time t, the out-weight of u is computed as $1 + \delta_{u,t}^{out}$ and the in-weight of v, $1 + \delta_{v,t}^{in}$. An edge is then formed between u and v with the probability of

$$\alpha \frac{(1 + \delta_{u,t}^{out})(1 + \delta_{v,t}^{in})}{t^2} \tag{4.2}$$

The second model (Model B) adds controls that allow independently controlled in- and out-degrees via the use of new parameters. Model C adds new controls that allow edge creation via four different mechanisms. These are

1. Edge from u and v randomly selected by probabilities based on the in-degree of u and the out-degree of v
2. Randomly selected vertex u connected to the newly formed vertex where u is selected probabilistically based on its out-degree.
3. A newly formed vertex is connected to a randomly selected vertex u such that its selection is probabilistically based on its in-degree.
4. A loop to the new vertex.

The final model (Model D) is a variant of Model C such that it constructs undirected networks. All models construct power-law distribution graphs with large components in the graph and utilize parameters to fluctuate the degree distributions.

It is said that the range of degree distributions obtainable from these models are greater than those which can be produced by stochastic Kronecker graphs [72]. Furthermore, [72] argues that the optimization of Chung-Lu graphs is simpler than that of a Kronecker graph. Regardless, this model assumes that complex networks follow a power-law distribution.

4.4 Evolutionary Modelling Strategies

In this section, two methods of modelling are presented which utilize evolutionary strategies, namely Genetic Programming (GP). These strategies are unique to the previous strategies and make fewer assumptions about the network being modelled [62]. While both methods employ GP, they do so in very different way.

The first method, proposed by Bailey *et al.* [11], uses GP to construct a complete program which when run constructs a graph. Generated models are evaluated based on their ability to reproduce specific network properties. These measures are the average geodesic path length, transitivity, and degree distribution of the network. A graph is provided as an example to the system and used to compare the output graphs of an evolved program. The methods proposed by Bailey *et al.* [11] provided the first attempts in automation of graph modelling using GP. However, the method utilized priority queues [10], and community detection algorithms Bailey *et al.* [12] to construct some types of networks.

Another method, recently proposed in [62], used symbolic regression to construct networks. The work proposed a model such that at each time step a vertex was added to the graph and an edge was added to the graph probabilistically from the possible edges not already constructed. The probability of an edge being formed from u to v was determined by a probability function which was evolved by the GP. Experiments were performed on this method using the Erdos-Renyi graph model and the Barabasi-Albert graph model.

4.5 Discussion

Many of the proposed methodologies for generating graph models infer that specific properties exist in complex networks. For example, Leskovec *et al.* [55] explain that

"Real networks exibit a long list of surprising properties: Heavy tails for the in- and out-degree distribution, heavy tails for the eigenvalues and eigenvectors, small diameters, and densification and shrinking diameters over time." Despite numerous examples [6, 13, 23, 67] which indicate that such properties hold, there exists no means to conclude that this is the case in general. A contradiction is found in the Ageing Preferential Attachment model, in the model there are situations where the power-law degree distribution does not hold [27].

Despite the specificity of the properties modelled by parametrized models of complex networks, they are important and provide a practical means of evaluating the theory in complex networks. Their existence suggests that knowledge about complex networks can be used successfully to construct accurate graph models.

On the other hand, GP methods for the automatic creation of graph models offers an attractive means for the construction of graph models that do not require the assumptions made by parametrized models. The evolutionary strategies by Bailey [10] do not require that a graph model have a heavily tailed degree distribution. Evidence for this is found in the ability to replicate the Erdos-Renyi model where the degree distribution is very different from the scale-free property observed in other networks. The symbolic regression method proposed by Menezes and Roth [62] takes a step back further from any assumptions of graph models. Graphs are constructed based on a probability formulas constructed by the system. However, the previous evolutionary strategies do not make use of what is known about a complex network being modelled specifically with respect to the process of the network. This means that regardless of what is known about the processes of a network being modelled the GP is made to determine all processes in the network.

No matter the method, modelling constructs only one of many possible ways of constructing a graph model and do not take into account the semantics of the network at hand [78]. While this thesis does not directly experiment with encoding the semantics of complex networks into solutions, it does leave room for such constructions. Moreover, it addresses the lack of expert knowledge encoded into GP methodologies via the proposal of a novel GP technique. Thus, the framework offered by the proposed methodology of this thesis (See Chapter 7) attempts to build upon the strengths of all modelling techniques. The framework facilitates the research to define the algorithmic structure of the graph model and, thereby, reduces the responsibility of the GP to evolve all processes of the graph model. Furthermore, it maintains the ability to construct a very diverse set of graph models as the expert knowledge provided to the system is varied.

Chapter 5

Genetic Programming

Genetic programming (GP) is a computational intelligence technique inspired by Darwinian evolution and introduced by Koza [43]. In genetic programming, a population of programs is randomly created and evolved using a survival of the fittest approach. That is, programs that more closely meet a desired criterion have better odds of passing their genetic code to the next generation.

GP has been used in a number of different applications such as curve fitting [9, 74, 87], the development of circuits [45], evolutionary art [16], modelling [11, 80], and the construction of artificial agents [41], among many others [45, 74].

In its original form, as proposed by Koza [43], GP forms tree-based programs which utilize a functional programming paradigm. However, this is not the only representation for programs in GP. In Sections 5.2, 5.3, and 5.4, a brief review of three of forms of GP representation will be given. In Section 5.2, the tree-based form will be discussed. Section 5.3 will provide an alternative to tree-based GP, linear GP. Finally, Section 5.4 provides a background on Object-Oriented GP, the paradigm adapted by this thesis.

5.1 The Genetic Programming Algorithm

While there are many approaches to GP, they all share a very general algorithm called an evolutionary algorithm. This is not to be mistaken with the related evolutionary computational technique, Genetic Algorithms. Instead, evolutionary algorithms, in the context of this section, refers to the general algorithm that loosely describes the processes of both genetic programming and Genetic Algorithms. In the evolutionary algorithm [35], a population (or sometimes populations) of programs are stochastically transformed to build new programs. Algorithm 9 provides a loosely constructed

evolutionary algorithm which describes the processes of the Genetic Algorithm.

Algorithm 9: The Evolutionary Algorithm

```
begin
    randomly create a population of individuals;
    while acceptable solution not found or other stopping criteria not met do
        Determine the fitness of each individual in the population;
        Probabilistically select individual(s) from the population base on fitness;
        Create new individual(s) using a genetic operator and selected
        individual(s);
    end while
    return best solution
end
```

Algorithm 9 begins with the random creation of a population of individuals. In GP, this is a population of randomly constructed programs [74]. The construction of programs is dependant upon the representation of the GP. In the later sections of this chapter, some representations available in GP will be discussed. Algorithm 9 also describes at each iteration (or in Genetic Algorithm terms, generation) that the fitness of each individual must be evaluated. In GP, this is accomplished by executing each program and evaluating its result [74].

5.1.1 Selection

Selection of individuals, the next step in Algorithm 9, is determined by a probability based on fitness. That is to say, the probability of selecting a highly fit individual should be greater than the probability of selecting an individual who is much less fit. Two methods for selecting individuals probabilistically are tournament selection and roulette selection.

In tournament selection, for each individual selection k individuals are randomly chosen from the population and the fittest wins and is selected. roulette selection selects individuals by determining their proportional fitness in the population and probabilistically selecting an individual based on their proportional fitness.

5.1.2 Genetic Operators

After having selected individuals. The operations that will be performed on a selected individual is decided by a probability of performing the genetic operator. The

probability of each operator's use is a parameter of the Genetic Algorithm. There are two major operators which are used: crossover and mutation.

crossover is the recombination operator which takes, usually, two individuals and transforms into one or more new individuals. The common idea of all crossover operations is that genetic information is taken from multiple individuals and recombined.

mutation is the application of random changes to an individual. This operation, usually, has a low probability of occurring and when it does occur only small changes are made to the individual.

In the case of both operations, their implementation is determined in part by the representation of the GP system at hand. As it will be demonstrated in the following sections of this chapters, the genetic representations of each approach reviewed are very different and therefore require different implementations of genetic operators that can work with the representation being employed.

5.2 Tree-Based Genetic Programming

Genetic programming is a tree-based system and was originally popularized by Koza [43]. In the vanilla GP proposed by Koza [43], functions used by a GP system were singly-typed, i.e., all elements of the program had the same type. In this form, the programs are comprised of functions and terminals. Functions are elements which have arguments and terminals are elements which do not. Figure 5.1 provides an example of a tree-based program typical of tree-based GP. In the program a function is constructed that is equivalent to the expression $x \times 3x + \frac{2}{x}$.

In a tree-based genetic program, crossover operations are used to recombine two programs to create two new programs. crossover is, usually, performed by selecting a subtree from each parent and swapping them to form two new programs, the children. This process is illustrated in Figure 5.2.

A variant of vanilla genetic programming, strongly-typed GP [65], introduces the ability to represent more sophisticated function definitions by facilitating the construction of program trees with more than one type. Instead of the set of functions and terminals all having the same type, the set contains functions and terminals which do not all have the same type. Also, the arguments of functions will require specific types. With this special consideration must be taken into account when constructing program trees.

Some users of GP utilize the strongly-typed GP system to control the follow of the program structure in GP. Recognizing this representation as cumbersome, some

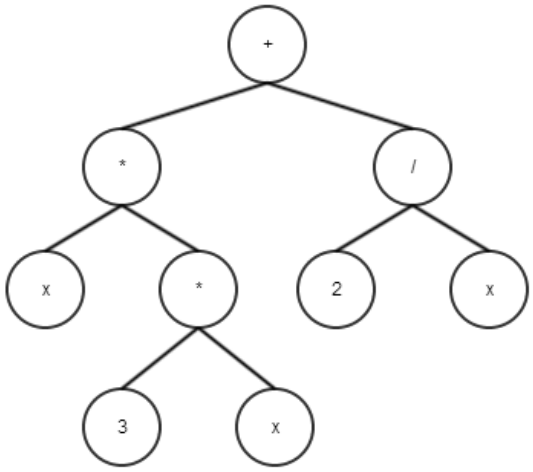


Figure 5.1: Singly-typed Tree-based Program

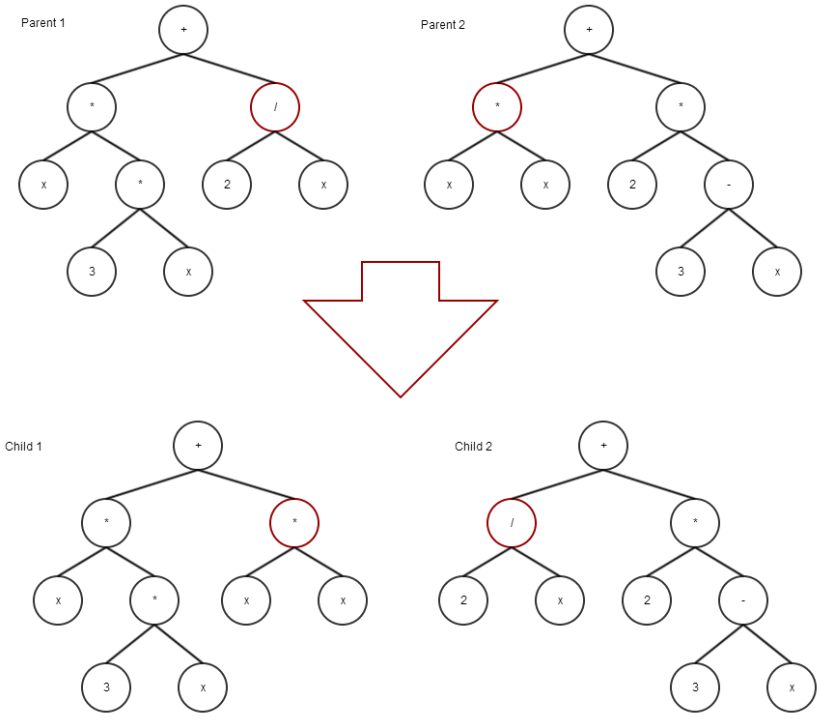


Figure 5.2: Tree-based GP Crossover Operation

researchers developed grammar-guided GP systems (GGGP) [18, 32, 92]. This form of tree-based system allowed the construction of programs that were built using context-free grammars (CFGs). Using GGGP, it is also possible to define sub-functions Wong and Leung [94]. One of the main role that GGGP systems serve, however, is to reduce the search space of the problem [59].

5.3 Linear Genetic Programming

Linear genetic programming [74] is an alternative to tree-based GP. Linear GP, like tree-based GP, produces a population of candidate programs to be evolved. However, programs in linear GP are represented as a linear array of instructions [19], rather than a tree structure. It is important to understand that linear refers to the structure of the program representation as an imperative representation and not a linear list of nodes [20]. In this sense, the building blocks of a linear GP system consist of registers (variables) and instructions. Many techniques have been proposed for linear GP [71], each with advantages and disadvantages. However, an important advantage of linear GP over tree-based GP is the ability to vary the destructiveness of crossover and mutation [19].

One approach to interpreting the chromosome is to use a function map. The chromosome representation in this approach is a linear array of bytes like in Figure 5.3. The interpretation of the bytes is determined by a table of instructions, for example, Table 5.1, which are assigned to the bytes. Similar to the representation of assembly code.

Figure 5.3: Chromosome Comprised of an Array of Bytes

Byte Code	Instruction
0001	1
0110	Add
1001	9
1011	x
1100	Subtract
1111	Multiply

Table 5.1: Example of Byte to Instruction Mapping

Interpreting Figure 5.3 via Table 5.1, the chromosome's program is $+ - 9 \times xx1$ or using infix notation $9 - x^2 + 1$. However, if not careful both the table used and the configuration of the chromosome will result in an infeasible program construction. The solution to this is to use a grammar that guides the construction of the chromosome.

Notably, linear GP systems are said to be fast systems as they consist of low-level simple instructions and written in languages such as C and C++ [20]. However, this is not the only reason that linear GP is considered in some applications of GP. It has been shown by Langdon and Banzhaf [50] that dead code is more readily identified in linear GP than in tree-based GP.

5.4 Object-Oriented Genetic Programming

In the previous sections, discussion surrounded GP systems which produced programs which had one behaviour, i.e. a single function. However, in practice, programming to solve a problem involves multiple functions and data structures. It is possible to evolve complex programs like this using other forms of GP [44, 64, 94]. However, it is also possible to construct representations of GP as objects. object-oriented genetic programming (OOGP) provides a framework with which multiple functions and/or data structures are evolved.

Object-oriented programming is a widely used programming paradigm where types are represented as entities with behaviours. In the object-oriented paradigm, instances of types, known as objects, are mutable by way of methods that are analogous to behaviours. The OOGP methodology [1] provides a framework to produce programs in such a paradigm. Past approaches with OOGP made use of a multi-tree representation, whereby a chromosome's consisting of several trees, each corresponding to a single method, are evolved. The multi-tree representation used by OOGP allows the simultaneous optimization of multiple methods, leading to the evolution of more complex programs.

An example of the application of OOGP can be drawn from [49]. In the work by Langdon [49], OOGP was employed to evolve data structures such as a stack. A stack has multiple functions such as pop and push. Simultaneously evolving both of these behaviours in conventional approaches to GP is not simple as traditional approaches such as tree-based and linear GP systems are designed to evolve a single function. Therefore, Langdon [49] proposed a representation that consisted of multiple trees; one for each function of the stack, for instance.

Other approaches have also been proposed [1, 21, 31]. However, none of these

proposed methods seem to have been widely adopted as searches on the topic do not reveal much further uses beyond initial proposals of these systems. This is unfortunate, as these systems facilitate an incorporation of additional knowledge not easily conveyed in other approaches. OOGP allows a specification of the behaviours of an object to be easily constructed which then in turn can be used to produce data structures that have multiple behaviours.

In the research of this thesis, such a structure has been adopted. A graph model is a data structure which has a set of processes that construct a graph. Therefore, a specification of the model can be constructed, and the implementation evolved from the specification. However, the current OOGP approaches available employ tree-based approaches. As explained in Section 5.2, tree-based approaches are often difficult to use as representation of the problem can be difficult. Furthermore, the crossover processes of tree-based GP is limiting and destructive [83]. Thus, we arrive at the motivation for this thesis in the proposal of a new OOGP system (see Chapter 6) that employs a linear-based OOGP and readily incorporates expert knowledge.

Chapter 6

LinkableGP – An Object-Oriented Linear Genetic Programming System

LinkableGP is an Object-Oriented Genetic Programming (OOGP) System that is designed to incorporate expert knowledge. It does so by utilizing an abstract class definition. The abstract class also provides a means of defining the structure of the programs evolved and allows some functions of the evolved program to have implementations *a priori*.

In the sections to come, the systems construction will be discussed in greater detail describing how expert knowledge is incorporated. Furthermore, the structure and representation of individuals as well as the genetic operators used during evolution will be described.

6.1 Facilitation of Expert Knowledge

To facilitate expert knowledge in GP, proposed is a novel object-orientated linear GP system, LinkableGP. LinkableGP makes use of 1) an object-orientation methodology to construct class-based programs and 2) linear GP to construct imperatively-defined methods within the class. As a result, LinkableGP allows the use of partially-implemented classes, i.e., abstract classes, whereby the user may incorporate their domain knowledge of the problem. The knowledge is embedded into the abstract class by providing implementation of methods where the desired functionality is known *a priori*. The remainder of the functionality is evolved using LinkableGP's evolutionary

strategies. Finally, LinkableGP produces a .NET-based[1] source file corresponding to the evolved implementation. The resulting source file is structured as expected for object-oriented, imperative style .NET source code.

6.2 Motivation

The primary motivation for allowing expert knowledge encoded within an evolved program stems from the difficulties experienced when controlling the flow of programs using traditional GP approaches. When a program that has several logical stages is evolved using traditional GP approaches, often it is necessary to construct a collection of rules for the set of functions and terminals [58]. Handling of the rules is typically managed by either employing a grammar to define valid program states or via the manipulation of types. However, decomposition of a program into logical methods, each with a set of functions and terminals specific to the associated task, will result in a more natural program structure.

Furthermore, interpretation of the algorithms produced by GP is often difficult due to the unintuitive program structure evolved. That is, code produced by GP is typically dissimilar to that of human-created code. Therefore, it is desirable to generate programs in a paradigm that is both familiar and intuitive. Having such programs facilitates expert evaluation of the outcome and allows for refinement of the strategies employed.

6.3 Structure and Representation

Unlike a traditional GP tree structure, which represents a single function, individuals in the LinkableGP population represent types or classes. The classes are implementations of a parent type. This parent type is an abstract class which allows the user to encode expert knowledge elegantly and in a manner familiar to most programmers.

The structure of individuals within a population in LinkableGP is inspired by both linear GP and OOGP representations. Furthermore, individuals are constructed using a collection of chromosomes, analogous to a DNA structure. The DNA structure of an individual defines the physical implementations of abstract methods from the parent type. That is, each chromosome corresponds to the code sequence of a single abstract method defined in the parent type. Each chromosome's genotype is an array of integers, which directly corresponds to a code sequence that constructs a single

[1] Refers to Microsoft's .NET framework

method in the resulting phenotype. A visualization of the chromosome structure and genotype to phenotype process is presented in Figure 6.1.

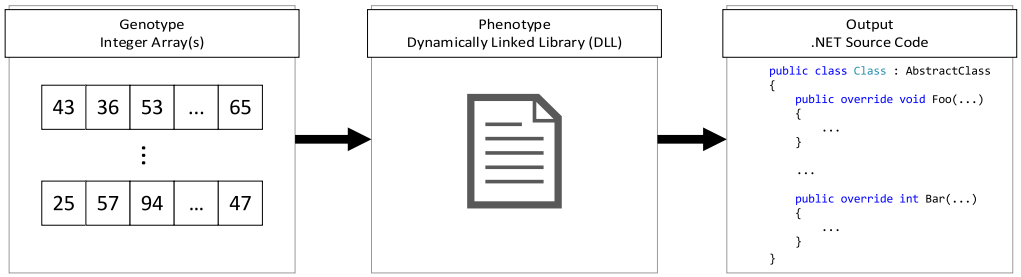


Figure 6.1: Visualization of the genotype to phenotype mapping for an individual in the LinkableGP system.

6.3.1 The Abstract Class

The abstract class in LinkableGP serves as an embryo for each individual. The class consists of any number of implemented or abstract methods, as is expected of an abstract class. Those methods that are marked abstract are to be implemented by the individuals and evolved by LinkableGP. That is, the genotype of each individual is defined by the number of abstract methods. The phenotype of the individual is the implementation of the abstract class as defined by the genotype.

6.3.2 The Genotype

The genotype of an individual is a collection of integer arrays, called chromosomes, one for each method marked abstract by the defining abstract class. The chromosomes are of variably-lengthed arrays to allow different lengths of programs. Each value in the chromosomes represents an element in the construction of the abstract method it is mapped to. The process that governs the conversion between chromosome to method implementation is explained in Subsection 6.3.4.

6.3.3 The Phenotype and the Language

The phenotype of an individual is the implementation of the abstract class. It is resulting source of the program that has been generated by the individual through the mapping of the genotype to the phenotype. In order to build a method, a language (i.e., a set of functions, constants, and constant generators) must be defined. Each

method utilizes a language set as the functions, constants, and other constructs for building one method and is not necessarily useful to another.

The language, however, is not static. Dependent on the context built-in mapping the genotype, new values may result. For example, if a function that is used in the method being implemented has a result either a mutable variable is selected from the language or a new one might be created. If a new variable is created, it is added to the instance of the language that is being used by this implementation. This change in context, however, does not prevail or transfer between chromosomes.

6.3.4 Mapping Genotype to Phenotype

The mapping of a genotype to phenotype transforms each chromosome in the genotype to a method in the phenotype. The process of mapping a chromosome to a method constructs single instructions iteratively. During the mapping of the chromosome, an index is maintained to the current position in the chromosome array. The index is iterated each time a chromosome value is used during construction. The first step in creating an instruction is to select a function from the language. The selection of a function from the language is restricted to those functions whose arguments can be immediately satisfied by the available variables in the variable bag. To determine which function is used from the subset, the next chromosome value mod the cardinality of the subset is computed. The computed value provides the index of the selected function from the subset.

Given the function for the instruction, the arguments of the function must now be satisfied using variables from the variable bag which have the same type as the argument. The selection of the variable is determined by the computed value of the next chromosome value mod the cardinality of the subset of variables. The selected variable is then the variable at the index of the computed value in the subset. The assignment of variables to arguments continues until all arguments are satisfied.

If the function in the current instruction has a result type that is not void, a variable must be selected to store the result. Like with arguments of a function, a subset is formed from the variable bag such that it contains only those variables which have the same type as the result type. However, there is an added constraint to variables in the subset that they also be mutable variables. The only variables which are mutable in LinkableGP are those that are constructed locally during mapping. Before a variable is selected from the subset, a new variable is added to the subset with the same type as the result type of the function. Then, as before, the next chromosome

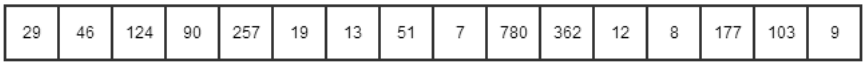


Figure 6.2: Example Chromosome

value mod the cardinality of the subset is computed. The selected variable for the result of the function is the variable at the index equal to the computed value from the subset of variables. If the variable selected is the newly created variable it is added to the variable bag. Since this variable was constructed locally, it will be marked as mutable and, therefore, usable as for future assignments in the construction of the method.

The process of constructing instructions continues until there the end of the chromosome is reached. If the end of a chromosome is reached during the construction of an instruction then, the chromosome array is extended with random values from an uniform distribution to allow the completion of the instruction. This extension remains as a part of the chromosome. If the method being constructed has a return value then, one chromosome value is reserved at the end of the chromosome array to allow the selection of the return variable from the variable bag. This value is selected, as before, by constructing a subset of variables that have the same type as the return type of the method. The variable is selected by computing the final chromosome value mod the cardinality of the subset and selecting the variable at the index equal to the computed value. If the subset is empty, an instruction is constructed using the same method as describe for constructing a typical instruction with the added requirement that the select function has the same return type as the method. In this case, the chromosome is extended sufficiently to allow the completion of the return instruction.

Example

In order to clearly understand the mapping process the following example is provided. Assume an abstract class that contains only one method that is marked abstract and has the signature void ***MagicNumber****(x:Integer, y:String)*. A set of functions for the language in the example is provided in Figure 6.1. The variable bag will initial only contain inputs variables x and y which are marked as immutable. The method will be mapped given the chromosome found in Figure 6.2.

Function Set

Function Name	Arguments	Return Type
Add	a:Integer, b:Integer	Integer
Sub	b:Integer, b:Integer	Integer
ToString	a:Integer	String
Print	a:String	void
ToInteger	a:String	Integer

Variable Bag

Variable Name	Mutable	Type
x	False	Integer
y	False	String

Table 6.1: Initial Example Language

The first instruction of the phenotype will use the chromosome values at indices $[0, 2]$ which are $\{29, 46, 124\}$.[2] First the function is selected from the function set. Given the variable bag, all arguments of all functions are satisfiable, so the subset of functions is the complete set of functions in the language. The cardinality of the set is 5 and so the index of the function selected is $29 \mod 5 = 4$ which is the ToInteger function. The type of the only argument for the ToInteger function is a String, so the subset of variables from the variable bag is $\{y\}$. The variable y is selected as the actual parameter of the function because $46 \mod 1 = 0$. This function has a result variable so we consider the set $\{a\}$ as the possible variables for assignment of the result for the ToInteger instruction. The variable a, is a new variable which is not added to the variable bag unless it is selected.[3] Since the subset of variables from the variable bag that are mutable and of type Integer is an empty set, no other variables are added to the set. The variable which will store the result is a because $124 \mod 1 = 0$ and there is only the one variable. After one instruction, the language is updated to match Table 6.2.

The next instruction in mapping the chromosome utilizes the chromosome values at indices $[3, 6]$ which are the values $\{90, 257, 19, 13\}$. The subset of functions usable for this instruction, as was the case with the first instruction, is the complete function set of the language. The index of the function for this instruction is computed as 90

[2]Please remark that at this point in the system it is not known how many chromosomes will be required to for an instruction. However, for the purpose of determining where values are obtained in the chromosome the values used for each instruction are provided before explaining the result.

[3]Variable names are determined as the next available letter in the alphabet which does not mask any other already used variable in the scope of the method. If z is reached in variable naming the convention is to append is to continue with $(aa, ab, ac, \ldots, az, ba, \ldots)$

Function Set

Function Name	Arguments	Return Type
Add	a:Integer, b:Integer	Integer
Sub	b:Integer, b:Integer	Integer
ToString	a:Integer	String
Print	a:String	void
ToInteger	a:String	Integer

Variable Bag

Variable Name	Mutable	Type
x	False	Integer
y	False	String
a	True	Integer

Table 6.2: Example Language after one instruction

mod $5 = 0$, thus, the function for this instruction is Add. The add instruction has two arguments both of which are Integers. Therefore, the subset of the variable bag is $\{x, a\}$ for both arguments. The variable selected for the first argument is a ($257 \mod 2 = 1$) as is the second argument ($19 \mod 2 = 1$). The set of variables considered for the result of the function, Add, is $\{a, b\}$ such that b is a potential new variable. The computed value, $13 \mod 2 = 1$, determines that b is selected to store the result. As a result of this instruction, the language is updated to reflect the new variable b as seen in Table 6.3.

Function Set

Function Name	Arguments	Return Type
Add	a:Integer, b:Integer	Integer
Sub	b:Integer, b:Integer	Integer
ToString	a:Integer	String
Print	a:String	void
ToInteger	a:String	Integer

Variable Bag

Variable Name	Mutable	Type
x	False	Integer
y	False	String
a	True	Integer
b	True	Integer

Table 6.3: Example Language after two instructions

The next instruction uses the chromosome values from indices $[7, 10]$ which are

$\{51, 7, 780, 362\}$ (See Figure 6.2). This instruction uses the function Sub as the index of the function from the subset of functions is 51 $\mod 5 = 1$. The arguments selected from $\{x, a, b\}$ for the function's invocation are a (7 $\mod 3 = 1$) and x (780 $\mod 3 = 0$). The result of the function is assigned to the variable a (362 $\mod 2 = 0$) from the set $\{a, b\}$. After this instruction, the language as provided in Table 6.3 is unchanged.

The fourth instruction in the mapping of the examples chromosome uses the chromosome values at indices $[11, 13]$ which are $\{12, 8, 177\}$. This function results in the use of the function ToString (12 $\mod 5 = 2$) using the argument b (8 $\mod 3 = 2$) from the subset of the variable bag $\{x, a, b\}$. The result of the ToString is assigned to the variable c (177 $\mod 1 = 0$) from the set $\{c\}$ which contains only a new variable as the variable bag does not contain a mutable String. The language is updated to include the new variable c and is provided in Table 6.4.

Function Set

Function Name	Arguments	Return Type
Add	a:Integer, b:Integer	Integer
Sub	b:Integer, b:Integer	Integer
ToString	a:Integer	String
Print	a:String	void
ToInteger	a:String	Integer

Variable Bag

Variable Name	Mutable	Type
x	False	Integer
y	False	String
a	True	Integer
b	True	Integer
c	True	Integer

Table 6.4: Example Language after four instructions

The final instruction in this example uses the last two values of the chromosome from Figure 6.2 which are $\{103, 9\}$. The selected function for this instruction is determined by the value 103 $\mod 5 = 3$ and is Print. This function, unlike the functions previously used, has a void return type. Thus, only the single argument is required to be assigned. For the example this is the String c selected using the value 9 $\mod 2 = 1$ from the set $\{y, c\}$.

At this point in the example, all chromosome values have been exhausted. Thus, the mapping of the chromosome is complete. The resulting implementation of the chromosome for the method, void ***MagicNumber****(x:Integer, y:String)*, is found in

Algorithm 10.

Algorithm 10: Resulting Algorithm of Example Genotype-Phenotype Mapping

```
Function MagicNumber(int x, int y)
    a=ToInteger(y);
    b=Add(a, a);
    a=Sub(a, x);
    c=ToString(b);
    print(c);
```

6.4 Operations

LinkableGP employs a typical, generational genetic algorithm that performs crossover, mutation, elitism, and selection operations. However, the crossover and mutation operations are modified to work with the DNA structure. A proportion of the best performing individuals from the previous generation is directly copied to the new population, via elitism while the remainder of the new population results from offspring of the crossover operations.

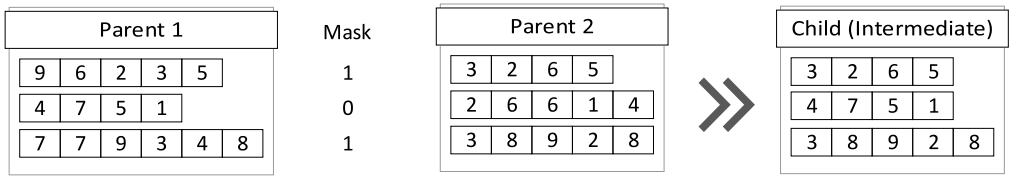
6.4.1 Crossover

Crossover, visualized in Figure 6.3, is a two-phased operation. Crossover, as performed in LinkableGP, results in the placement of a single new chromosome in the new population constructed in a generation.

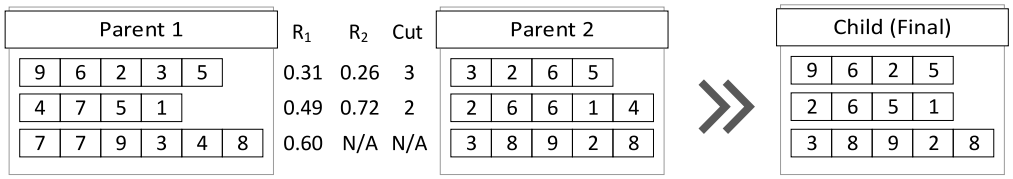
The first phase referred to as mating, selects two parents using a standard tournament selection operator and constructs a bit-mask to determine the chromosomes that are to be inherited from each parent. Each bit in the randomly generated bit-mask acts as parent discriminator. The chromosome value at 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 6.3a. For some chromosomes of an individual, this might be the only phase of the crossover performed. A random value, R_1 is selected uniformly and if the value is greater than ρ the second phase will not occur.

During the second phase called mixing, a chromosome in the child is replaced by the offspring resulting from a one-point crossover operation between the parents. A random cut-point is selected within the shortest parent chromosome. The mixing phase occurs 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 selection of the mixing method is determined by the random value, R_2 If $R_2 < 0.5$ the first method is used; otherwise the second method is used. The mixing phase is depicted in Figure 6.3b.



(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 6.3: Visualization of the two-phase crossover operation in the LinkableGP system.

6.4.2 Mutation

Mutation in LinkableGP is done either by extending/contracting the length of a chromosome or by randomly changing values within a chromosome. There is a chance of mutation for every chromosome of every individual added to the population in a generation. Any chromosome may be affected by any combination of either mutation approach.

In the extension/contraction mutation, a chromosome has either an integer appended to it or removed from it. The operation performed is selected uniformly. The chance of any chromosome of individual having this operation perform is determined by the system parameter m_1.

Alternatively, with probability m_2 a chromosome of an individual could have a value within its array randomly changed. The element that is mutated is determined uniformly as is its replacement value.

Chapter 7

Proposed Methodology

This thesis proposes a methodology for the automatic inference of graph models for directed complex networks using Object-Oriented Genetic Programming. This approach both facilitates and benefits from the incorporation of expert knowledge. Thus, the methods proposed in this chapter utilize the knowledge gained from the examination of existing graph models representing complex networks.

In Chapter 3, several models were examined which provide inspiration for the methods described in this chapter. The examination of these models offers insights that will form the basis of an abstract graph model (see Section 7.1) that will be used by LinkableGP to implement models targeted. Also from the examination of the models, a collection of function sets (the language) will be proposed for LinkableGP, which will provide the building blocks for the implementation of the abstract methods. Details of the make-up of the language will be provided in Section 7.2.

When generating a graph model, there is a target complex network. However, if the complex network was known then modelling it would be trivial. Therefore, sample data must be relied on. Using a representative graph, a *target graph*, from the network being model provides such sample data.

In order for LinkableGP to evaluate how well a program is modelling the complex network at hand, graphs must be produced by the program. The graph produced by the program can then be evaluated, statistically, to the target graph. The evaluation will provide LinkableGP an indication of the fitness of the program. In Section 7.3 the methods for evaluating the graphs produced by programs will be explained in detail.

Armed with an abstract class, language, and fitness function, the inference of a graph model begins with the construction of a population of programs. Then each program is evaluated against the target graph and ranked relative to the population.

The best-performing programs are immediately moved to a new population. In order to complete the construction of a new population, genetic operators are applied to the existing population to form new programs. The completed new population then replaces the old population, and the process begins again from evaluating and ranking. The algorithm continues the process of construction populations, evaluating and, ranking for some prescribed number of iterations. The final result is a graph model which should model the targeted network.

7.1 Abstract Graph Model

In order for LinkableGP to exploit utilize knowledge about the behaviour of complex networks, an abstract class definition is required as a partial representation of the knowledge. In this section, one representation will be proposed. It is stressed that this is only one of many possible representations, and the focus of this representation is relevant to the models evolved in Chapter 8.

Algorithm 11 presents the abstract class that will be used in this thesis. The first function, *GenerateModel*, serves as the calling function to generate a graph produced by an implementation of the class. In this function, construction of a graph begins with the creation of an initial graph. For the purposes of this thesis, this is always an empty graph.

The body of the function in Algorithm 11, iterates over t time steps, each time adding a new vertex, v, to the graph. The iterations are done this way because all the complex networks experimented with in Chapter 8 follow a growing behaviour (assumed to be one vertex each iteration). After vertex addition, a collection of vertices, W, are selected to form new edges with the new vertex. However, edge creation has the possibility of being probabilistic and thus there exists a possibility that no edge will be formed. The process of selecting vertices for edge creation is something observed in all the graph models found in Chapter 3, however it is a step that is performed differently depending on the network. Therefore, this function, named *SelectVertices*, is marked abstract and to be evolved by LinkableGP.

After the initial creation of W in the function, *GenerateModel* in Algorithm 11, an iteration over the W vertices are performed. In this iteration, the current vertex w from W and v are passed to a function *AddEdge* which will return an edge. However, this edge might be an empty edge which gives the effect of no edge being added. This function is marked abstract and is evolved by LinkableGP because the criteria in which vertex addition and the direction of the edge is unknown and dependent on

Algorithm 11: Abstract Graph Model for Complex Networks

Function `GenerateModel(`*int t*`)`**return graph**
  $G \leftarrow InitialiseGraph()$;
  for $i \leftarrow 1$ **to** t **do**
    $v \leftarrow Vertex()$;
    $W \leftarrow SelectVertices(G)$;
    while $|W| > 0$ **do**
      $w \leftarrow Next(W)$;
      $E \leftarrow E \cup AddEdge(v, w)$;
      $SecondaryActions(W, w)$;
    end while
    $V \leftarrow V \cup v$;
  end for
  return G

abstract Function `SelectVertices(`*Graph g*`)`**return vertices**

abstract Function `AddEdge(`*Vertex v1,Vertex v2*`)`**return edge**

abstract Procedure `SecondaryActions(`*Vertex v, Vertices V*`)`

the network. In the Erdos-Renyi model, the creation of edges is probabilistic, while the Growing Random model's edge creation given two vertices is definite. Therefore, LinkableGP should be responsible for determining how edge creation should be performed.

In the same iteration where the *AddEdge* function is called, a procedure, *SecondaryAction*, is also invoked with the arguments W, the collection of vertices being iterated, and the vertex, w. The purpose of this procedure is to perform any other actions that happen as a consequence of the edge creation of w and v. In the Forest-Fire graph model, this would be where some neighbours of w not already visited by v (i.e., not already added to W) would be appended to W. As this procedure is not apparent in all graph models found in Chapter 3 and would be dependent on the network being modelled, it is marked abstract.

7.2 GP Language

The functions *SelectVertices*, *AddEdge*, and *SecondaryActions* that were marked as abstract in the abstract class defined in Section 7.1 each required a function set for use in LinkableGP. In order to describe all the functions and terminals used by each abstract function, like functions and terminals are grouped, for convenience, into the

following collections.

- Arithmetic functions
  - Func<Vertex, double> Add(f:Func<Vertex, double>, g:Func<Vertex, double>)
  - Func<Vertex, double> Sub(f:Func<Vertex, double>, g:Func<Vertex, double>)
  - Func<Vertex, double> Mult(f:Func<Vertex, double>, g:Func<Vertex, double>)
  - Func<Vertex, double> Constant(c:double)
  - Func<Vertex, double> Pow(f:Func<Vertex, double>, g:Func<Vertex, double>)
- Vertex property evaluators
  - Func<Vertex, double> InDegree()
  - Func<Vertex, double> OutDegree()
  - Func<Vertex, double> Degree()
  - Func<Vertex, double> Age() (only in Ageing Preferential Attachment)
- Collection providers
  - TabooVertexCollection GetRandomQueue(g:Graph, n:int)
  - TabooVertexCollection GetRouletteQueue(g:Graph, n:int, F:Func¡Vertex,double¿)
  - TabooVertexCollection GetRandomStack(g:Graph, n:int)
  - TabooVertexCollection GetRouletteStack(g:Graph, n:int, F:Func¡Vertex,double¿)
- Edge creation functions
  - Edge CreateEdge(v:Vertex, w:Vertex)
  - Edge CreateEdgeWithProbability(v:Vertex, w:Vertex, prob:double)
- Vertex providers
  - void AddPredecessor(V:TabooVertexCollection, n:int, a:Vertex)
  - void AddSuccessor(V:TabooVertexCollection, n:int, a:Vertex)
  - void AddNeighbour(V:TabooVertexCollection, n:int, a:Vertex)
- Random value providers
  - int GetRandomValue(min:int,max:int)

  - int GetGeometricValue(prob:double)
  - int GetBernoulliValue(prob:double)

- Constant Generators
  - Floating-Point Number Generator
  - Integer Number Generator

The collection of arithmetic functions consists of operations for constructing expression that evaluate a vertex. These expressions are Add, Multiply, Constant, and Pow. Add and Multiple both take two expressions and add/multiply their results. Pow evaluates two expressions and uses the first expression as the base, b, and the second as the exponent, e then evaluates b^e. Constant takes in a value, returning the same value and represents a terminal in the expression tree allowing a value to be converted to an expression. The conversion is important so that Add, Multiply and Pow functions can use constants.

The collection of vertex property evaluators serves as a collection of terminals used in co-operation with the arithmetic functions previously described. The vertex properties consist of local properties of vertices. These are the in- and out-degrees of the given vertex as well as the combined in/out degree of the vertex and its local transitivity. When combined with the arithmetic functions these form an expression tree where a vertex can be evaluated to a value that can be used for roulette selection to select vertices probabilistically to pair with a new vertex.

The roulette selection is one of the two base selection methods in the collection providers, the other being uniform random. The roulette selection uses an expression given to determine a value for each vertex in the graph. It then selects probabilistically one or more of the vertices based on the proportional value resultant of the expression evaluation of each vertices. Whether a roulette selection or a random selection, each has implemented using a stack and a queue. It is also important to note that these collections implement a taboo. A vertex may be added at most once into an instance of a collection even if removed from the collection. The functions, where g is a graph and n is the number of vertices to add initially to the collection, are

Edge creation functions are simply those functions responsible for creating new edges. The collection consists of probabilistic edge creation, and definite edge creation and empty edge creation (handled by the system as no new edge i.e. $E \leftarrow E \cup \emptyset$).

The vertex providers take a vertex collection and a vertex, v and add n vertices related to v by some edge association. These associations consist of predecessor,

Abstract Method	Collections
SelectVertices	random value provider, vertex property provider, vertex collection provider, arithmetic functions, integer constants, and real constants.
AddEdge	edge creation functions, and real constants.
SecondaryActions	vertex providers, random value providers, integer constants, and real constants.

Table 7.1: Accessible Collections of Functions and Terminals by Abstract Method

successor, or neighbour. Since the vertex collections are taboo collections, the n vertices selected will be distinct to the collection thereby avoiding multi-edges.

Finally, the random value providers are responsible for producing values. These providers, in contrast to generated constant, provide random values at runtime of the algorithm. Three providers are implemented serving Integer values from one of the following: an uniform distribution, geometric distribution, or Bernoulli trials. Further to these collections, generated constants in the form of Integers, and Real (double-precision floating-point) numbers are provided to some abstract methods.

Each abstract method is given access to only some collections that have been described. Table 7.1 outlines the collections accessible to each method.

7.3 Fitness Evaluation

It is not possible to directly compare a program to an unknown complex network, the target, no matter what is known of the network. Therefore, it is necessary to compare the program to the target graph using *instance graphs* of the program being evaluated. An instance graph refers to a graph generated by an evolved program. The means of comparison, the fitness evaluation, is established based on previous works [10] in the inference of graph models and modified for the application in directed complex networks.

The fitness of the program is the mean difference in average geodesic path length (Equation 7.1), network average clustering coefficient (Equation 7.2), and degree distributions (both in- and out-degrees) (Equations 7.3 and 7.4) of m instance graphs (X_i) against the target graph (T) (See Chapter 2 for details in computing each value). In Equations 7.3 and 7.4, the function F refers to the cumulative empirical distribution function of the in-degree and G to the cumulative empirical distribution function of the out-degree.

Solution Fitnesses	f_1 rank	f_2 rank	f_3 rank	Sum of Ranks
(3, 2, 2)	3	2	2	7
(1, 3, 4)	2	3	3	8
(0, 0, 1)	**1**	**1**	**1**	**3**

Table 7.2: Example of Sum of Ranks

$$F_1(X,T) = |\frac{1}{m}\sum_{i=1}^{m} l(X_i) - l(T)| \tag{7.1}$$

$$F_2(X,T) = |\frac{1}{m}\sum_{i=1}^{m} C(X_i) - C(T)| \tag{7.2}$$

$$F_3(X,T) = \frac{1}{m}\sum_{i=1}^{m} D_{n,n'}(F_n(X_i), F_{n'}(T)) \tag{7.3}$$

$$F_4(X,T) = \frac{1}{m}\sum_{i=1}^{m} D_{n,n'}(G_n(X_i), G_{n'}(T)) \tag{7.4}$$

Since this problem is multi-objective in nature, a means of comparing solutions qualities is necessary. Means of comparison are established using the Sum-of-Ranks approach to multi-objective optimization. The Sum-of-Ranks algorithm sums the relative rank of each solution (or in this case graph model) in a population over each of the established objectives. For example, Table 7.2 describes a population of three solutions with three objectives where each objective goal is minimization. The ranking value of each solution is presented in the columns of f_1 rank, f_2 rank, and f_3 rank, respectively. The ranks are summed and shown in the column Sum of Ranks. In this example, the 3^{rd} solution is determined the best as it has the lowest summed rank value.

After each new population is constructed, all programs are evaluated over the objectives: F_1, F_2, F_3, and F_4. Then, each program is ranked relative to its population using the previously described method, Sum-of-Ranks. Moreover, now each program is comparable relative to the other programs in its population.

Chapter 8

Experiments – Reproducing Graph Models

In order to determine the utility of the methodology proposed by this thesis, graph models were evolved from target graphs generated by known graph models. For experimentation, the models selected were the Growing Random model (GR), Barabasi-Albert model (BA), the Forest Fire model (FF), and the Ageing Preferential Attachment model (APA) described in Chapter 3. Known models were select as a more thorough validation can be performed as limitless number of graphs could be used for comparisons. The GR model was selected as it is a random model with only a growing property making it the simplest to model. The BA model uses a vertex selection process which requires the LinkableGP to find a formula to represent it. The FF model is the most complex of the four models selected. It has complex patterns for the addition of edges to the graph and makes it the most challenging of the models for LinkableGP. Finally, the APA adds a new property (age) to the modelling and therefore requires the system to discover new processes not seen in other models.

For each model, four target graphs were generated with 100, 250, 500, and 1000 vertices, respectively. LinkableGP was run 30 times for each target graph, resulting in a collection of evolved models. For each target size of each known model, a single evolved model was chosen from the target's respective collection of evolved models. The final evolved model for each target was selected by way of a Sum-of-Ranks test on the collection of evolved models. The evolved model ranked the highest was selected.

A number of graphs (1000) were generated using each final evolved with the same number of vertices as its respective target graph and compared to the target graph. These comparisons provide a means of verifying that the final fitness of the graph was not a chance occurrence.

The aim of validating the final evolved model is to determine whether it reproduces the graph model that produced the target graph used. Logically, there exist many ways that validation schemes might be designed, some more rigorous than other. One such way is visual examination of the graph model algorithms produced and the graphs it produces. However, visual inspection is not a very formal approach to validation but does reveal some information about the model and the graphs it produces. Therefore, only a brief visual comparison for each set of experiments will be provided. A complete set of algorithms from the evolved models is found in Appendix A. Also, side by side visualization of target graphs and graphs produced by each evolved graph model are found in Appendix B.

The approach of this thesis in the validation of models is to compare the structure of the graphs produced by evolved model to those of the targeted model statistically. In order to do so, each final evolved model was used to produce 1000 graphs of each size (100, 250, 500, 1000 vertices). The same was done with the targeted graph model. The evolved graph model and the targeted graph model were then compared over each same-vertices-sized set. Comparison was performed by way of t-tests to compare the average geodesic path lengths and the clustering coefficients and using KS tests for the in- and out-degree distributions. The sample sizes were selected as they provided a sample power of 0.971. The effect size for Welch t-tests was 0.2 with a significance level of 0.01. The effect size for chi-squared tests was determined to be 0.1 with a significance level of 0.01. Furthermore, where a t-test was employed, each sample was tested using the Shapiro-Wilk test and found to be normally distributed.

In the remainder of this chapter, the results of the experiments on each model will be presented. In Section 8.1, the setup for experimentation will be described. Thereafter, Sections 8.2, 8.3, and 8.4 will present the results for experimentation in reproducing the GR, BA, and FF models. Finally, Section 8.6 will provide some discussion on the overall results of experiments.

8.1 Experimental Setup

For all experiments performed, LinkableGP was configured with the parameters found in Table 8.1. The system parameters for LinkableGP used in experimentation were established empirically during small-scale experiments in the earlier stages of experimentation. The resulting evolved models were named based on their target model and target graph size as listed in Table 8.2.

Parameter	Value
Population	50
Generations	30
Crossover Rate	0.9
Mutation Rate(m_1, m_2)	0.1
Tournament Size	3
Elitism	0.1
Initial Chromosome Length Ranges	
SelectVertex	[1, 15]
AddEdge	[1, 5]
SecondaryAction	[1, 25]

Table 8.1: LinkableGP Parameters

Each target graph was generated using igraph[1]. The settings used to construct each target graph within a model only differed by number of vertices. Table 8.2 lists the settings used for each model.[2] The m values used in the settings were all 1 and was chosen as to remove the responsibility of the LinkableGP from having to determine the number of initial vertex candidates. The remaining settings were chosen based purely upon the example uses provided in the igraph documentation. Beyond variation of the graph size, no experimentation with graph model parameters was performed.

8.2 Growing Random Model

The Growing Random model constructs graphs by adding vertices at each time step and forming an uniform and randomly chosen vertex with which to form an edge. Of the models experimented with, it is the simplest. However, there are certain constraints that evolved models must follow if they are to be considered feasible given the parameters used to construct the target graphs (See Section 8.1 for parameter details).

- Each vertex must have exactly one out-bound edge except the first vertex added
- A graph produced may not contain any transitivity

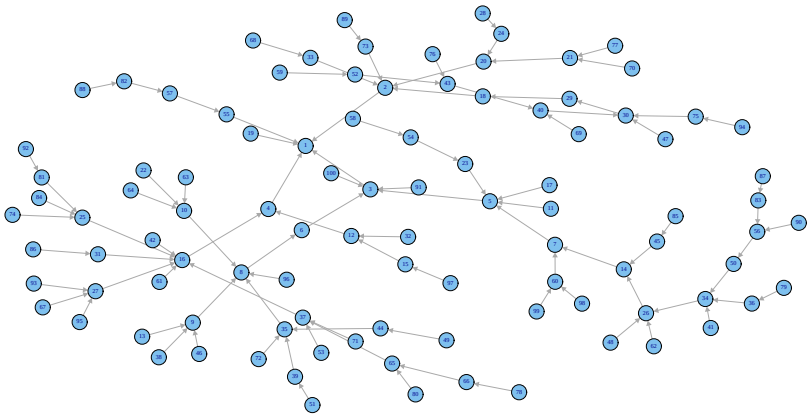
[1]igraph is a library with an implementation in R for applications with graphs. Information about the library can be found at http://www.igraph.org

[2]For more details on the implementation and the application of the parameters for each of this graph models, see Chapter 3.

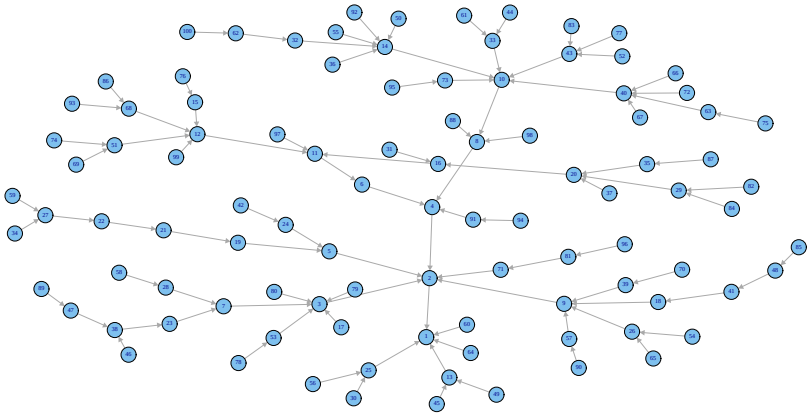
Experiment Id	Model	Parameters
GR-100	Growing Random Graph Model	$n = 100$, $m = 1$
GR-250	Growing Random Graph Model	$n = 250$, $m = 1$
GR-500	Growing Random Graph Model	$n = 500$, $m = 1$
GR-1000	Growing Random Graph Model	$n = 1000$, $m = 1$
BA-100	Barabasi-Albert Model	$n = 100$, $m = 1$, $\alpha = 1$, $a = 1$, $c = 1$
BA-250	Barabasi-Albert Model	$n = 250$, $m = 1$, $\alpha = 1$, $a = 1$, $c = 1$
BA-500	Barabasi-Albert Model	$n = 500$, $m = 1$, $\alpha = 1$, $a = 1$, $c = 1$
BA-1000	Barabasi-Albert Model	$n = 1000$, $m = 1$, $\alpha = 1$, $a = 1$, $c = 1$
FF-100	Forest Fire Model	$n = 100$, $m = 1$, $p = 0.37$, $r = \frac{0.32}{0.37}$
FF-250	Forest Fire Model	$n = 250$, $m = 1$, $p = 0.37$, $r = \frac{0.32}{0.37}$
FF-500	Forest Fire Model	$n = 500$, $m = 1$, $p = 0.37$, $r = \frac{0.32}{0.37}$
FF-1000	Forest Fire Model	$n = 1000$, $m = 1$, $p = 0.37$, $r = \frac{0.32}{0.37}$
APA-100	Ageing Preferential Attachment Model	$n = 100$, $m = 1$, $\alpha = 1$, $a = 1$, $c = 1$, $\beta = -1$, $b = 1$, $d = 0$
APA-250	Ageing Preferential Attachment Model	$n = 250$, $m = 1$, $\alpha = 1$, $a = 1$, $c = 1$, $\beta = -1$, $b = 1$, $d = 0$
APA-500	Ageing Preferential Attachment Model	$n = 500$, $m = 1$, $\alpha = 1$, $a = 1$, $c = 1$, $\beta = -1$, $b = 1$, $d = 0$
APA-1000	Ageing Preferential Attachment Model	$n = 1000$, $m = 1$, $\alpha = 1$, $a = 1$, $c = 1$, $\beta = -1$, $b = 1$, $d = 0$

Table 8.2: Experiments

One of the selected best-evolved models, the **GR-100**, provides an initial perspective on the quality of solutions that will be seen in this section. Figure 8.1 shows the target graph and a graph produced from the **GR-100** model. It is observed that both have similar types of spanning trees and have a number of vertices that have 3 to five in-degrees that serves as a sort-of hub. Examination of Algorithm 12 representing the **GR-500** model shows that a great deal of benign code was produced but, importantly, variable b in SelectVertices generates a random selection of vertex and then an edge is created by AddEdge which is similar to the processes of GR.



(a) Growing Random Model with 100 Vertices



(b) Evolved Growing Random Model with 100 Vertices

Figure 8.1: Graphs from the GR model and the **GR-100** model

In examining the results of **GR-100**, **GR-250**, **GR-500**, and **GR-1000** found in Table 8.3, based on fitness objectives F_2, and F_4, it is found that the constraints are satisfied. If any of the models produced graphs which violated the out-bound degree constraint, F_4 would show a D-Value that sometimes deviated from 0.

In the case of transitivity, if a model sometimes produced transitivity there would be some deviation found in the difference in clustering coefficients (F_2).

Algorithm 12: GR-500

Function `GenerateModel(`*int t*`)`**return graph**
  $G \leftarrow InitialiseGraph()$;
  for $i \leftarrow 1$ **to** t **do**
    $v \leftarrow Vertex()$;
    $W \leftarrow SelectVertices(G)$;
    while $|W| > 0$ **do**
      $w \leftarrow Next(W)$;
      $E \leftarrow E \cup AddEdge(v, w)$;
      $SecondaryActions(W, w)$;
    end while
    $V \leftarrow V \cup v$;
  end for
  return G

override Function `SelectVertices(`*Graph g*`)`**return vertices**
  $a \leftarrow GeometricValue(0.402696)$;
  $b \leftarrow GetRandomQueue(g)$;
  $c \leftarrow GetLocalTransitivity()$;
  return b
override Function `AddEdge(`*Vertex v1,Vertex v2*`)`**return edge**
  $a \leftarrow CreateEdge(v1, v2)$;
  return a
override Procedure `SecondaryActions(`*Vertex v, Vertices V*`)`
  $a \leftarrow GeometricValue(0.886134)$;
  $a \leftarrow RandomValue(a, a)$;
  $a \leftarrow BernoulliValue(0.93379)$;

	F_1		F_2		F_3		F_4	
Experiment	$\bar{x}$	s	$\bar{x}$	s	$\bar{x}$	s	$\bar{x}$	s
GR-100	0.07	0.342	0	0	0.037	0.014	0	0
GR-250	0.702	0.338	0	0	0.032	0.011	0	0
GR-500	0.339	0.351	0	0	0.024	0.009	0	0
GR-1000	0.08	0.344	0	0	0.016	0.006	0	0

Table 8.3: Final evolved GR models results comparing 1000 graphs produce by the best-evolved model vs. its target graph

The results in Table 8.3, with respect to F_3, presents the average and standard deviation of the D-Value of the in-degree distribution of 1000 graphs from the evolved model versus its respective target graph. The mean results found in Table 8.3 are fount to be more than 3 standard deviations less than the relevant critical values as given in Table 8.4. This observation shows that the in-degree distributions produced by the evolved models are similar to those found in the target graph.

Interpretation of the results for F_1 is difficult to interpret via comparison to the model. The values are optimal relative to the other evolved models not selected to produce average path lengths similar to those found in the target graph. However, there is no means of determining if these differences are fit relative to the target graph and therefore the target model. Instead, it is necessary to defer to the validation of the evolved models to determine the how well the average geodesic path lengths compare to the observed average geodesic path lengths of the GR.

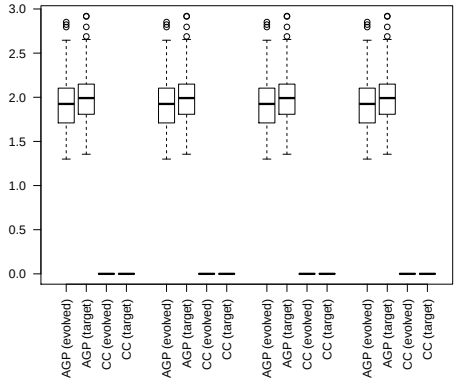
In Figure 8.2, the results of the comparison between the best-evolved model for each GR target is shown for their distribution of average geodesic path length and network average clustering coefficient. This figure shows that all evolved models produce the clustering coefficients expected. Comparing the network average clustering coefficient between the evolved models the growing random model the seem to be similar except the **GR-1000**. The path lengths observed in this evolved model exceed those observed in the actual growing random model(See Subfigure 8.2d).

$n_1 \backslash n_2$	100	250	500	1000
100	0.192	0.161	0.149	0.143
250	0.161	0.122	0.105	0.096
500	0.149	0.105	0.086	0.074
1000	0.143	0.096	0.074	0.061

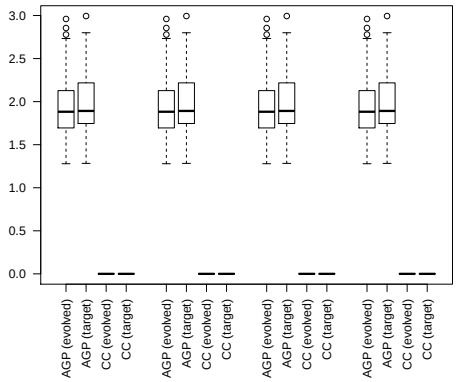
note: the critical values with a significance of 0.05 are computed by

$$D_{crit} = 1.36\sqrt{\frac{n_1+n_2}{n_1 n_2}}$$

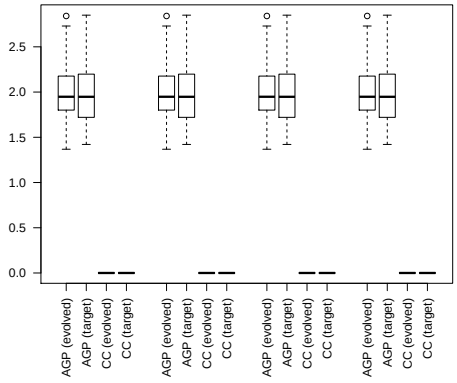
Table 8.4: D_{crit} Values



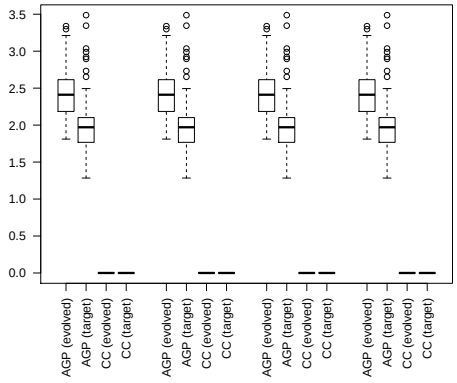
(a) **GR-100**



(b) **GR-250**



(c) **GR-500**



(d) **GR-1000**

Figure 8.2: Box plots of the average geodesic path length (AGP) and network average clustering coefficient (CC) for **GR** experiments (evolved) versus the growing random model(target). In each chart the four groups of box plots represent, in order from left to right, graphs with 100 vertices, 250 vertices, 500 vertices, and 1000 vertices

A t-test comparing the samples of average geodesic path lengths from the evolved models and the growing random model was performed, and the results are found in Table 8.5. The results confirm the previous observations of Figure 8.2, that with a 99% confidence the average geodesic path lengths of the **GR-100**, **GR-250**, and **GR-500** evolved models are similar to those observed in the growing random model.

The final area properties to inspect in validation for the evolved models of the growing random model are the in- and out-degree distributions. To compare this

Model \Size	100	250	500	1000
GR-100	1.285	0.322	0.196	0.903
GR-250	0.609	1.287	0.358	0.454
GR-500	0.366	0.125	1.622	0.955
GR-1000	8.486	9.59	7.536	9.524

Table 8.5: T-Values of the comparisons between GR evolved models and the growing random model. The critical T-Statistic for a 0.01 significance test with 1000 degrees of freedom is 2.326.

1000 graph pairs were compared at each vertex size for each model. In 100% of cases all four evolved models were able to replicate the degree distributions observed in the growing random model for graphs with 100, 250, 500, 1000 vertices.

8.3 Barabasi-Albert Model

The Barabasi-Albert Model is graph model that models growing preferential attachment networks. A highlight of these networks is that the network is scale-free meaning that the in-degree of the network follows a power law distribution, a common property of social networks [69].

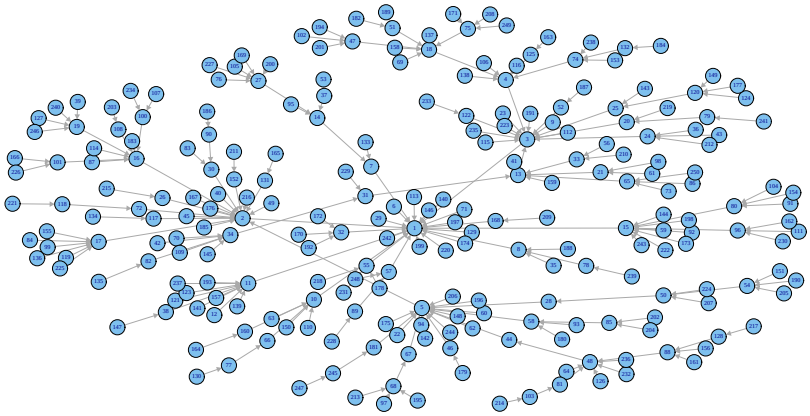
In the experiments conducted with this model, there are a few constraints which are resultant of the parameters used in constructing the target graph. As a result, evolved models that do not observe these constraints can be said to be invalid. For the target graphs in this section those constraints are

- Each vertex must have exactly one out-bound edge except the first vertex added
- A graph produced may not contain any transitivity

Comparing the graphs produced by the BA model and the **BA-250** model found in Figure 8.3 shows that the graph of the evolved model produces a few high in-degree vertices like the graph of the BA model. The figure also shows that the spanning trees expected in a BA graph are also in the evolve model's graph. The algorithm of the **BA-500** shows that the equation expected to compute the probabilities of selection are being discovered. **BA-500** pseudo-code is found in B as Algorithm 18.

Examination of Table 8.6 confirms that all evolved model satisfied the constraints of the out-degree and transitivity. It is apparent that the evolved models did not show deviation from the constraints because the values of F_2, F_3 are zero for their mean and standard deviation. The lack of deviation means that the models all showed in

(a) Barabasi-Albert Model with 250 Vertices



(b) Evolved Barabasi-Albert Model with 250 Vertices

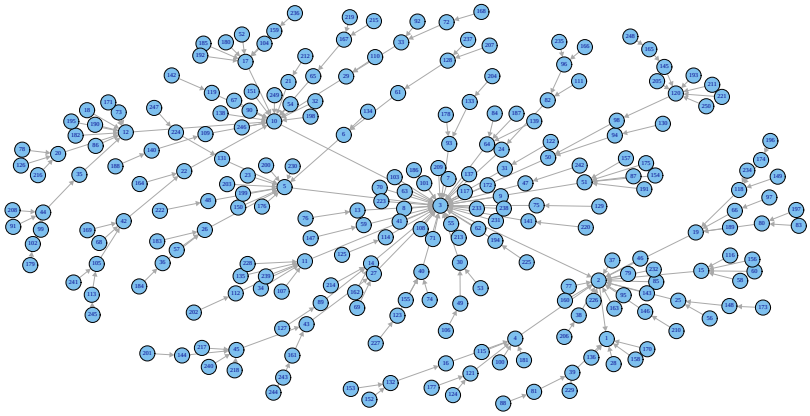


Figure 8.3: Graphs from the BA model and the **BA-250** model

	F_1		F_2		F_3		F_4	
Experiment	$\bar{x}$	s	$\bar{x}$	s	$\bar{x}$	s	$\bar{x}$	s
BA-100	0.046	0.338	0	0	0.08	0.025	0	0
BA-250	0.245	0.284	0	0	0.042	0.017	0	0
BA-500	0.311	0.314	0	0	0.018	0.007	0	0
BA-1000	0.314	0.326	0	0	0.014	0.005	0	0

Table 8.6: Final evolved BA models results comparing 1000 graphs produce by the evolved model vs. its target graph

their 1000 tests that the clustering coefficients and the out-degree distributions were identical to the target.

For the fitness objective, F_3, Table 8.6 provides the mean and standard deviation of the 1000 comparisons to the target graph. Examination of these values demonstrated that even +3 standard deviations are below the critical D-Values for the KS test (see table 8.4). Thus, the evolved model always showed similar in-degree distributions to the target graph.

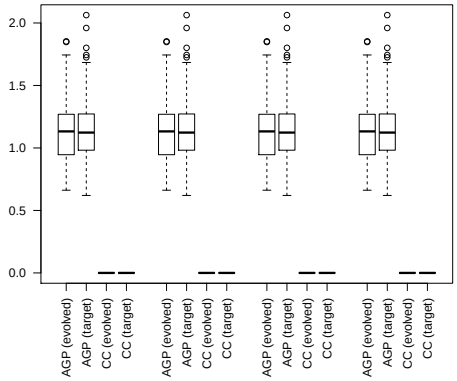
As was the case in Section 8.2, comparing the average geodesic path lengths of the evolved model to the target graph reveals little about the quality of the model. However, the validation using the target model will help to determine the quality of the average geodesic path lengths. Comparison of the properties of the average geodesic path lengths of the actual Barabasi-Albert model and the evolved models will provide a much more meaningful measure of performance.

Figure 8.4 shows the observations during validation of the evolved models targeting the Barabasi-Albert model for the average geodesic path length and the clustering coefficient. The box plots show that the distribution of the observed average geodesic path lengths of the evolved models compared to those of the Barabasi-Albert model are quite similar. Validation by way of the t-test reveals that the mean observations are not statistically different. Table 8.7 shows the results of this test.

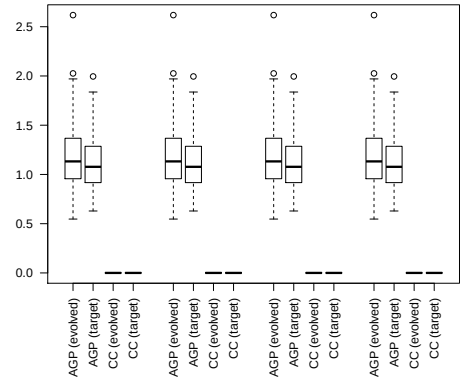
As was observed in the initial testing of the evolved model versus the target graph, validation shows that the constraints with respect to out-degree and clustering coefficients are respected. The clustering coefficients of all four evolved graph models were always 0. The out-degree distributions of the evolved models were, also, always identical to the graphs produced by the Barabasi-Albert model. The KS-test of the in-degree distributions of the evolved model versus those of the Barabasi-Albert model showed that 100% of the graphs tested had statistically similar distributions.

Model \Size	100	250	500	1000
BA-100	0.507	0.538	0.245	0.038
BA-250	1.951	0.993	0.322	0.123
BA-500	1.749	0.429	1.914	1.14
BA-1000	1.898	1.617	0.653	0.029

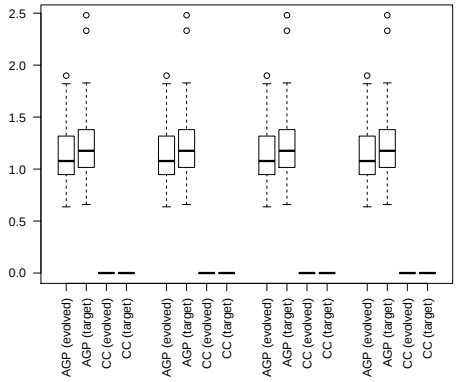
Table 8.7: T-Values of the comparisons between BA evolved models and the Barabasi-Albert model. The critical T-Statistic for a 0.01 significance test with 1000 degrees of freedom is 2.326.



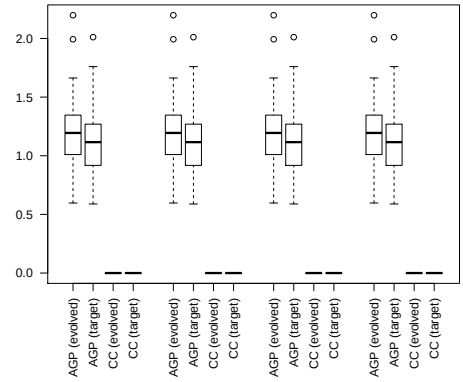
(a) **BA-100**



(b) **BA-250**



(c) **BA-500**



(d) **BA-1000**

Figure 8.4: Box plots of the average geodesic path length (AGP) and network average clustering coefficient (CC) for **BA** experiments (evolved) versus the growing random model(target). In each chart the four groups of box plots represent, in order from left to right, graphs with 100 vertices, 250 vertices, 500 vertices, and 1000 vertices

8.4 Forest Fire Model

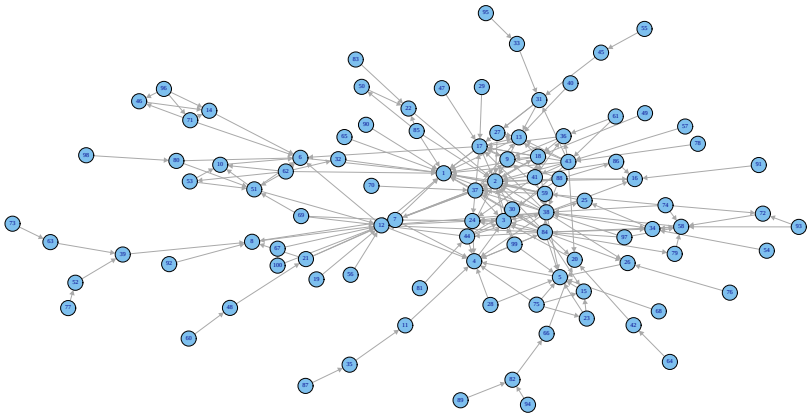
The Forest Fire Model presents a new set of challenges in modelling that were not experienced with the Barabasi-Albert and growing random models. In the Forest Fire model, there is a clustering coefficient and a non-regular out-degree. This model presents the greatest challenge in modelling because the processes in the FF model exhibit recursion, use floating point constants that are very sensitive, and there are multiple processes that select vertices for attachment. Algorithm 13 shows one of the most successful and simplest solutions found. This algorithm illustrates that LinkableGP had to evolve a number of constants in order to be able to construct a good model.

Algorithm 13: FF-500

```
Function GenerateModel(int t)return graph
    G ← InitialiseGraph();
    for i ← 1 to t do
        v ← Vertex();
        W ← SelectVertices(G);
        while |W| > 0 do
            w ← Next(W);
            E ← E ∪ AddEdge(v, w);
            SecondaryActions(W, w);
        end while
        V ← V ∪ v;
    end for
    return G

override Function SelectVertices(Graph g)return vertices
    a ← BernoulliValue(0.02819);
    a ← GeometricValue(0.941932);
    a ← BernoulliValue(0.347325);
    b ← GetRandomStack(g);
    c ← GetRandomQueue(g);
    return c
override Function AddEdge(Vertex v1,Vertex v2)return edge
    a ← CreateEdge(v1, v2);
    return a
override Procedure SecondaryActions(Vertex v, Vertices V)
    a ← GeometricValue(0.63182);
    a ← GeometricValue(0.681023);
    AddSuccessor(V, a, v);
    AddPredecessors(V, b, v);
```

(a) Forest Fire Model with 100 Vertices



(b) Evolved Forest Fire Model with 100 Vertices

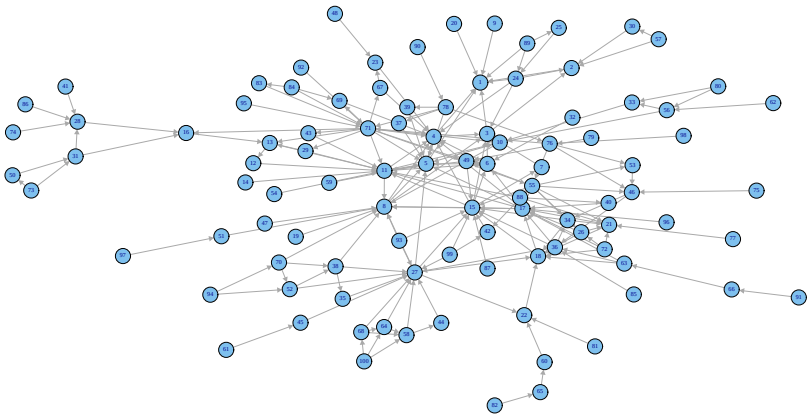


Figure 8.5: Graphs from the FF model and the **FF-100** model

Despite some successes in evolving models that replicated the FF model, not all solutions were successful. Figure 8.5 shows one such example. The graph produced by the **FF-100** shown in the figure has a noticeable difference from the graph produced by the FF model. The graph from the **FF-100** does not have the same type of dense connection sets as in the graph from the FF model.

Examination of the evolved models compared to the target graph reveals the increased difficulty in modelling the Forest Fire. Table 8.8 shows the results of the comparison between the evolved models and the target graph. This table shows that the models display a difference in average geodesic path lengths to the target that is similar to those observed in Tables 8.3 and 8.6 which is a good initial indicator. However, a more thorough examination of the average geodesic path lengths against that of the actual model is required. Likewise, the difference in clustering coefficient values seem good but comparison against the target graph does not add much value to the model.

	F_1		F_2		F_3		F_4	
Experiment	$\bar{x}$	s	$\bar{x}$	s	$\bar{x}$	s	$\bar{x}$	s
FF-100	0.182	0.165	0.055	0.054	0.079	0.04	0.081	0.035
FF-250	0.127	0.115	0.082	0.053	0.191	0.069	0.118	0.02
FF-500	0.123	0.123	0.004	0.02	0.045	0.018	0.047	0.02
FF-1000	0.138	0.123	0.001	0.016	0.04	0.018	0.037	0.015

Table 8.8: Final evolved FF models results comparing 1000 graphs produced by the evolved model vs. its target graph

Fitness objectives F_3 and F_4, in Table 8.8 do, however, provide some valuable insights into the model. The mean value of the objectives for each model are below the relevant critical D-values for the KS test found in Table 8.4. Even one standard deviation is still below the critical D-value. However, unlike the results of the previous models, a computed value of +3 standard deviations from the mean values of F_3 and F_4 for each model shows that the values exceed the D-critical value. This result means that sometimes the evolved models can produce degree distributions that are not similar those observed in the target graph.

In order to validate the ability of the evolved models to replicate the degree distributions expected of the target model, an experiment was designed. In the experiment, a graph from an FF evolved model was compared to a graph from the FF model using the KS test. The KS test was perform 1000 times each with a new graph from the evolved model and the FF model. For each test performed a count of how many times the KS test found the graphs to have similar degree distributions was recorded. The same experiment was performed on where both graphs in the KS test came from the FF model to provide a baseline of how often the model generated statistically similar degree distributions. The proportion of tests that showed a similar degree distribution is found in Table 8.9.

A Pearson's chi-squared test [73] was performed to determine if the number of occurrences where the evolved model had a similar degree distribution was at least as much as when the graphs both came from the FF model. The Pearson's chi-square test examines categorical variables to determine how likely the difference observed between groups occurred by chance. Each one-sided Pearson's chi-squared test performed for an evolved model compared the differences observed in the frequency of KS test that should statistically similar degree distributions versus the same for the target model. The results shown in brackets in Table 8.9 reveals that the **FF-100** and **FF-250** models often reject the null hypothesis (p-value less than 0.01). The evolved model differences did not occur by chance and were found to be less than expected by the

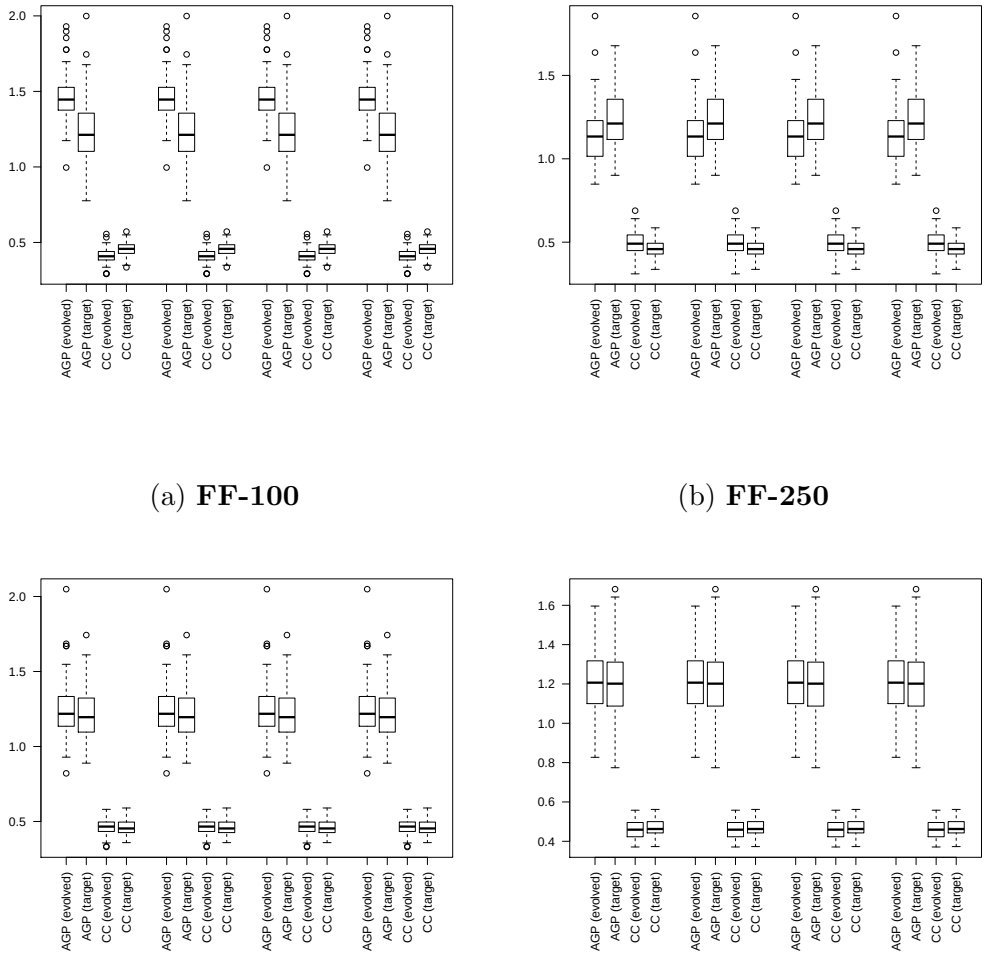
Model \Size	100	250	500	1000
In-Degree				
FF-100	0.99 (1)	0.95 (0.963)	0.8 (0)	0.48 (0)
FF-250	0.83 (0)	0.48 (0)	0.14 (0)	0 (0)
FF-500	1 (1)	0.98 (1)	0.96 (1)	0.96 (1)
FF-1000	0.99 (1)	0.95 (0.963)	0.999 (1)	0.89 (1)
Target Model	0.96	0.93	0.87	0.81
Out-Degree				
FF-100	1 (1)	0.97 (0.863)	0.92 (0.004)	0.79 (0)
FF-250	0.95 (0.015)	0.69 (0)	0.27 (0)	0 (0)
FF-500	0.99 (0.999)	0.99 (1)	0.96 (0.834)	1 (1)
FF-1000	0.99 (0.999)	1 (1)	0.95 (0.5)	0.99 (1)
Target Model	0.97	0.96	0.95	0.93

Table 8.9: Results of 1000 KS-tests comparing degree distributions of the evolved models and the forest fire model. The values in this table are the proportion of tests that showed that each graph pair had the similar degree distributions. Values in brackets are the computed p-value of a one-sided Pearson's chi-square test which tested the null hypothesis that the evolved model passed the KS test at least as often as the target model.

target model. However, the **FF-500** and **FF-1000** did show no statistically significant differences and slightly more often was able to match the degree distributions more steadily the target model compared to itself.

Validation of the models by way of the average geodesic path lengths and the clustering coefficients, provides further evidence that the forest fire model has not been successfully modelled by the **FF-100** and **FF-250** evolved models. Figure 8.6 provides box plots of the average geodesic path lengths and clustering coefficients observed during the validation. The figure shows just how significant the difference in the average path lengths and clustering coefficients are for the **FF-100** and **FF-250**.

However, Figure 8.6 also shows that the **FF-500** and **FF-1000** evolved have similar distributions for the average path lengths and clustering coefficients to the forest fire model. T-Tests performed on these distributions confirm that the means of the **FF-500** and **FF-1000** evolved models average geodesic path lengths and clustering coefficients are not statistically different.



(a) **FF-100** (b) **FF-250**

(c) **FF-500** (d) **FF-1000**

Figure 8.6: Box plots of the average geodesic path length (AGP) and network average clustering coefficient (CC) for **FF** experiments (evolved) versus the growing random model(target)

note: In each chart the four groups of box plots represent, in order from left to right, graphs with 100 vertices, 250 vertices, 500 vertices, and 1000 vertices

8.5 Ageing Preferential Attachment Model

The preferential attachment is closely related to the Barabasi-Albert model however it differs as it considers the age of vertices. The vertex property makes this model very different than the previously examined models of this chapter. In order to

successfully model the network produced by the ageing preferential attachment model, it is necessary to add new constructs to the language and the abstract class.

The first construct that needs to be added is in the abstract class. Algorithm 11 is amended to track time in the graph now and provide vertices with a birth value. The change is reflected in Algorithm 14. In order for a model to utilize this information, the language must also be updated. Therefore, the Vertex property evaluators, is amended to include a function that provides the age of a vertex. The age of a vertex is computed as the difference in the current time set for the graph and the birth value of the vertex.

Algorithm 14: Modified Abstract Graph Model for Complex Networks

```
Function GenerateModel(int t)return graph
    G ← InitialiseGraph();
    for i ← 1 to t do
        v ← Vertex();
        SetBirth(v, i);
        SetTime(g, i);
        W ← SelectVertices(G);
        while |W| > 0 do
            w ← Next(W);
            E ← E ∪ AddEdge(v, w);
            SecondaryActions(W, w);
        end while
        V ← V ∪ v;
    end for
    return G

abstract Function SelectVertices(Graph g)return vertices
abstract Function AddEdge(Vertex v1,Vertex v2)return edge
abstract Procedure SecondaryActions(Vertex v, Vertices V)
```

The target graphs produce by the ageing preferential attachment model used the same settings as the Barabasi-Albert target graphs (see intro of this chapter) and an age preference exponent of -1. The value of the age preference exponent means that over time new vertices are less likely to attach to older vertices. Also, the target graphs produce have a regular out-degree and no clustering coefficients. Thus, evolved models should also exhibit these characteristics.

Immediately, it is observed in Table 8.10 that the expected out-degree and clustering coefficients of the targeted models are reproduced. If there were evidence that an evolved model produced a clustering coefficient or had an out-degree greater than one

	F_1		F_2		F_3		F_4	
Experiment	$\bar{x}$	s	$\bar{x}$	s	$\bar{x}$	s	$\bar{x}$	s
APA-100	1.38	1.048	0	0	0.23	0.007	0	0
APA-250	1.701	1.198	0	0	0.022	0.008	0	0
APA-500	1.461	0.093	0	0	0.022	0.008	0	0
APA-1000	1.56	0.902	0	0	0.023	0.009	0	0

Table 8.10: Final evolved APA models results comparing 1000 graphs produce by the evolved model vs. its target graph

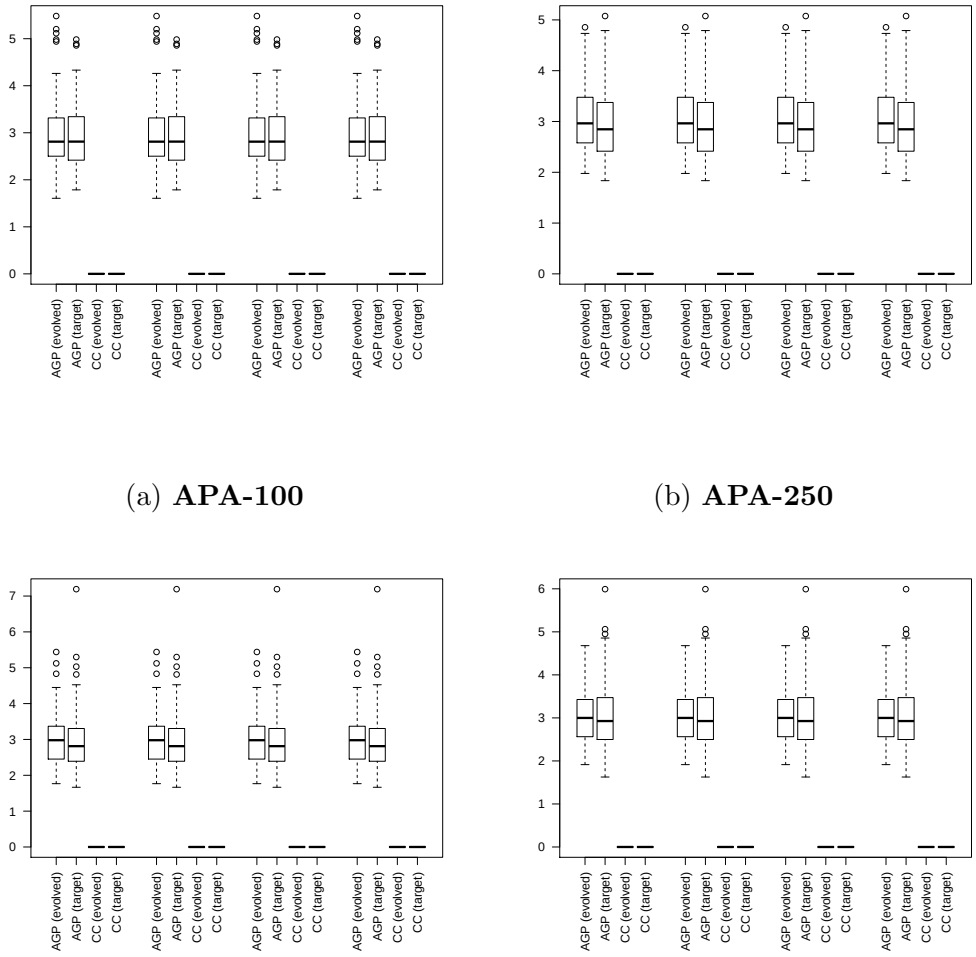
Model \Size	100	250	500	1000
APA-100	0.374	0.981	0.223	0.431
APA-250	1.538	0.454	0.341	1.035
APA-500	0.37	1.048	0.782	1.125
APA-1000	0.265	1.836	0.549	1.183

Table 8.11: T-Values of the comparisons between APA evolved models and the growing random model. The critical T-Statistic for a 0.01 significance test with 1000 degrees of freedom is 2.326.

there would be evidence of a mean or standard deviation different than 0 in columns F_2 and F_4 of Table 8.10.

Further examination of Table 8.10 shows that F_3, the D-value computed for the difference in in-degree distributions, for all evolved models is fit to the target. Adding three standard deviations to the mean over each model is still well below the relevant critical D-value. However, notably, the difference in average geodesic path lengths of these evolved models to the target graph is much greater than those reported for evolved of the GR, BA, and FF.

In the validation, it is discovered that the distribution of average geodesic path lengths for the ageing preferential attachment model are much wider than the previous models. Thus, examining the box plots found in Figure 8.7 reveals that the average path length differences may be acceptable. Table 8.11 shows the results of the t-tests performed on the observed average geodesic path lengths of each evolved model and the target model. The analysis confirms that there are no statistically significant differences in the average geodesic path lengths of the models compared to those of the target model.



(a) **APA-100** (b) **APA-250**

(c) **APA-500** (d) **APA-1000**

Figure 8.7: Box plots of the average geodesic path length (AGP) and network average clustering coefficient (CC) for **APA** experiments (evolved) versus the growing random model(target)

note: In each chart the four groups of box plots represent, in order from left to right, graphs with 100 vertices, 250 vertices, 500 vertices, and 1000 vertices

Finally, the KS-tests performed for validation of the in-degree distributions of the evolved models showed no statistical difference in the in-degree distributions in any of the four evolved model compared to the target model. Thus, with respect to the fitness measures employed, it is concluded that the four models are valid replications of the ageing preferential attachment model.

8.6 Overall Performance

The results discussed in this chapter have made no remark of the performance of the average runs produced by the methods of this thesis. In this section, the general perform of the GP system will be analysed. First, the average final result of trial runs will be examined. Then, the performance by generation of the system will be discussed.

Fitness Objective	Mean	Std. Dev	Median	3^{rd} Quantile	Skewness	Kurtosis
GR-100						
F_1	6.532	24.853	0.001	0.002	3.474	13.070
F_2	0.000	0.000	0.000	0.000	NA	NA
F_3	0.080	0.155	0.020	0.002	2.124	5.878
F_4	0.045	0.131	0.000	0.000	2.646	8.049
GR-250						
F_1	0.030	0.071	0.007	0.016	1.145	20.502
F_2	0.011	0.059	0.000	0.000	5.199	28.034
F_3	0.041	0.098	0.012	0.016	3.589	15.240
F_4	0.036	0.114	0.000	0.000	3.038	10.650
GR-500						
F_1	82.817	1888.323	0.007	0.039	1.789	4.200
F_2	0.006	0.033	0.000	0.000	5.199	28.034
F_3	0.073	0.131	0.010	0.013	1.962	5.894
F_4	0.037	0.093	0.000	0.000	3.480	15.83
GR-1000						
F_1	232.560	428.636	0.0167	0.720	1.261	2.590
F_2	0.000	0.001	0.000	0.000	4.967	26.305
F_3	0.071	0.104	0.007	0.133	1.108	2.502
F_4	0.034	0.077	0.000	0.007	2.762	10.818

Table 8.12: Overall Final Fitness Results of **GR-100**, **GR-250**, **GR-500**, and **GR-1000**

In order to have some indication of the average system performance, descriptive statistics of the final fitness values of each experiment were computed. The descriptive statistics are reported in Tables 8.12, 8.13, and 8.14. In many of the cases, the mean performance of LinkableGP at modelling was poor and had a very high standard deviation. However, the skewness of the results indicates that the average performance of LinkableGP was heavily and positively skewed. Also, the kurtosis of the results show that the majority of the results were distributed about the median. Thus, the median fitness values are a much better indication of the average performance. Detailed

evidence of the skewed distributions and the tendency about the median is provided in Appendix C via charts of the empirical cumulative distributions of the results.

Fitness Objective	Mean	Std. Dev	Median	3^{rd} Quantile	Skewness	Kurtosis
BA-100						
F_1	6.609	25.047	0.004	0.044	3.474	13.071
F_2	0.000	0.000	0.000	0.000	NA	NA
F_3	0.092	0.070	0.050	0.147	1.164	3.640
F_4	0.021	0.082	0.000	0.000	3.625	14.510
BA-250						
F_1	16.701	63.057	0.025	0.370	3.474	13.071
F_2	0.030	0.079	0.000	0.000	2.268	6.342
F_3	0.086	0.006	0.078	0.156	0.139	1.228
F_4	0.130	0.213	0.000	0.291	1.152	2.599
BA-500						
F_1	55.317	153.011	0.0540	0.135	2.517	7.519
F_2	0.005	0.028	0.000	0.000	5.199	28.034
F_3	0.112	0.123	0.137	0.146	2.925	14.460
F_4	0.046	0.126	0.000	0.000	2.575	8.045
BA-1000						
F_1	64.879	249.191	0.506	0.898	3.545	13.569
F_2	0.092	0.195	0.000	0.011	2.182	6.96
F_3	0.109	0.065	0.143	0.153	-0.800	1.857
F_4	0.154	0.245	0.000	0.300	1.620	5.388

Table 8.13: Overall Final Fitness Results of **BA-100**, **BA-250**, **BA-500**, and **BA-1000**

Tables 8.12, 8.13, and 8.14 show that the median performance is much better. In most cases the median, and often the 3^{rd} quantile, final fitness values were good enough to produce fit models, based on observations of the best performing evolved models. The only exceptions to the number of quality solutions was experiments with the Forest Fire model. The routine quality of the results shows that the system can consistently work towards the optimization of graph models given a single target graph.

When working with evolutionary systems such as GP, it is interesting to examine if the system is improving over time. In order to determine how well LinkableGP improved solutions over time, convergence plots were produced. These plots examine the average fitness values of the best solutions of each trial run over each generation. Figure 8.8 provides an example of the typical convergence for an experiment. The figure illustrates that during the first ten generations, solutions improved a great deal

Fitness Objective	Mean	Std. Dev	Median	3^{rd} Quantile	Skewness	Kurtosis
FF-100						
F_1	0.0482	0.143	0.010	0.032	4.856	25.632
F_2	0.33	0.062	0.003	0.012	1.722	4.265
F_3	0.186	0.219	0.095	0.188	1.725	4.854
F_4	0.234	0.239	0.160	0.328	1.267	3.704
FF-250						
F_1	0.549	2.820	0.011	0.032	5.195	28.002
F_2	0.019	0.044	0.005	0.009	3.799	17.792
F_3	0.242	0.210	0.220	0.356	1.072	3.489
F_4	0.221	0.198	0.212	0.292	1.318	4.656
FF-500						
F_1	3.329	17.554	0.011	0.094	5.197	28.018
F_2	0.094	0.122	0.038	0.153	1.289	3.561
F_3	0.208	0.203	0.118	0.328	1.805	6.035
F_4	0.342	0.254	0.274	0.448	0.775	2.326
FF-1000						
F_1	0.192	0.358	0.075	0.197	3.755	17.332
F_2	0.146	0.159	0.089	0.261	0.732	1.922
F_3	0.248	0.198	0.236	0.315	1.144	4.217
F_4	0.355	0.252	0.340	0.471	0.821	2.869

Table 8.14: Overall Final Fitness Results of **FF-100**, **FF-250**, **FF-500**, and **FF-1000**

and then came to a slow convergence. However, sometimes the fitness objectives F_1 and F_4 will diverge late in the run in exchange for improved values of F_2 and F_3. Figure 8.9 provides an example of this phenomenon. Often, individual runs that experience the divergence of F_1 and F_4 resulted in infeasible models. The convergence plots for each experiment is available in Appendix D.

8.7 Summary

In this chapter experiments were conducted with the proposed methodology to replicate several known graph models: the growing random, Barabasi-Albert, Forest Fire, and Ageing Preferential Attachment. In each experiment, evolved graph models were compared to a target graph generated by the relevant known graph models by way of 1000 graphs from the evolved model compared to the target graph. In every case it was demonstrated, where applicable, that the evolved models matched the target graph well. However, some measures were not so easily compared such as the cluster-

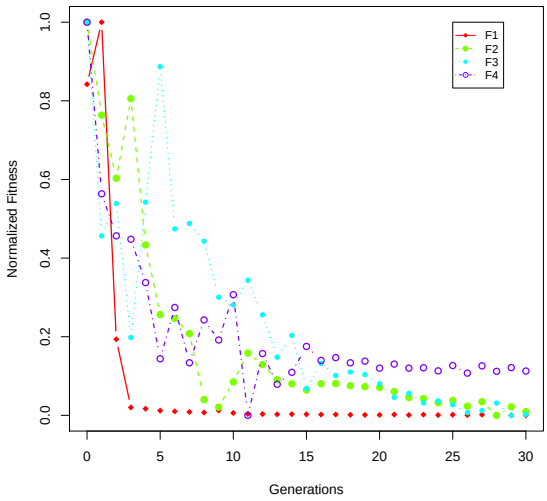


Figure 8.8: Convergence Plot for **FF-500**

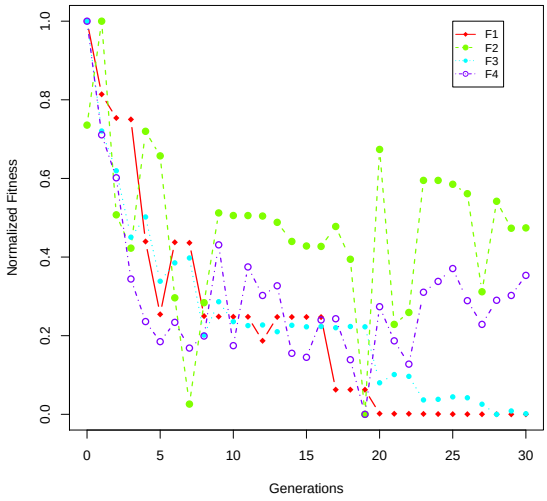


Figure 8.9: Convergence Plot for **BA-250**

ing coefficient and the average geodesic path length as there was no clear threshold for determining fitness.

The objective of this chapter was to demonstrate that the proposed methodology could replicate the model using only a single target graph. Having benefited from the use of known graph models, the chapter was able to provide a validation of evolved models against the actual model. Thus, the evolved models were compared statistically to the respective graph model. It was demonstrated that all evolved

models, except the **FF-100** and **FF-250**, had statistically similar values to their target model by way of t-tests comparing the differences in the average geodesic path lengths and the clustering coefficients.

In the case of the Forest Fire Model, additional statistical test was employed to demonstrate the fitness of in- and out-degree of the evolved models. It was desired to know if the frequency at which a graph from the evolved model and a graph from the target model had similar degree distributions was statistically different. A chi-squared test was employed to determine if the frequency was less than the frequency of statistically similar degree distributions between two graphs from the target model. It was shown that the evolved models trained with more vertices were able to satisfy all tests however those evolved models trained with 250 or fewer vertices failed.

The flexibility of the proposed methodology and LinkableGP was shown in experimentation with the ageing preferential attachment model. The ageing preferential attachment model introduces a new characteristic, namely vertex age, which require modifications to the system not required for experimentation with the other known graph models. Again through the same statistical approaches used with the other models it was demonstrated that the proposed methodology was able to produce statistically similar models respective to the fitness measures employed.

Finally, it was shown in Section 8.6 that the overall results of the trial runs routinely produced graphs that might have passed the validation criteria employed with the best models as many of the final fitness values or trial runs were comparable to the best models. The descriptive statistics of the final fitness values for the trial runs demonstrated a bias towards producing models that had very similar fitness values to those observed in the best model. While the results did show that some trial runs performed poorly, not only was it the case that the poor performance was obvious but also that the more than 75% of the results were comparable to the best evolved model.

Chapter 9

Comparison of Proposed Methods to Other Modelling Approaches

In Chapter 8, the proposed methods were empirically examined by replicating existing and well-known graph models. The experiments showed that the proposed methods were able to replicate those models with respect to the fitness measures used. However, these are not the only fitness measures that can be used. Moreover, in the case of the forest fire model, the fitness measures did not take into account all properties of the graph model.

Recently, work has been undertaken to try and determine a framework for deciding which network measures will provide a good fitness of evolved graph models. This work done by Harrison [36] focusses on a large number of measures of network properties and provides insights on they prevalence in complex networks.

Harrison [36] utilizes the study on networks measures and the methods proposed of this thesis. His experiments in replication of existing and well-known undirected graph models and in the construction of graph models for real-world complex networks provides further evidence that the proposed methods of this thesis are capable of producing models for a wide variety of networks. Additionally, his work demonstrates that the framework proposed by this thesis is flexible as his approach uses a different version of the abstract class proposed than proposed in this thesis.

Other modelling techniques, as presented in Chapter 4, have also shown promise in constructing graph models for complex networks. Techniques, such as Kronecker graphs [55], have also been widely applied in modelling real-world networks. This is thesis and no other work has yet compared this technique to other in terms of performance and accuracy. However, it can be said that this technique proposes a flexible framework for embedding specific knowledge about complex networks that

is not as easily reproduced by other works. Furthermore, when such knowledge is embedded into solutions it is not as clear what their impact on the network is.

9.1 Incorporation of Expert Knowledge

When constructing a graph model manually, it is necessary to collect a great deal of knowledge about the network at hand. This knowledge arises from insights about complex networks in general and the data being explored. In Chapter 7, an abstract class was introduced that encapsulates knowledge about the complex networks that would be later experimented with. Some of the abstract class included general knowledge with respect to graph models and other parts were very specific to the types of graph models that would later be the subject of experimentation.
Knowledge added about graph models in general included:

- Discrete time steps
- Specific placement of vertex selection
- Specific placement of edge creation/modification/removal

Knowledge specific to the models explored included:

- Vertex growth over time
- Restriction to edge creation only
- Allowance additional actions after edge creation
- Addition of age information (only employed in Ageing Preferential Attachment)

This thesis makes no claim that this is the only nor best way to generalize graph models for complex networks. It is simply an introduction to the concept and a demonstration of the application. The abstract form utilized is quite rigid and restrictive to work with the models relevant to the experiments in Chapter 8. However, modification to the algorithm is simple and sometimes necessary. In Section 8.5, a modification of the algorithm was incorporated to handle age of a vertex. In Harrison's [36], modifications were made to work with non-growth type algorithms as well as undirected networks.

The idea of incorporating expert knowledge into solutions is informed a great deal by the previous automatic inference methods as proposed by Bailey [10]. His

Algorithm 15: Recreation of Algorithm 3 from Bailey [10]

```
begin
    SET_GROW_NODES;
    for each node n do
        CONNECT_STUB_PERSIST(FLOAT_TO_PROB(0.02), TRUE);
    end for
end
```

Algorithm 16: BA-500

```
Function GenerateModel(int t)return graph
    G ← InitialiseGraph();
    for i ← 1 to t do
        v ← Vertex();
        W ← SelectVertices(G);
        while |W| > 0 do
            w ← Next(W);
            E ← E ∪ AddEdge(v, w);
            SecondaryActions(W, w);
        end while
        V ← V ∪ v;
    end for
    return G

Function SelectVertices(Graph g)return vertices
    b ← GetInDegree();
    c ← Constant(1);
    c ← Add(b, c);
    d ← GetRouletteQueue(g, c);
    return d
override Function AddEdge(Vertex v1,Vertex v2)return edge
    a ← CreateEdge(v1, v2);
    return a
override Procedure SecondaryActions(Vertex v, Vertices V)
```

work utilized a priority queue structure which may have been responsible for creating the preferential attachment properties in experiments at replicating the Barabasi-Albert model. Bailey [10] remarks that a function CONNECT_STUB_PERSIST is responsible for determining the vertices selected from a priority queue and are responsible for the preferential attachment. Bailey [10] also explains that a function SET_GROW_NODES is responsible for the growth process. In Algorithm 15, a recreation of one of the results from Bailey's [10] thesis is provided.

The algorithm shows a great deal of knowledge that is being incorporated into the system as the two instructions of the algorithm each do a great deal of work that is not a product of the evolutionary algorithm. Algorithm 16, a similar result from this thesis, shows the growth strategy (implemented *a priori*) and the preferential attachment (implemented as a roulette selection), also, simplifying the efforts of the GP. However, Algorithm 16 provides a more transparent understanding of the processes at work that is easily modifiable and controllable.

In the parametrized models discussed in Chapter 4, a great deal of expert knowledge for the general understanding of complex networks is being employed as they were formed on from a very intimate understanding of complex networks in general. However, the techniques do not lend themselves to modification for knowledge of specific complex networks. The models rely solely on the parameter tuning to determine the model for a specific complex network. The evolutionary strategy that uses symbolic regression [62] also employs a very specific algorithm that is dependent on the tuning of an equation.

9.2 Performance

Previous approaches such as those presented in Chapter 4 have been demonstrated to be successful a modelling both directed and undirected networks with the exception of Bailey [10] which has only been tested with the latter. Moreover, all have been demonstrated to work with real-world networks. Application in real-world networks is an important step for any modelling technique as it is the motivation of all graph modelling techniques.

In this thesis the proposed method has only been tested with directed networks produced by well-known graph models. However, the methods of this thesis have already been applied to work successfully with undirected complex networks and real-world networks [36]. It cannot be said that this method outperforms or underperforms compared to any other method in terms of successfully generation of graph

models. However, it can be said that this method has demonstrated that it works on a variety of different complex networks.

A comparative study of approaches in modelling complex networks is a relevant future study. However, as an interim comparison, this thesis' method proposes means of easily incorporating partial information about the structure and process of a complex network that is not readily incorporated into other approach. The abstract class representation provides an simple tool unlike any other approach where very little to a great deal of *a priori* knowledge can be encoded into solutions. This is a means that finding a solution is reduced to only what is unknown about the network. It is not impossible for other techniques to incorporate short-cuts equivalent to those achieved by the abstract class but the framework offered in this thesis provides an clear and easy means of doing so.

Chapter 10

Conclusion

In this thesis, a new methodology for the inference of graph models for complex networks was proposed. In doing so, the thesis also proposes a novel approach to GP, Object-Oriented GP (OOGP). The goal of the thesis was to demonstrate the proposed methodology that allowed expert knowledge of complex networks. Moreover, specific knowledge of a complex network being modelled could be incorporated into the construction of a fit graph model.

In the experimentation via the replication of known graph models, it was demonstrated that the methods employed were able to produce fit graph models compared to the targeted well-known graph models. The selected best-evolve models had final fitness values that did not occur by random chance as shown by post-run comparisons to its target graph. It was, also, demonstrated that through validation that the best-evolved models were able to reproduce certain structural properties expected from an accurately replicated model.

It was demonstrated, in the graph model replication experiments that trial runs routinely produced results comparable to those of the best-evolved models. This work also demonstrated that the final models from trials often had a similar fitness to the best models. While it was not confirmed with the same rigour employed with best-evolved models, it is likely that most of the final models would have also been shown to be accurately replicated models.

An important goal of this work was to develop a methodology that offered flexibility to the researcher to embed their expert knowledge into the graph model solutions produced by the system. The development of LinkableGP formed a framework that was conducive to incorporating knowledge. The methods employed by the system allows researchers to develop a structure for a graph model, provide it with a general algorithm, and restrict the search of LinkableGP in more expressive ways. An

initial example of how LinkableGP accomplishes this was outlined in Chapter 7. The chapter introduces a generalized and abstracted form of a graph model for complex networks with growing degrees that when tested is shown to produce valid models. Further evidence of the power of LinkableGP's representation of knowledge is demonstrated in the adaptations of the graph model abstraction for experiments with the ageing preferential attachment model. The model was selected as a demonstration of additional properties of a graph model. LinkableGP was demonstrated to be quickly and easily adapted to work with the new properties while still producing fit models.

In previous works [10], ad-hoc systems were necessary such as heap structures and community detection algorithms. Under the traditional GP approaches employed, these served to convolute the graph model and made it hard to understand their impact clearly on the evolved algorithm. However, this methodology presents a means to define the structure and a partial definition of the graph model. Thus, the effects of additional algorithms are more readily assessable in evolved solutions. This does not mean that the methods of this thesis are superior nor is there evidence to suggest it is inferior. The discussion in Chapter 9, instead, suggests that they are different and that understanding the impact of expert knowledge is easier with the methods of this thesis. Moreover, it is arguable that the code produced by LinkableGP is much closer to that expected of a programmer. It has been remarked by reviewers of the publications produced from this work that the design of LinkableGP has a good focus on the generation of real code in line with the way programming is done. The code in solutions is able to be modularized and have a well-defined structure. Having such code means that when additional strategies are employed that do not rely completely on evolution, it is clear and easy to understand the role in the resultant model.

While this thesis did not focus on applications of LinkableGP outside the context of graph model inference, LinkableGP does have application outside complex networks such as the evolution of data structures [60], and multi-behaviour artificial agents. This thesis and the publications derived from the development of LinkableGP serve as evidence that it addresses some shortcomings of other GP approaches. While other GP approaches can evolve multiple functions simultaneously and incorporate expert knowledge, LinkableGP provides a means of doing that shares the programming task with the human that is more accessible. LinkableGP, inspired by object-oriented programming, introduces the means to define more fluidly a program structure. In the opinion of this author, the approach, in contrast to other approaches, is more conducive to solving more complex problems that were tenable before by GP. It facilitates the incorporation of expert knowledge in a meaningful way and alleviates

the pressure from the GP to evolve parts of the solution already known. It is the expectation of this author that continued study will help to further prove the powers of LinkableGP.

Finally, This thesis has also served as an introduction to the inference of graph models for direct complex networks. It introduced a modified set of fitness measures from previous works (see [10]) to accommodate the differences between directed and undirected networks. It also introduced a new set of functions necessary to construct directed networks. It was demonstrated that the proposed methodology was able to work with these changes and still produce models that were accurate to the expected behaviours of the measures of each evolved model.

10.1 Limitations

In the development of a novel GP system which was well suited for the evolution of graph models for complex networks, there is much that must be considered. While this thesis does attempt to address representation and structure of GP in its role for the evolution of graph models, it has placed little focus on rigorous evaluation of the evolved models.

The thesis only manages to validate models in the context of the fitness measures used to construct the model during evolution. It is important to note that the four measures of fitness do not fully represent each model. For example, the Forest Fire model is known to produce models with community structures. In validating evolved models, no attempt was made to examine whether these communities were appropriately formed. The approach of fitness evaluation and validation does not allow the claim that an evolved model is an accurate replica of its target. Instead, it is only possible to say that the methods successfully replicated the properties measured of the models.

Furthermore, this thesis does not explore real-world networks. Insufficient time was able to be dedicated to experimentation with real-world data and therefore no analysis of the methods could be conducted on real-world experiments.

In this thesis, an abstract class that embedded some knowledge of graph models was proposed. This model relied on assumptions that the networks being modelled were growing networks that only grew one vertex at a time. In general, complex networks do not behave this way. While other abstract classes were experimented with, it was decided to use this model as a short-cut to evolution providing it with the most knowledge that was common between all targeted models. When modelling real-

world networks it is not likely that such knowledge could be assumed. For examples of other abstract classes please see [61] and [36].

10.2 Future Work

The results of this work have not been able to address a number of open problems in the inference of graph models for complex networks. Furthermore, it has also served to open more questions both in the graph model inference and the study of GP.

Several times through this work, it may have been apparent to the reader that evolved graph models were only compared based on a limited set of fitness measures that covered some very general structural properties of complex networks. However, these measures are not known to be sufficient to model accurately a specific complex network let alone complex networks in general. For example, the Forest Fire model generates graphs that are known to contain communities. The methodology for measuring the fitness of graph models used in this work does not incorporate this. Thus, two questions can be asked

1. Are the fitness measures sufficient to evolve models that produce graphs that exhibit properties that are not measured during evolution?

2. Are there other fitness measures required to ensure a model is accurate to the complex network at hand?

Further study into the fitness measure sufficiency is required. Although, a study has already been completed [36] which identifies which are the most useful measures found using well known graph models, it is not clear if these are sufficient measures. Studies should investigate which fitness measures will most accurately capture the properties of complex networks in general or provide a framework for determining what fitness measures are necessary for a specific complex network.

It is also necessary to expand efforts in the inference of graph models of complex networks to other types of networks. So far research has investigated unweighted networks both undirected and directed. Studies involving complex networks which are weighted, or where vertices have traits are important areas to investigate. In practical application for research, complex networks often are weighted and (especially social networks) have traits. An approach that is demonstrated to model these types of networks would prove a tool.

With respect to GP, this work has an important contribution as it has introduced another technique in evolving programs. However, the scope of this work is limited

to evolving complex networks and, therefore, does not fully investigate the approach presented. Further study using LinkableGP should investigate its application in other problems such as evolving data structures or other complex algorithms. Exploration into the genetic operators should also be conducted which focuses on understanding the impact on operators in the translation from genotype to phenotype. The mapping of the genotype to the phenotype is sensitive to the genetic operators but to what extent is not clearly known.

Bibliography

[1] Russ Abbott. Object-oriented genetic programming, an initial implementation. In *International Conference on Machine Learning: Models, Technologies and Applications*, pages 26–30, 2003.

[2] William Aiello, Fan Chung, and Linyuan Lu. A random graph model for massive graphs. In *Proceedings of the thirty-second annual ACM symposium on Theory of computing*, pages 171–180. Acm, 2000.

[3] William Aiello, Fan Chung, and Linyuan Lu. A random graph model for power law graphs. *Experimental Mathematics*, 10(1):53–66, 2001.

[4] William Aiello, Fan Chung, and Linyuan Lu. Random evolution in massive graphs. In *Handbook of massive data sets*, pages 97–122. Springer, 2002.

[5] Reka Albert and Albert-Laszlo Barabasi. Statistical mechanics of complex networks. *Reviews of Modern Physics*, 74:47–97, 2002.

[6] Réka Albert, Hawoong Jeong, and Albert-László Barabási. Internet: Diameter of the world-wide web. *Nature*, 401(6749):130–131, 1999.

[7] John C Almack. The influence of intelligence on the selection of associates. *School and Society*, 16:529–530, 1922.

[8] L. A. N. Amaral, A. Scala, M. Barthelemy, and H. E. Stanley. Classes of small-world networks. *PNAS*, 97:11149–11152, 2000.

[9] D.A. Augusto and H. J C Barbosa. Symbolic regression via genetic programming. In *Neural Networks, 2000. Proceedings. Sixth Brazilian Symposium on*, pages 173–178, 2000.

[10] Alexander Bailey. Automatic inference of graph models for complex networks with genetic programming. Master's thesis, Brock University, 2013.

[11] Alexander Bailey, Mario Ventresca, and Beatrice 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, pages 711–718, New York, NY, USA, 2012. ACM.

[12] Alexander Bailey, Beatrice Ombuki-Berman, and Mario 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, pages 893–900, New York, NY, USA, 2013. ACM. ISBN 978-1-4503-1963-8.

[13] Albert-László Barabási and Réka Albert. Emergence of scaling in random networks. *Science*, 286(5439):509–512, 1999.

[14] M. Barthelemy and L. A. N. Amaral. Small-world networks: Evidence for a crossover picture. *Physical Review Letters*, 82:3180–3183, 1999.

[15] P. S. Bearman, J Moody, and K Stivel. Chains of affection: The structure of adolescent romantic and sexual networks. *American Journal of Sociology*, 100: 44–91, 2004.

[16] Steven Bergen and BrianJ. Ross. Evolutionary art using summed multi-objective ranks. In Rick Riolo, Trent McConaghy, and Ekaterina Vladislavleva, editors, *Genetic Programming Theory and Practice VIII*, volume 8 of *Genetic and Evolutionary Computation*, pages 227–244. Springer New York, 2011. ISBN 978-1-4419-7746-5.

[17] Stefano Boccaletti, Vito Latora, Yamir Moreno, Martin Chavez, and D-U Hwang. Complex networks: Structure and dynamics. *Physics reports*, 424(4):175–308, 2006.

[18] Walter Bohm and Andreas Geyer-Schulz. Exact uniform initialization for genetic programming. *Foundations of Genetic Algorithms IV*, pages 379–407, 1997.

[19] Markus Brameier and Wolfgang Banzhaf. Explicit control of diversity and effective variation distance in linear genetic programming. In JamesA. Foster, Evelyne Lutton, Julian Miller, Conor Ryan, and Andrea Tettamanzi, editors, *Genetic Programming*, volume 2278 of *Lecture Notes in Computer Science*, pages 37–49. Springer Berlin Heidelberg, 2002.

[20] Markus F Brameier and Wolfgang Banzhaf. *Linear genetic programming.* Springer, 2007.

[21] Wilker Shane Bruce. *The Application of Genetic Programming to the Automatic Generation of Object-Oriented Programs.* PhD thesis, School of Computer and Information Sciences, Nova Southeastern University, Fort Lauderdale, Florida, USA, 1995.

[22] Robert Cairns, Hongling Xie, and Man-Chi Leung. The popularity of friendship and the neglect of social networks: Toward a new balance. *New Directions for Child and Adolescent Development*, 1998(81):25–53, 1998.

[23] Fan Chung, Linyuan Lu, T Gregory Dewey, and David J Galas. Duplication models for biological networks. *Journal of computational biology*, 10(5):677–687, 2003.

[24] Stanley Dagley, Donald Elliot Nicholson, *et al. An introduction to metabolic pathways.* Blackwell Scientific Publications, Ltd., Oxford, 1970.

[25] Derek de Solla Price. Networks of scientific papers. *Science*, 149:510–515, 1965.

[26] Reinhard Diestel. *Graph Theory*, volume 173 of *Graduate Texts in Mathematics.* Springer-Verlag, Heidelberg, third edition, 2005.

[27] S. N. Dorogovtsev and J. F. F. Mendes. Evolution of networks with aging of sites. *Phys. Rev. E*, 62:1842–1845, Aug 2000. doi: 10.1103/PhysRevE.62.1842.

[28] P Erdos and A Renyi. On random graphs i. *Publ. Math. Debrecen*, 6:290–297, 1959.

[29] Paul Erdős and Alfred Renyi. On the strength of connectedness of a random graph. *Acta Mathematica Hungarica*, 12(1):261–267, 1961.

[30] Ernesto Estrada. *The structure of complex networks: theory and applications.* Oxford University Press, 2011.

[31] Jinfei Fan and Franz Oppacher. Object oriented genetic programming, June 1 2009. US Patent App. 12/475,819.

[32] Andreas Geyer-Schulz and A Geyer-Schulz. *Fuzzy rule-based expert systems and genetic machine learning.* Physica-Verlag Heidelberg, 1997.

[33] E. N. Gilbert. Random graphs. *The Annals of Mathematical Statistics*, 30(4): 1141–1144, 12 1959. doi: 10.1214/aoms/1177706098.

[34] Michelle Girvan and Mark EJ Newman. Community structure in social and biological networks. *Proceedings of the National Academy of Sciences*, 99(12): 7821–7826, 2002.

[35] David Goldberg. Genetic algorithms: search and optimization algorithms, 1989.

[36] Kyle Robert Harrison. Network similarity measures and automatic construction of graph models using genetic programming. 2014.

[37] Bernardo A Huberman and Lada A Adamic. Internet: growth dynamics of the world-wide web. *Nature*, 401(6749):131–131, 1999.

[38] Norman P Hummon and Patrick Dereian. Connectivity in a citation network: The development of dna theory. *Social Networks*, 11(1):39–63, 1989.

[39] David R Hunter, Mark S Handcock, Carter T Butts, Steven M Goodreau, and Martina Morris. ergm: A package to fit, simulate and diagnose exponential-family models for networks. *Journal of Statistical Software*, 24(3):nihpa54860, 2008.

[40] Henry Kautz, Bart Selman, and Mehul Shah. Referral web: combining social networks and collaborative filtering. *Communications of the ACM*, 40(3):63–65, 1997.

[41] Gul Muhammad Khan, Julian Francis Miller, and David M. Halliday. Coevolution of intelligent agents using cartesian genetic programming. In *Proceedings of the 9th Annual Conference on Genetic and Evolutionary Computation*, GECCO '07, pages 269–276, New York, NY, USA, 2007. ACM. ISBN 978-1-59593-697-4.

[42] Andrej Nikolaevič Kolmogorov. Foundations of the theory of probability. 1950.

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

[44] John R Koza. *Genetic programming II: automatic discovery of reusable programs*. MIT press, 1994.

[45] John R. Koza, III Bennett, Forrest H., and Oscar Stiffelman. Genetic programming as a darwinian invention machine. In Riccardo Poli, Peter Nordin, William B. Langdon, and Terence C. Fogarty, editors, *Genetic Programming*, volume 1598 of *Lecture Notes in Computer Science*, pages 93–108. Springer Berlin Heidelberg, 1999. ISBN 978-3-540-65899-3.

[46] Paul L Krapivsky and Sidney Redner. Organization of growing random networks. *Physical Review E*, 63(6):066123, 2001.

[47] Paul L Krapivsky, Sidney Redner, and Francois Leyvraz. Connectivity of growing random networks. *Physical review letters*, 85(21):4629, 2000.

[48] James Ladyman and James Lambert. What is a complex system? Technical report, University of Bristol, 2011. URL `http://http://philsci-archive.pitt.edu/9044/4/LLWultimate.pdf`.

[49] William B Langdon. Evolving data structures with genetic programming. In *Proceedings of the 6th International Conference on Genetic Algorithms (ICGA95)*, pages 295–302, 1995.

[50] William B Langdon and Wolfgang Banzhaf. Repeated sequences in linear genetic programming genomes. *Complex Systems*, 15(4 (c)):285–306, 2005.

[51] Jure Leskovec and Christos Faloutsos. Scalable modeling of real graphs using kronecker multiplication. In *Proceedings of the 24th international conference on Machine learning*, pages 497–504. ACM, 2007.

[52] Jure Leskovec and Julian J Mcauley. Learning to discover social circles in ego networks. In *Advances in neural information processing systems*, pages 539–547, 2012.

[53] Jure Leskovec, Jon Kleinberg, and Christos 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*, pages 177–187. ACM, 2005.

[54] Jure Leskovec, Jon Kleinberg, and Christos Faloutsos. Graph evolution: Densification and shrinking diameters. *ACM Trans. Knowl. Discov. Data*, 1(1), March 2007. ISSN 1556-4681. doi: 10.1145/1217299.1217301.

[55] Jure Leskovec, Deepayan Chakrabarti, Jon Kleinberg, Christos Faloutsos, and Zoubin Ghahramani. Kronecker graphs: An approach to modeling networks. *The Journal of Machine Learning Research*, 11:985–1042, 2010.

[56] Heike K Lotze, Marta Coll, and Jennifer A Dunne. Historical changes in marine resources, food-web structure and ecosystem functioning in the adriatic sea, mediterranean. *Ecosystems*, 14(2):198–222, 2011.

[57] AR Mashaghi, Abolfazl Ramezanpour, and V Karimipour. Investigation of a protein complex network. *The European Physical Journal B-Condensed Matter and Complex Systems*, 41(1):113–121, 2004.

[58] Robert I. McKay, Nguyen Xuan Hoai, Peter Alexander Whigham, Yin Shan, and Michael O'Neill. Grammar-based genetic programming: a survey. *Genetic Programming and Evolvable Machines*, 11(3-4):365–396, 2010.

[59] Robert I Mckay, Nguyen Xuan Hoai, Peter Alexander Whigham, Yin Shan, and Michael O?Neill. Grammar-based genetic programming: a survey. *Genetic Programming and Evolvable Machines*, 11(3-4):365–396, 2010.

[60] Michael Richard Medland, Kyle Robert Harrison, and Beatrice Ombuki-Berman. Incorporating expert knowledge in object-oriented genetic programming. In *Proceedings of the 2014 conference companion on Genetic and evolutionary computation companion*, pages 145–146. ACM, 2014.

[61] Michael Richard Medland, Kyle Robert Harrison, and Beatrice 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, pages 305–311. IEEE, 2014.

[62] Telmo Menezes and Camille Roth. Symbolic regression of generative network models. *Scientific reports*, 4, 2014.

[63] Stanley Milgram. The small world problem. *Psychology today*, 2(1):60–67, 1967.

[64] Julian F Miller and Peter Thomson. Cartesian genetic programming. In *Genetic Programming*, pages 121–132. Springer, 2000.

[65] David J Montana. Strongly typed genetic programming. *Evolutionary computation*, 3(2):199–230, 1995.

[66] Jacob L Moreno and Helen Hall Jennings. Who shall survive? 1934.

[67] Amit A Nanavati, Siva Gurumurthy, Gautam Das, Dipanjan Chakraborty, Koustuv Dasgupta, Sougata Mukherjea, and Anupam Joshi. On the structural properties of massive telecom call graphs: findings and implications. In *Proceedings of the 15th ACM international conference on Information and knowledge management*, pages 435–444. ACM, 2006.

[68] M. E. J. Newman. The structure and function of complex networks. *SIAM Review*, 45:167–256, 2003.

[69] M. E. J. Newman. *Networks: An Introduction*. Oxford University Press, 2010.

[70] Mark Newman, Albert-Laszlo Barabasi, and Duncan J. Watts. *The Structure and Dynamics of Networks*. Princeton University Press, 2006.

[71] Mihai Oltean and Crina Grosan. A comparison of several linear genetic programming techniques. *Complex Systems*, 14(4):285–314, 2003.

[72] Ali Pinar, C Seshadhri, and Tamara G Kolda. The similarity between stochastic kronecker and chung-lu graph models. *CoRR, abs/1110.4925*, 2011.

[73] Robin L Plackett. Karl pearson and the chi-squared test. *International Statistical Review/Revue Internationale de Statistique*, pages 59–72, 1983.

[74] Riccardo Poli, W William B Langdon, Nicholas F McPhee, and John R Koza. *A field guide to genetic programming*. Lulu. com, 2008.

[75] Stephen R Proulx, Daniel EL Promislow, and Patrick C Phillips. Network thinking in ecology and evolution. *Trends in Ecology & Evolution*, 20(6):345–353, 2005.

[76] Patrick Reynolds and Amin Vahdat. Efficient peer-to-peer keyword searching. In *Proceedings of the ACM/IFIP/USENIX 2003 International Conference on Middleware*, pages 21–40. Springer-Verlag New York, Inc., 2003.

[77] Matei Ripeanu. Peer-to-peer architecture case study: Gnutella network. In *Peer-to-Peer Computing, 2001. Proceedings. First International Conference on*, pages 99–100. IEEE, 2001.

[78] Garry Robins, Pip Pattison, Yuval Kalish, and Dean Lusher. An introduction to exponential random graph (p^*) models for social networks. *Social networks*, 29(2):173–191, 2007.

[79] Garry Robins, Tom Snijders, Peng Wang, Mark Handcock, and Philippa Pattison. Recent developments in exponential random graph (p^*) models for social networks. *Social networks*, 29(2):192–215, 2007.

[80] Kumara Sastry and DavidE. Goldberg. Probabilistic model building and competent genetic programming. In Rick Riolo and Bill Worzel, editors, *Genetic Programming Theory and Practice*, volume 6 of *Genetic Programming Series*, pages 205–220. Springer US, 2003.

[81] Thomas C Schelling. Dynamic models of segregation? *Journal of mathematical sociology*, 1(2):143–186, 1971.

[82] Ray Solomonoff and Anatol Rapoport. Connectivity of random nets. *The bulletin of mathematical biophysics*, 13(2):107–117, 1951.

[83] Terence Soule, James A Foster, *et al.* Code size and depth flows in genetic programming. *Genetic Programming*, pages 313–320, 1997.

[84] Olaf Sporns and Jonathan D Zwi. The small world of the cerebral cortex. *Neuroinformatics*, 2(2):145–162, 2004.

[85] Jeffrey Travers and Stanley Milgram. An experimental study of the small world problem. *Sociometry*, 32(4):pp. 425–443, 1969.

[86] Bimal Viswanath, Alan Mislove, Meeyoung Cha, and Krishna P. Gummadi. On the evolution of user interaction in facebook. In *Proceedings of the 2nd ACM SIGCOMM Workshop on Social Networks (WOSN'09)*, August 2009.

[87] E.J. Vladislavleva, G.F. 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*, 13(2): 333–349, April 2009. ISSN 1089-778X.

[88] Andreas Wagner and David A Fell. The small world inside large metabolic networks. *Proceedings of the Royal Society of London. Series B: Biological Sciences*, 268(1478):1803–1810, 2001.

[89] Stanley Wasserman. *Social network analysis: Methods and applications*, volume 8. Cambridge university press, 1994.

[90] Duncan J Watts and Steven H Strogatz. Collective dynamics of ?small-world?networks. *nature*, 393(6684):440–442, 1998.

[91] Beth Wellman. The school child's choice of companions. *The Journal of Educational Research*, pages 126–132, 1926.

[92] Peter A Whigham *et al.* Grammatically-based genetic programming. In *Proceedings of the workshop on genetic programming: from theory to real-world applications*, volume 16, pages 33–41, 1995.

[93] Charles H Wolfgang. *Solving discipline and classroom management problems*. Wiley Global Education, 2008.

[94] Man Leung Wong and Kwong Sak Leung. Applying logic grammars to induce sub-functions in genetic programming. In *Evolutionary Computation, 1995., IEEE International Conference on*, volume 2, pages 737–740. IEEE, 1995.

Appendix A

Select Evolved Graph Models

The algorithms presented here are selected from the experiments in Chapter 8. They are implementations of Algorithms 11, or 14 (APA only). These models were chosen as they were characteristic of the best evolved models for their targeted model.

Algorithm 17: GR-500

```
Function GenerateModel(int t)return graph
    G ← InitialiseGraph();
    for i ← 1 to t do
        v ← Vertex();
        W ← SelectVertices(G);
        while |W| > 0 do
            w ← Next(W);
            E ← E ∪ AddEdge(v, w);
            SecondaryActions(W, w);
        end while
        V ← V ∪ v;
    end for
    return G

override Function SelectVertices(Graph g)return vertices
    a ← GeometricValue(0.402696);
    b ← GetRandomQueue(g);
    c ← GetLocalTransitivity();
    return b
override Function AddEdge(Vertex v1,Vertex v2)return edge
    a ← CreateEdge(v1, v2);
    return a
override Procedure SecondaryActions(Vertex v, Vertices V)
    a ← GeometricValue(0.886134);
    a ← RandomValue(a, a);
    a ← BernoulliValue(0.93379);
```

Algorithm 18: BA-500

Function `GenerateModel(`*int t*`)`**return graph**
$G \leftarrow InitialiseGraph()$;
for $i \leftarrow 1$ **to** t **do**
$v \leftarrow Vertex()$;
$W \leftarrow SelectVertices(G)$;
while $|W| > 0$ **do**
$w \leftarrow Next(W)$;
$E \leftarrow E \cup AddEdge(v, w)$;
$SecondaryActions(W, w)$;
end while
$V \leftarrow V \cup v$;
end for
return G

Function `SelectVertices(`*Graph g*`)`**return vertices**
$a \leftarrow BernoulliValue(0.590228)$;
$a \leftarrow RandomValue(10, 2)$;
$b \leftarrow GetInDegree()$;
$c \leftarrow Constant(1)$;
$c \leftarrow Add(b, c)$;
$d \leftarrow GetRouletteQueue(g, c)$;
return d
override Function `AddEdge(`*Vertex v1,Vertex v2*`)`**return edge**
$a \leftarrow CreateEdge(v1, v2)$;
return a
override Procedure `SecondaryActions(`*Vertex v, Vertices V*`)`
$a \leftarrow BernoulliValue(0.350639)$;

Algorithm 19: FF-500

Function `GenerateModel(`*int t*`)`**return graph**
$G \leftarrow InitialiseGraph()$;
for $i \leftarrow 1$ **to** t **do**
$v \leftarrow Vertex()$;
$W \leftarrow SelectVertices(G)$;
while $|W| > 0$ **do**
$w \leftarrow Next(W)$;
$E \leftarrow E \cup AddEdge(v, w)$;
$SecondaryActions(W, w)$;
end while
$V \leftarrow V \cup v$;
end for
return G

override Function `SelectVertices(`*Graph g*`)`**return vertices**
$a \leftarrow BernoulliValue(0.02819)$;
$a \leftarrow GeometricValue(0.941932)$;
$a \leftarrow BernoulliValue(0.347325)$;
$b \leftarrow GetRandomStack(g)$;
$c \leftarrow GetRandomQueue(g)$;
return c
override Function `AddEdge(`*Vertex v1,Vertex v2*`)`**return edge**
$a \leftarrow CreateEdge(v1, v2)$;
return a
override Procedure `SecondaryActions(`*Vertex v, Vertices V*`)`
$a \leftarrow GeometricValue(0.63182)$;
$a \leftarrow GeometricValue(0.681023)$;
$AddSuccessor(V, a, v)$;
$AddPredecessors(V, b, v)$;

Algorithm 20: APA-1000

```
Function GenerateModel(int t)return graph
    G ← InitialiseGraph();
    for i ← 1 to t do
        v ← Vertex();
        SetBirth(v, i);
        SetTime(g, i);
        W ← SelectVertices(G);
        while |W| > 0 do
            w ← Next(W);
            E ← E ∪ AddEdge(v, w);
            SecondaryActions(W, w);
        end while
        V ← V ∪ v;
    end for
    return G

override Function SelectVertices(Graph g)return vertices
    a ← GetAge();
    b ← GetInDegree();
    c ← Constant();
    d ← Add(b, c);
    e ← GeometricValue(0.347125);
    b ← Div(d, b);
    e ← GetRoulleteQueue(g, c);
    return e
override Function AddEdge(Vertex v1,Vertex v2)return edge
    a ← CreateEdge(v1, v2);
    return a
override Procedure SecondaryActions(Vertex v, Vertices V)
    a ← GeometricValue(0.157924);
    a ← RandomValue(a, 1);
    b ← GeometricValue(0.785169);
```

Appendix B

Graph Visualizations of Evolved Model vs. Real Model by Number of Vertices

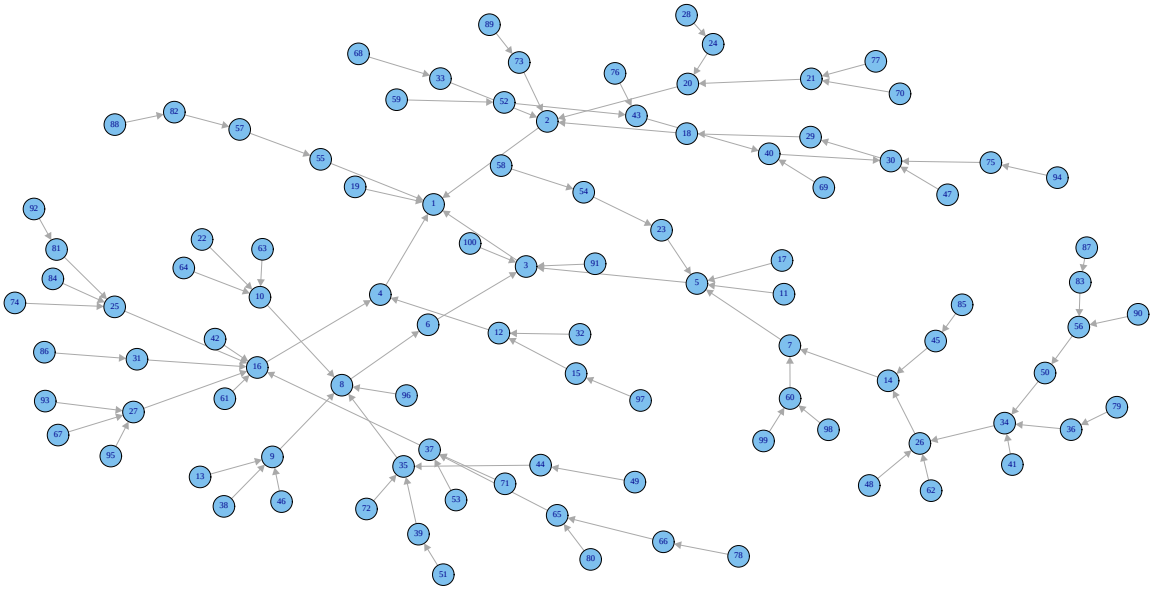


Figure B.1: Growing Random Model with 100 Vertices

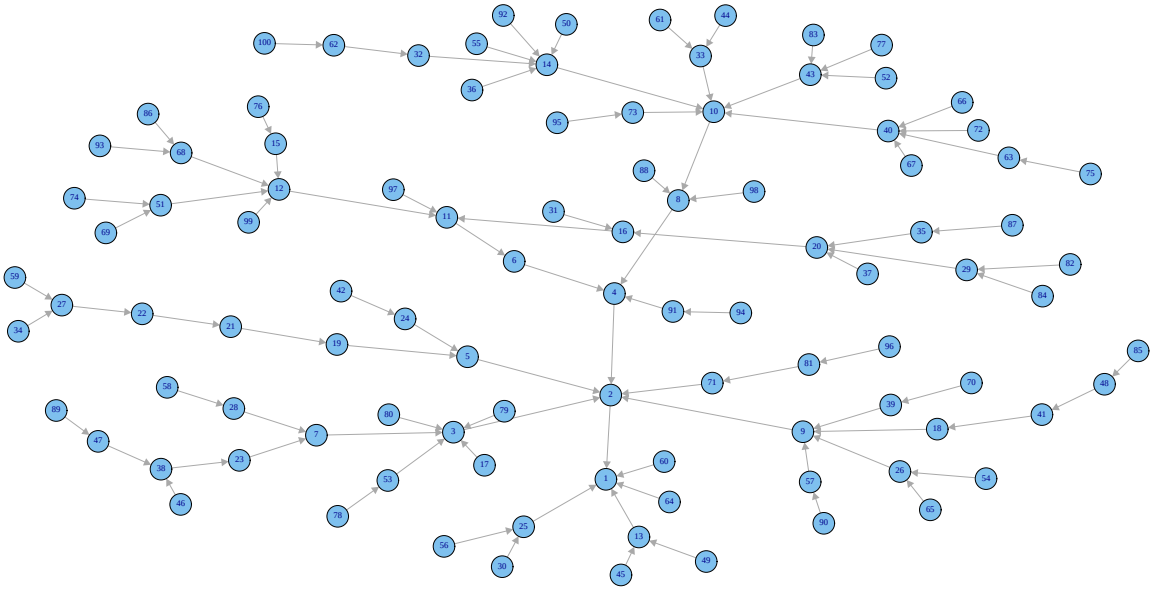


Figure B.2: Evolved Growing Random Model with 100 Vertices

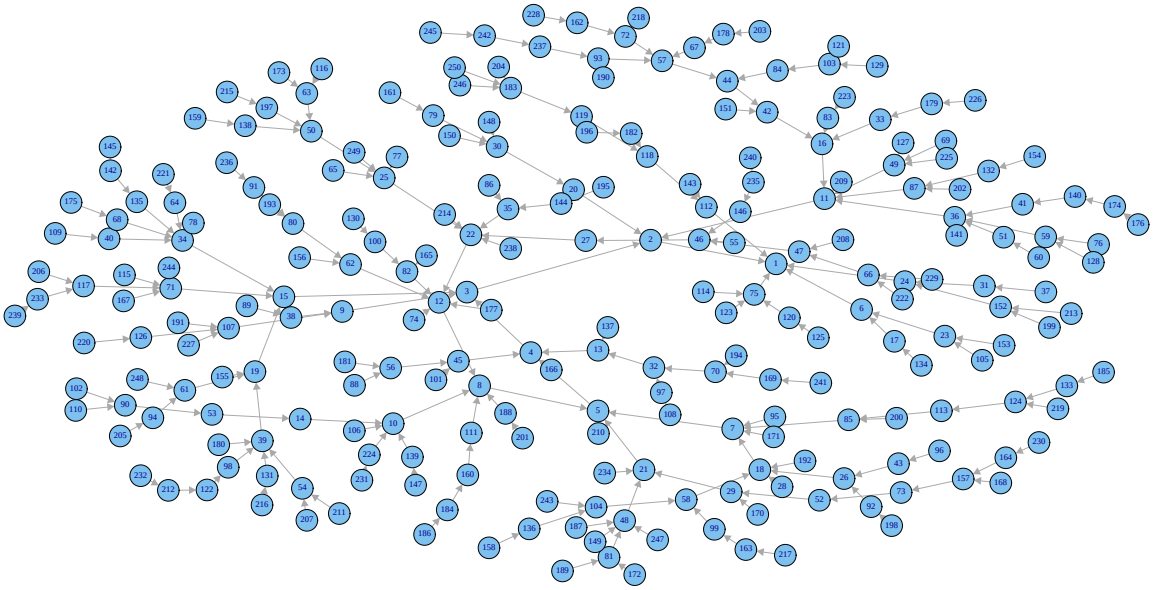


Figure B.3: Growing Random Model with 250 Vertices

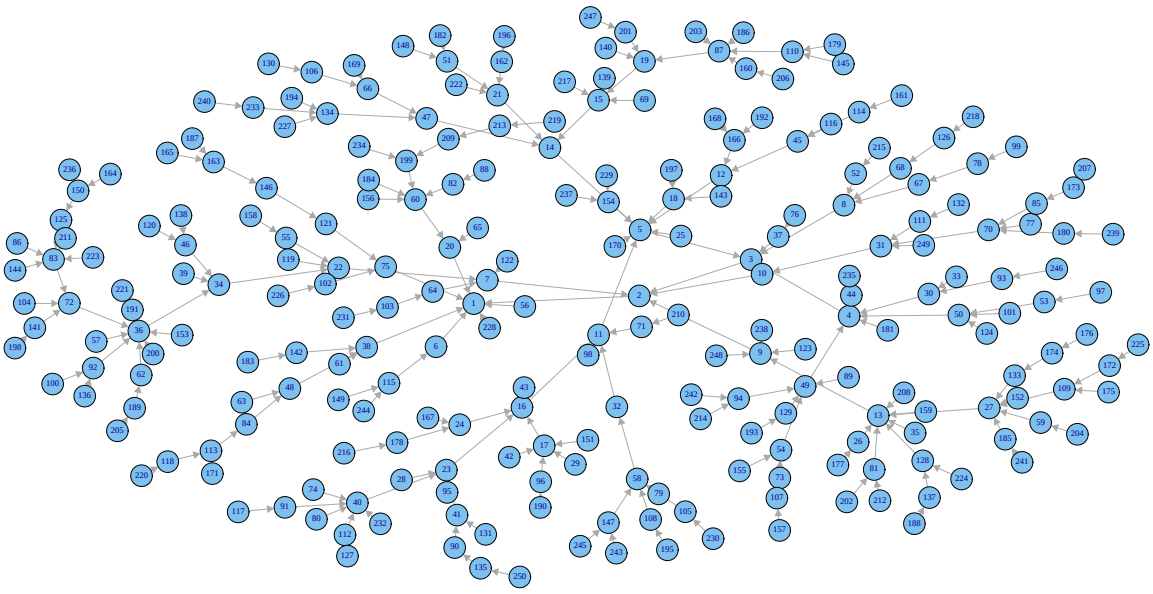


Figure B.4: Evolved Growing Random Model with 250 Vertices

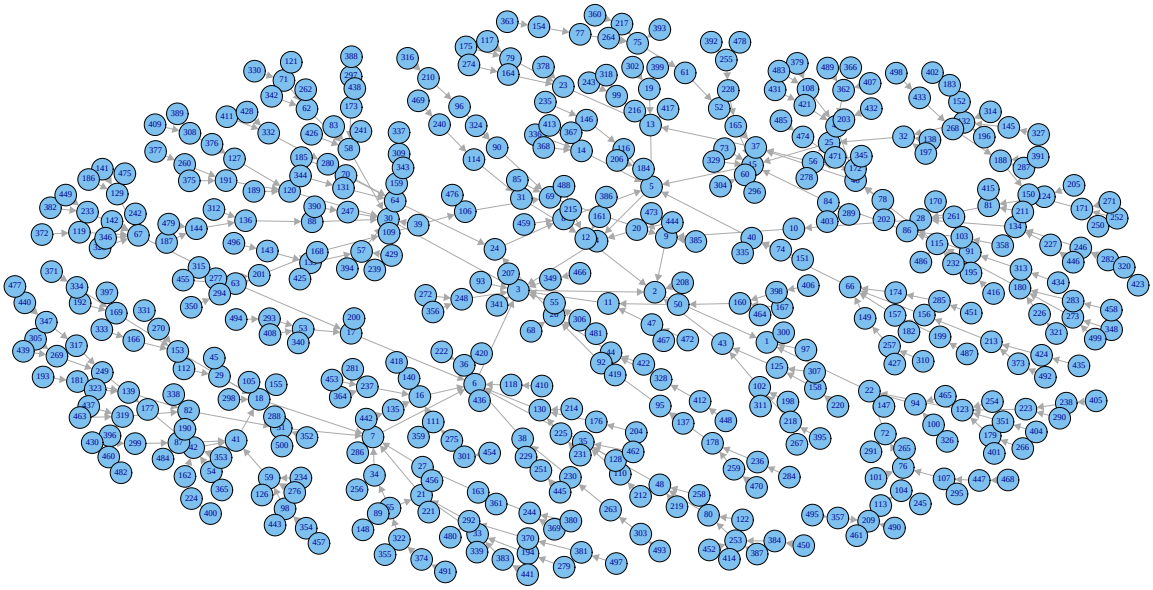


Figure B.5: Growing Random Model with 500 Vertices

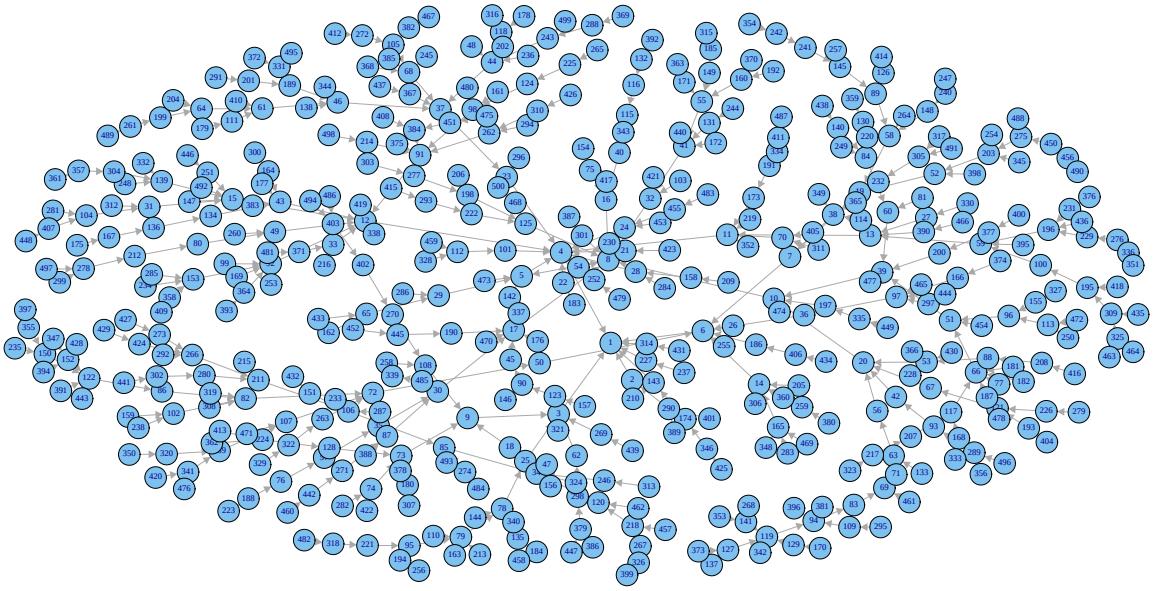


Figure B.6: Evolved Growing Random Model with 500 Vertices

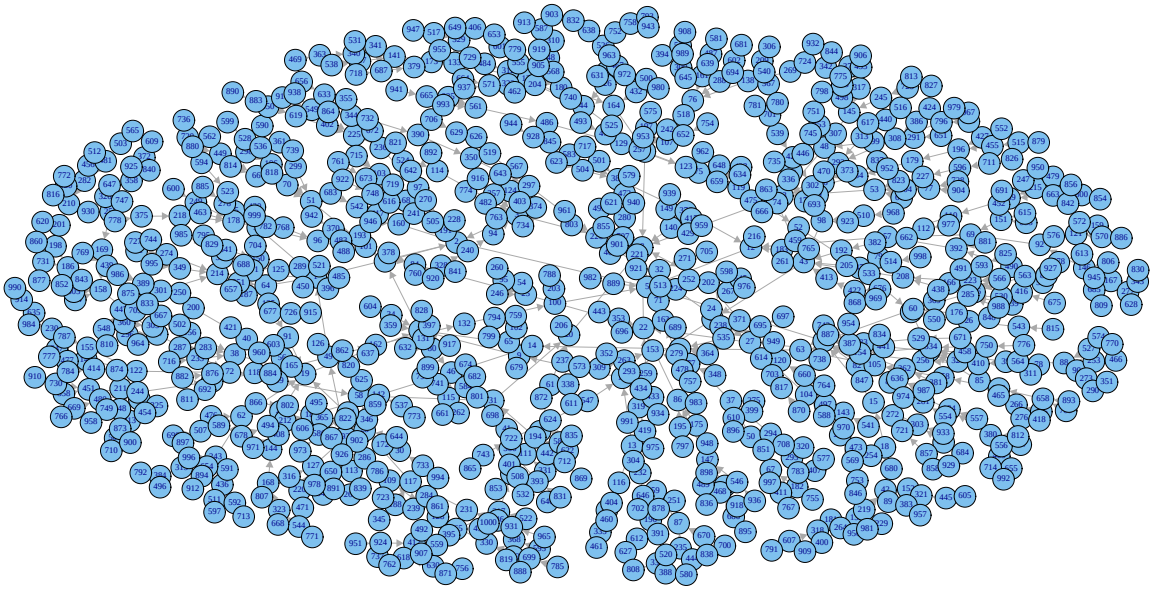


Figure B.7: Growing Random Model with 1000 Vertices

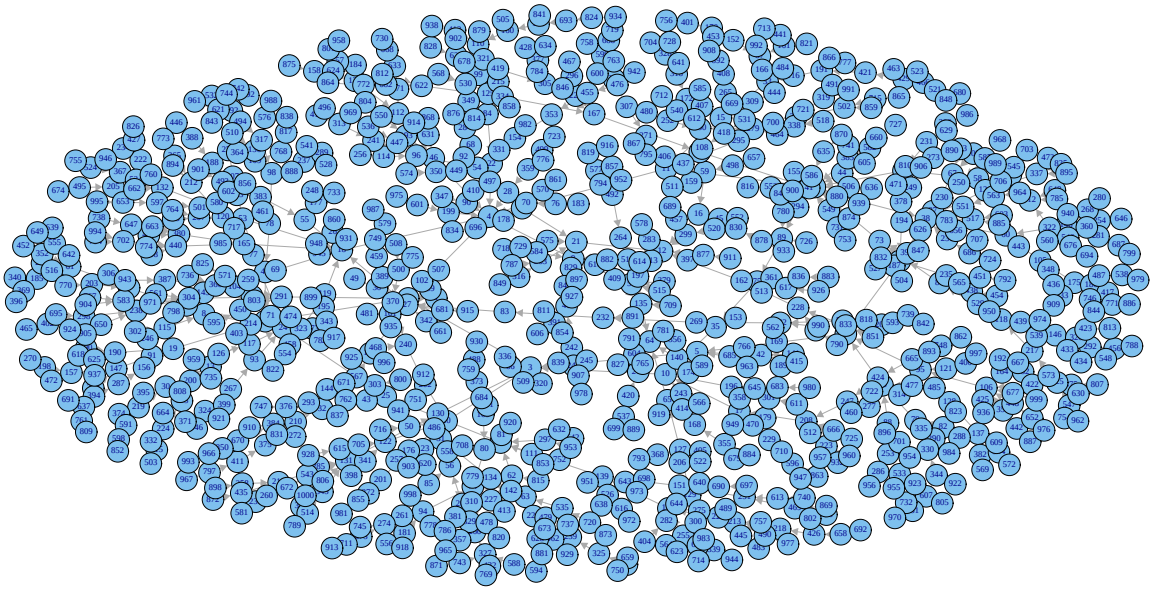


Figure B.8: Evolved Growing Random Model with 1000 Vertices

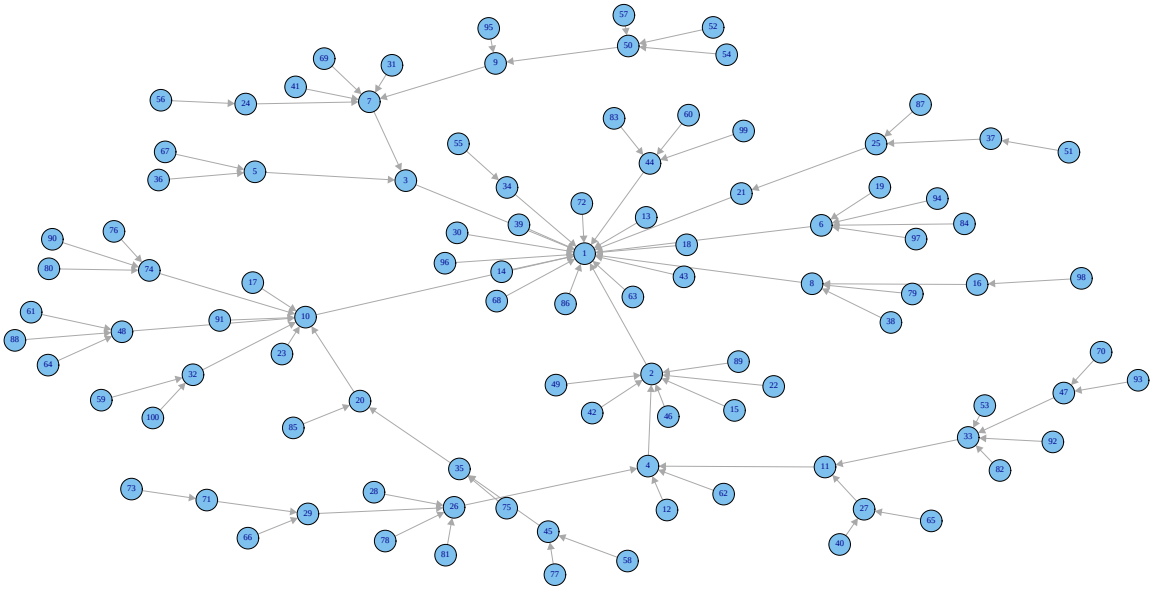


Figure B.9: Barabasi-Albert Model with 100 Vertices

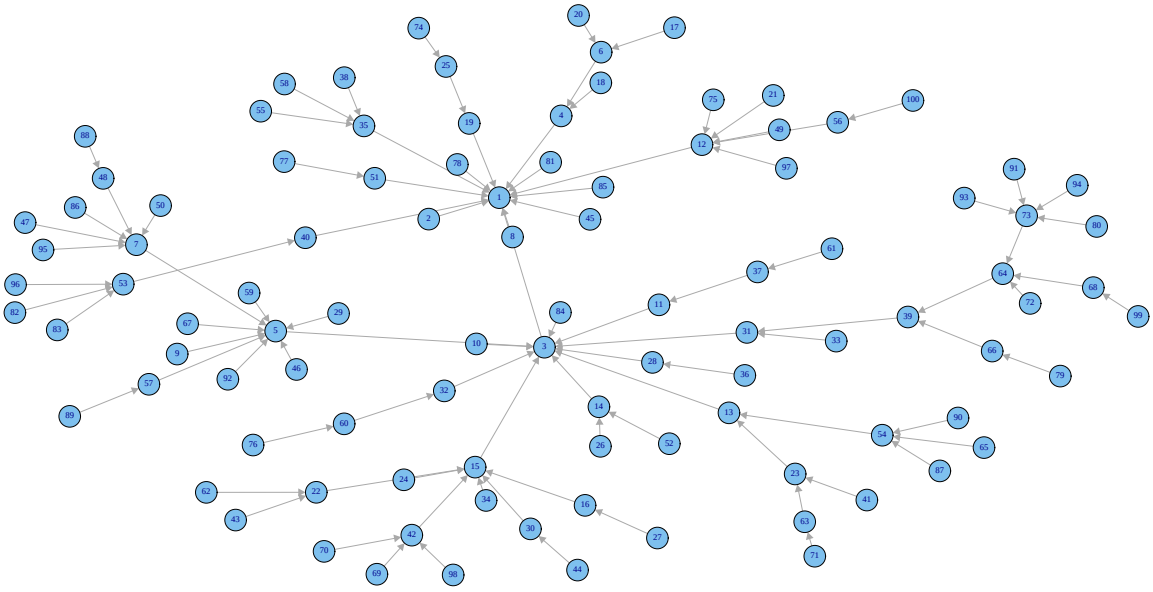


Figure B.10: Evolved Barabasi-Albert Model with 100 Vertices

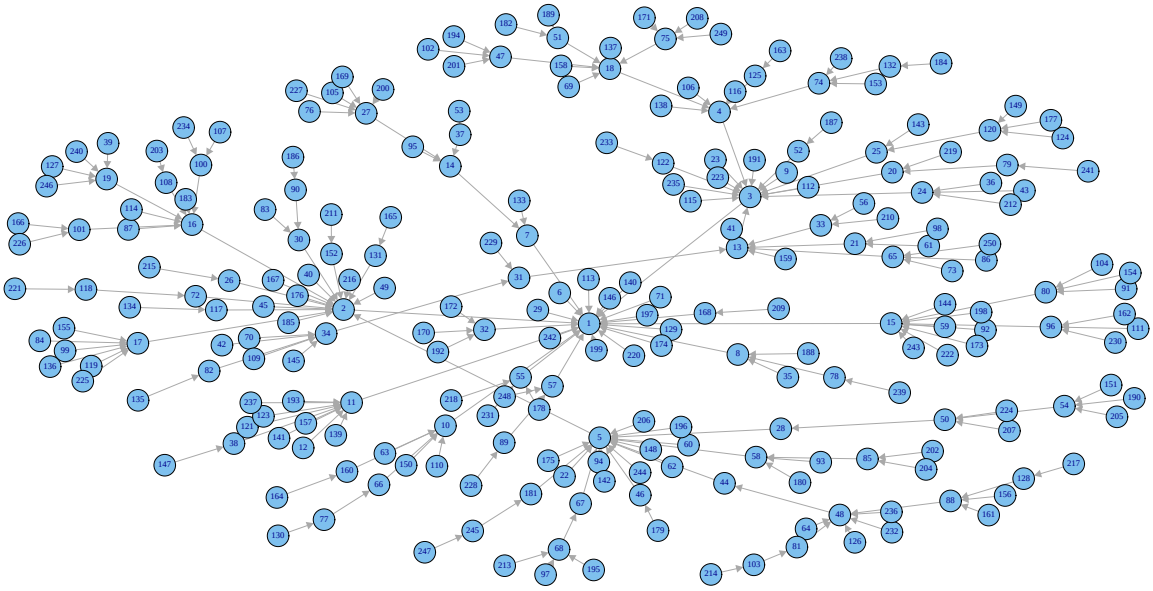


Figure B.11: Barabasi-Albert Model with 250 Vertices

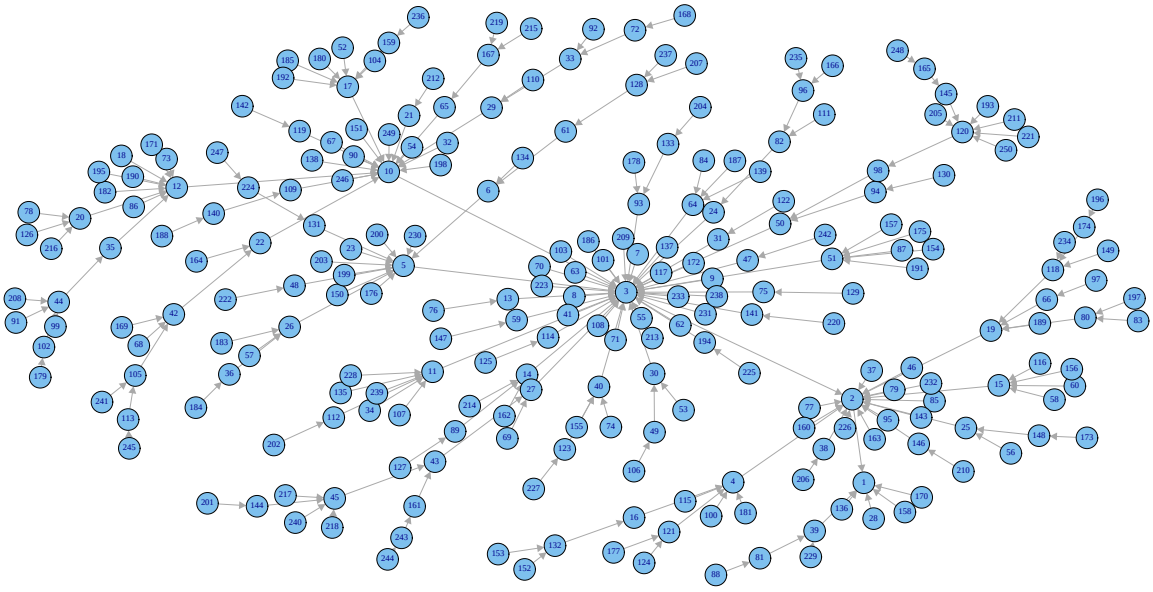


Figure B.12: Evolved Barabasi-Albert Model with 250 Vertices

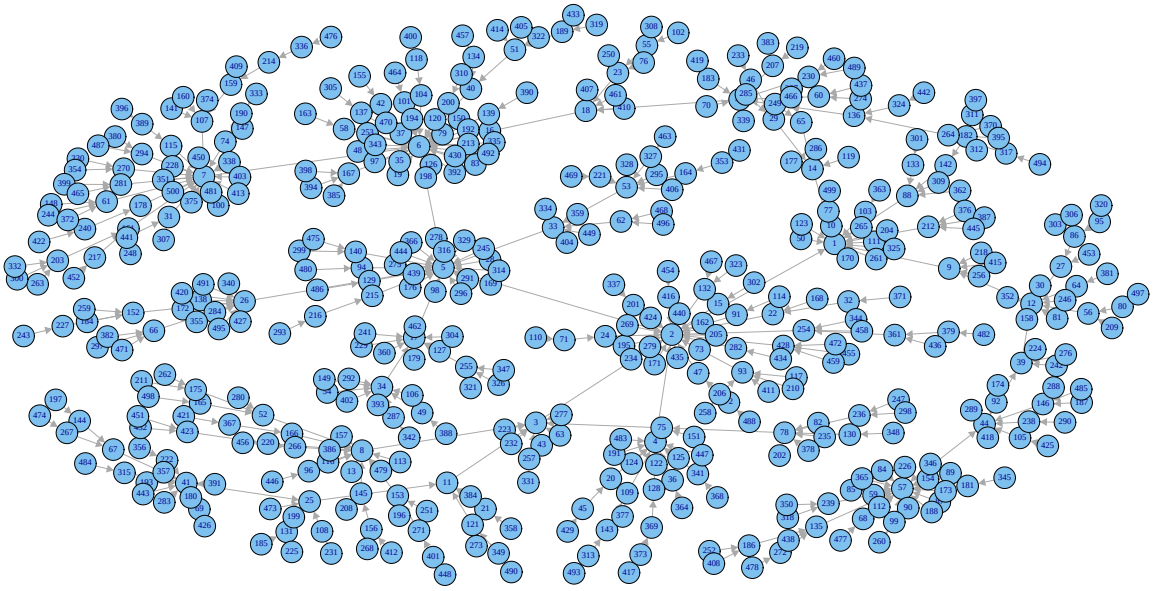


Figure B.13: Barabasi-Albert Model with 500 Vertices

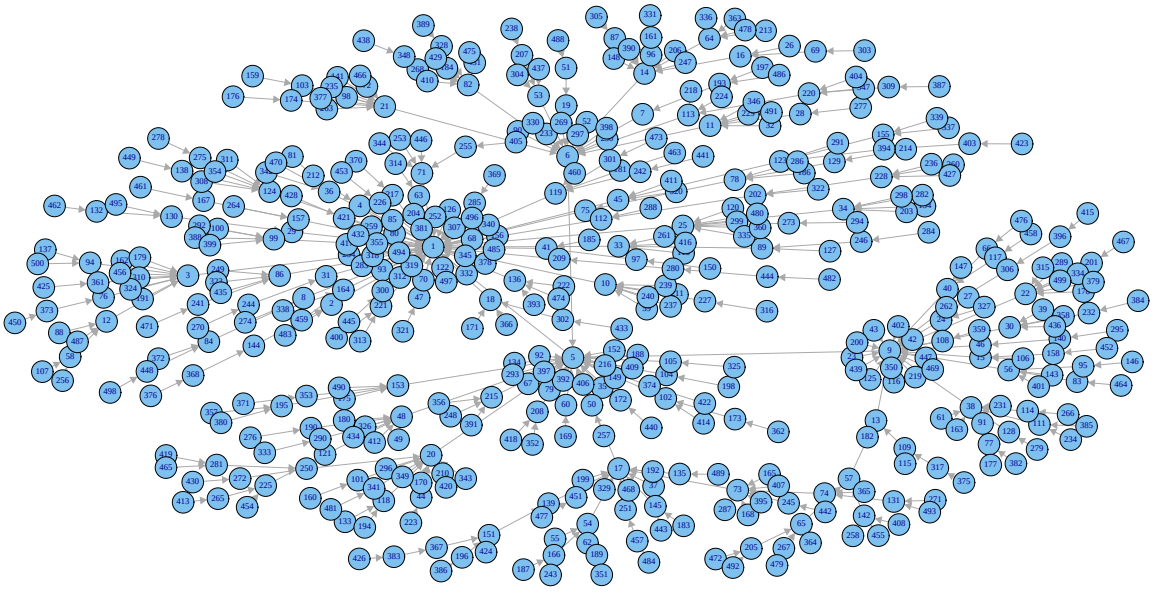


Figure B.14: Evolved Barabasi-Albert Model with 500 Vertices

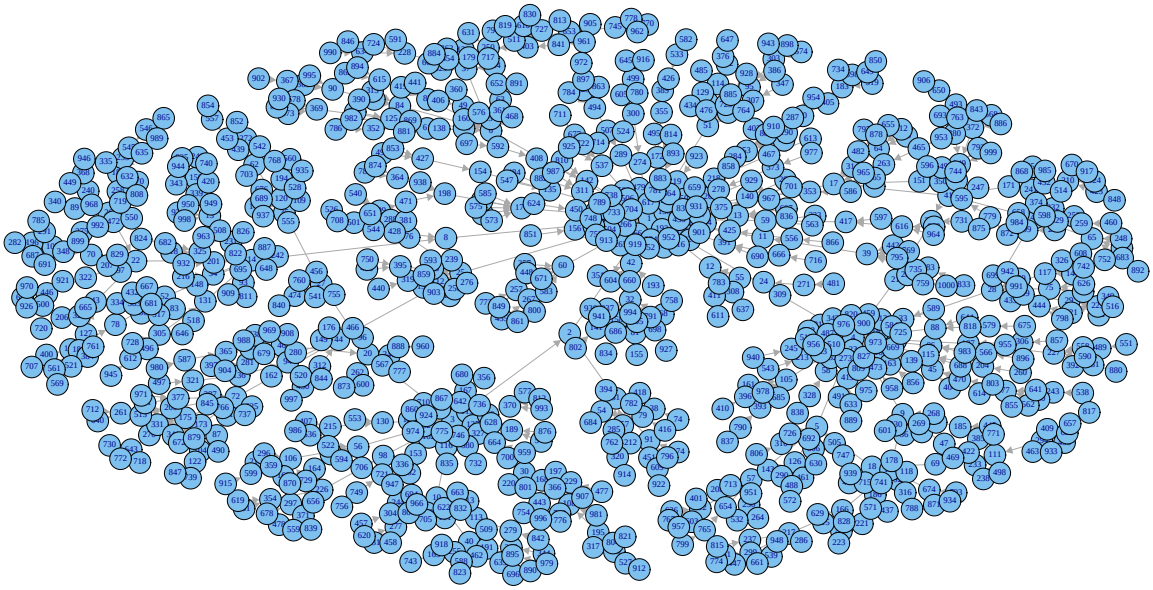


Figure B.15: Barabasi-Albert Model with 1000 Vertices

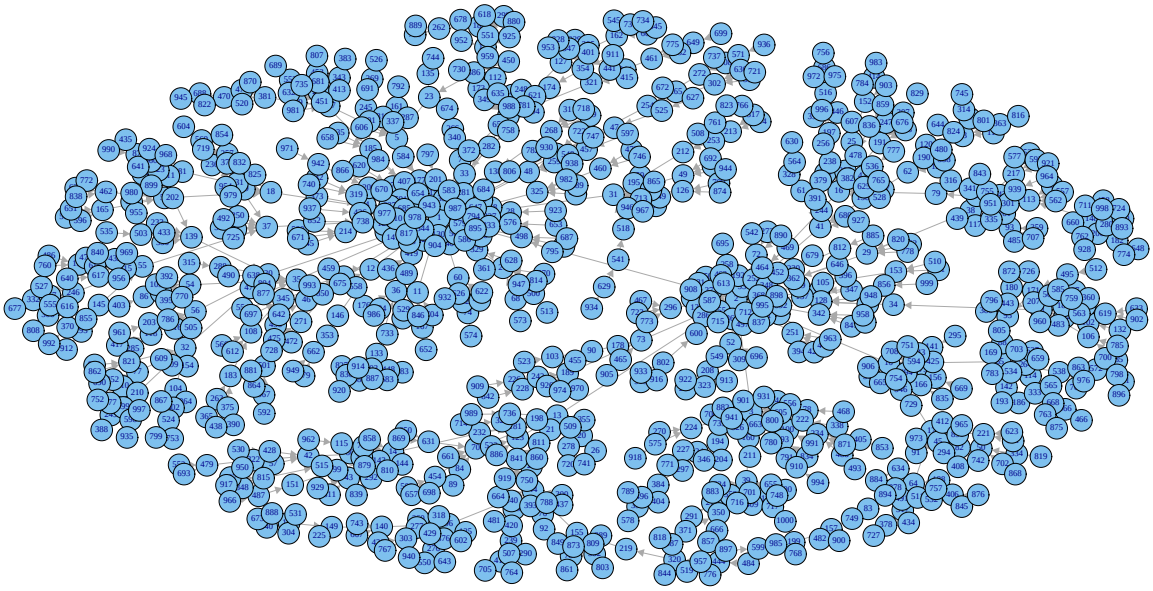


Figure B.16: Evolved Barabasi-Albert Model with 1000 Vertices

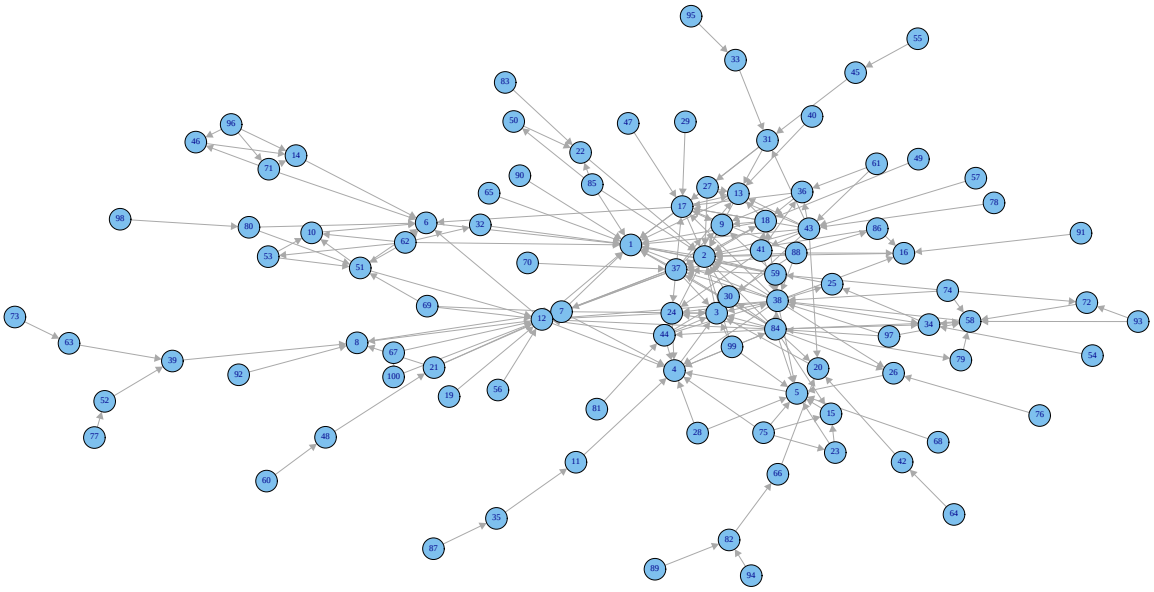


Figure B.17: Forest Fire Model with 100 Vertices

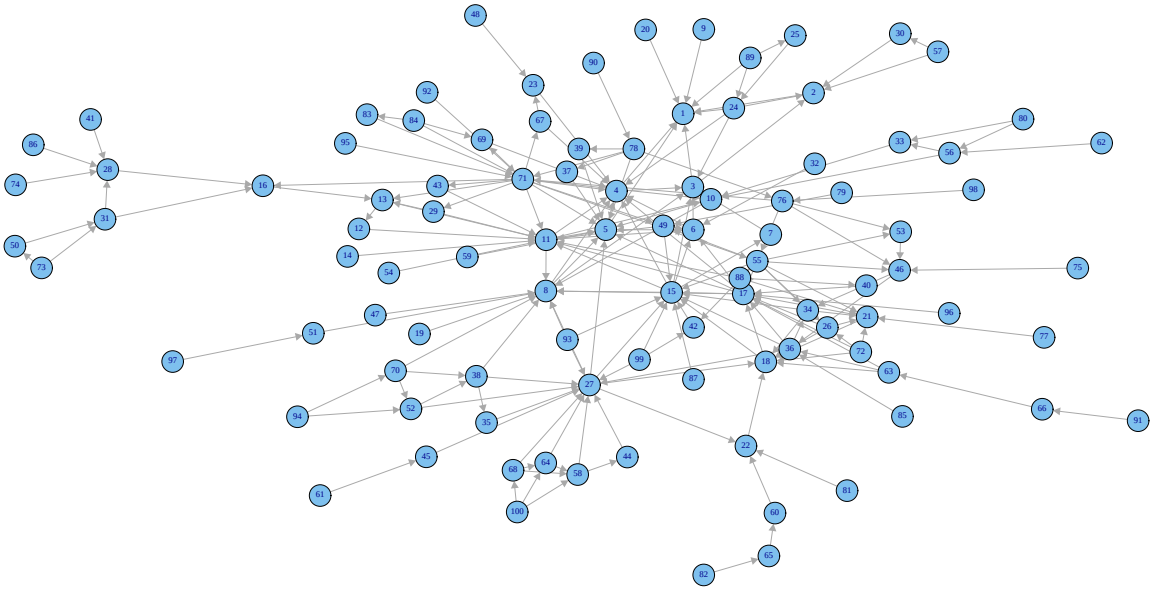


Figure B.18: Evolved Forest Fire Model with 100 Vertices

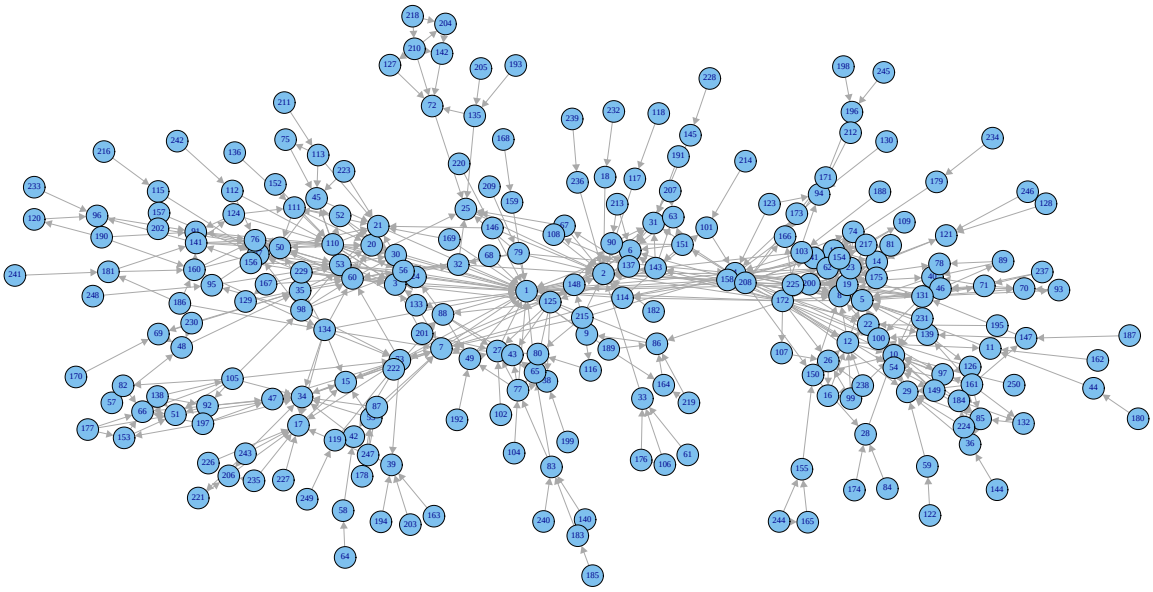


Figure B.19: Forest Fire Model with 250 Vertices

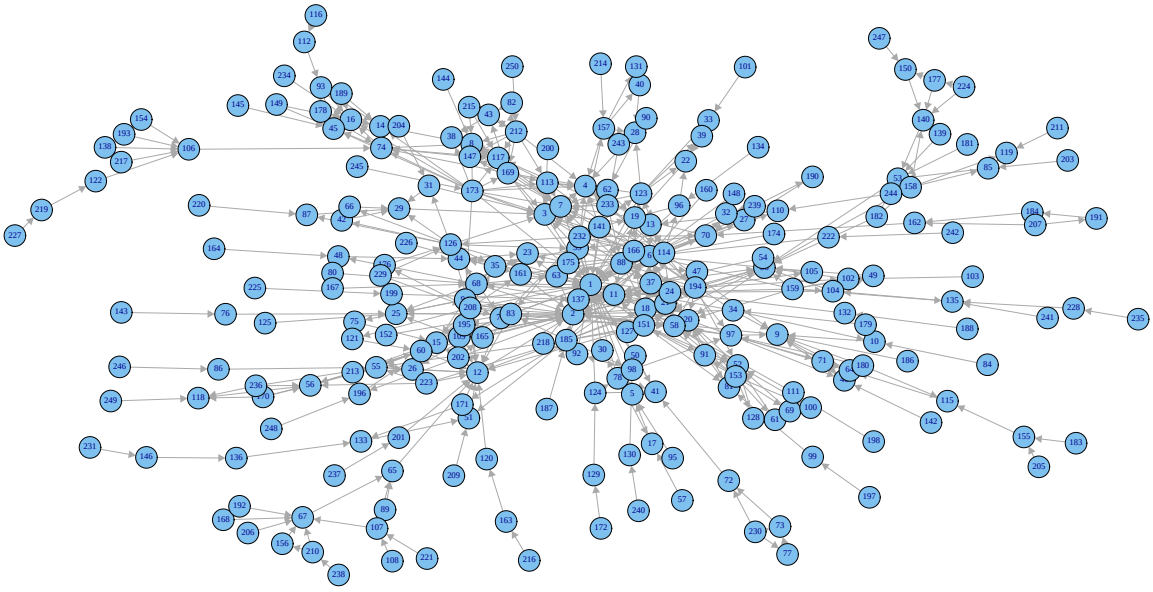


Figure B.20: Evolved Forest Fire Model with 250 Vertices

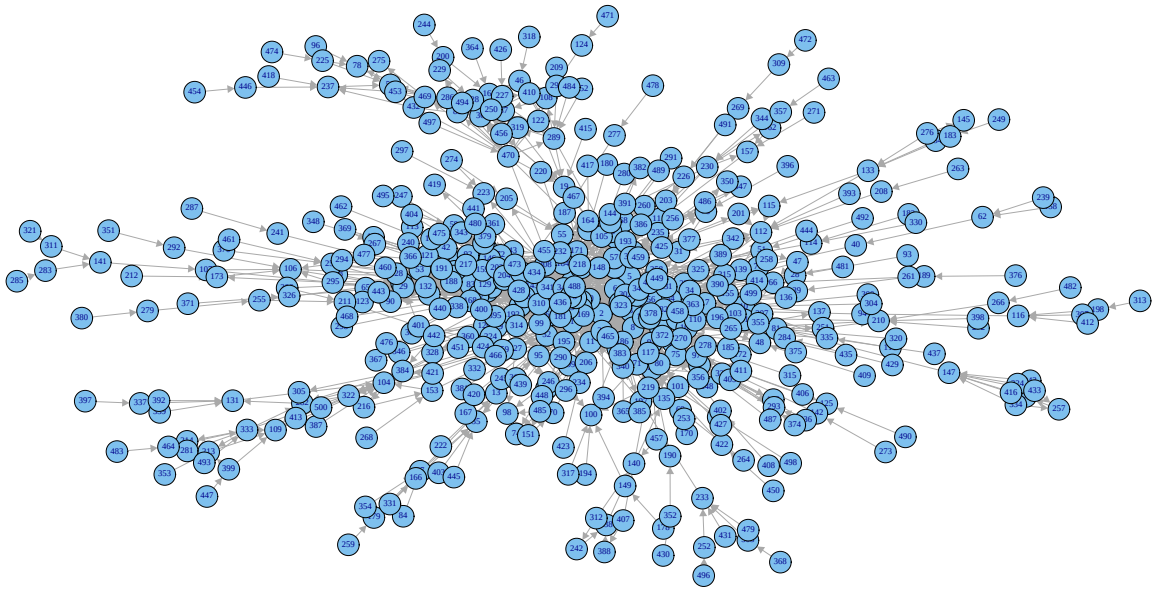


Figure B.21: Forest Fire Model with 500 Vertices

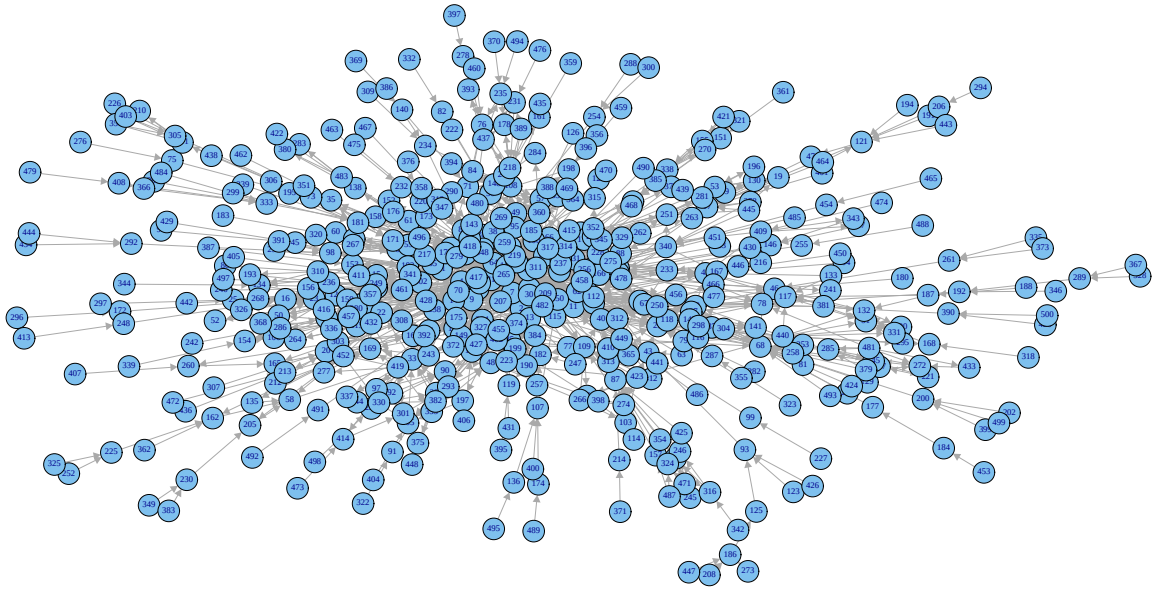


Figure B.22: Evolved Forest Fire Model with 500 Vertices

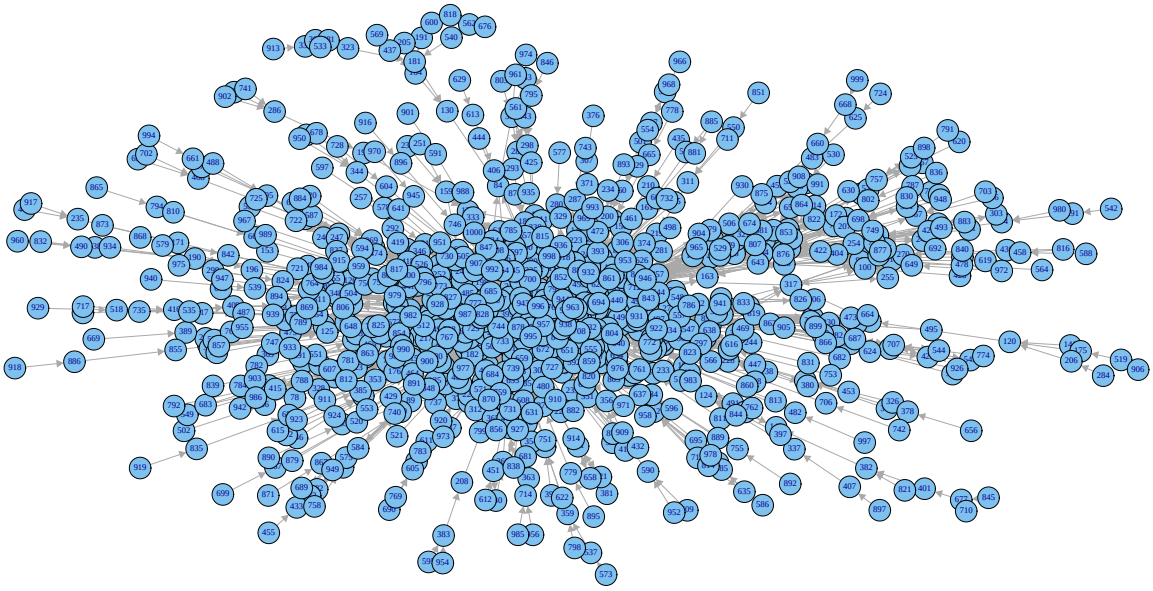


Figure B.23: Forest Fire Model with 1000 Vertices

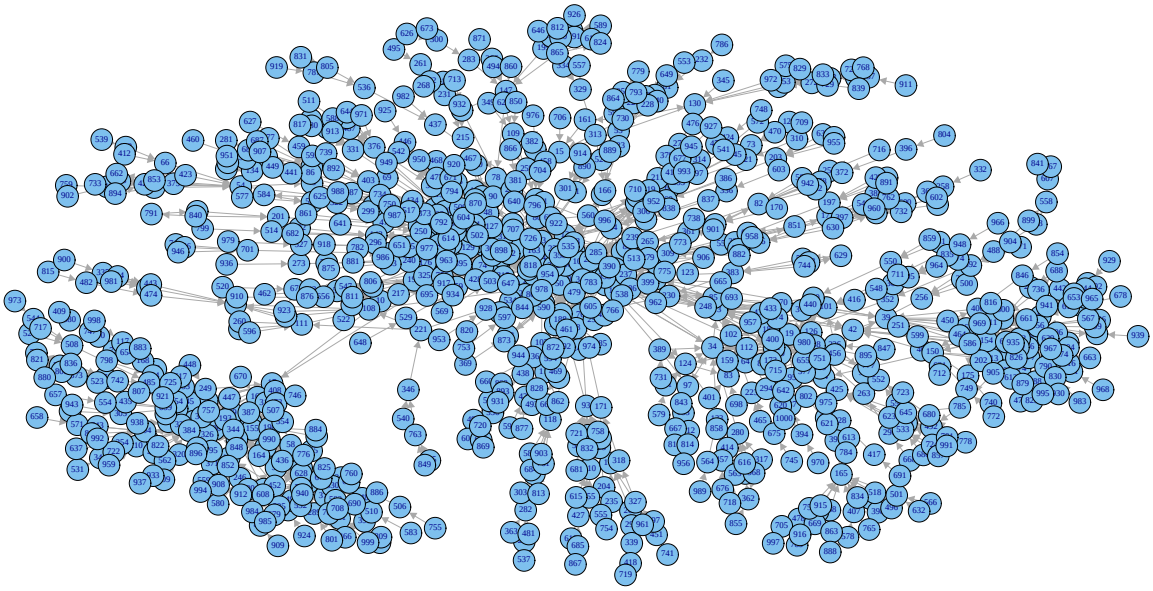


Figure B.24: Evolved Forest Fire Model with 1000 Vertices

Appendix C

Empirical Cumulative Distributions of Fitness Results by Experiment and Objective

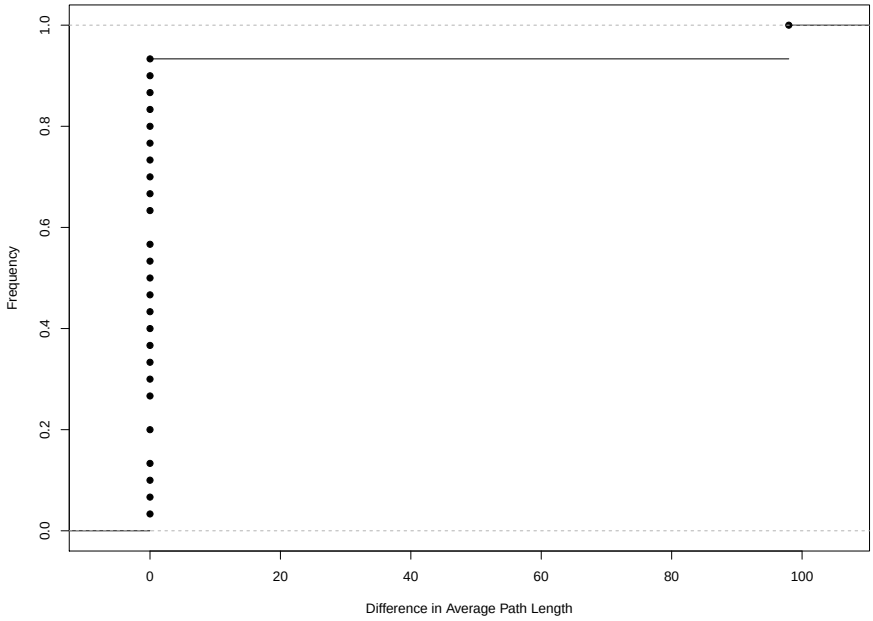


Figure C.1: Distribution of Differences in Average Geodesic Path Lengths for **GR-100**

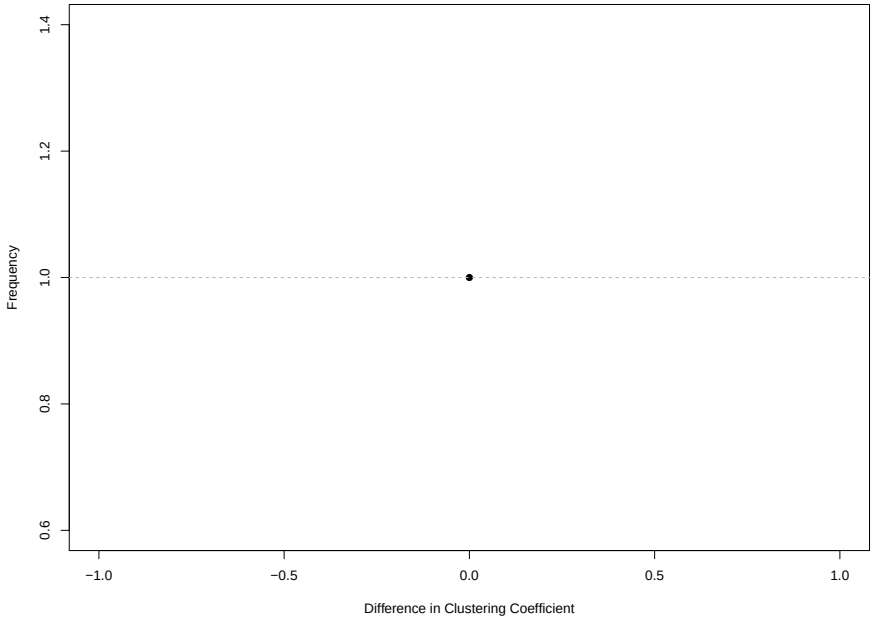


Figure C.2: Distribution of Differences in Clustering Coefficients for **GR-100**

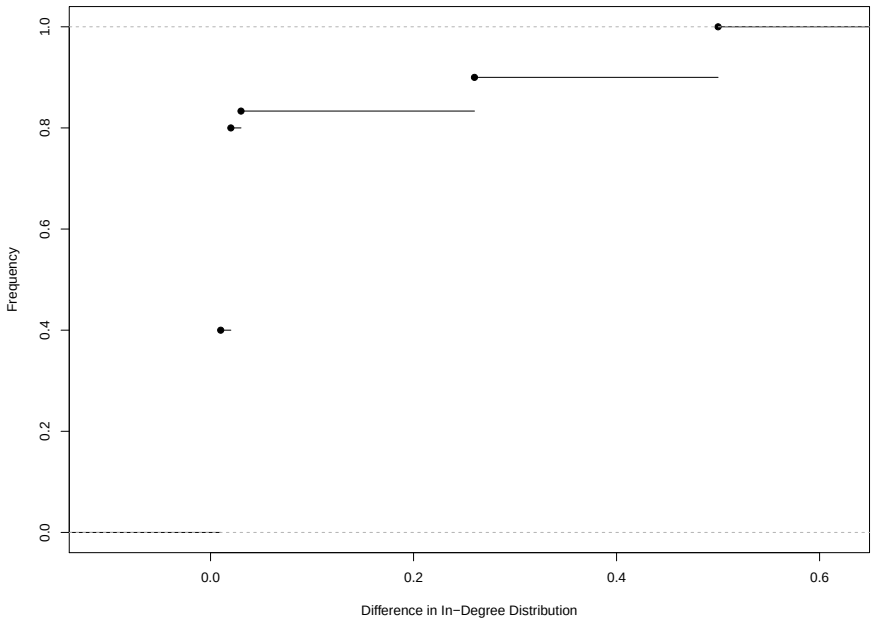


Figure C.3: Distribution of Differences in In-Degree Distributions for **GR-100**

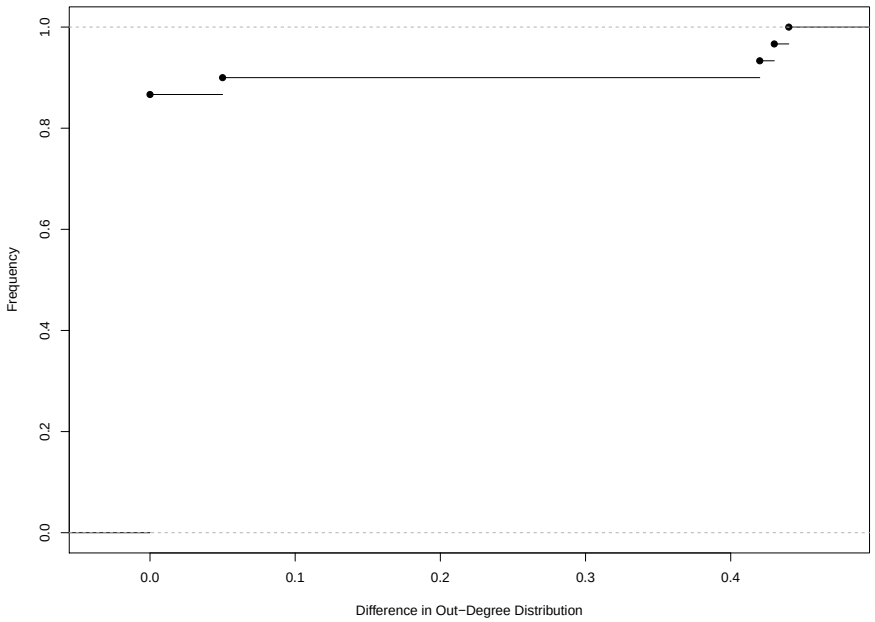


Figure C.4: Distribution of Differences in Out-Degree Distributions for **GR-100**

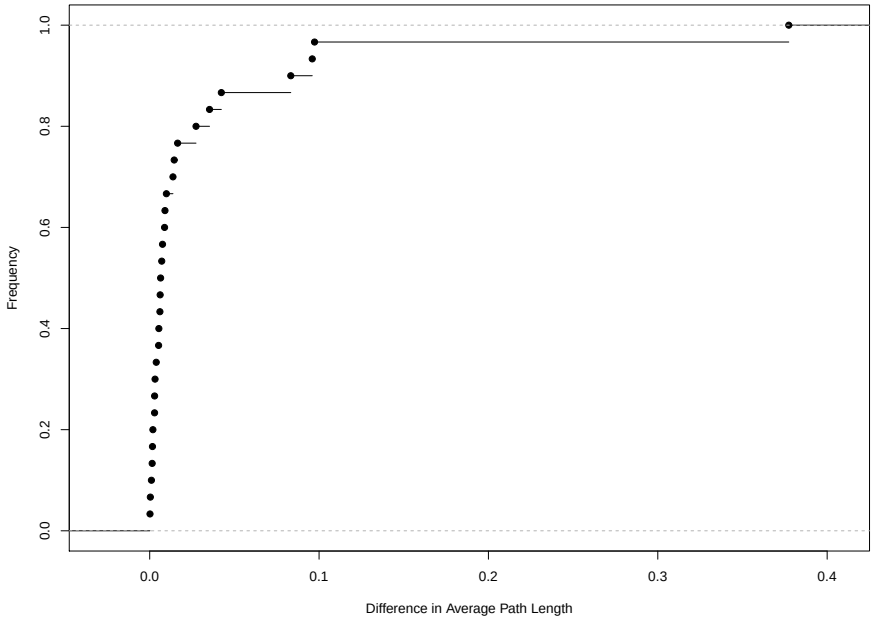


Figure C.5: Distribution of Differences in Average Geodesic Path Lengths for **GR-250**

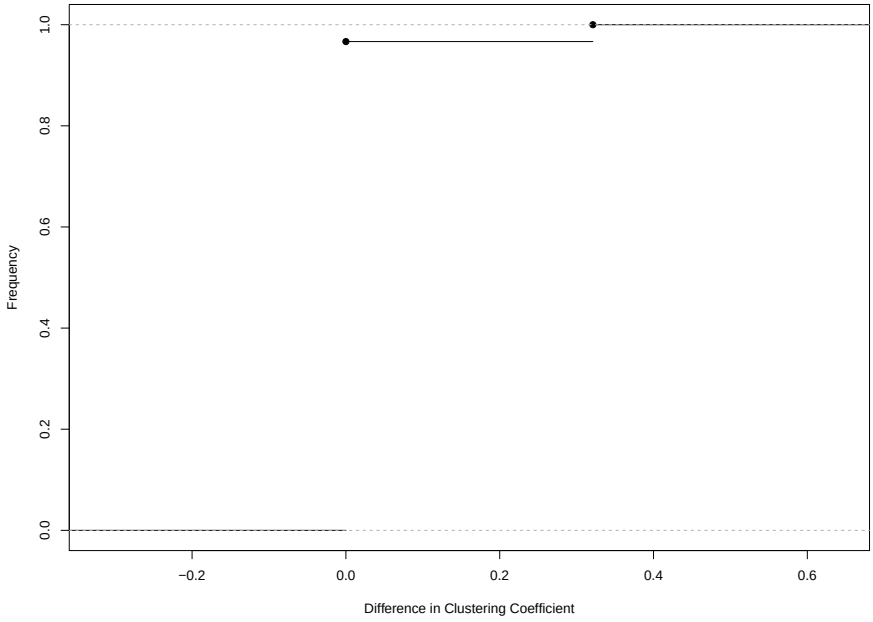


Figure C.6: Distribution of Differences in Clustering Coefficients for **GR-250**

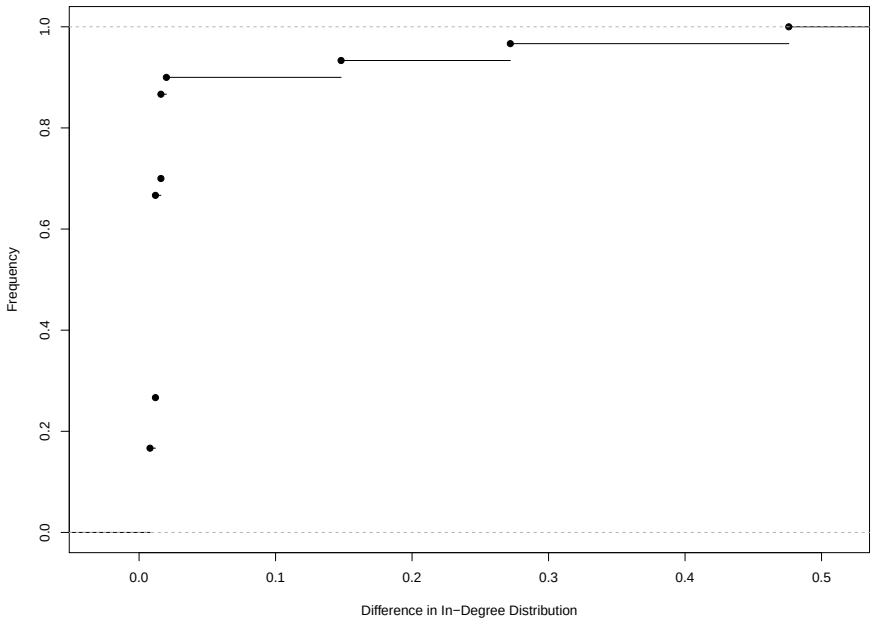


Figure C.7: Distribution of Differences in In-Degree Distributions for **GR-250**

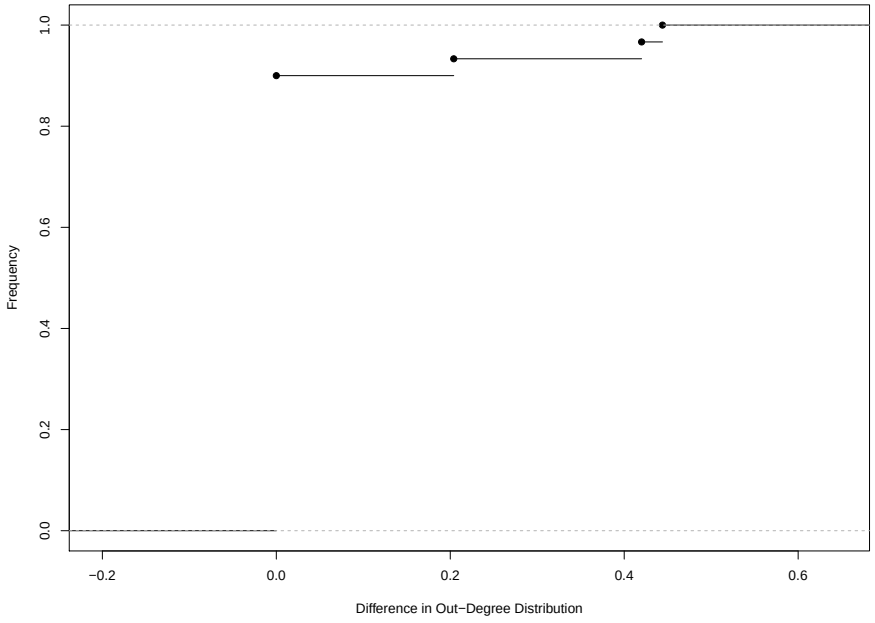


Figure C.8: Distribution of Differences in Out-Degree Distributions for **GR-250**

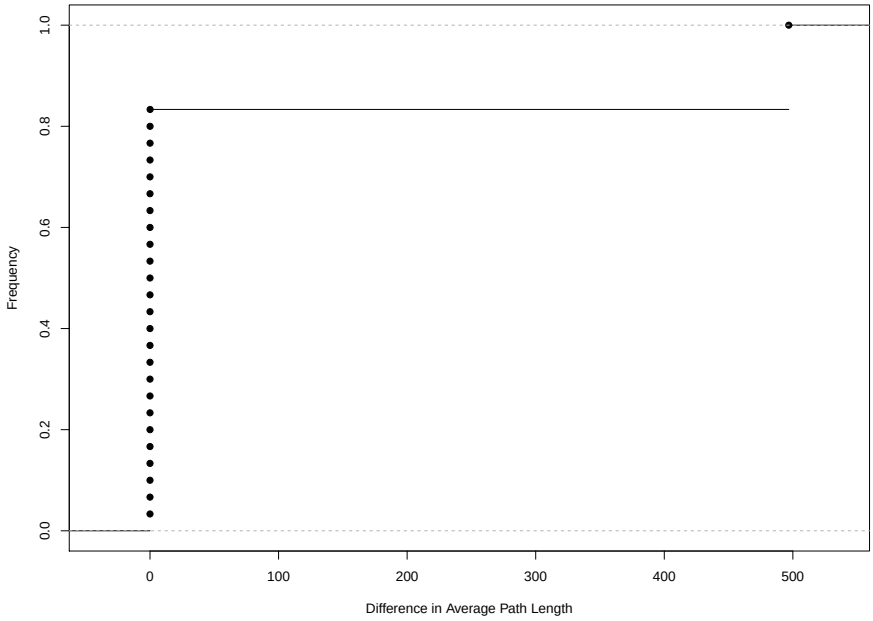


Figure C.9: Distribution of Differences in Average Geodesic Path Lengths for **GR-500**

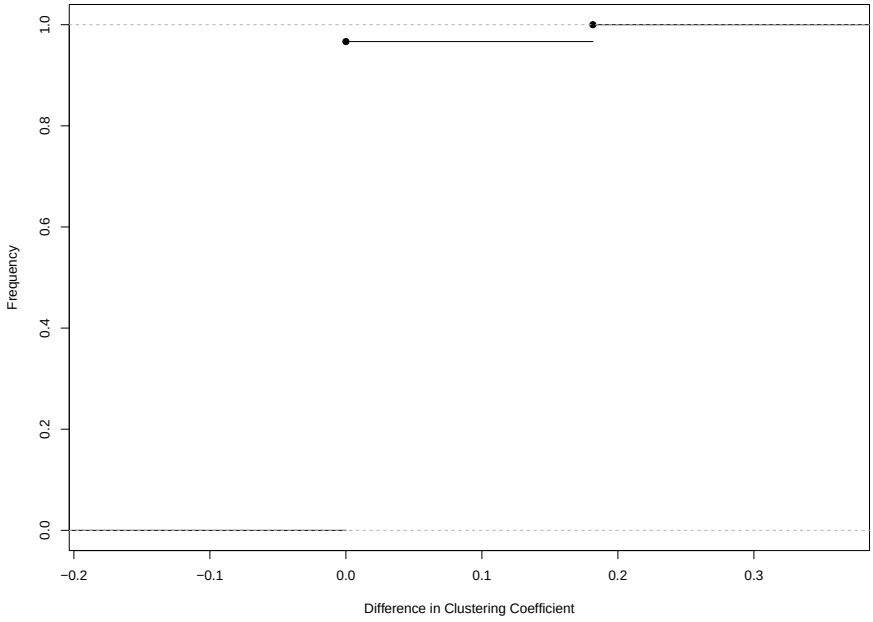


Figure C.10: Distribution of Differences in Clustering Coefficients for **GR-500**

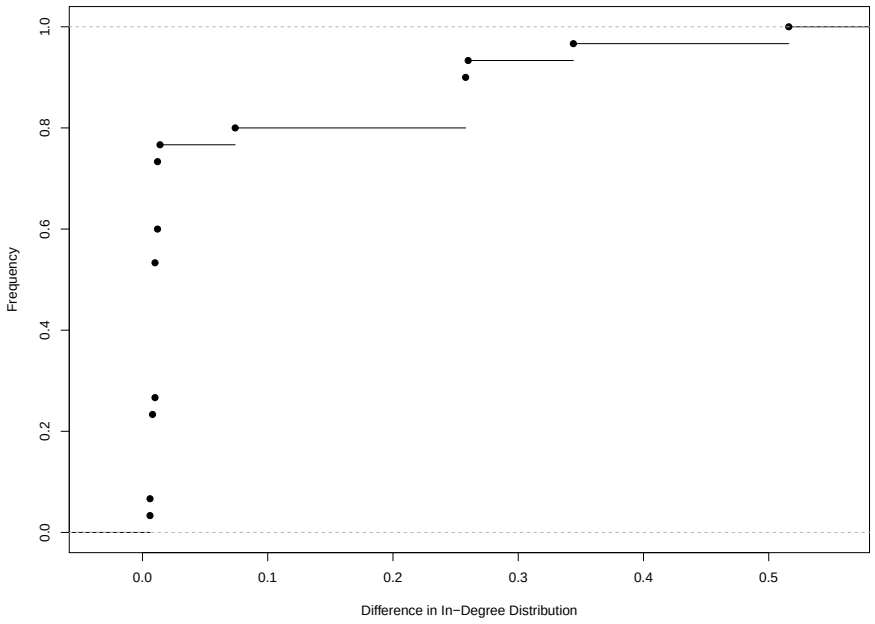


Figure C.11: Distribution of Differences in In-Degree Distributions for **GR-500**

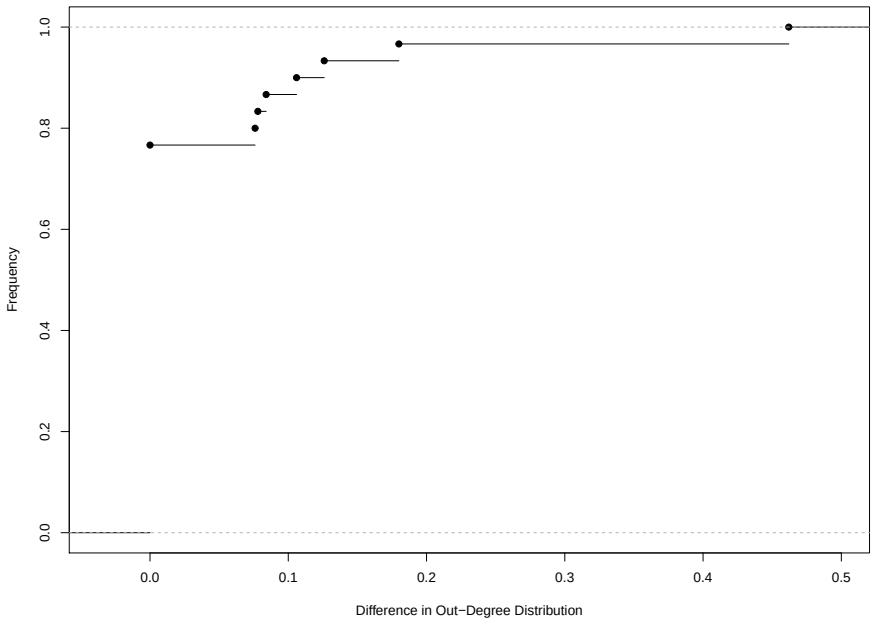


Figure C.12: Distribution of Differences in Out-Degree Distributions for **GR-500**

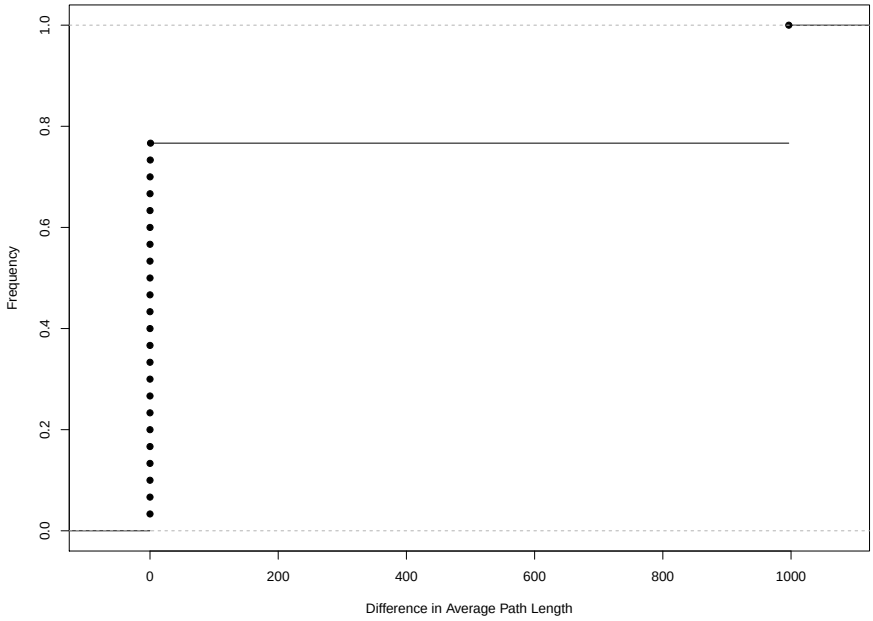


Figure C.13: Distribution of Differences in Average Geodesic Path Lengths for **GR-1000**

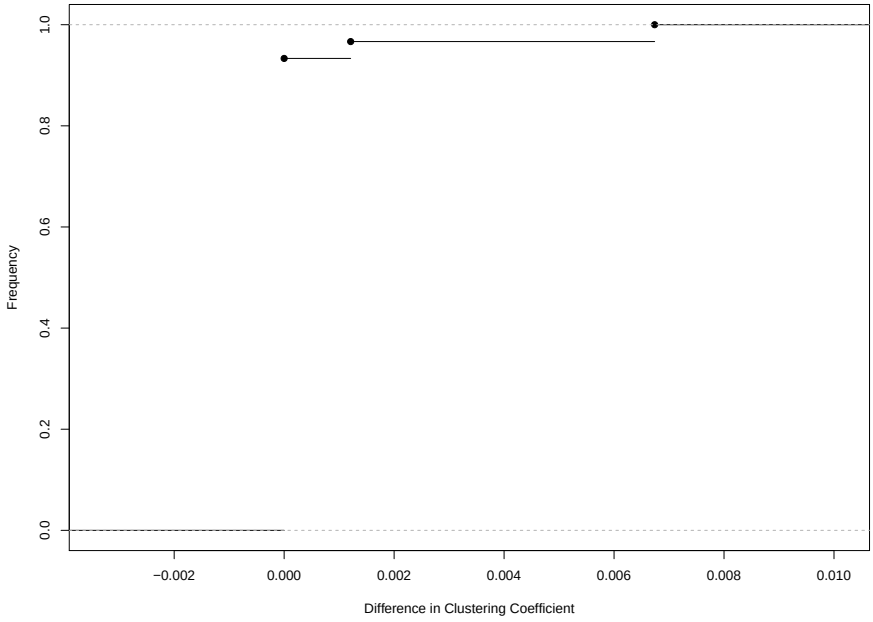


Figure C.14: Distribution of Differences in Clustering Coefficients for **GR-1000**

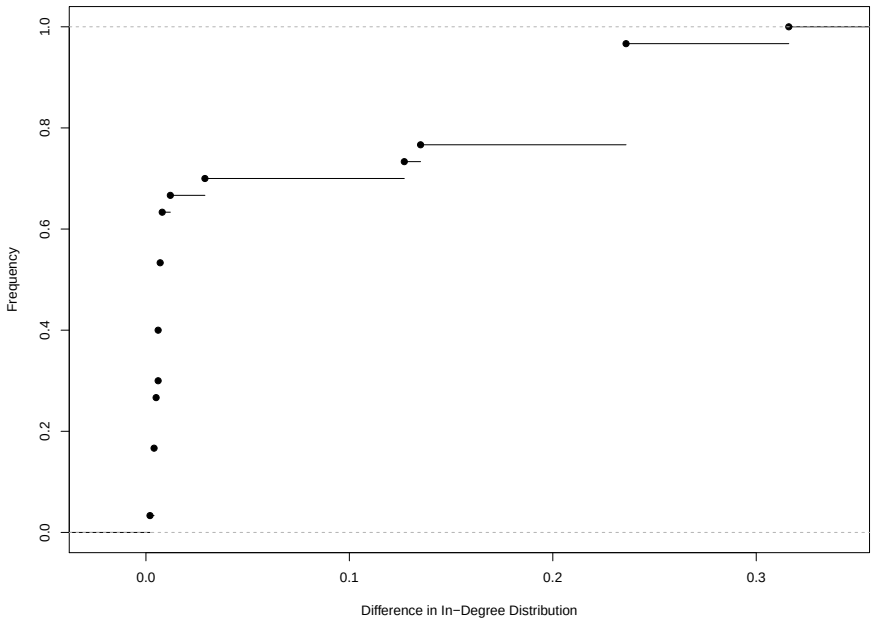


Figure C.15: Distribution of Differences in In-Degree Distributions for **GR-1000**

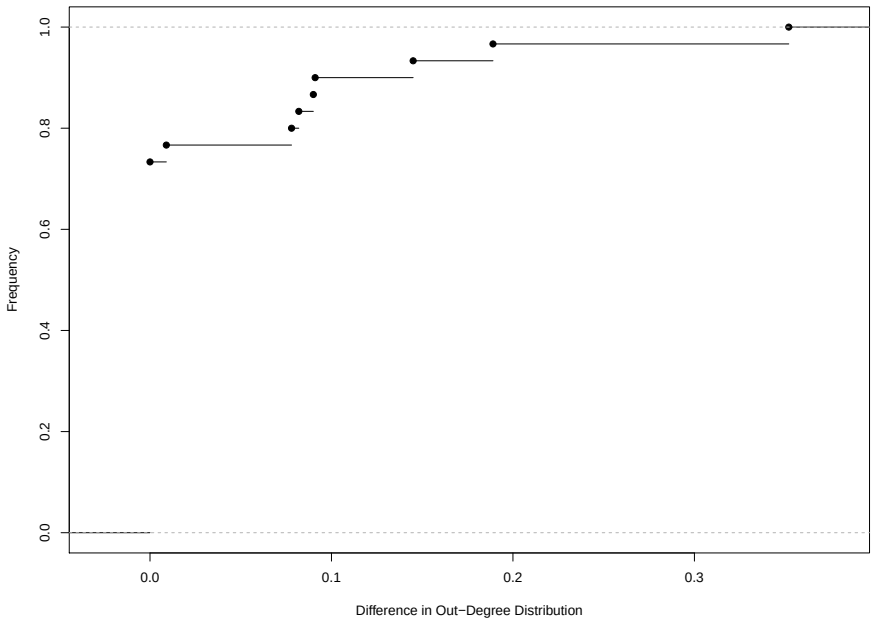


Figure C.16: Distribution of Differences in Out-Degree Distributions for **GR-1000**

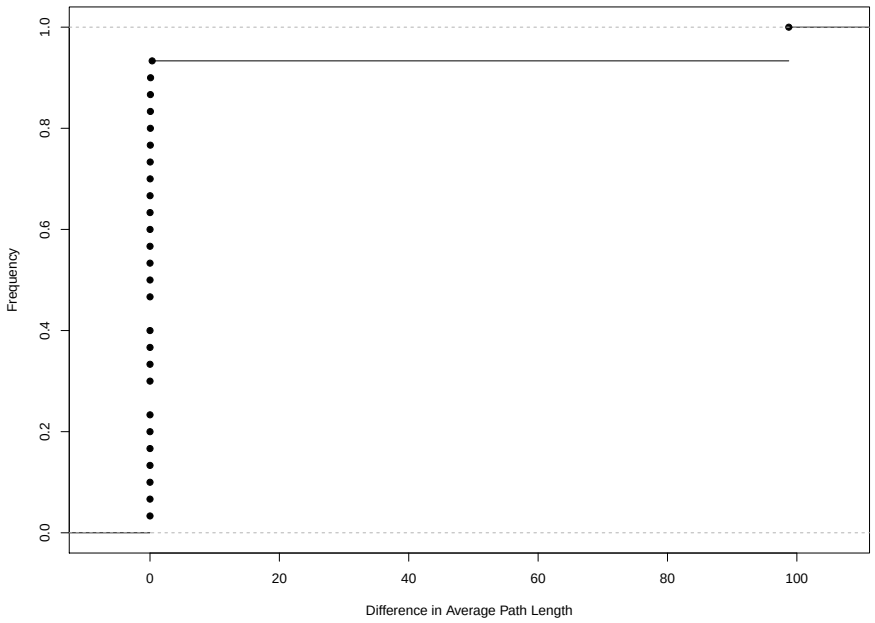


Figure C.17: Distribution of Differences in Average Geodesic Path Lengths for **BA-100**

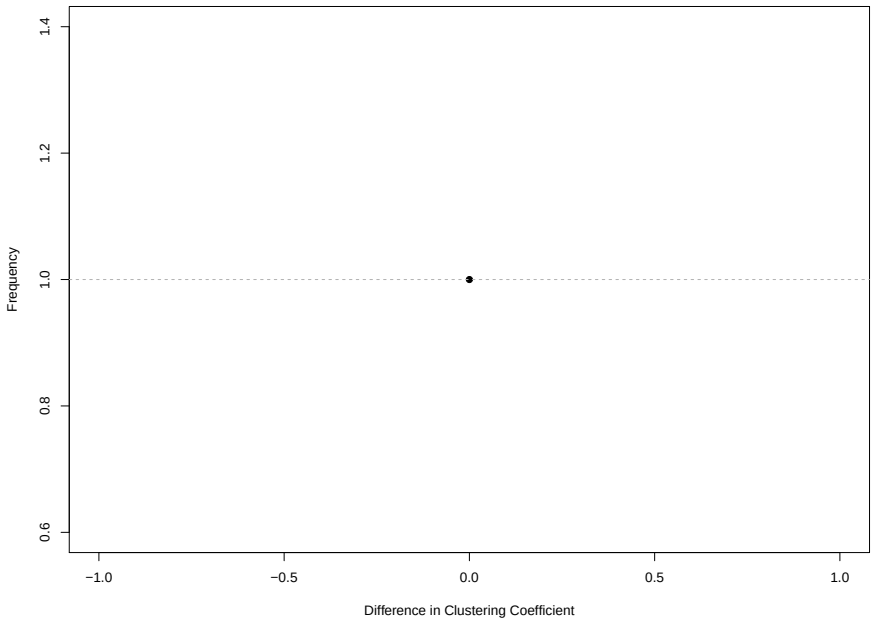


Figure C.18: Distribution of Differences in Clustering Coefficients for **BA-100**

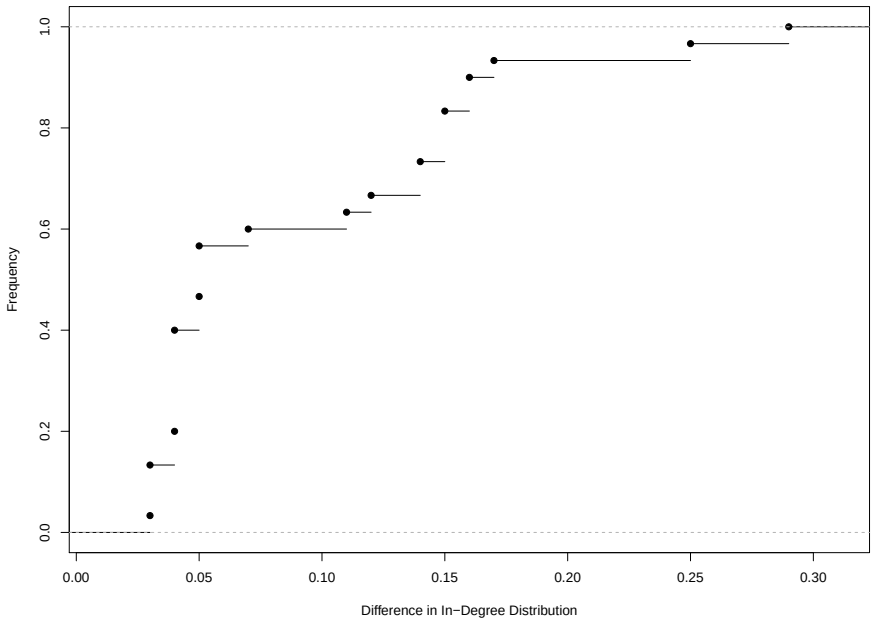


Figure C.19: Distribution of Differences in In-Degree Distributions for **BA-100**

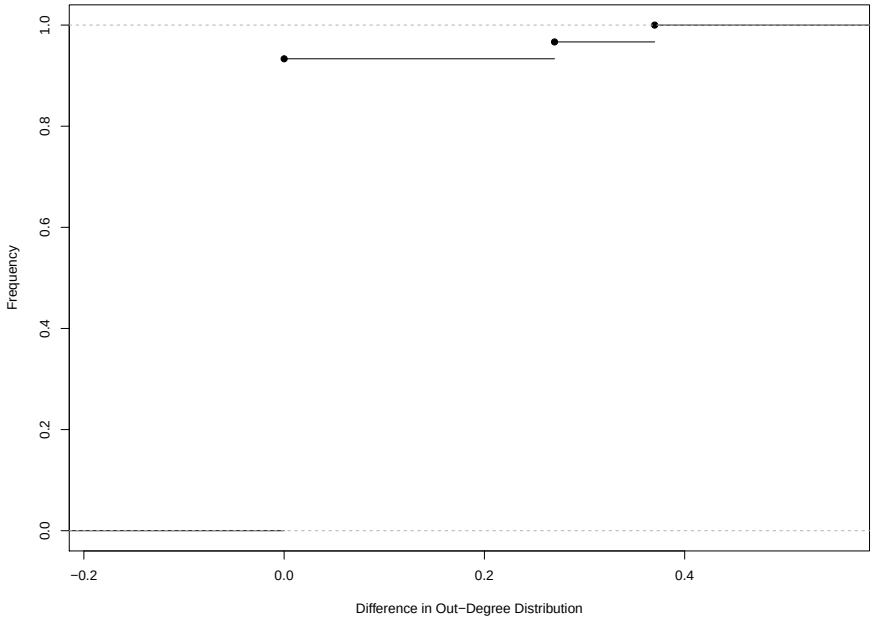


Figure C.20: Distribution of Differences in Out-Degree Distributions for **BA-100**

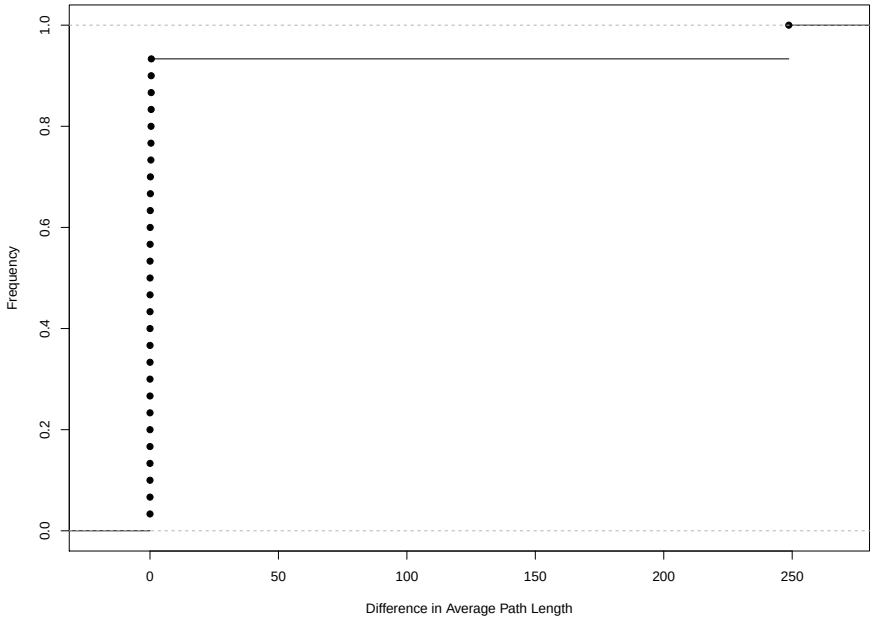


Figure C.21: Distribution of Differences in Average Geodesic Path Lengths for **BA-250**

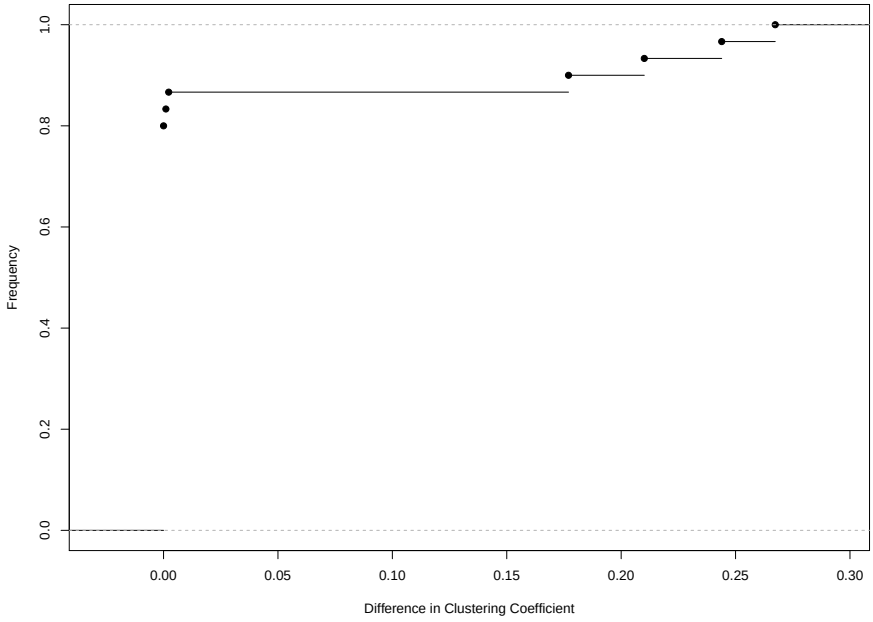


Figure C.22: Distribution of Differences in Clustering Coefficients for **BA-250**

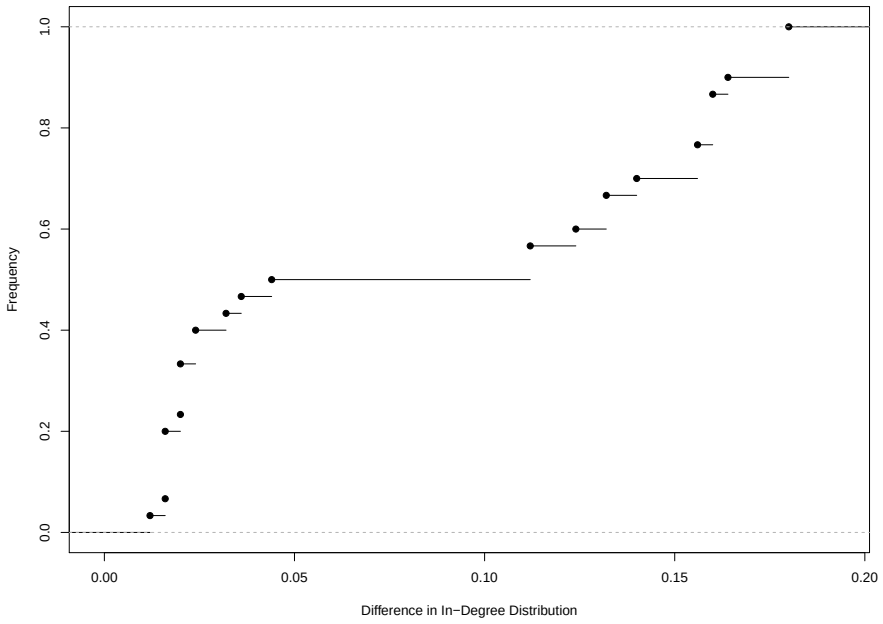


Figure C.23: Distribution of Differences in In-Degree Distributions for **BA-250**

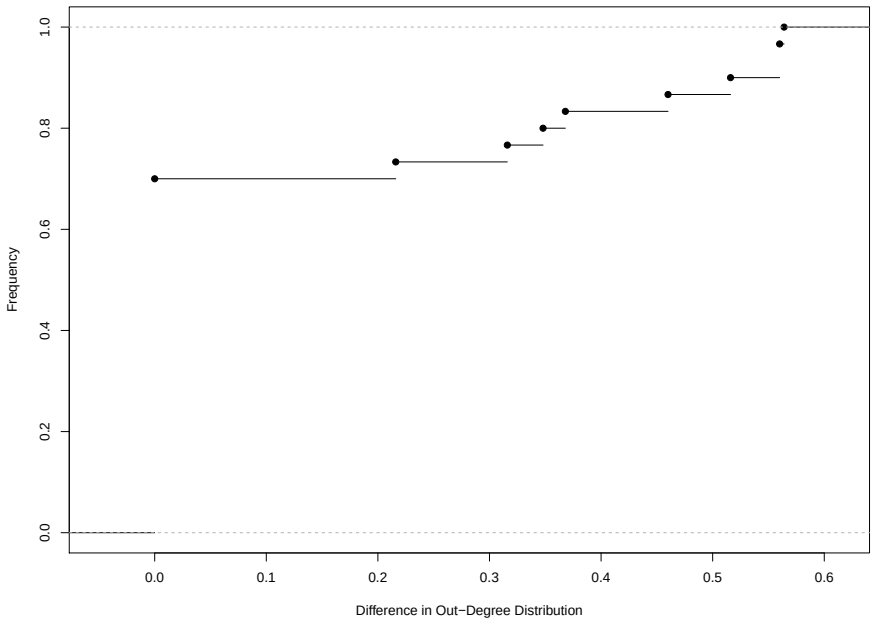


Figure C.24: Distribution of Differences in Out-Degree Distributions for **BA-250**

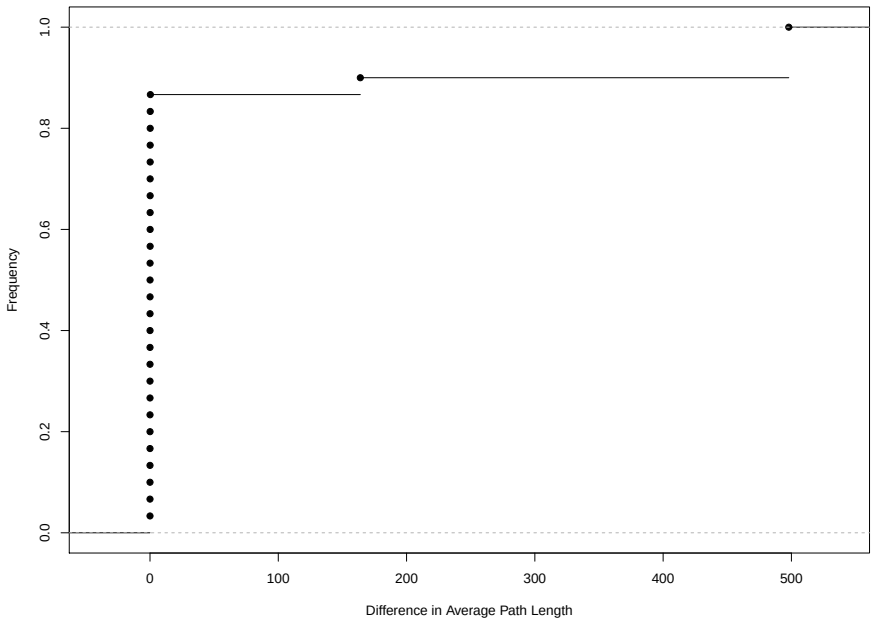


Figure C.25: Distribution of Differences in Average Geodesic Path Lengths for **BA-500**

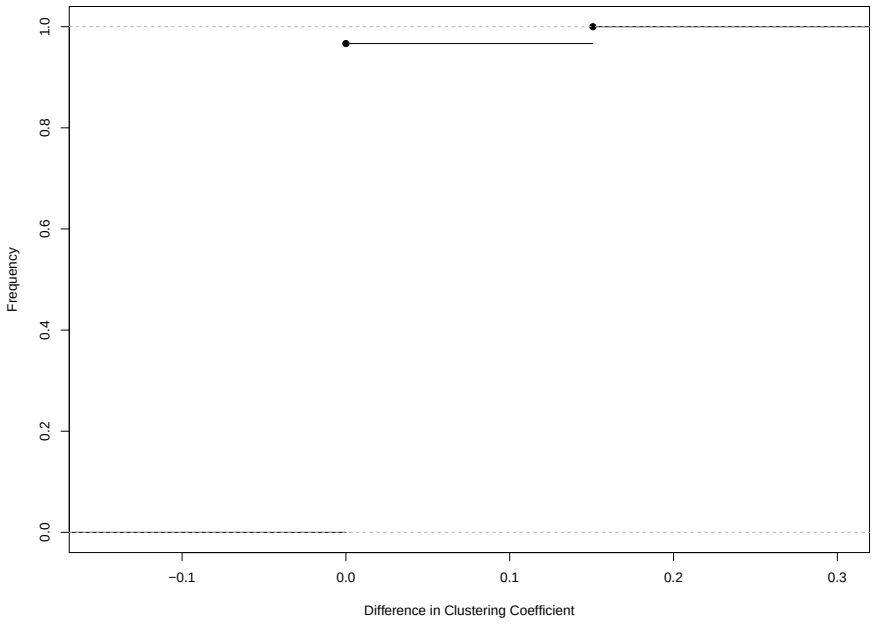


Figure C.26: Distribution of Differences in Clustering Coefficients for **BA-500**

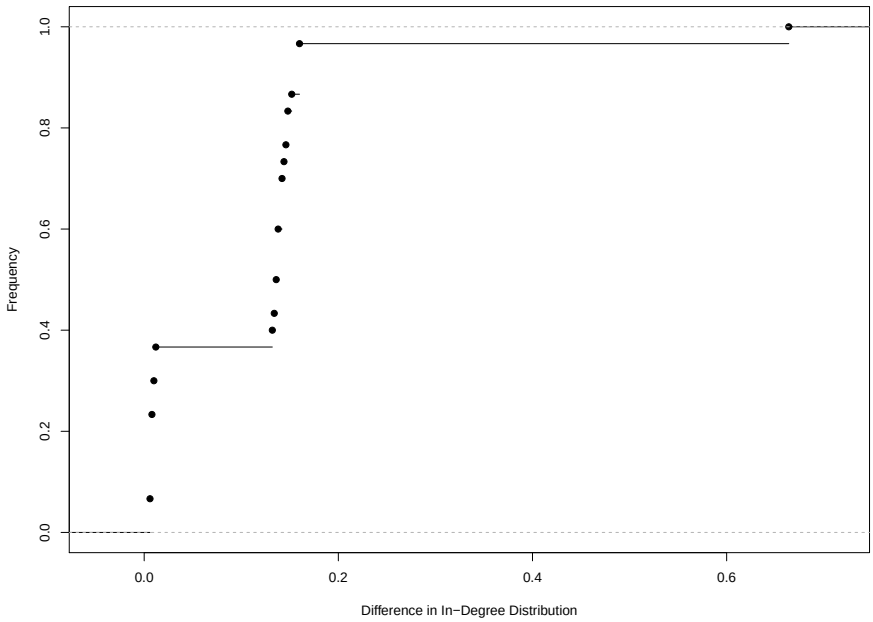


Figure C.27: Distribution of Differences in In-Degree Distributions for **BA-500**

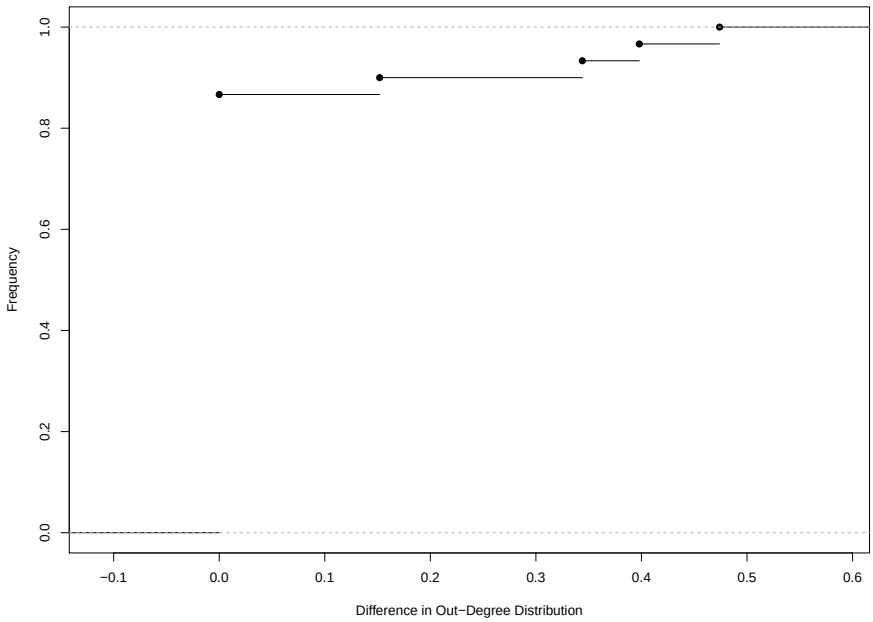


Figure C.28: Distribution of Differences in Out-Degree Distributions for **BA-500**

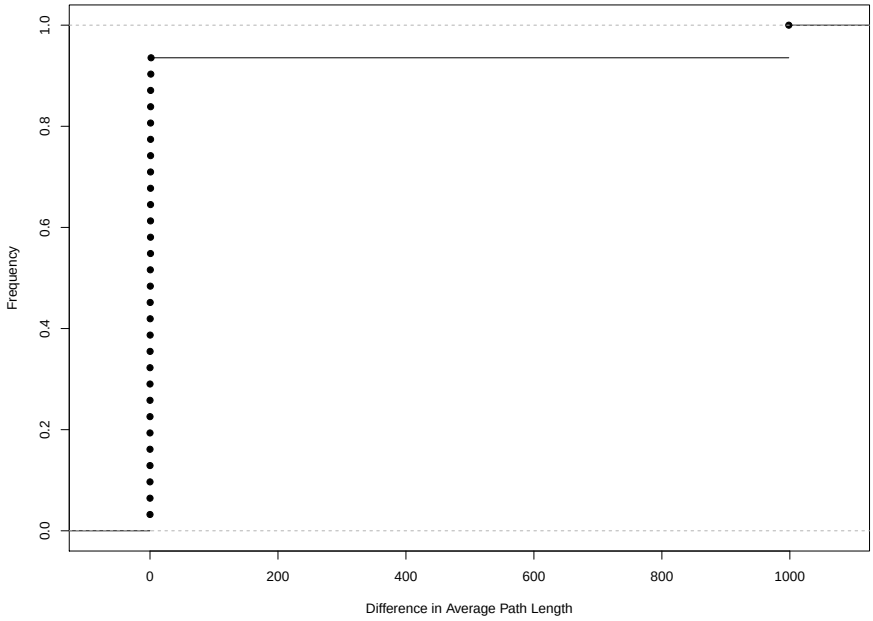


Figure C.29: Distribution of Differences in Average Geodesic Path Lengths for **BA-1000**

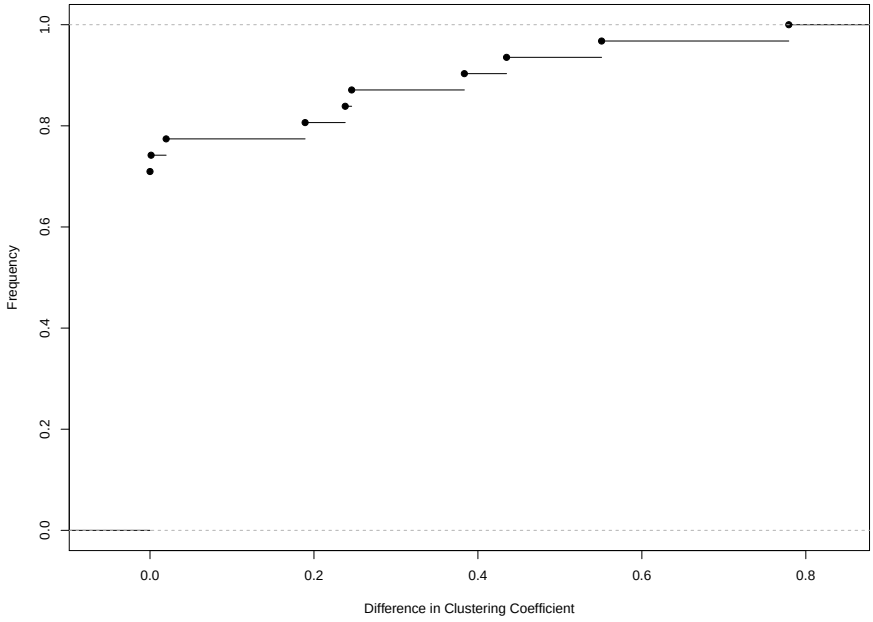


Figure C.30: Distribution of Differences in Clustering Coefficients for **BA-1000**

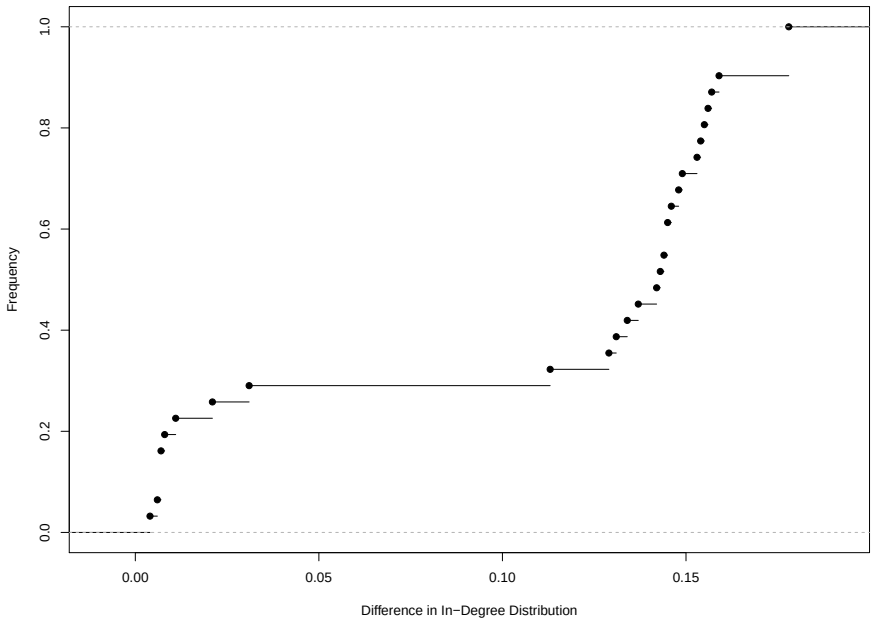


Figure C.31: Distribution of Differences in In-Degree Distributions for **BA-1000**

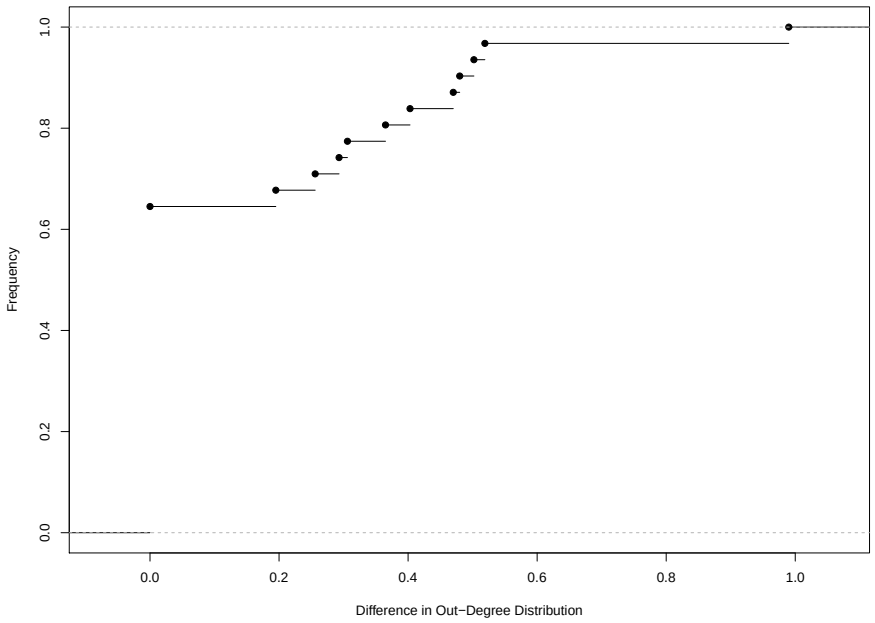


Figure C.32: Distribution of Differences in Out-Degree Distributions for **BA-1000**

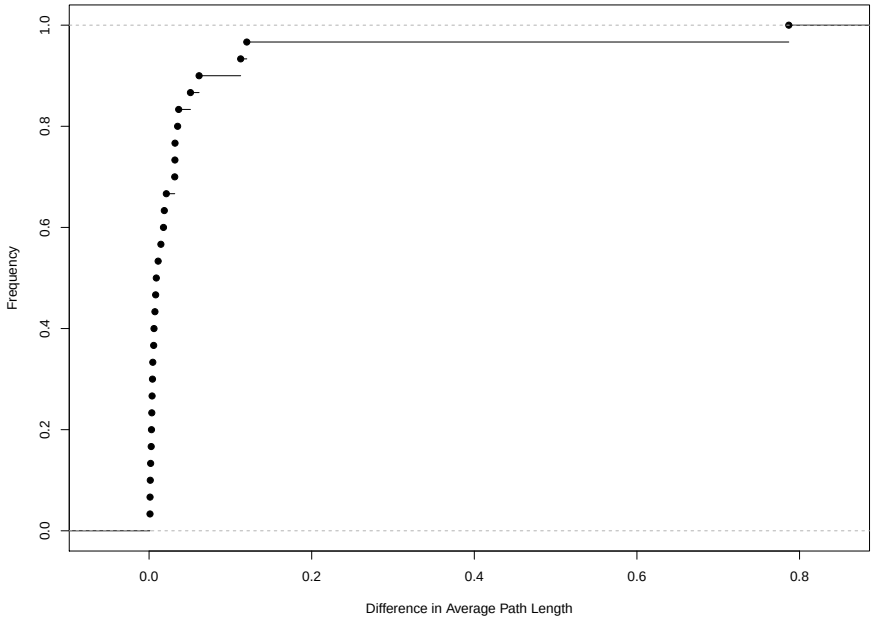


Figure C.33: Distribution of Differences in Average Geodesic Path Lengths for **FF-100**

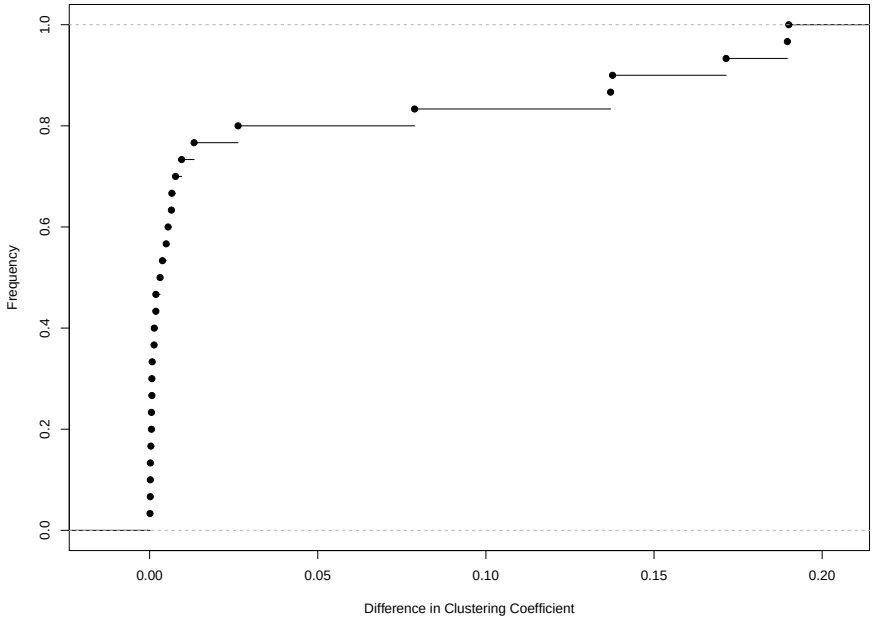


Figure C.34: Distribution of Differences in Clustering Coefficients for **FF-100**

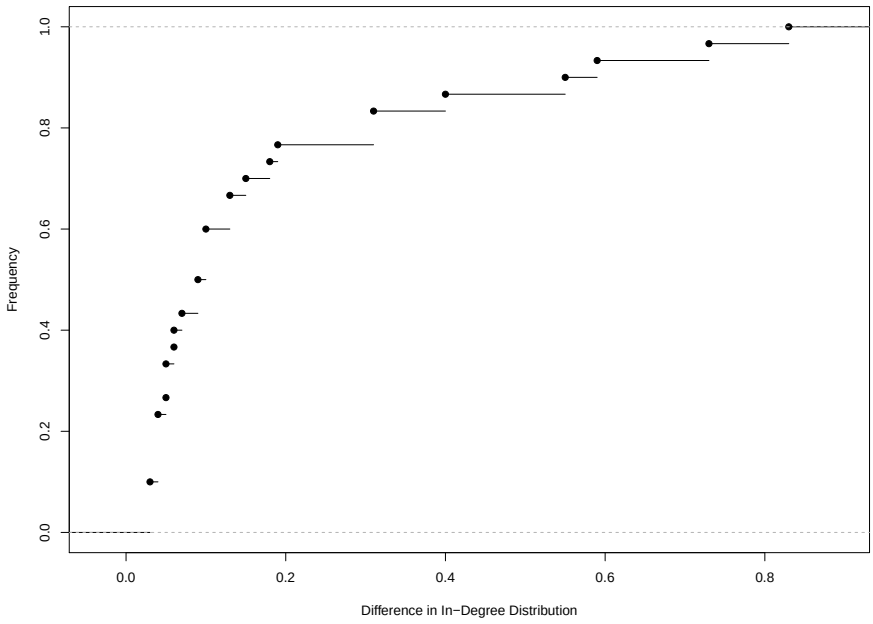


Figure C.35: Distribution of Differences in In-Degree Distributions for **FF-100**

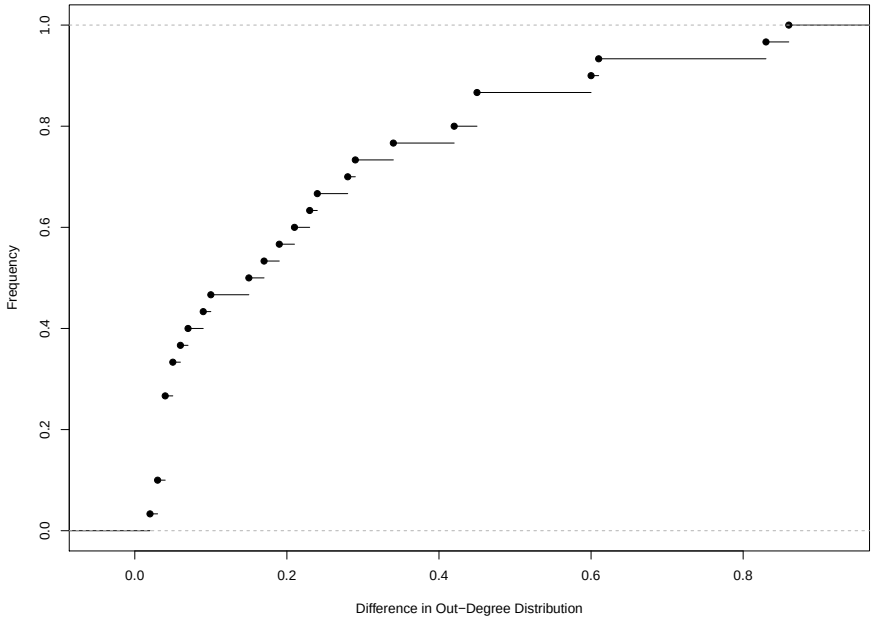


Figure C.36: Distribution of Differences in Out-Degree Distributions for **FF-100**

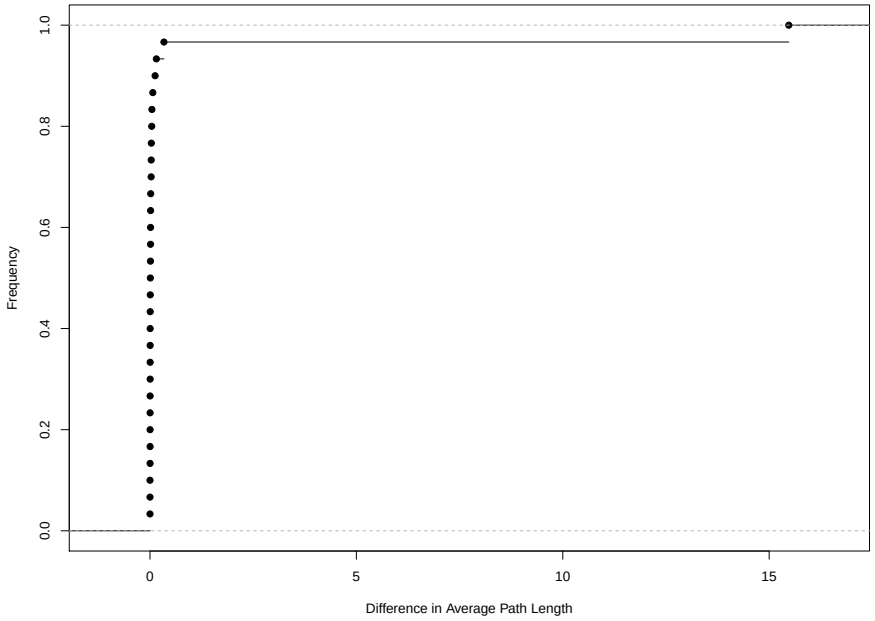


Figure C.37: Distribution of Differences in Average Geodesic Path Lengths for **FF-250**

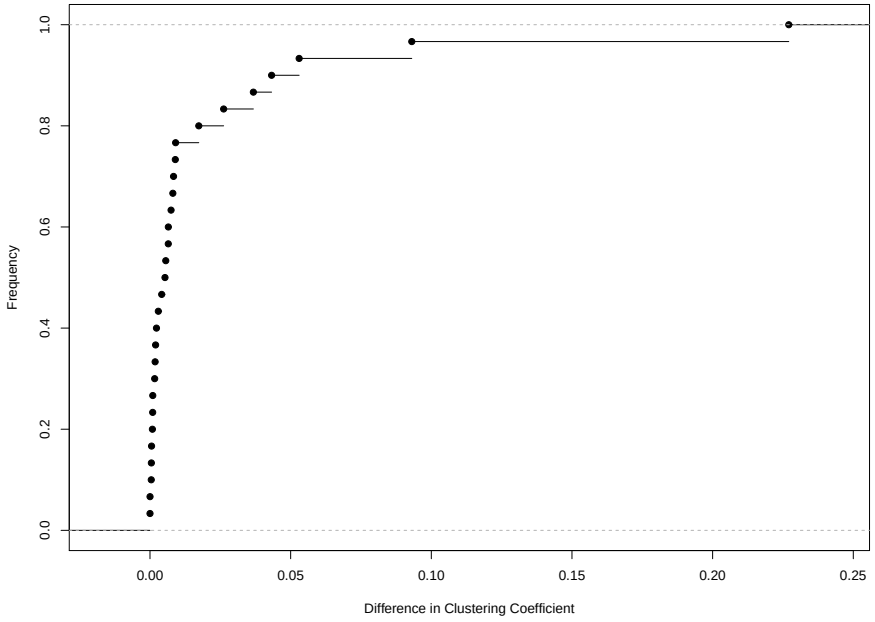


Figure C.38: Distribution of Differences in Clustering Coefficients for **FF-250**

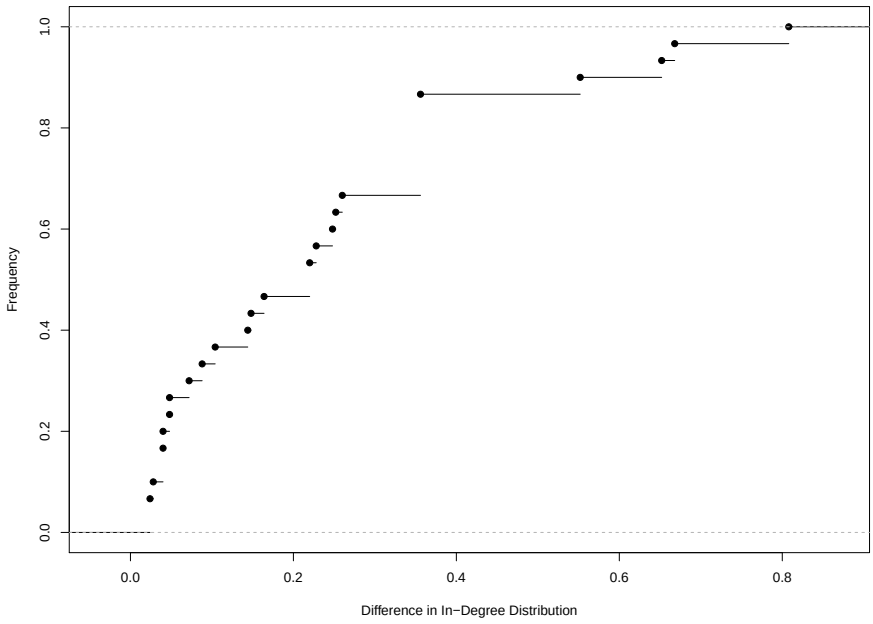


Figure C.39: Distribution of Differences in In-Degree Distributions for **FF-250**

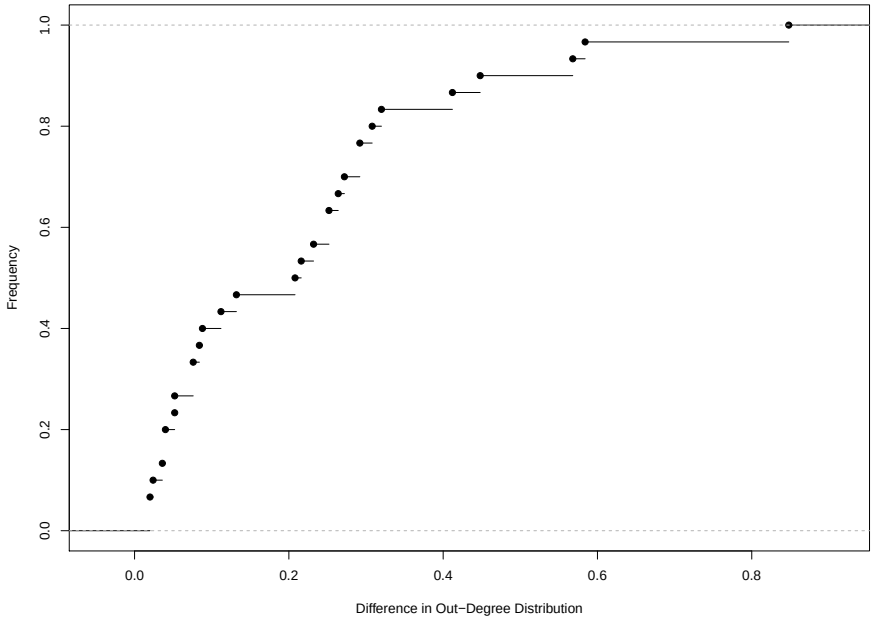


Figure C.40: Distribution of Differences in Out-Degree Distributions for **FF-250**

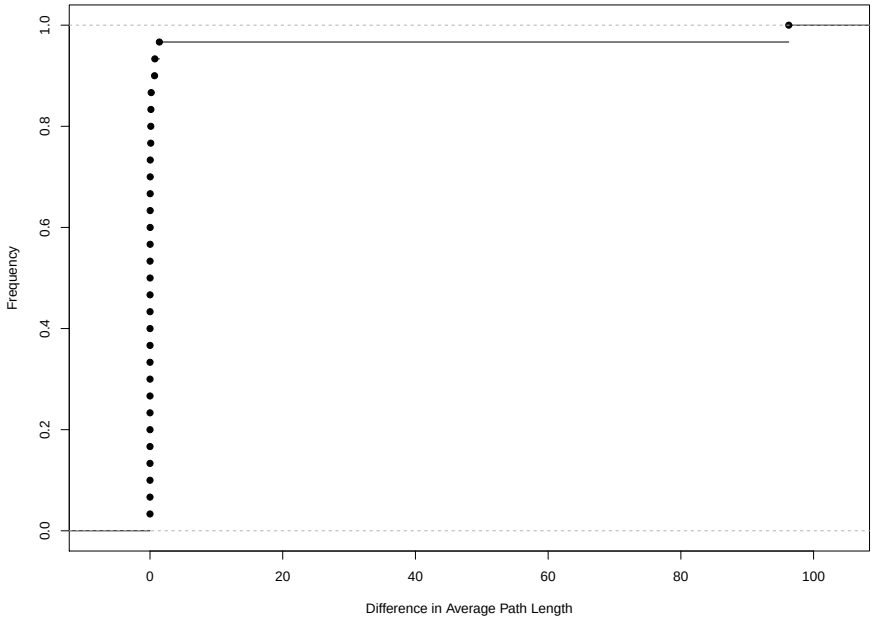


Figure C.41: Distribution of Differences in Average Geodesic Path Lengths for **FF-500**

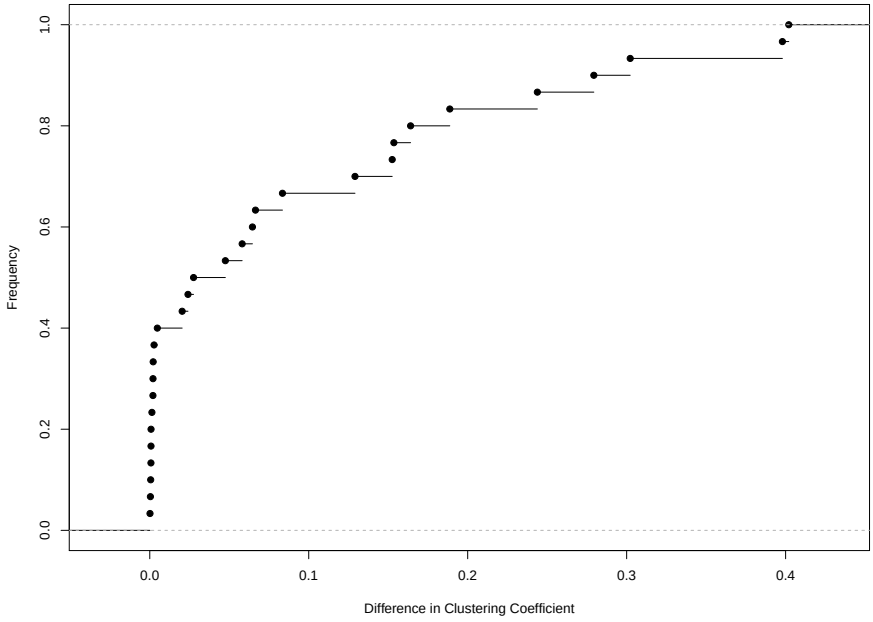


Figure C.42: Distribution of Differences in Clustering Coefficients for **FF-500**

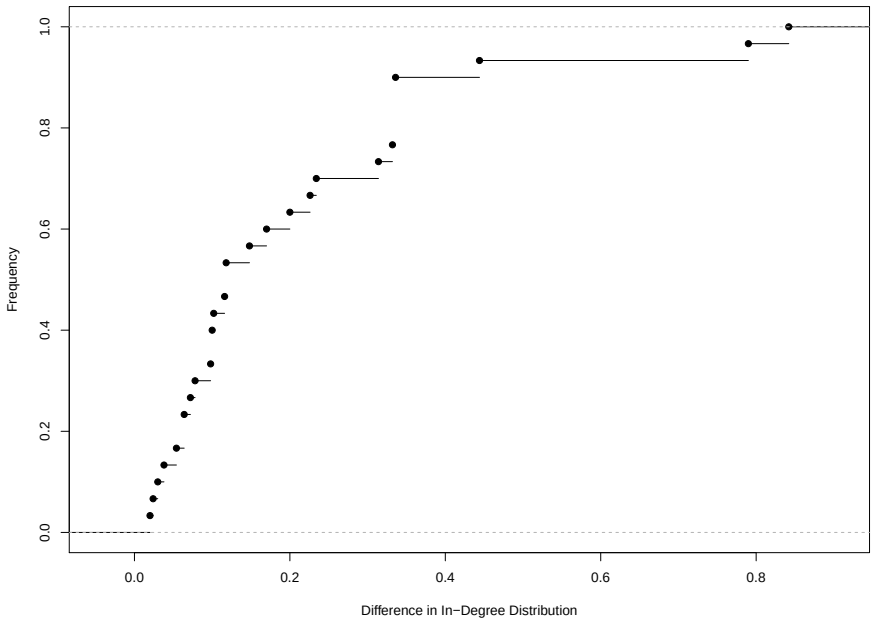


Figure C.43: Distribution of Differences in In-Degree Distributions for **FF-500**

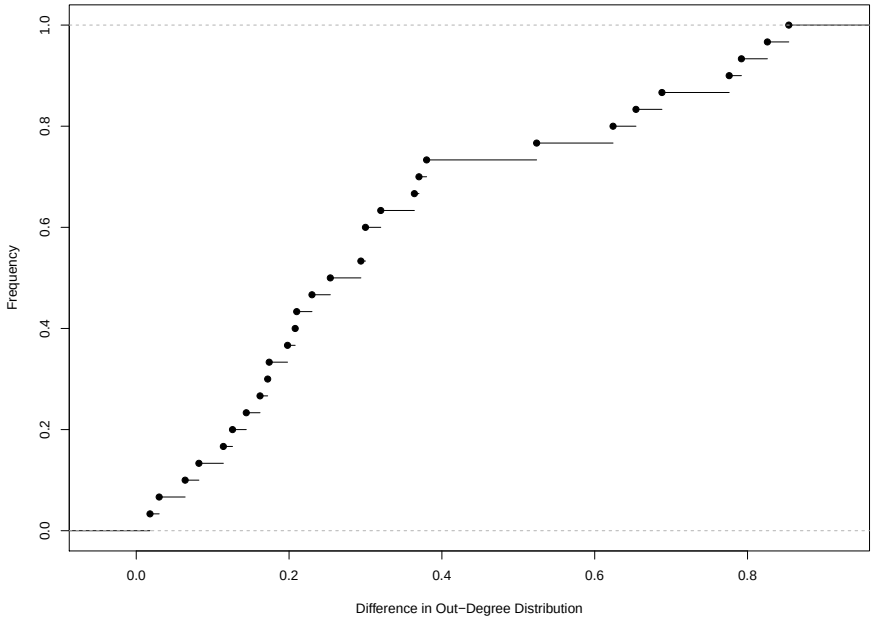


Figure C.44: Distribution of Differences in Out-Degree Distributions for **FF-500**

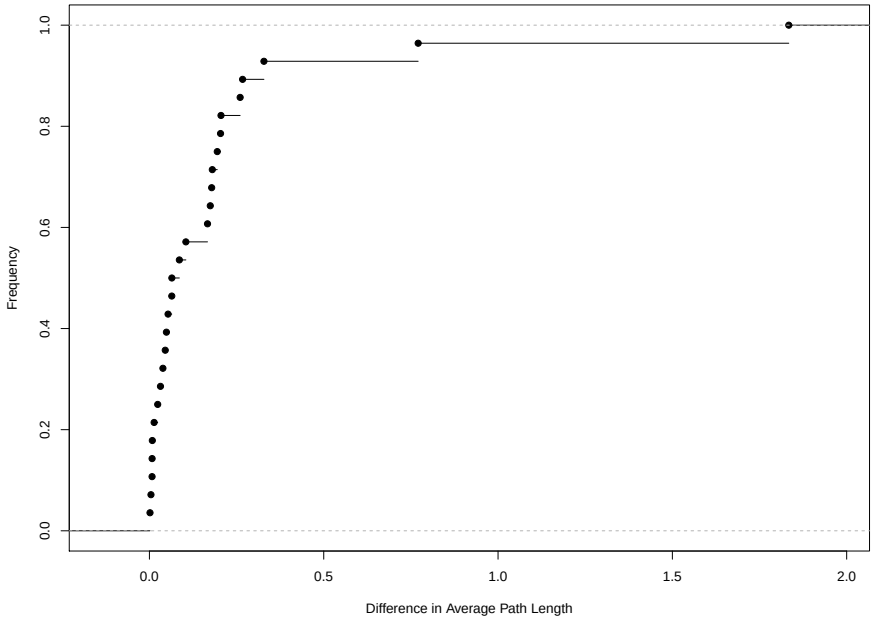


Figure C.45: Distribution of Differences in Average Geodesic Path Lengths for **FF-1000**

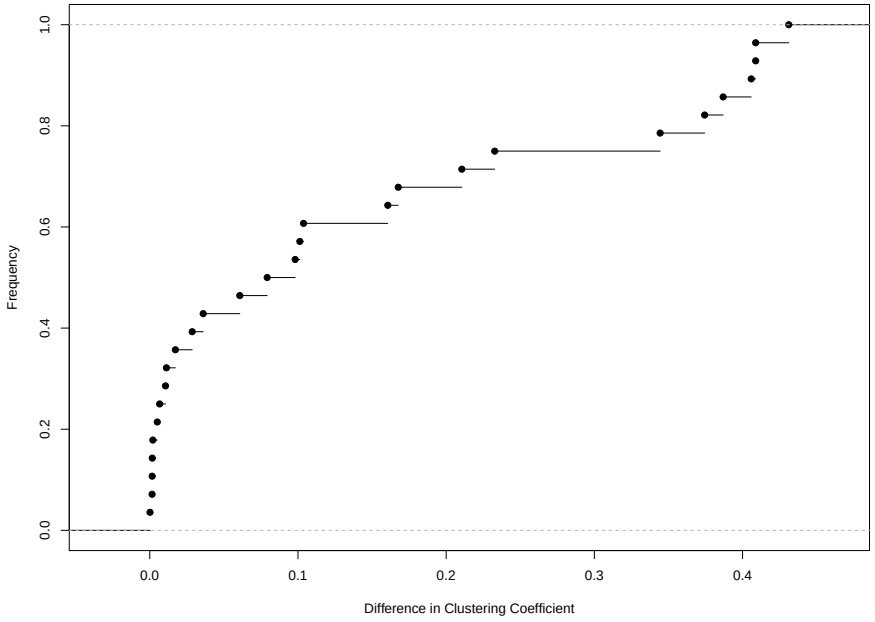


Figure C.46: Distribution of Differences in Clustering Coefficients for **FF-1000**

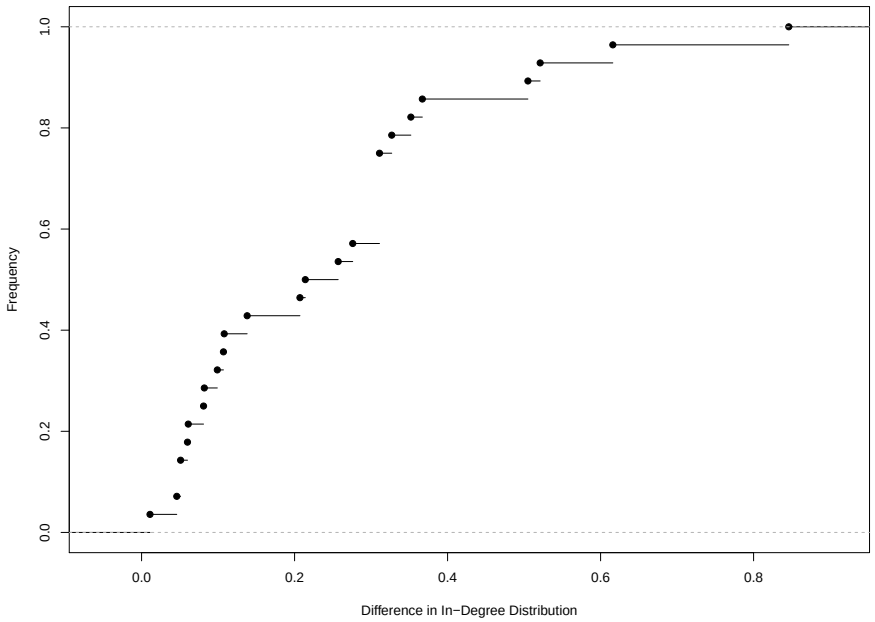


Figure C.47: Distribution of Differences in In-Degree Distributions for **FF-1000**

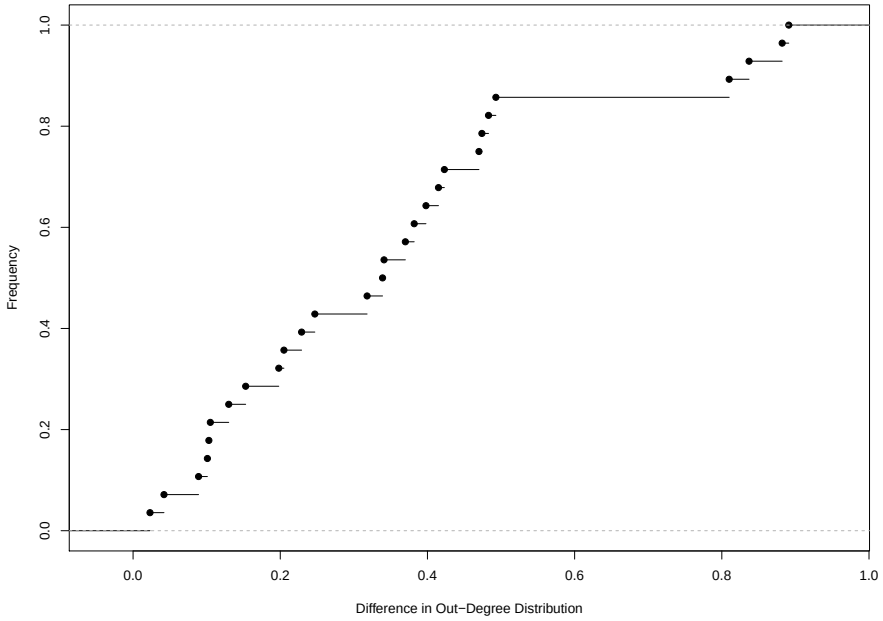


Figure C.48: Distribution of Differences in Out-Degree Distributions for **FF-1000**

Appendix D

GP Convergence Plots

In this appendix, the average best fitness values by generation are presented for all runs of each experiment. The fitness values have been normalized using the maximum and minimum value of the averages. This was done so that each fitness value was easily plotted in the same chart.

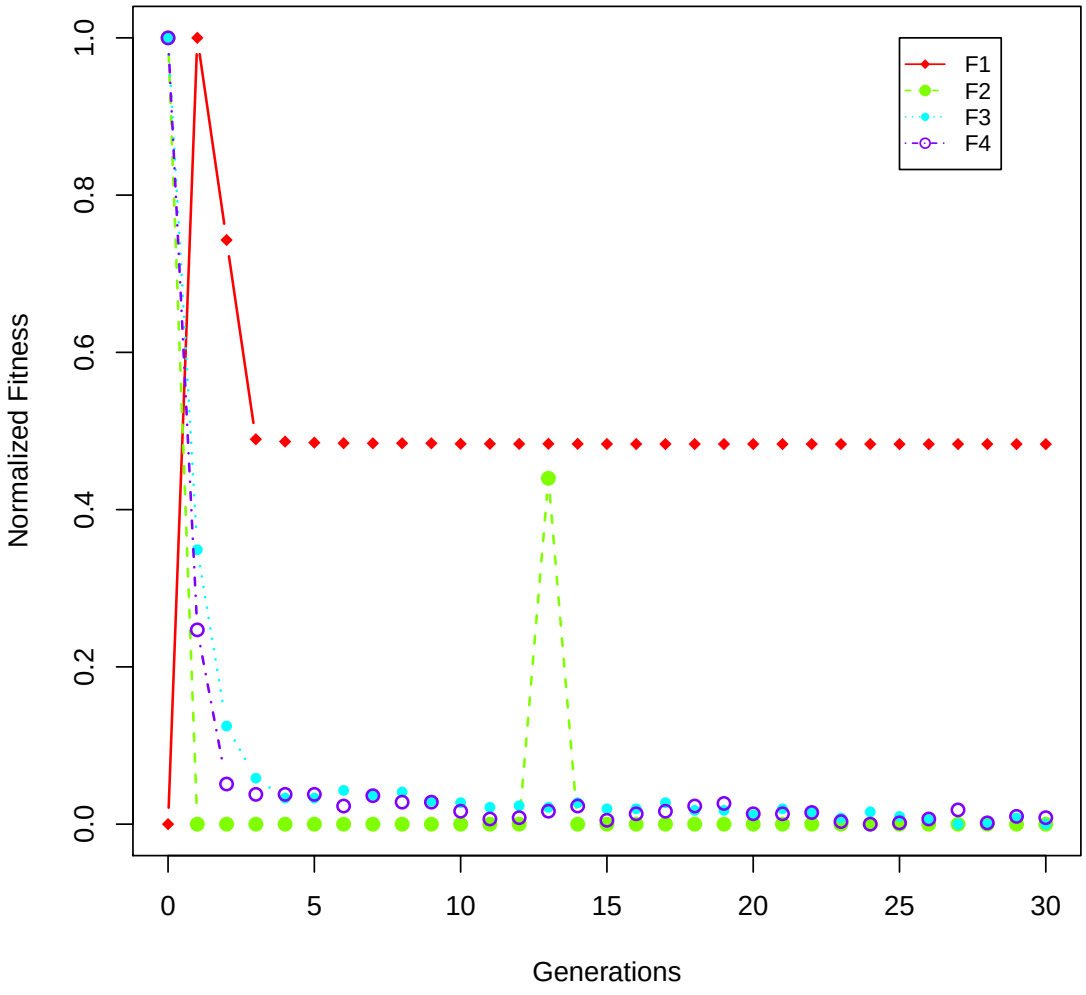


Figure D.1: **GR-100** Convergence Plot

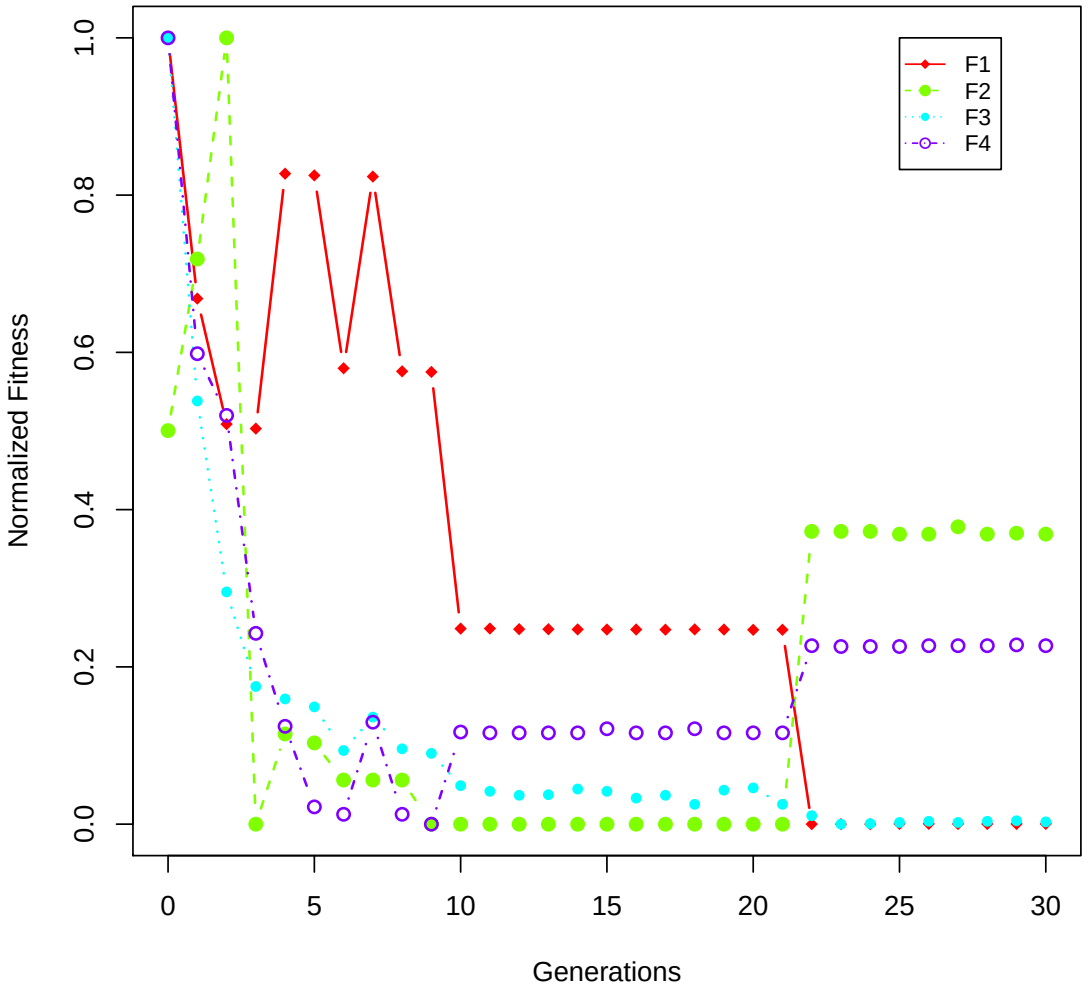


Figure D.2: **GR-250** Convergence Plot

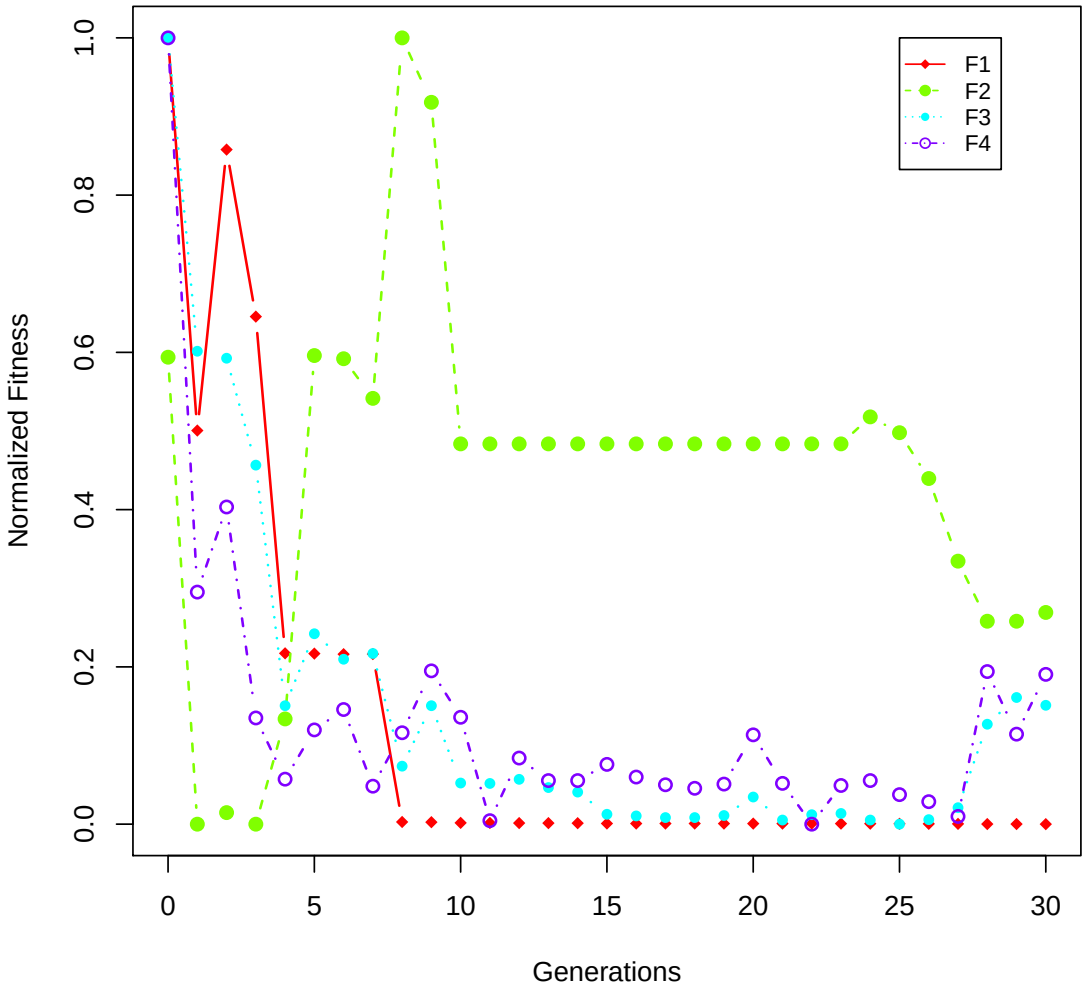


Figure D.3: **GR-500** Convergence Plot

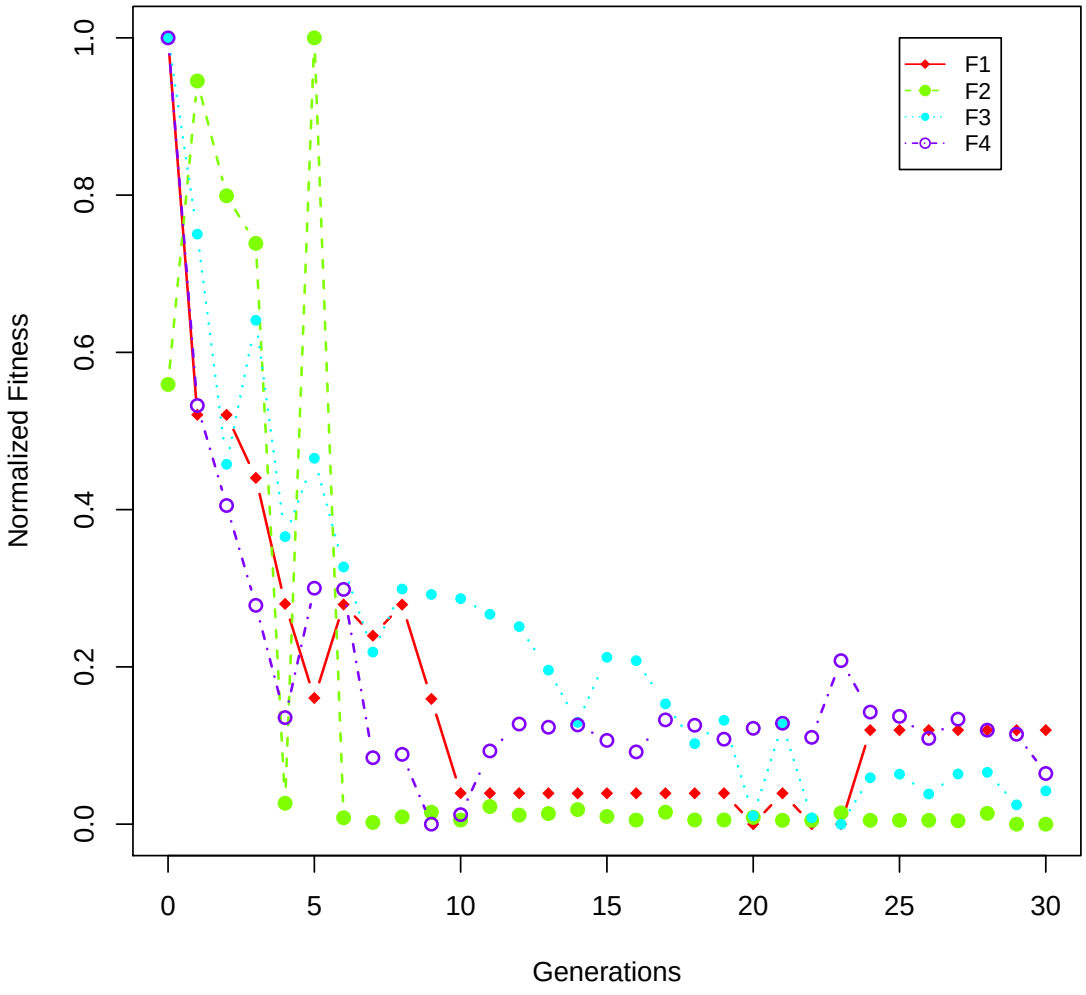


Figure D.4: **GR-1000** Convergence Plot

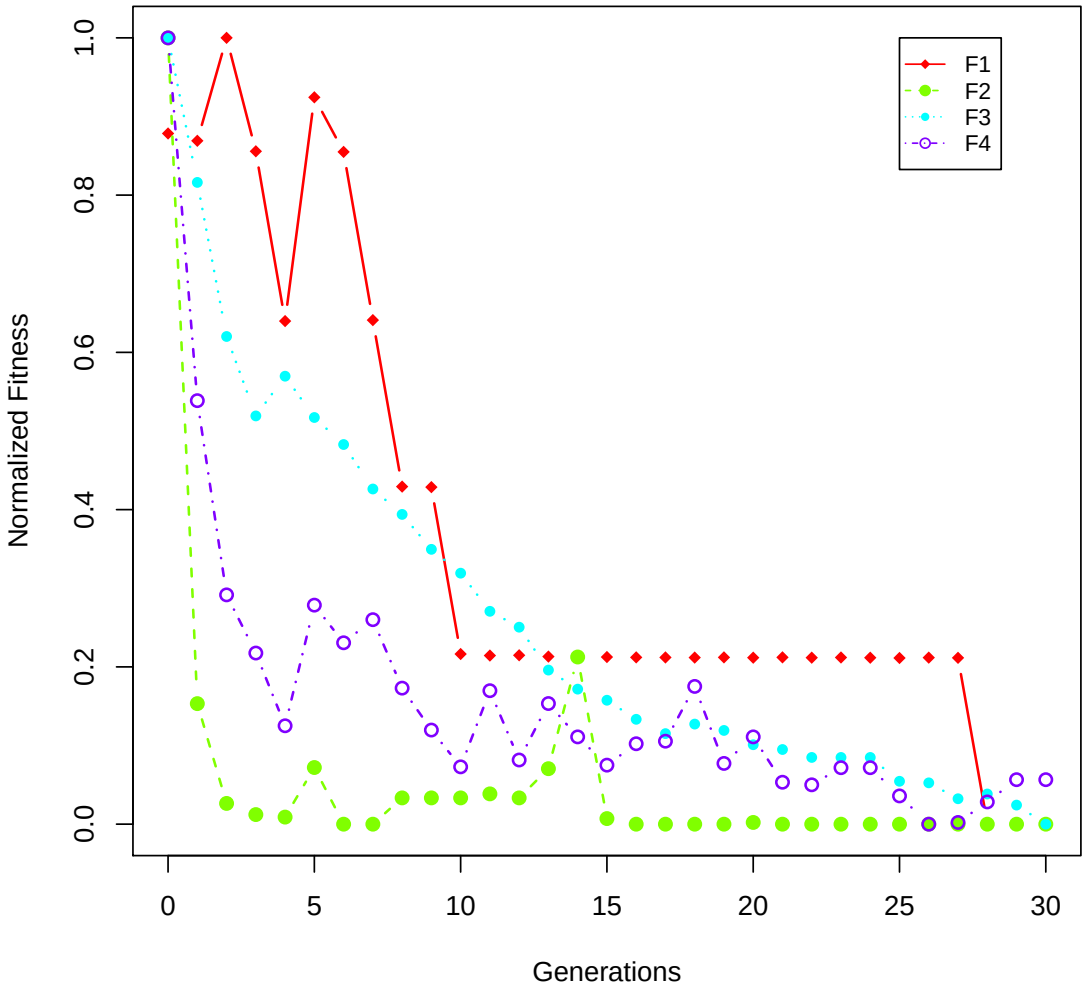


Figure D.5: **BA-100** Convergence Plot

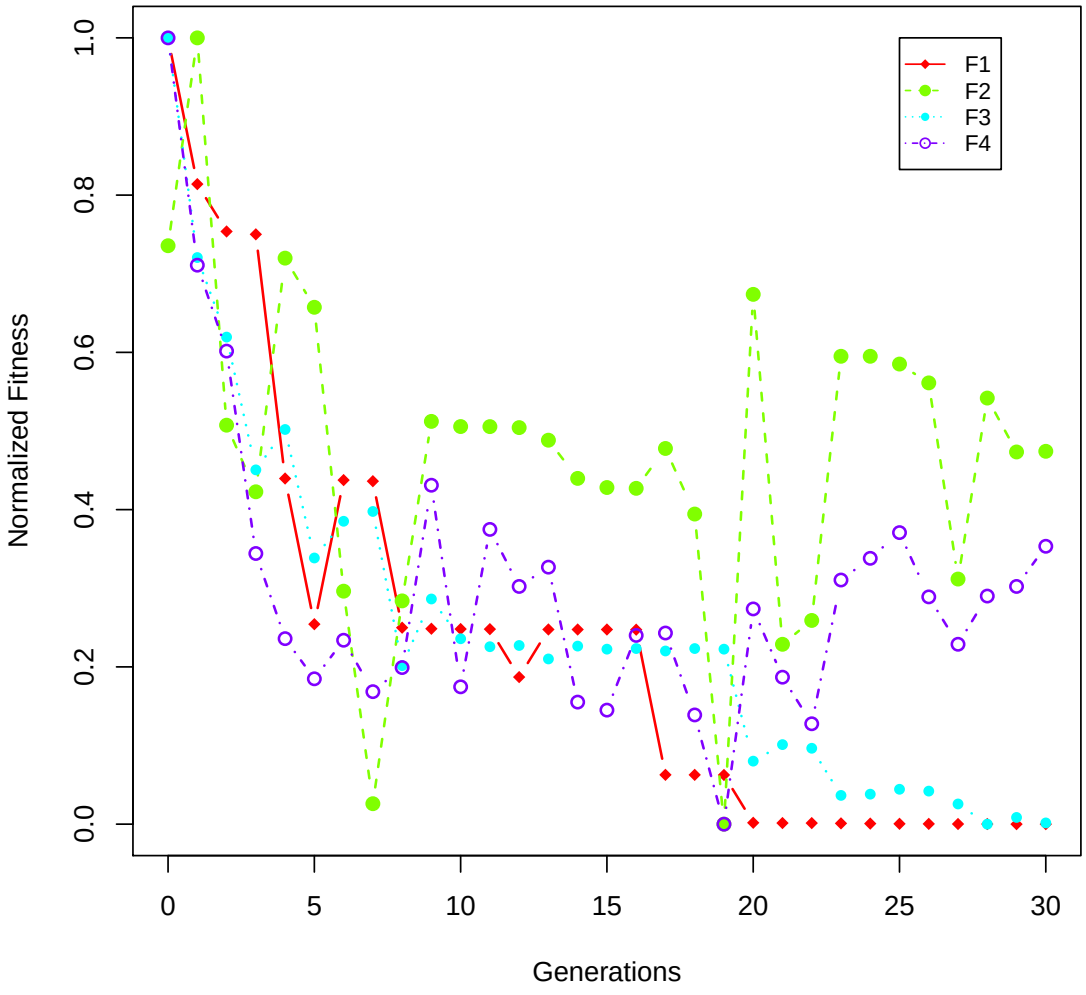


Figure D.6: **BA-250** Convergence Plot

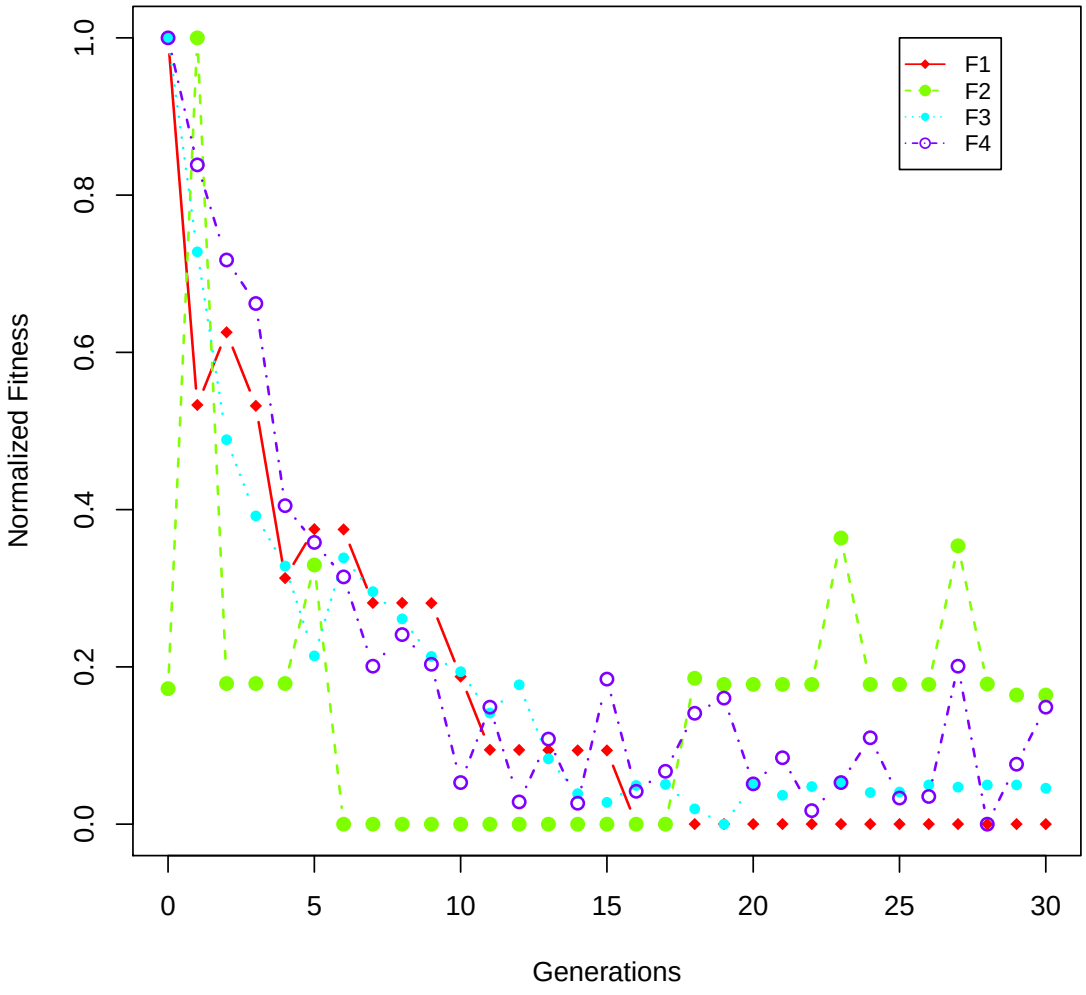


Figure D.7: **BA-500** Convergence Plot

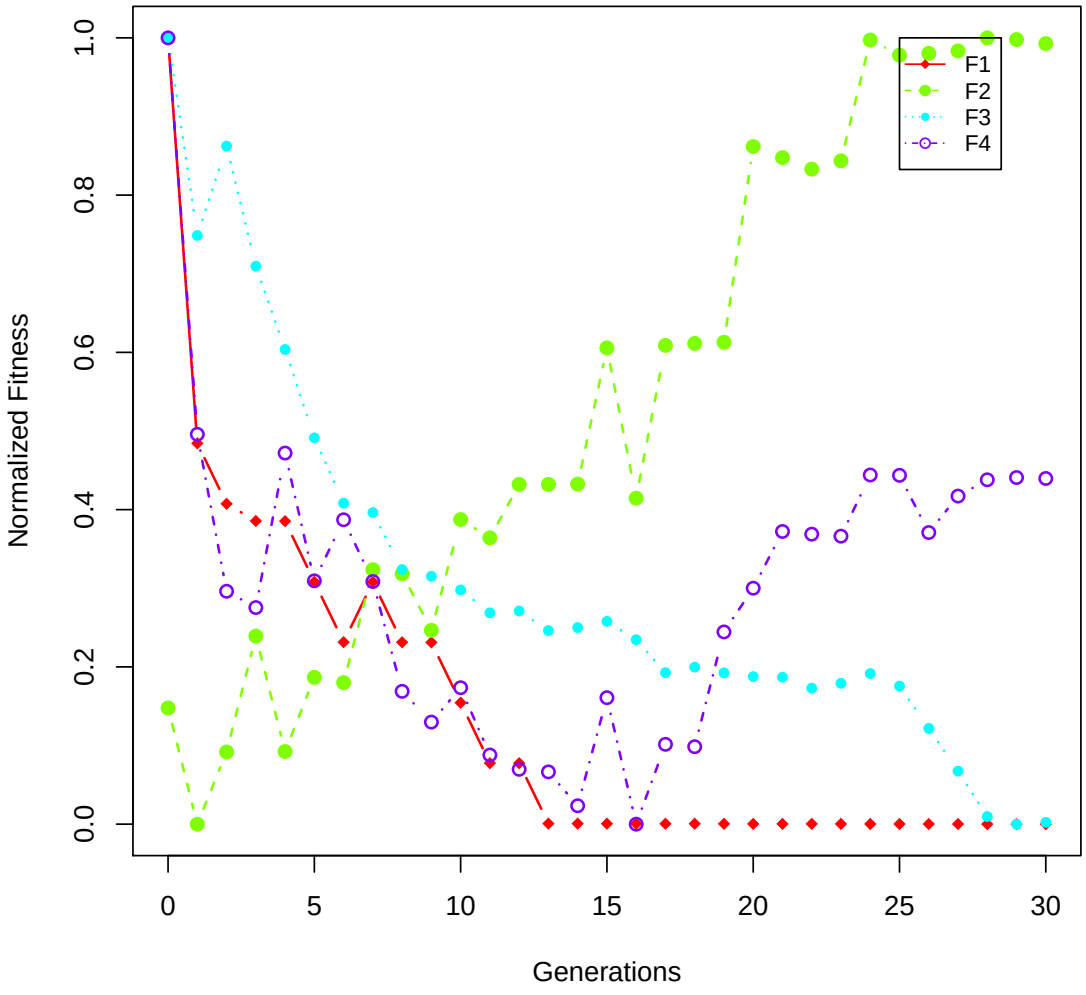


Figure D.8: **BA-1000** Convergence Plot

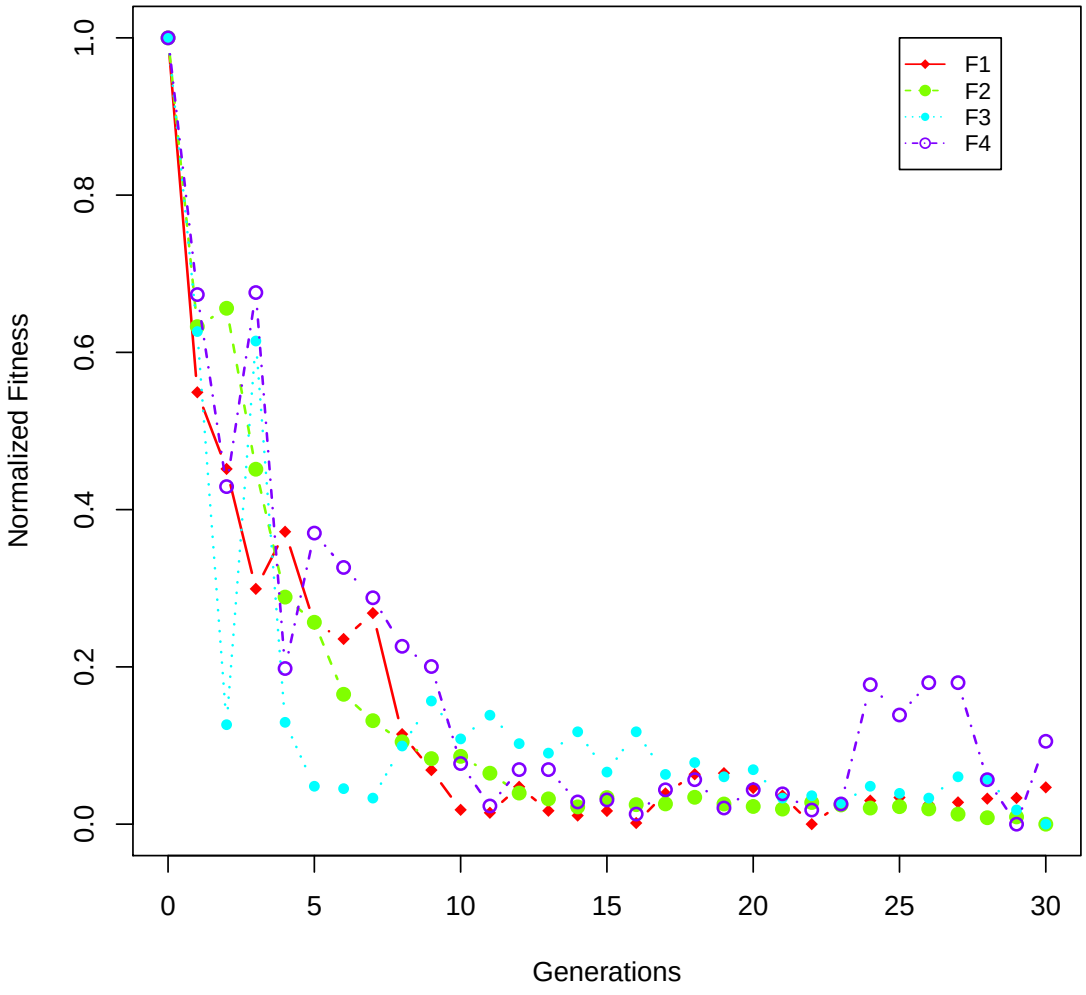


Figure D.9: **FF-100** Convergence Plot

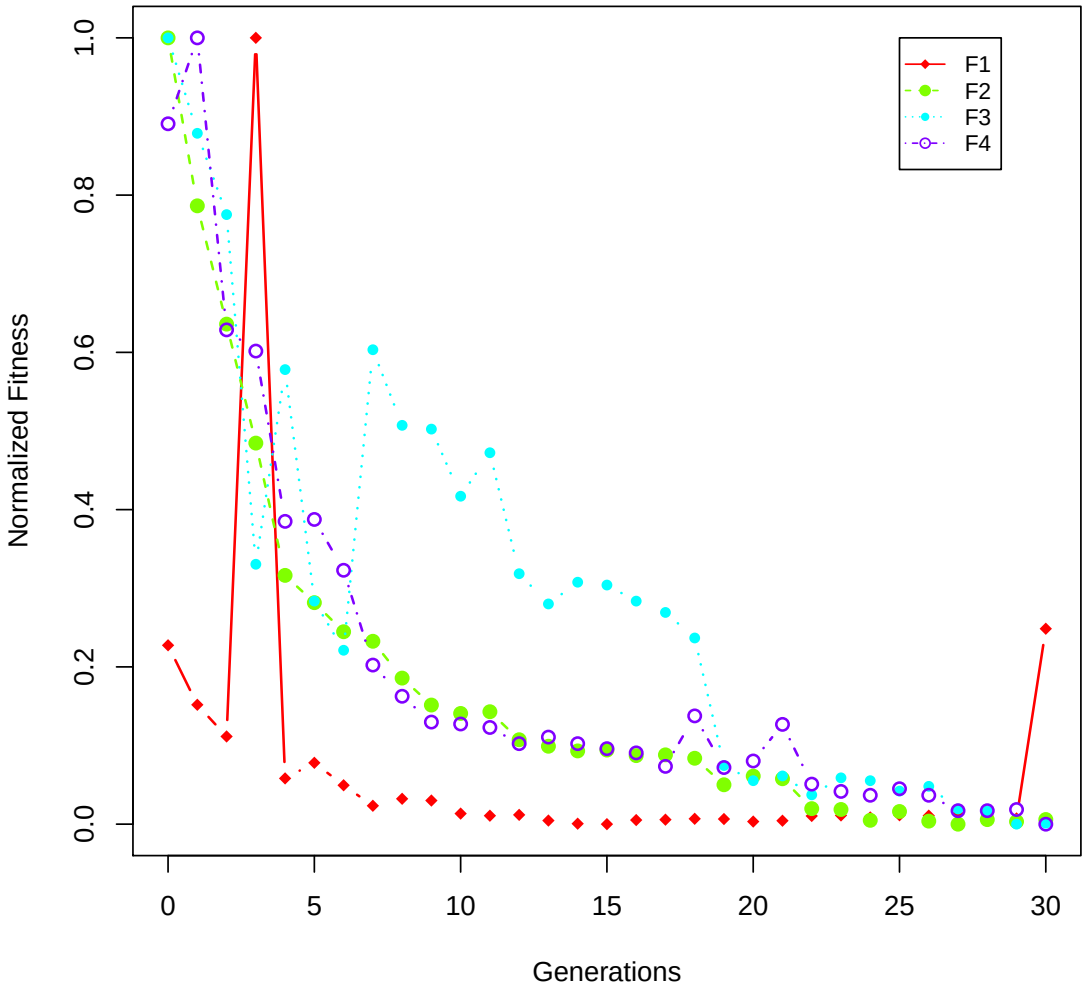


Figure D.10: **FF-250** Convergence Plot

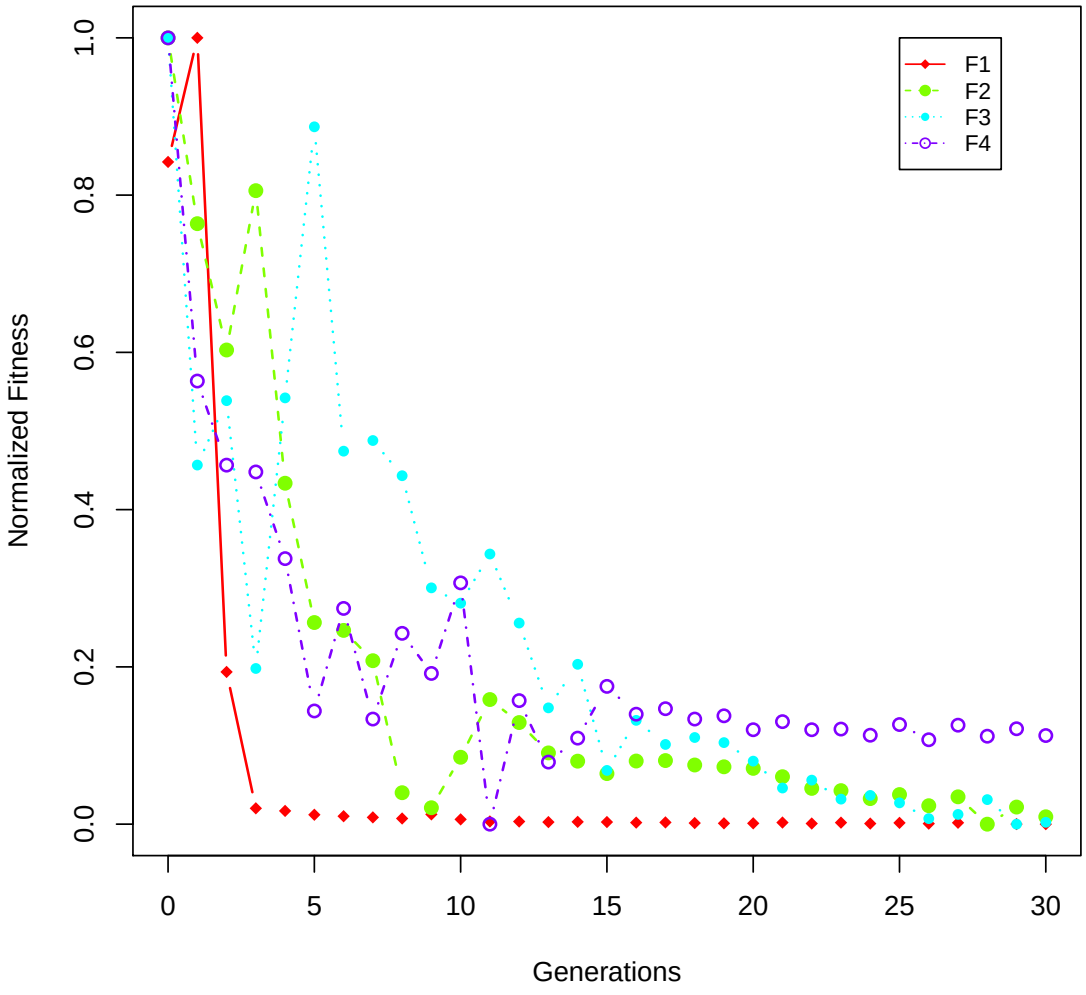


Figure D.11: **FF-500** Convergence Plot

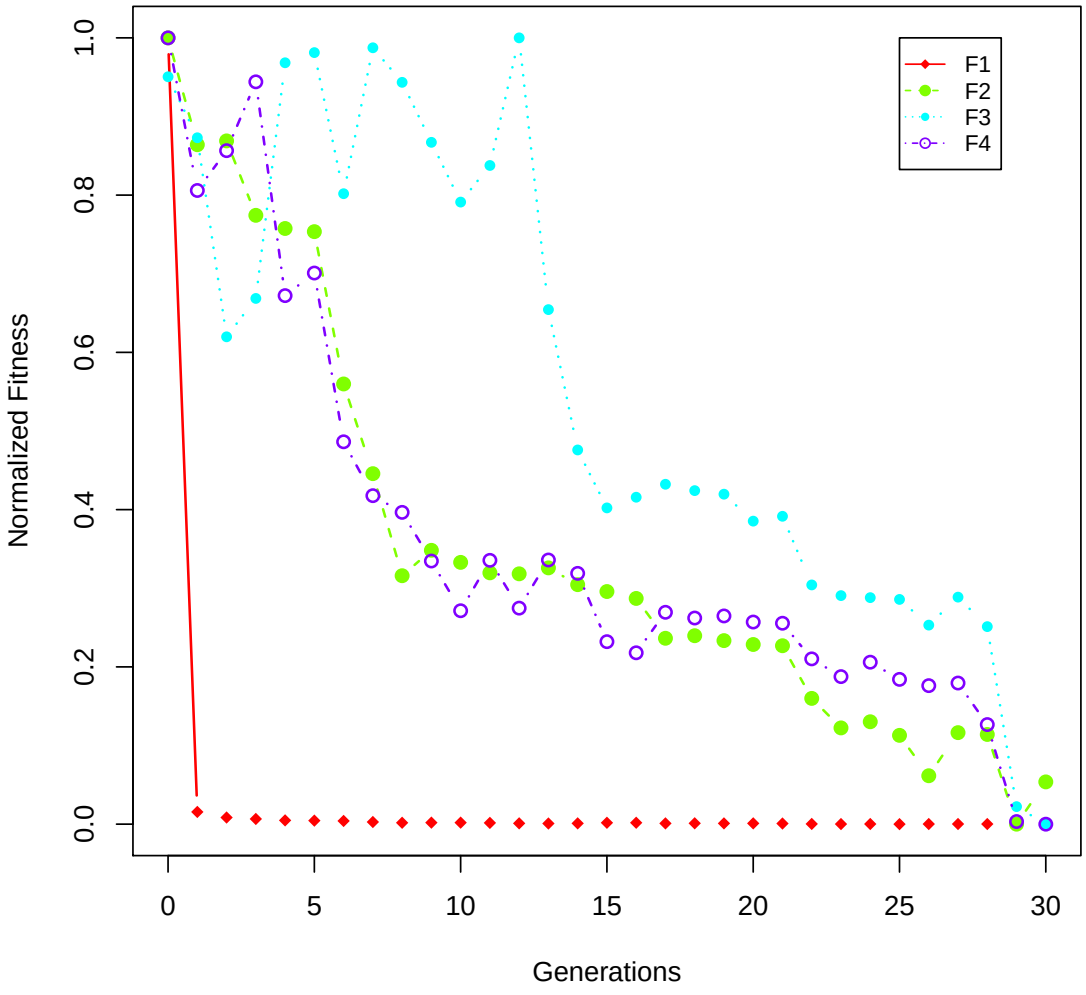


Figure D.12: **FF-1000** Convergence Plot

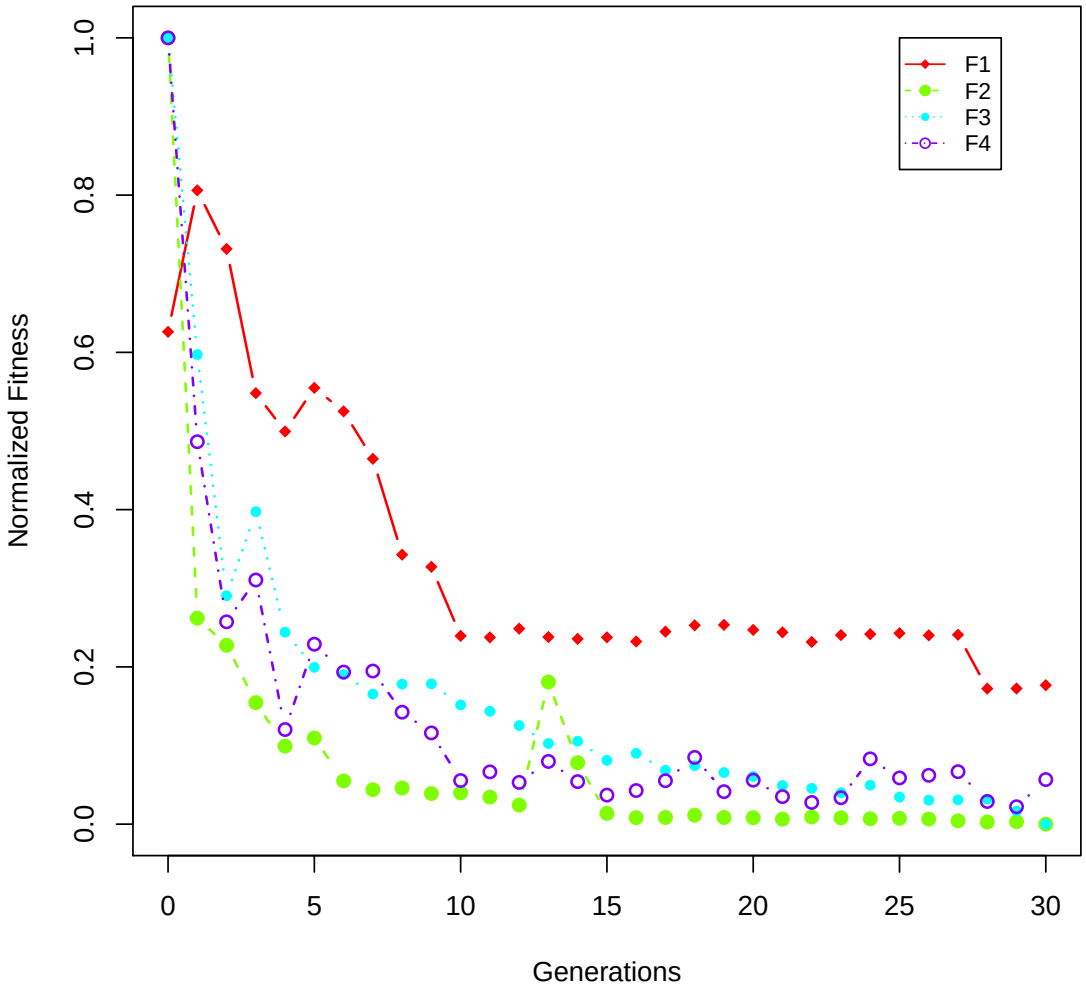


Figure D.13: **APA-100** Convergence Plot

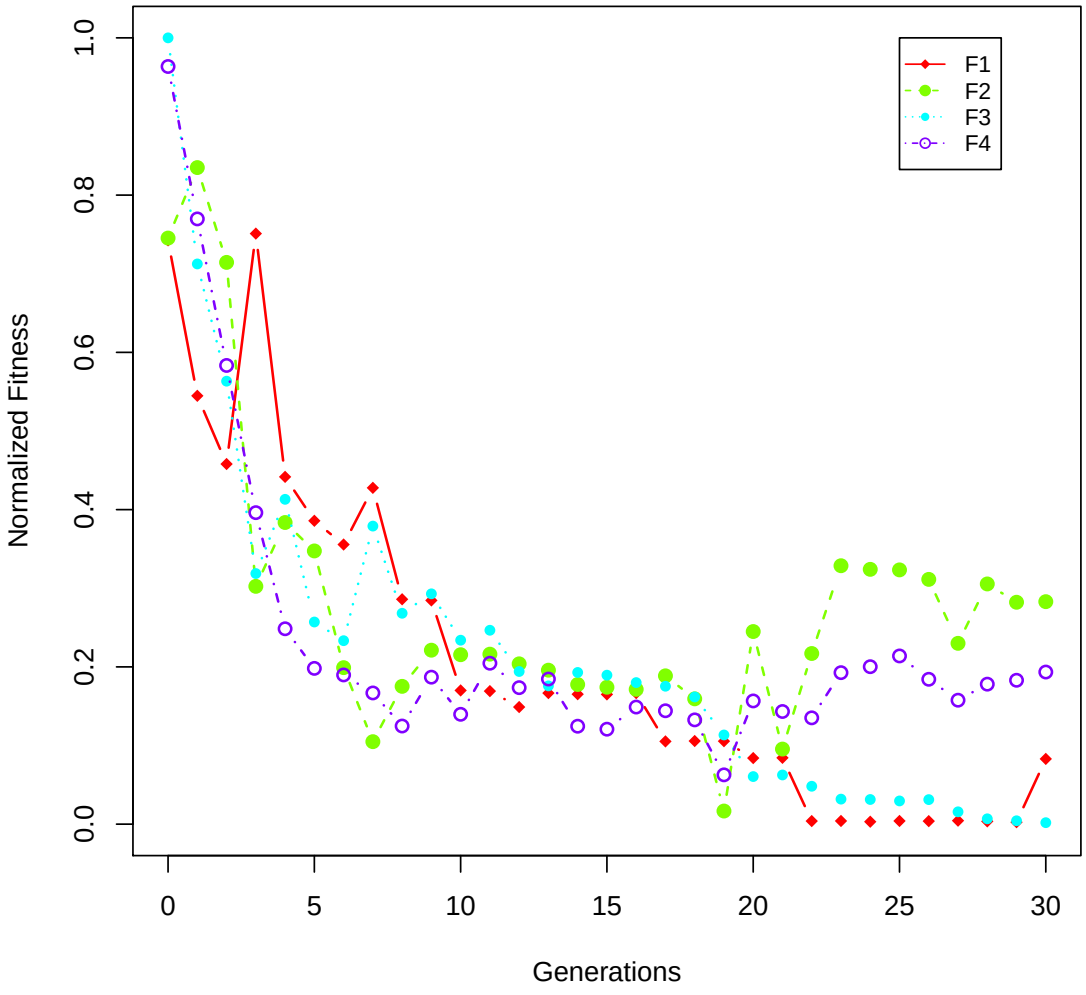


Figure D.14: **APA-250** Convergence Plot

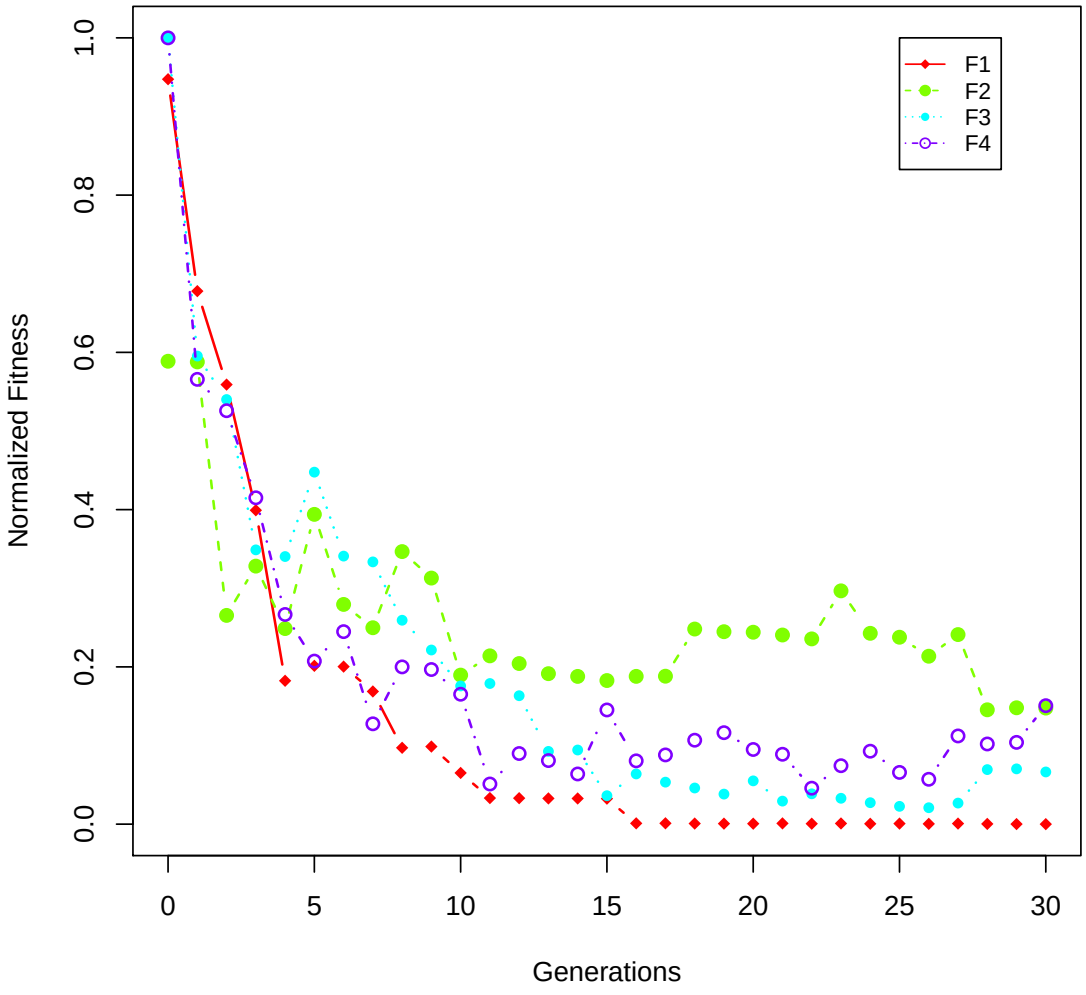


Figure D.15: **APA-500** Convergence Plot

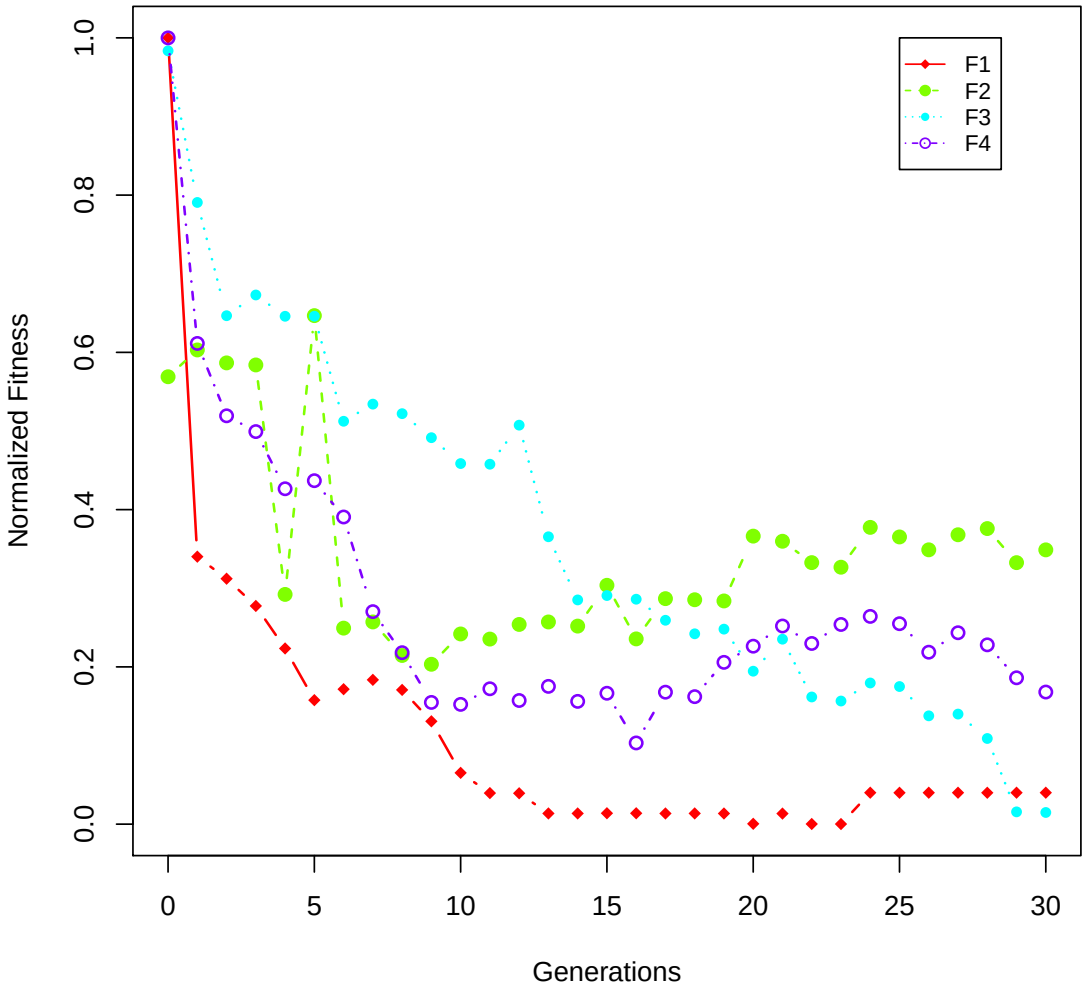


Figure D.16: **APA-1000** Convergence Plot