STRUCTURE AND RANDOMNESS IN COMPLEX NETWORKS APPLIED TO THE TARGET SET SELECTION PROBLEM

A thesis presented in partial fulfilment of the requirements for the degree of
Doctor of Philosophy
in
Computer Science
at Massey University, Manawatu,
New Zealand.

Callum William Scudamore Lowcay

2014

Abstract

Advances in technology have enabled the empirical study of large, so-called 'complex' networks with tens of thousands to millions of vertices, such as social networks and large communications networks. It has been discovered that these networks share a non-random topology characterised mainly by highly skewed, heavy-tailed degree distributions and small average distances between vertices. The work of this thesis is to attempt to leverage the well-known topological properties of complex networks to efficiently solve difficult NP-complete problems, with the aim of obtaining better or faster solutions than would be possible for general graphs.

Two related NP-complete problems are selected for study: the minimum target set problem, and the maximum activation set problem. Both problems relate to finding a 'target set' of vertices which is capable of initiating a spreading process (such as the spread of a rumour) that reaches a large proportion of the network. This thesis introduces several novel heuristics for these two problems inspired by the topology of complex networks. It is discovered that in many (but not all) cases it is possible to make relatively small alterations to the network that enable the computation of a considerably smaller target set than would be possible on general graphs.

The evaluation of the various heuristics is entirely experimental, which required the development of procedures to generate 'random' networks that can be used as experimental controls. This thesis includes a survey of several popular techniques for generating random networks and finds all but one (random rewiring) to be unsuitable as controls. The validity of random rewiring relies on a somewhat obscure theorem. Although a proof of the theorem (essentially an existence proof) is already known, this thesis offers a constructive algorithmic proof. The new proof advances on the old by providing an upper bound on the maximum number of rewiring operations required to transform between networks of the same degree-sequence, whereas an upper bound could not be determined under the old proof.

Acknowledgements

I would like to thank my two supervisors Dr. Catherine McCartin and Professor Stephen Marsland. The timely completion of this thesis would not have been possible without their efforts.

I would also like to thank Professor Mike Langston and his research group for hosting and welcoming me at the University of Tennessee in Knoxville, where we collaborated on the minimum vertex cover problem.

I was supported for the duration of my PhD research by the Massey University Vice Chancellor's Doctoral Scholarship. I was also supported by Stephen Marsland's Marsden grant MAU0908.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Advances in technology have enabled the study of large networks, such as social networks, communications networks, and various kinds of biological networks (including neural networks). These networks range in size from a few hundred, to tens of thousands or even millions of vertices. It was once common to assume that all networks of such size are essentially random in their topology (i.e. the pattern of connections in the network). However, empirical studies of large networks have shown this assumption to be false (Newman, 2003; Boccaletti et al., 2006). In fact, most large networks that have been studied have been found to have a topology that is qualitatively distinct from the old random models of network topology. These kinds of networks, with non-random topology, are now referred to in the literature as *complex networks*.

It is important to determine which topological features of a complex network are particular to the network in question, and which are merely due to chance. In this thesis, this is done by empirically comparing the topologies of complex networks with suitable random networks. A random network is a network chosen randomly from the space of all possible networks with some particular property, such as all the networks with a chosen degree distribution, or all the networks of a particular size.

Studies of complex networks from many disparate domains (including social networks, computer networks, biological networks, and others) have revealed two topological properties that are near-universal in complex networks: the heavy-tailed degree distribution, and the small-world effect. These properties will be discussed in detail in Section 2.2. Most complex networks exhibit both properties simultaneously. That these two topologies should be present in such a wide range of networks has attracted much interest, and has spawned a great deal of research into complex networks.

1.1 Motivation

Most of the complex networks literature focuses on identifying networks that exhibit typical complex network topologies (the heavy-tailed degree distribution and the small-world effect), and on explaining how those topologies arise in complex networks.

This thesis asks a different question: knowing that a network has typical complex network topology, can that structure be leveraged by suitable algorithms to achieve faster solutions than would be possible without the complex network topology.

Consider, for example, the problem of network search in the world-wide-web network. The goal is to find a particular web page by following the links from one page to the next. Standard search algorithms (depth-first search and breadth-first search) take $O(m)$ time to do this, where m is the size of the network in edges. These algorithms are optimal for the general case where nothing is known about the structure of the network. However, in the case of complex networks the topology can be used to guide the search, resulting in search algorithms with expected sublinear running time on complex networks.

Adamic et al. (2001) investigate search in the context of a heavy-tailed network topology, and find that simple algorithms (specifically a random walk search and a degree directed search) have expected running times that are polylogarithmic in the size of the network. Kleinberg (2006) surveys the search problem for networks exhibiting the small-world effect and finds that sublinear search is possible for some, but not all, small-world topologies.

Although this thesis seeks general techniques for leveraging complex network topology (techniques that could be applied to a variety of problems), two specific problems are chosen for study. It is hoped that the techniques developed for these problems can be applied more broadly. The two problems chosen for study (described in Section 5) are the minimum target set selection problem and the maximum activation set problem. Both problems concern the spreading of information in a network, such as the spreading of rumours in a social network.

1.2 Choice of applications

The minimum target set selection problem seeks to find a minimum size subset of the network, a ‘target set’, so that a spreading process starting from the target set will eventually affect the entire network. The maximum activation set problem asks for a target set of a fixed size k, so that the largest possible fraction of the network is affected by a spreading process starting with the target set. Chapter 5 includes a more detailed description of the two problems.

Unlike other well-studied network problems (such as the minimum dominating set problem), the target set problems are specific to complex networks. In a random network, for example, any two elements chosen at random would be expected to have a similar influence over the network. Hence, any randomly selected target set would be about as good as any other. In complex networks, on the other hand, it is known that some elements are considerably more influential than others (Kitsak et al., 2010). How to identify a highly influential subset of a network is, therefore, an interesting and highly relevant problem in the field of complex networks.

As an example of the target set problem in practice, consider modelling the spread of disease, or modelling the uptake of new ideas in social networks. If it is possible to find a small target set that has a large influence on the network, then that has important implications for example in epidemiology, or 'viral' marketing.

Another interesting aspect of the target set problems for this thesis is that they are both NP-complete, which implies that exact solutions cannot be efficiently computed on general networks. Furthermore, minimum target set selection is APX-hard (or inapproximable), so it is not even possible to find good approximation algorithms in general.

Although the minimum target set and maximum activation set problems are computationally hard, there is some evidence that they may be easier on complex networks than on general networks. Nichterlein et al. (2010) identify three classes of network where minimum target set can be computed exactly in polynomial time. It is not known whether or not any real-world complex networks belong to one or more of these classes, and determining this is one of the objectives of this thesis.

The algorithms of Nichterlein et al. (2010) are based on the theory of parametrized complexity. In the field of parametrized complexity, an aspect of the problem input (in this case an aspect of the topology of the network) is encoded as some *parameter* of the input, then algorithms can be designed where the running time is a function of both the size of the input (as usual), *and* the parameter. An algorithm is referred to as FPT (Fixed-Parameter-Tractable) if it is polynomial in the size of the input. It may be exponential or worse in the parameter. If the parameter is small enough, then such an algorithm effectively runs in polynomial time.

Since the target set selection problems are both NP-complete, and minimum target set is APX-hard, the two most promising algorithmic techniques that remain are the FPT approach (which may or may not be applicable to complex networks) and the use of heuristics. Both FPT algorithms, and heuristics, rely on exploiting structure inherent in the problem. Hence, this thesis will be an attempt to develop heuristics, FPT algorithms, or a combination of both, to exploit complex network topology and to thereby enable faster, or better (i.e. closer to optimal) solutions for the target set problems. It also is hoped that the techniques developed will be applicable to other problems besides the target set problems.

1.3 Aims and objectives

1. To determine if there is any previously unnoticed topological structure in complex networks that could be used to inform the design of algorithms for complex networks. In particular, to determine if complex networks belong to any of the classes of network for which minimum target set selection can be solved in polynomial time.

2. To determine if, and how, the known structure of complex networks can inform the design of improved algorithms for the target set selection problems, when restricted to complex networks. Specifically, this thesis aims to apply complex network theory to the design of algorithms that are faster, or that produce better (closer to optimal) solutions than existing heuristics.

This thesis will take an experimental approach. Hence, the objectives in order to meet the first aim are as follows:

- To select a range of publicly available network datasets that are representative of the major categories of complex network.

- To determine whether or not any of the selected networks exhibit topological structures that are known to enable FPT solutions to minimum target set. This will be done by measuring the relevant network parameters.

For the second aim of this thesis, it will be necessary to compare the performance of algorithms on complex networks, and *non*-complex networks. In this way, it is possible to confirm that some proposed algorithm really does leverage complex network structure, and that it would not work as well on networks that lack that structure. As usual in complex networks research, *random networks* will be used as the null model.

Two varieties of random network will be used: random networks drawn from the space of all networks of a particular size, and random networks drawn from the space of all networks with a particular degree distribution. Random networks with particular degree distributions are included because the degree distribution property is considered to be a fundamental property of complex networks (Li et al., 2005; Boccaletti et al., 2006).

By repeating experiments on these two varieties of random network, and a selection of empirical network datasets, it will be possible to see the relative effects of: no complex network structure, some complex network structure, and full complex network structure. It is the opinion of the author that these random networks should be *uniformly* random, i.e. drawn uniformly at random from the relevant space of networks.

The objectives for the second aim are thus:

- To determine the most practical procedure for generating the two varieties of random network, with the constraint that the procedure must generate random networks uniformly.
- To design new heuristic algorithms for minimum target set selection, and maximum activation set, based on the known properties of complex networks.
- To compare the performance of the newly designed algorithms with known algorithms (from the literature on target set selection), on both empirical network datasets, and random networks.

1.4 Roadmap

The remainder of this thesis is organized as follows:

A range of publicly available real-world network datasets are introduced in Section 2.1. These networks include communications networks, social networks, and biological networks. Some of these networks are directed (the connections can only be followed in one direction), others have connection weights, multiple connections between some pairs of vertices, and even *self-loops* (vertices that are connected to themselves). To enable comparisons across such a variety of networks, a decision has been made to project every network down to a simple graph, where each edge is undirected, and there are no multiple connections, self-loops, or weights.

Section 2.2 is a review of the literature on complex networks. The two major properties of complex networks: the heavy-tailed degree distribution and the small-world effect are described in detail. Several other properties are discussed as background to the following chapters.

Chapter 3 concerns the problem of generating random graphs. Since it was decided to model the empirical complex network datasets as simple graphs, it is necessary for the random graphs to also be simple. However, it turns out that generating uniformly random *simple* graphs is surprisingly difficult. A variety of methods, popular in the literature, are reviewed. It is found that only one model is able to generate the required random networks, while also being practical to implement: the degree-preserving rewiring model of Gkantsidis et al. (2003).

The Gkantsidis et al. method relies on a somewhat obscure theorem by Taylor (1980). A novel alternative proof of that theorem is presented in Chapter 4. This new proof permits the computation of upper bounds for the number of rewiring operations required (which was not investigated by Taylor).

Chapter 5 introduces the minimum target set selection, and maximum activation set problems. Section 5.3 of that chapter describes the topological conditions under which minimum target set selection can be solved exactly in polynomial time by known FPT algorithms. It then goes on to determine whether or not any of the complex networks datasets introduced in Chapter 2 meet these conditions.

Chapter 6 mainly concerns the minimum target set selection problem. The best known heuristic algorithms from the literature are introduced and described. Section 6.2 introduces a novel heuristic for minimum target set selection that can be applied in a distributed manner. This is clearly relevant in the context of large communications networks, for example. An experimental comparison of the known heuristics, and the novel heuristic reveals that, surprisingly, the distributed heuristic computes the smallest target set in the least time (on the network datasets selected in Chapter 2).

Experience detailed in Section 5.3 suggests that complex network structure may be too weak to leverage directly. This is already known to be the case for certain classes of optimisation problem (Shen et al., 2012). Instead, the remaining chapters of this thesis investigate a class of algorithms which make small modifications to the network, in order to create structure that permits faster or better solutions.

Section 6.3 examines how the structure of a complex network changes as vertices are removed in order from the highest to lowest degree (i.e. in order of how many connections they have to the rest of the network). Chapter 7 continues the theme of making small network modifications, this time adding new edges (i.e. new connections) instead of removing vertices.

Ultimately, it is discovered that it is often possible in complex networks to make relatively small modifications to a network in order to achieve substantial improvements in the performance of suitably tuned algorithms. In particular, experiments from Chapter 6 show that the removal of a relatively small number of high-degree vertices from a complex network simplifies the structure of the network to the point that an exact solution can be computed for the bulk of the network. Experiments described in Chapter 7 show that adding edges to a complex network is another way to achieve better performance for suitable algorithms. However, this kind of modification applies less broadly than the removal of hubs. Finally, it is shown that these results are specific to complex networks as the techniques employed rely on the underlying complex network topology.

Chapter 2

Complex networks

A network consists of vertices connected by edges. The vertices represent the networked entities. For example, the vertices of a social network are the people who make up the network. The edges represent interactions of some sort between the vertices. Depending on which interactions are used to form edges, a variety of networks can be drawn even from the same system.

Complex networks are distinguished from other networks by their topology, i.e. the pattern of connections in the network. There has been much research comparing the topology of large empirical networks with random network topologies. This has lead to the discovery of several properties that occur frequently in large networks, and which cannot be explained as arising by chance (Newman, 2003; Boccaletti et al., 2006). Surprisingly, the *same* topological features have been discovered in a broad range of networks. These features (described in detail in Section 2.2 of this chapter) are now considered to be characteristic of 'complex networks' (Newman, 2003).

This chapter introduces eight complex network datasets (see Table 2.1 for a full list), from a broad range of domains. These datasets are used throughout this thesis as examples of real data. They are contrasted with the *random* networks discussed in Chapter 3. The eight datasets introduced in Section 2.1 are chosen to represent the main kinds of complex networks reported in the literature. They are large enough to be computationally challenging, but not so large as to require unreasonable computational resources.

In this thesis, networks are modelled as graphs in the graph theory sense. A graph $G = (V, E)$ is defined by a set of V of vertices, and a set E of edges, where each edge connects two vertices. The degree of a vertex is equal to the number of edges incident on that vertex. In the complex networks literature it is common to see the terms "graph" and "network" used interchangeably.

A graph can be *directed* or *undirected*, depending on whether or not the edges have directions associated with them. Some graphs may have *multiple edges* between the

Network	Vertices	Edges	Density	Mean degree	Max degree
Physicists 1	40,421	175, 693	2.15×10^{-4}	8.69	278
Physicists 2	34,546	420,877	7.05×10^{-4}	24.4	846
Enron	36,692	183,831	2.73×10^{-4}	10.0	1,383
Gnutella	10,876	39,994	6.76×10^{-4}	7.35	103
Blogs	1,490	16,715	151×10^{-4}	22.43	351
Internet	27,719	41,684	1.09×10^{-4}	3.01	1,644
Neural	297	2,148	490×10^{-4}	14.5	134
Metabolic	453	2,025	198×10^{-4}	8.94	237

Table 2.1: The eight network datasets used in the experiments throughout this thesis. Basic metrics are reported for each network: number of vertices, number of edges, density (ratio of edges to maximum possible number of edges), mean degree, and the maximum degree.

same pair of vertices, or *self-loops* (vertices connected to themselves). Sometimes a number known as a *weight* is associated with the edges or vertices of a graph, for example to represent the relative strengths of the connections in a network. An undirected graph with no multiple-edges, self-loops, or edge weights is referred to as a *simple* graph.

In order that the disparate network datasets introduced in this chapter can be treated equivalently, it was decided early on in this research to reduce all the datasets to simple graphs. This is achieved by replacing all directed edges with undirected edges, removing any edge weights, removing any self-loops, and replacing multiple-edges with single edges.

The first part of this chapter, Section 2.1, provides a summary of the network datasets that were selected for study. The second part, Section 2.2, explains some of the important topological properties of complex networks. Also included in Section 2.2 are the results of computations showing that the network datasets selected in Section 2.1 exhibit all the classic properties of complex networks.

The main aim of this thesis is to apply knowledge of complex network topologies to problems in the design of algorithms for complex networks. It was found that some of the topological properties introduced in this chapter were helpful in that goal, whereas others could not be so easily harnessed.

2.1 Network datasets

Many kinds of complex network are reported in the literature, and they commonly fall into three broad categories: *social networks*, *communication networks*, and *biological networks* (Boccaletti et al., 2006). In social networks, the vertices represent people and the edges are interactions between those people. In communication networks the vertices are elements in a telecommunications system, such as routers in the Internet. The edges in this case are communications links. Biological networks are the broadest category of all; the vertices are elements of some biological system such as neurons in a brain. In the neural network example, the edges are connections between neurons.

In order to study as wide a range of complex networks as possible, eight publicly available datasets were selected. See Table 2.1 for a full list of the networks chosen. Table 2.1 also reports the number of vertices and edges in each of the networks, along with the density, mean degree, and maximum degree.

The eight networks fall nicely into the three categories previously mentioned: The Physicists 1, Physicists 2, and Enron networks are social networks; the Gnutella, Blogs, and Internet networks are communications networks; and the Neural and Metabolic networks are biological. The remainder of this section is a basic description of the network datasets, and how they are derived.

2.1.1 Social networks

Of the social networks, the most studied variety is the *collaboration network*. In a collaboration network, the edges represent collaborations between people. For example, the famous IMDB (Internet Movie Database) network of movie actors (`http://www.imdb.com/`). In the IMDB network, there is an edge between two actors if they co-starred in a movie. The network is known to have a small diameter, and it was at one time a popular game to find short paths connecting actor Kevin Bacon to other, more obscure, actors.

The IMDB network is not used for the experiments in the thesis, as it is too large given the computational resources available to the author. Instead, two smaller collaboration networks were used, the Physicists 1 and Physicists 2 networks:

Physicists 1 is a co-authorship network of physicists (a collaboration network) posting preprints on the arXiv (`http://arxiv.org`) in the category of "condensed matter physics". It includes papers from 1995 to 2005 (Newman, 2001).

Physicists 2 is another co-authorship network of physicists on the arXiv, this time in the category of "high energy physics". It includes papers published prior to 2003 (Gehrke et al., 2003; Leskovec et al., 2005). It is quite a lot denser than the *Physicists 1* network, which can lead to noticeably different behaviour.

Due to the rise of social media platforms, such as Facebook and twitter, it has become possible to study large friendship networks, where two people are linked by an edge when they report that they are "friends". Published research shows that the Facebook network, for example, behaves much like other, more widely studied social networks (Ugander et al., 2011). Due to the proprietary nature of these networks, even anonymised data is not typically available to the public. Hence it was not possible to include such a network in this thesis.

Another way to derive a social network is to examine who talks to who. For example, an edge could be created when two people send emails to each other. This is how the *Enron* social network is derived (Leskovec et al., 2009):

In the *Enron* network, each vertex represents an email address, and there is an edge between two addresses if an email was sent between them (Klimt and Yang, 2004; Leskovec et al., 2009).

The Enron network is popular for study because the raw data was made public during the trial that followed the bankruptcy of the Enron corporation (Klimt and Yang, 2004).

2.1.2 Communication networks

The most thoroughly studied of the communication networks is the Internet. Due to the multilayer architecture of the Internet, several complex networks can be derived. At the router level network, every vertex represents a router that has a connection to the Internet. Edges are created between routers when they communicate directly to each other.

The next layer up is the AS (Autonomous System) level. An autonomous system is a single Internet connected network (for example, an ISP). The Internet is in fact composed of many thousands of autonomous systems, and they exchange traffic directly with each other in a relationship known as "peering". Since it is impractical for an AS to have a direct link to every other AS, a packet travelling through the Internet may traverse many autonomous systems along the way. In the AS network, the vertices are autonomous systems, and the edges represent the peering relations between them. From Table 2.1:

Internet is a 2009 snapshot of the Internet at the AS level, provided by CAIDA (Hyun et al., 2009). Each vertex represents an autonomous system, and there is an edge between two systems if they exchange traffic. It is by far the sparsest of the networks studied in this thesis.

A number of networks are built on top of the Internet infrastructure. The most obvious example is the world wide web network, where the vertices are web pages and the edges are hyperlinks between those web pages. The world wide web has been extensively studied, which has given rise to algorithms to rank search engine results for example. This thesis includes a network derived from a small piece of the world wide web:

Blogs is a directed network of American political blogs, recorded in 2005 by Adamic and Glance (2005). The vertices represent individual blogs, and the edges are hyperlinks between the blogs. Thus, it is a very small snapshot of the World Wide Web. This network was intended to demonstrate community structure; Adamic and Glance found two distinguishable communities. Perhaps due to the divided nature of the network, some interesting behaviour is exhibited in some of the experiments later in this thesis.

Another variety of network on the Internet is a P2P (peer-to-peer) network. In a P2P network, traffic is exchanged directly between client machines (peers), rather than being routed through a central server. The lack of a central server means that network discovery services (for example) must be distributed across the peers, which are sparsely connected. The exact details of how the peers are connected depends on the specific P2P protocol being used. Examples include bittorrent and the older Gnutella network:

Gnutella is a 2002 snapshot of part of the Gnutella network (Ripeanu et al., 2002; Leskovec et al., 2007), which is a P2P network used mainly for file-sharing.

2.1.3 Biological networks

Many biological networks have been described in the literature. Two are used in this thesis, both derived from the biology of the *C.Elegans* nematode worm. The *C.Elegans* worm was the first organism to have its full neural network mapped out (White et al., 1986). In a neural network, the vertices are neurons in brain, and the edges represent connections between the neurons. Such networks are now known as *connectomes*, in analogy to the genome of a species. There is ongoing effort to map the neural networks of more advanced species. Other varieties on biological network (not studied in this thesis) include protein interaction networks, and food webs.

The two biological networks used in this thesis are as follows:

Neural is the neural network of the *C. Elegans* nematode worm, compiled by Watts and Strogatz (1998) from the work of White et al. (1986). The vertices are neurons, and the edges represent synaptic connections between the neurons. This network is the smallest and densest of the networks used in this thesis. It is sometimes observed to show unusual behaviour as compared to the other networks (see Figure 2.3 for example). However, the most important complex network properties do hold (degree structure and the small world effect, explained in Section 2.2).

Metabolic is the metabolic network of the *C. Elegans* nematode worm (Duch and Arenas, 2005). As with the neural network, it is smaller than the other networks, although the density (as measured by mean degree) is similar to the other networks. The behaviour of this network appears to be similar to the other networks studied in this thesis.

2.1.4 A note on density

Notice in Table 2.1 that the maximum degree is much larger than the mean degree. This is due to the highly skewed distribution of vertex degrees observed in these networks (more in section 2.2.1). Vertices with much higher degree than the mean are referred to as *hubs*. Due to the presence of these hubs, the mean degree cannot be interpreted as the 'typical' vertex degree.

Density is the ratio of the number of edges to the maximum possible number of edges; i.e. $d = m/\frac{n(n-1)}{2}$ where n is the number of vertices and m is the number of edges. As seen in Table 2.1, all the networks have extremely low density (which is to say they are all very sparse). The density metric is thus unsuited to distinguishing the fine (but important) differences in densities of these networks.

Instead, twice the ratio of edges to vertices ($2m/n$) is used throughout this thesis as a measure of density. The reason for using $2m/n$ is that this is also the mean degree (since in any graph, total degree equals $2m$). Thus, the mean degree turns out to be convenient as a measure of density in sparse networks.

2.2 Properties of complex networks

Despite the disparate origins of the various categories of complex networks, a range of topological properties have been found to be common to all such networks. This section reports and discusses the key properties of complex networks as observed in the literature. These properties inform the work in the later chapters of this thesis.

Part of the original hope of the research that lead to this thesis was to discover previously unknown topological structures in complex networks. Some discussion of this work will be found in Section 5.3. However, no such structures could be found, and it is now the opinion of the author that the properties listed in this section are most likely all that one can expect to see in general in a complex network.

2.2.1 Degree distribution

The *degree distribution* $p(k)$ of a network is the probability that a randomly selected vertex has degree k; that is, the probability that it has k neighbours. Related to the degree distribution is the *degree sequence*. The degree sequence of a network is a sequence of n integers representing the degrees of the n vertices in the network. A degree sequence is by convention ordered from largest to smallest degree. Some authors use the degree distribution and degree sequence interchangeably, but in this thesis a

Network	Mean Distance	Clustering Coefficient	Mean Clustering	Assortativity Coefficient
Physicists 1	5.50	0.245	0.719	0.186
Physicists 2	4.33	0.146	0.296	−0.00629
Enron	4.03	0.0853	0.716	−0.111
Gnutella	4.64	0.00540	0.00804	−0.0132
Blogs	2.74	0.226	0.360	−0.221
Internet	4.49	0.00876	0.203	−0.142
Neural	2.46	0.181	0.308	−0.163
Metabolic	2.66	0.124	0.655	−0.226

Table 2.2: Measurements of key metrics on the chosen complex networks, independently computed for this thesis. The small-world effect can be seen by observing that the mean distance is very small in all the networks measured, but the clustering coefficient and mean clustering are relatively high. The Gnutella network is an exception in terms of clustering, which may relate to the function of that network. Degree correlations are observed in all but two of the networks.

Network	Power-law exponent	Power-law p-value
Physicists 1	3.50	0.22
Physicists 2	3.50	0.16
Enron	1.97	0.00
Gnutella	3.50	0.03
Blogs	3.50	0.48
Internet	2.17	0.25
Neural	3.34	0.63
Metabolic	2.63	0.01

Table 2.3: Attempted power-law fit to the degree distributions of the chosen complex networks, independently computed for this thesis according to the method of Clauset et al. (2009). The p-value is the probability that the data (the vertex degrees of the network) are drawn from a discrete power-law distribution. Thus, only a *large* p-value of a least 0.10 is considered compatible with the power-law hypothesis. Of the networks that do exhibit true power-law behaviour, the power-law exponents are within the 1.5 to 3.5 range reported in the literature.

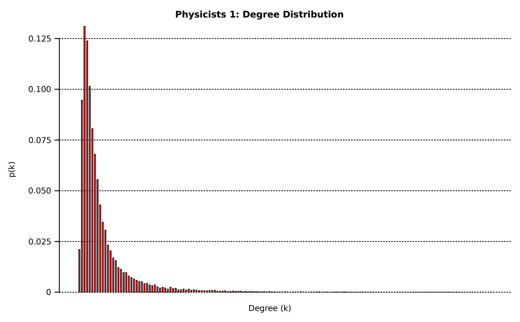

Figure 2.1: The degree distribution of the Physicists 1 network. Notice that most vertices are likely to have low degree, but there are a small number of vertices with very high degree. This is the classic heavy-tailed degree distribution of a complex network.

degree distribution will always be a probability distribution, and a degree sequence will always be a sequence of integers.

The degree distributions of complex networks are typically highly skewed, with wide variability. As an example, the degree distribution of the Physicists 1 network was computed and is shown in Figure 2.1. Such distributions are said to be heavy-tailed. Due to these two properties (highly skewed, heavy-tailed), the mean degree cannot be interpreted as a measure of the 'typical' degree of vertices in the network. The mean degree does, however, find use in this thesis as a measure of density for sparse networks that is somewhat independent of the size of the network.

The degree distribution in many complex networks has been shown to follow a power-law $p(k) = \alpha k^{-\gamma}$ where α and γ are parameters of the distribution (Newman, 2003). The parameter γ is referred to as the exponent, or the scaling index, and is typically in the range $1.5 \leq \gamma \leq 3.5$ (Newman, 2003). See, for example, the values for the power-law exponent reported in Table 2.3.

Power-law distributions are also referred to as scale-free distributions, because the parameter γ is invariant when the distribution is multiplied by a scaling factor. This has lead to the term "scale-free-network" for a network with a power-law degree distribution. This is poor terminology, because it implies that complex networks have a fractal structure, which has never been shown to be the case in general.

Power-law degree distributions (and other heavy-tailed degree distributions) are interesting for several reasons. The power-law implies that even in relatively small complex networks there will be a small, but significant, number of very high degree vertices, which are known as *hubs*. These hubs are not present in randomized networks (discussed in Chapter 3), which implies that they constitute a structure particular to complex networks (Newman, 2003).

One consequence of the presence of hubs is that such networks are resilient to random failures, but vulnerable to targeted attacks (Newman, 2003). Most of the vertices have relatively low-degree, so when a vertex is removed at random, usually only a small number of edges are removed from the network. If, however, hubs are removed from highest to lowest degree, large numbers of edges will be removed from the network, rapidly causing it to become disconnected. This property is of obvious importance in communication networks, such as the Internet.

Since there are a small number of hubs relative to the other vertices, hubs must be mainly connected to low-degree vertices. Hence, low-degree vertices typically have short paths back to the hubs. This leads to short average distances between randomly selected vertices in complex networks (more in Section 2.2.4). This suggests a simple local strategy (which can be employed without requiring knowledge of the entire network) to find hubs: choose a vertex at random, then pick the neighbour of that vertex with the highest degree.

The hubs can be used to guide the design of algorithms on complex networks. For example, an optimised depth-first search on a network with a power-law degree distribution completes in $O(n^{1/2})$ time at worst ($O(log(n))$ time at best) (Adamic et al., 2001; Newman, 2003). The same algorithm takes linear ($O(n)$) time on general networks (with arbitrary degree distributions). This result, and others, suggest that some problems may be computationally easier on power-law networks than on general networks.

Unfortunately, several major NP-complete optimisation problems are known to be no easier to approximate in power-law networks than in general networks (Ferrante et al., 2008; Shen et al., 2012). The problem is that the constraint of a power-law degree distribution is not restrictive enough to prevent the embedding of certain problematic substructures. Experiments described in Section 3.2 suggest that in fact a wide range of topologies are possible given a fixed degree sequence.

Although it seems that a power-law degree distribution alone is not enough to allow for better approximations or faster exact solutions, note that other structures are also known to be present in complex networks. It is possible that a combination of properties may provide the desired algorithmic benefits. Chapter 6.3 considers how the hubs can be used as a basis for modifying the network so that it contains structure that is known to be useful algorithmically.

Power-laws in empirical networks

The traditional way to check for a power-law degree distribution was to plot the frequency of the vertex degrees on a log-log scale, then do a linear regression to fit a straight line to the data. The slope of the line is equal to the power-law exponent. This method is known to be statistically unreliable, and so it is likely that many claimed power-laws are in fact not (Li et al., 2005).

A reliable procedure for fitting a power-law distribution to data is given by Clauset et al. (2009), this procedure takes into account the difficulties peculiar to fitting a discrete power-law (since vertex degrees are always integers, we must fit the discrete version of the power-law distribution).

The method of Clauset et al. also gives a p-value that measures the likelihood that the data are in fact drawn from a power-law distribution. A p-value of 1 indicates that the data follow a power-law, a p-value of 0 indicates that they do not. A *large* p-value of at least 0.10 is considered compatible with the power-law hypothesis. This is different to the usual concept of a p-value (where a small p-value indicates statistical significance), because this p-value does *not* represent a null hypothesis. This unusual situation reflects the difficulty of distinguishing between a true power-law, and a large number alternative distributions with similar properties (especially the log-normal distribution).

Power-law fits according to the Clauset et al. method are given in Table 2.3. The code used to compute these values was adapted from the partial R implementation available from http://www.santafe.edu/~aaronc/powerlaws/.

One of the difficulties in fitting a power-law distribution to data is that there are a number of similar distributions that can be difficult to distinguish, for example the log-normal (Clauset et al., 2009). Thus, and especially considering the generally low p-values in Table 2.3, the possibility cannot be ruled out that *none* of these networks have power-law degree distributions.

While a power-law degree distribution is convenient for theoretical analysis, it is probably not so important in practice. Li et al. (2005) make a strong case that in the degree distributions of complex networks, the important properties are that they are strongly skewed and heavy-tailed, regardless of what specific probability distribution lies behind those features. Thus, the term *heavy-tailed* will be used throughout this thesis to refer generally to networks with these kinds of degree distributions.

2.2.2 Degree correlations

Correlations in the degrees of adjacent vertices are widely observed in complex networks (Newman, 2003; Boccaletti et al., 2006). These correlations can be positive (adjacent vertices are likely to have similar degree), or negative (adjacent vertices are likely to have dissimilar degree). Networks with positive degree correlations are referred to as *assortative* networks, whereas networks with negative correlations are known as *disassortative* networks (Newman, 2002).

The degree correlations are quantified by the assortativity coefficient r, introduced by Newman (2002). Newman's assortativity coefficient is equivalent to the Pearson's correlation coefficient of the degrees of adjacent vertices, which lies in the range

$-1 \leq r \leq 1$. In assortative networks, r is positive, in disassortative networks it is negative. Values for empirical networks are reported to be in the range $-0.3 \leq r \leq 0.3$. Table 2.2 includes independently computed values for the assortativity coefficient of several empirical networks.

Although the correlations are weak, they capture important variations in the structure of complex networks, beyond what is implied by the degree distribution. This is explored in the work of Li et al. (2005), where a not-normalised version of the assortativity coefficient is introduced, the *s-metric*.

Li et al. studied the relationship between their s-metric and a network performance metric (intended to measure the throughput performance of communication networks). They found that networks with high s (more assortative) have relatively poor performance, but that there is more variation in the structure of those networks. High performing networks have low s (more disassortative). Indeed, relatively strong disassortativity is a common feature of networks intended for communication (see Table 2.2).

In a power-law network, most vertices have relatively low degree with a few high degree hubs. Assuming there are no self loops (vertices connected to themselves) or multiple edges (between the same vertex pair), that implies that in an assortative network, the hubs are strongly connected to each other so that the connections between hubs and low degree vertices can be minimised. Conversely, in a highly disassortative network, the highest degree hubs are directly connected to the lowest degree vertices, with the medium sized hubs connected to each other (a typical structure for a communications network).

Newman's assortativity coefficient is normalised against a multigraph, where self loops and multiple edges are allowed. Thus, a highly assortative network is likely to have many self loops and multiple edges. All the networks used in this thesis are projected down to simple graphs by removing any self loops and multiple edges. This process creates a bias towards disassortativity which is explored further in Chapter 3. The same effect is observed by Maslov et al. (2004) and Li et al. (2005).

2.2.3 Extension to dK distributions

In their analysis of the Internet AS-level network, Mahadevan et al. discovered empirically that the *joint-degree distribution* determines most of the important topological properties of that network (Mahadevan et al., 2006b). The joint-degree distribution $p(j, k)$ is a two dimensional probability distribution defined as the probability that two adjacent vertices have degrees j and k. The degree distribution can be computed from the joint-degree distribution as $p(k) = \sum_j p(j, k)$. The joint-degree distribution also contains enough information to compute the assortativity coefficient.

The joint-degree distribution can be extended again into three dimensions. This time a pair of distributions are required, one to describe the degrees of vertices connected in triangles, the other to describe connected triples that do not form triangles. In this manner, Mahadevan et al. define a series of distributions that capture successively more information about the topology of a network (Mahadevan et al., 2006a).

They name these distributions "dK distributions", where d is the number of dimensions. Thus, the joint-degree distribution is the $2K$ distribution, the ordinary degree distribution is the $1K$ distribution, and the mean degree can be considered as the $0K$ distribution. Each successive distribution captures all the information from all the lower distributions. For example, the mean degree, degree distribution, and joint-degree distribution of a network can all be computed given the $3K$ distribution of that network.

Multidimensional probability distributions are inconvenient to work with, so dK distributions for $d > 1$ are not used in this thesis. However, the theory is used to inform the design of some experiments.

There is empirical evidence that a wide range of important local and global topological properties of complex networks are determined by the $3K$ or $4K$ distributions (Mahadevan et al., 2006a). This can be taken as further evidence that global properties of complex networks (such as a small diameter) are a result of the local organisation of the network.

2.2.4 The small-world effect

The small-world effect is the most popularly known property of complex networks (specifically, social networks, although the effect is seen in all varieties of complex network). The effect is also known by the phrase "six degrees of separation", following a 1960s experiment by Stanley Milgram which found that on average, any two Americans can be connected by a chain of six acquaintances (Milgram, 1967).

In the terminology of graph theory, a chain of edges connecting two vertices (which may not be adjacent) is a *path*. The number of edges in a path is the *length* of the path. Finally, the *distance* between two vertices is the length of the shortest path between those vertices. Additionally, the *diameter* of a network can be defined as the longest shortest-path (or farthest distance between any two vertices) in the network. This replaces the informal notion of "degrees of separation".

The expected distance between any two vertices in a network is measured by the *mean distance*: the mean of the distances between every pair of vertices in the network. To compute the mean distance, one must find the shortest paths between every pair of vertices. For simple graphs, such as those studied in this thesis, this can be computed in $O(n \cdot m)$ time by running a breadth-first search algorithm once from every vertex (n runs each taking $O(m)$ time). This method was used to compute the mean distance figures reported in Table 2.2.

For larger networks, where even this runtime is problematic, a sampling approach suggested by Newman (2003) may be used. A sample of N vertices is chosen uniformly at random, then a breadth-first search is run from each sampled vertex v to find the shortest paths from v to every other vertex in the graph. Every shortest path is equally likely to appear in the sample. Ultimately, none of the networks studied in this thesis were large enough to require sampling.

The mean distance between vertices in most complex networks has been found to be extremely short, especially considering the large size of some of these networks: see Table 2.2 for example. Additionally, most of these networks are known to have small diameters (Boccaletti et al., 2006). This seems counter-intuitive (particularly when one discovers short paths in one's personal network of acquaintances), but in fact it should not be so surprising.

Suppose, for example, that every vertex in a network has approximately 10 neighbours. Define the *1-hop* neighbourhood of a vertex v as all the vertices that can be reached from v by a path of length 1. Similarly, the *k-hop* neighbourhood of v is defined as all the vertices that can be reached from v by a path of length $\leq k$. The size of the 1-hop neighbourhood is approximately 10. The size of the 2-hop neighbourhood, assuming there are not many edges within the 1-hop neighbourhood, is $10 \times 10 = 100$. The size of the k-hop neighbourhood is thus 10^k, and so the maximum distance from v to any other vertex in the graph is approximated by $\log_{10}(n)$ where n is the size of the network. A version of this argument is included in Newman (2003).

There is a common misconception that a small-world network is any network that has short average distances, but short distances alone do not say much about the structure of a network. The small-world effect, as recognised by Milgram (1967) and later formalised by Watts and Strogatz (1998), refers to the presence of short average distances in combination with structure that would seem to exclude the possibility of such short distances.

In social networks, for example, most edges cover only very short geographical distance; and yet the average (network) distances between geographically distant people remain small. Both these properties are observed in the global Facebook network, for instance (Ugander et al., 2011). This is the paradox of the small-world effect.

Furthermore, the assumption that there are few edges between the neighbours of a vertex in complex networks has been shown to be false. The degree to which the neighbours of vertices are interconnected is known as *clustering*, and there are several ways to measure it.

In a simple graph (no self loops or multiple edges), the local clustering C_v of a vertex v is the ratio of the number of edges between the neighbours of v and the maximum number of such edges $\binom{d(v)}{2}$ where $d(v)$ is the degree of v. Hence, the clustering of a vertex measures how close the vertex is to forming a clique with its neighbours. C_v is normalised to fall in the range $0 \le C_v \le 1$. For multigraphs a subtly different definition is required (since there is no maximum number of edges): the ratio of the number of triangles containing v to the number of pairs of adjacent edges centred on v.

The clustering of an entire network can be summarised in two different but similar ways, which are often confused in the literature (Newman, 2003). The first metric, which will be referred to in this thesis as the *clustering coefficient*, is 3 $\times$ the number of triangles $\div$ the number of pairs of adjacent edges. The factor of 3 normalises the clustering coefficient C so it lies in the range $0 \le C \le 1$ (since each triangle contributes three pairs of adjacent edges). The second metric is the *mean local clustering*, which is defined as the mean of C_v for all vertices v in the network.

Both clustering metrics have been computed on a range of networks, and the values are reported in Table 2.2. Clearly, there is a significant amount of clustering in all the networks (except curiously the Gnutella network). This is in line with published literature (Boccaletti et al., 2006).

Notice that the mean local clustering is greater than the clustering coefficient in all the networks in Table 2.2. This is a common pattern in heavy-tailed networks. It arises because, due to degree constraints, the neighbours of the hubs cannot be highly interconnected. Thus, most of the clustering is in the low degree vertices. Therefore, in a highly clustered heavy-tailed network, the majority of the vertices have high clustering, but there are a small number of vertices with very low clustering. The mean local clustering emphasises the contribution of the highly clustered low degree vertices, whereas the clustering coefficient is more affected by the poorly clustered hubs.

The paradoxical coexistence of high clustering and short average distances was explored by Watts and Strogatz (1998). They developed a simple network model which demonstrates a mechanism by which the small-world effect can arise. The Watts-Strogatz model starts with a large number of vertices connected in a ring. From each vertex, edges are added connecting to all the k-hop neighbours, for some small k. This creates a network with a high clustering coefficient and high average distances. Watts and Strogatz then proceed to rewire the edges at random. The rewiring operation is as follows: pick a vertex v and a vertex u adjacent to v, then remove the edge uv and replace it with vx for a randomly chosen vertex x.

This causes the mean local clustering of the network to fall. However, the rewired edges act as *shortcuts* to distant parts of the network, causing the average distance to fall as well. As it happens, the average distance falls much faster than the mean local clustering, thus it is possible for a network to exhibit both high clustering and short average distances.

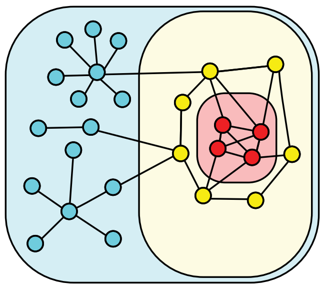

Figure 2.2: The k-core decomposition of a graph. The blue (biggest) box encloses the 1-core, the yellow box encloses the 2-core, and the red (smallest) box encloses the 3-core (which is the k-core of this graph). Additionally, the vertices of the 1-shell are coloured blue, the vertices of the 2-shell are coloured yellow, and the vertices of the 3-shell are coloured red. Notice that in this case, the highest degree vertex is in the 1-shell.

The small-world effect is an example of how various topological properties can interact in unexpected ways in large networks. The addition of a very small number of shortcuts relative to the size of the network leads to a large change in the global topology of the network (as measured by mean distance and diameter). The idea of making small changes to a network to achieve dramatic changes in behaviour is revisited later in this thesis, in Chapters 6 and 7.

It must be noted that the Watts-Strogatz model is not the only process that can lead to a small-world network (for example the configuration model described in Section 3.2.2 generates small-world networks). In fact, Watts-Strogatz style models are rarely used in recent literature as they fail to capture other important topological properties of complex networks, most importantly the heavy-tailed degree distribution (Newman, 2003). In heavy-tailed networks, it is most likely the hubs that create the shortcuts that bind the network together.

2.2.5 k-Core and k-shell decomposition

The k-core decomposition of a network captures important connectivity patterns. It has a variety of uses in network visualisation, fingerprinting, and estimating the influence of a vertex (Alvarez-Hamelin et al., 2005; Kitsak et al., 2010). Although not so widely studied as degree distributions and the small-world effect, the k-core decomposition reveals topological effects that are not apparent from the more widely reported metrics. In fact, in the search for previously unrecognised topological structure in complex networks, the author ended up independently rediscovering a crude form of the k-core decomposition.

The k-core decomposition divides the vertices of a network into a series of overlapping sets known as the k-cores. For any positive integer i, the i-core is a subset of the $(i-1)$-core, and the 0-core contains all the vertices in the network. Thus, each core is nested inside the previous core. By convention, the "k-core" of a network is the i-core with highest i (this can be confusing because the lower i-cores are also referred to as the k-core*s* of the network).

The i-core is computed according to a simple iterative process: Every vertex with degree less than i is removed from the network. This step is repeated on the new network until there are no vertices left with degree less than i. At that point, the remaining vertices constitute the i-core of the network.

k-cores are related to a well-known metric from graph theory, *degeneracy*, although degeneracy is not commonly applied to complex networks. A network is d-degenerate if the vertices can be ordered such that every vertex v is adjacent to no more than d vertices ahead of v in in the ordering. Degeneracy is computed by iteratively removing the lowest degree vertex, noting the maximum lowest degree encountered. This is in fact a variation on the process used to compute the k-core decomposition. Thus, the value of k for the k-core of a network is d. Furthermore, every k-core represents the part of the network left over when the $(k-1)$ degenerate part is removed. Note that a network with *low* d is said to be *highly* degenerate, and vice versa.

As well as k-cores, one can also study k-shells. The i-shell for some integer i is the i-core minus the $(i+1)$-core. If the i-core does not contain at least one vertex of degree i, then the i-core is equal to the $(i+1)$-core and the i-shell will be empty. Every vertex belongs to one i-shell, and i is known as the *k-shell number* for that vertex. Figure 2.2 illustrates the k-core and k-shell decompositions of a graph, and how the two relate to each other.

The k-core decomposition captures the connectivity patterns of the network. The strongly-skewed nature of complex network degree distributions implies that there are many low degree vertices, which are relatively poorly connected connected to the rest of the network. These vertices must be found in the low k-shells. The higher minimum degree of the high k-shells implies that they are relatively dense, and that vertices in these shells are probably well connected to each other. Note that it is possible for vertices with very high degree to appear in the low shells, for example in Figure 2.2, where the highest degree vertex is in the 1-shell. Such vertices have relatively poor connectivity despite their high degree.

Many complex networks are known to have a hierarchical structure, consisting of a densely connected *core*, surrounded by a loosely connected *periphery*. An excellent way to quantify this property is to measure the sizes of all the k-shells of a network. This was first done by Alvarez-Hamelin et al. (2008) using the Internet AS-level graph (note that there are preprints of the paper going back to 2005). k-shell decompositions for a wider variety of networks are given by Dorogovtsev et al. (2008). Dorogovtsev et al. (2006) also note that in complex networks, the value of k for the k-core is much higher than in random networks, i.e. complex networks have relatively high degeneracy.

The k-shell decompositions of all the networks listed in Table 2.1 were computed for this chapter, and the results are reported in Figure 2.3. For most of the networks the majority of the vertices are in the lowest shells, but all of the networks have the relatively high degeneracy observed by Dorogovtsev et al. (2006). This can be interpreted as saying that most of the networks have a large and poorly connected periphery, combined with a small, dense, highly connected core. This trend is present even in the Physicists 2 network, although it is weak, and the scale makes it difficult to see.

The exceptions to this rule are the two biological networks. In the Neural network, most of the vertices are in the core. This tells us that most neurons in the network have good connectivity to other neurons, and there is no periphery to speak of. This may be related to the function of the network. The Metabolic network exhibits even stranger behaviour. The largest shells fall in the middle of the range, telling us that most vertices are neither well connected nor poorly connected. There is neither a periphery nor a core.

One might expect that in a heavy-tailed network, the hubs would naturally make up the core of the network, and the low degree vertices would form the periphery. Hence, the k-shell decomposition would follow a similar pattern to the degree distribution. Although this is seen in the majority of the k-shell decompositions reported in Figure 2.3, there are enough exceptions to conclude that a variety of k-shell topologies are possible given the same degree distribution. In particular, a glance at Table 2.3 reveals that the Physicists 1, Physicists 2, and Neural networks all have strong power-laws in their degree distributions, and yet exhibit notably different k-shell topologies.

The k-shell topology of complex networks has obvious and important consequences for algorithm design, although unfortunately the author was not able to derive any direct applications from the theory. In a typical complex network, most vertices have low k-shell numbers, so greedy algorithms that iteratively remove low degree vertices can be highly effective.

An example of an algorithm that benefits from the typical k-shell topology of complex networks is the standard vertex-cover kernelization algorithm. The vertex cover problem, and the kernelization algorithm are discussed in more detail in Section 5.3.1. The kernelization algorithm operates by iteratively removing very low and very high degree vertices from the network according to carefully designed rules. It has been noted that this algorithm is more effective in practice than the theory would suggest (Abu-Khzam et al., 2004). This is most likely due to k-shell topology. Since the low

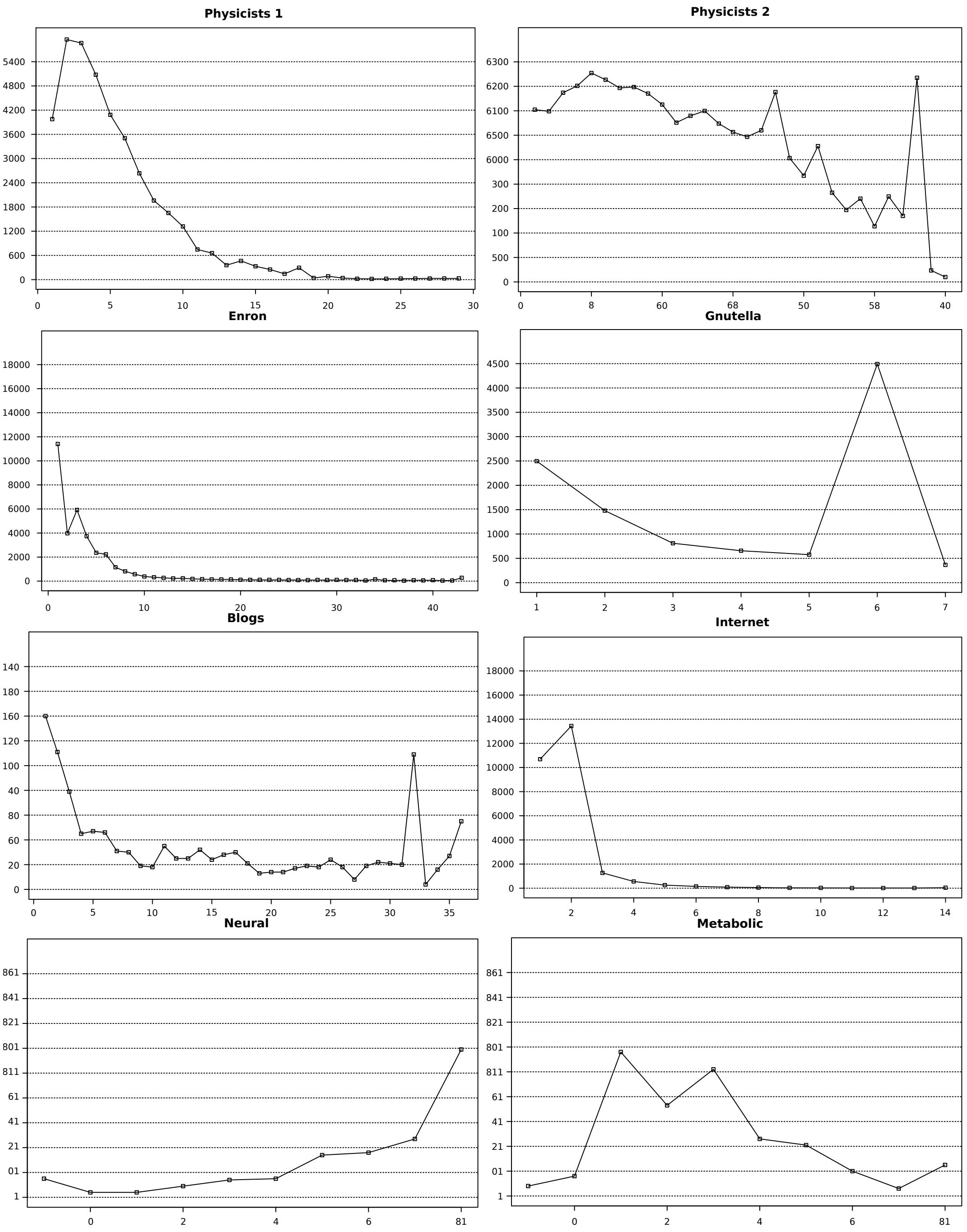

Figure 2.3: k-shell decompositions of several complex networks. Each chart shows the number of vertices (x-axis) per shell (y-axis) for one of the networks described in Section 2.1.

shells are very large, many low degree vertices can be removed from the graph, including vertices that didn't have low enough degree at the beginning of the process.

2.2.6 Centrality metrics

The centrality of a vertex in a network refers to the influence of that vertex on the structure or function of the network. Many metrics on complex networks show strongly-skewed behaviour, such as the degree distribution and the k-shell decomposition. This leads to a small number of vertices having significantly higher values than do typical vertices for these metrics. Such metrics can be used to quantify aspects of centrality. For example, a vertex with high *degree centrality* has high degree. A vertex with high *k-core centrality* has a high k-shell number.

Another widely used centrality metric is the *betweenness centrality.* The betweenness of a vertex v is the fraction of shortest paths between *all* pairs of vertices in the network that pass through v. The betweenness of an edge can be defined similarly, although the utility of edge-betweenness as a measure of centrality was debunked by Newman (2003). The naïve way to compute betweenness, by finding then counting all the shortest paths, is too slow to be practical on large networks, but an efficient $O(m \cdot n)$ algorithm to compute betweenness is given by Brandes (2001).

Betweenness is intended to measure the role of a vertex in transmitting information from one part of the network to another. In a social network, for example, high betweenness vertices might act as gatekeepers between communities. In the case of the Internet, all traffic is routed via the shortest path, so high betweenness vertices are expected to see high levels of traffic. For this reason, betweenness is of particular interest to Internet researchers (Mahadevan et al., 2006b).

A *betweenness distribution* can be defined in a similar way to the degree distribution. The betweenness distribution has been shown to follow a power-law distribution in complex networks (Goh et al., 2002; Mahadevan et al., 2006b).

Kitsak et al. (2010) compared the relative utility of degree centrality, k-shell centrality, and betweenness centrality with regards to two simple models of infectious disease spreading. A good centrality measure should predict which vertices are likely to lead to larger outbreaks, should they become infected. Kitsak et al. found that the k-shell number of a vertex largely determines the size of the outbreak. In addition, the betweenness was found to be a very poor predictor of the size of the outbreak.

This thesis aims to apply the known topological properties of complex networks to two problems related to spreading processes, described in Chapter 5. Since betweenness centrality seems to be unrelated to spreading, it is not investigated further in this thesis, despite the popularity of the metric.

2.2.7 Community structure

Intuitively, one would expect to find evidence of community structure in at least some kinds of complex networks. A community here is defined as a set of vertices that are more connected to each other than to the rest of the network. For example, a community in a social network might represent a group of friends. In the world-wide-web network, groups of pages on related topics might form communities.

The problem of dividing a network into communities is known as clustering, and there is extensive literature on the topic. The earlier approaches operate by computing a *connection strength* metric for every edge in the network. The edges can then be added in order of decreasing connection strength to build the communities from the bottom up (Newman and Girvan, 2004). Alternatively, one can rank the edges by edge-betweenness or some similar metric, and remove the edges in decreasing order of betweenness. In this way, the network is gradually broken down into smaller and smaller communities.

With these approaches, the number of communities found increases or decreases depending on how many edges are added or removed. The complete set of communities can be captured as a tree structure, known as a dendrogram. The root is the entire network, and the leaves are individual vertices. The inside nodes represent the merging of multiple communities. Using probabilistic methods, it is possible to directly compute a maximum likelihood dendrogam, without requiring a connection strength or betweenness metric (Clauset et al., 2008).

The problem with all these approaches is knowing which level of the dendrogram represents the true community structure of the network. Newman and Girvan (2004) give a simple *modularity* metric, which measures how many more edges there are within the communities than between them. This leads to another clustering algorithm which operates by directly optimising the modularity metric (Newman, 2006). Newman's modularity metric is now very widely used as the most convenient way to uncover community structure in a complex network.

It was hoped in the early stages of the research presented in this thesis, that community structure theory could be applied to the design of algorithms targeted for complex networks. If the communities are dense, and there are few connections between them, then it would be possible to compute optimal solutions within the communities, then combine the solutions across the boundaries. However, even in the blogs network, which is strongly divided into two clear communities (Adamic and Glance, 2005), the connections within the communities are still relatively sparse, and there are still large numbers of connections between the communities.

The k-shell structure goes some way towards explaining the weakness of communities in complex networks. Since the vertices of a community are required to be well connected to each other, the centres of the communities lie in the high k-cores. However, the high connectivity of the k-core implies that relatively dense parts of the communities must be highly interconnected with each other. Thus, the strong isolation desired is not seen in typical complex networks.

2.3 Summary

This chapter has introduced a selection of eight publicly available network datasets that will be used for experimentation throughout this thesis. These networks come from a range of domains, including social networks (Section 2.1.1), communications networks (Section 2.1.2), and biological networks (Section 2.1.3). These categories include the main varieties of complex networks reported on in the literature (Boccaletti et al., 2006; Newman, 2003).

The second part of this chapter (Section 2.2) described a number of topological properties that are commonly observed in complex networks, including the networks described in Section 2.1. The degree distribution (Section 2.2.1), and the small world effect (Section 2.2.4) are the most widely reported of these properties, but the other properties are also of relevance to the remainder of this thesis.

The next chapter will survey the literature on random networks. Random networks serve two purposes in the study of complex networks: as null models to show that the distinctive topologies of complex networks are not due to chance, and as models of how those topologies can emerge from simple rules applied stochastically.

Chapter 3

Methodology using random graphs

As discussed in Section 2.1, a range of eight empirical networks were selected for use in the experiments that come later in this thesis. These networks were selected mainly for pragmatic reasons: the datasets are publicly available, the networks have convenient size and density, and they cover a broad range of domains. However, generalising the conclusions of experiments is difficult for such a small and varied selection of networks.

For example, this thesis aims to make statements that apply to most complex networks, but topological variations in particular networks can act as confounding factors. These confounding factors can be eliminated by using constructed networks, designed such that all the structural properties are known and understood. It is also necessary to include a set of *control* graphs in every experiment. An ideal control graph should have as little topology as possible, so that one can distinguish between effects that are due to complex network topology, and those that are not.

Constructed networks, and control networks for the experiments are generated using random graphs models. This thesis proposes to use two random graph models, which will be referred to as *0K random graphs* and *1K random graphs* using the *dK* terminology of Mahadevan et al. (2006a) (described in Section 2.2.3). 0K random graphs have a fixed 0K distribution, which is to say a fixed mean degree. 1K random graphs have a fixed degree distribution. A complete description and justification of these particular models is included in Section 3.1.

By comparing the results obtained on the real networks with random networks that have the same 0K and 1K distributions, the results of the experiments can be generalised. It becomes possible to distinguish between effects that are due to the degree structure of complex networks, effects that are exclusive to particular networks, and effects that are not due to any specific topology.

Having determined what kinds of random graphs are required, the next question is how to generate those graphs. Many random-graph models are described and analysed in the complex networks literature. Depending on one's perspective, a random-graph model can be thought of as a stochastic procedure for generating graphs, or the ensemble of graphs generated by that procedure. (The computer scientist's view versus the physicist's view). This thesis takes the procedural view.

Two approaches to generating random graphs are described in this chapter. The generative models of Section 3.2 construct new graphs that meet some criteria (determined by the parameters of the individual models). The rewiring models discussed in Section 3.3 begin with an existing graph, then randomly rewire the edges in such a way as to preserve certain topological structures, while randomizing everything else.

This chapter begins by outlining the experimental methodology used throughout the remainder of this thesis, and describing the uniform random graphs required for that methodology. As it turns out, generating uniform random graphs that meet the requirements of this methodology is a difficult problem. Sections 3.2 and 3.3 review the relevant random-graph models from the literature, while analysing their suitability for the methodology of this thesis.

The degree-preserving rewiring model described by Gkantsidis et al. is found to be the most practical solution to the problem of generating uniform random graphs. This rewiring scheme relies on two theorems, first proved by Taylor (1980). Copies of Taylor's proofs are somewhat difficult to obtain, and they are in spirit existence proofs. For these reasons, independent constructive proofs of the two theorems are presented in Chapter 4

3.1 Random graphs as null models

The experiments that follow in later chapters of this thesis are of two kinds: those that compute metrics on complex networks (to determine the presence or absence of certain structures), and those that evaluate the performance of algorithms on complex networks. Two kinds of random graph are used in the experiments, these will be referred to as 0K and 1K random graphs.

The methodology is as follows: for each of the empirical networks listed in Table 2.1, two random graphs are computed, a 0K random graph and a 1K random graph. The 0K random graph has the same size and mean degree as the empirical network. The 1K random graph has the same size and degree distribution as the empirical network.

The relevant metrics are computed on the empirical networks and the random networks. It is then possible to compare the results for each empirical network with its two random networks, and also to compare the results across all the empirical networks, across all the 1K networks, or across all the 0K networks.

Notice that the 0K random graphs in general have *no* local topology, such as clustering or community structure. Thus, by including 0K random graphs in the experiments, it becomes possible to distinguish between effects that are due to the specific topology of the empirical complex networks, and effects that are not due to topology.

1K random graphs are included because (as discussed in Section 2.2.1) the heavy-tailed degree distribution is considered by many to be a defining characteristic of complex networks (Li et al., 2005; Dorogovtsev and Mendes, 2000; Newman, 2003). Including 1K random graphs allows one to draw conclusions from the experiments concerning heavy-tailed graphs in general, rather than just a few specific empirical examples. Additionally, it becomes possible to determine whether an effect is due to the degree distribution of the network, or some other topological property.

A logical extension of this methodology would be to also include 2K and 3K random graphs with the same 2K and 3K distributions as the empirical networks. This would allow one to observe how assortativity (determined by the 2K distribution) and clustering (determined by the 3K distribution) affect the results. Including these random graph models is not done as it would be beyond the scope of this thesis, which is concerned with techniques that are broadly relevant to all complex networks. Additionally, there is little literature on how to generate the required random graphs (Stanton and Pinar (2012) were the first to give a rigorous and practical method to generate uniformly random 2K graphs).

A clear problem with this approach is that only one random graph of each type is computed per empirical network. It is possible that an atypical graph might arise by chance and cause misleading results. This possibility is avoided by repeating every experiment 10 times to confirm that the results are typical. This works because, as will be seen later in this chapter, most random graphs derived from any particular model are almost indistinguishable (using known topological metrics for complex networks).

Statistical hypothesis testing is not used in the experiments presented in this thesis, which is concerned mainly with finding practical techniques that can be applied to complex networks. Thus, the experiments are usually looking for qualitative effects of *practical* significance.

Where it *is* necessary to measure the difference between means of large datasets, confidence intervals are computed using the conventional t-distribution, and plotted with the experimental results. In practice the confidence intervals are usually very tight. This is because, as previously mentioned, most random graphs of any particular kind exhibit very similar topology. Since the intervals are so tight, significance can be easily determined by sight. If the intervals do not overlap, the effect is significant, otherwise it is not.

The 0K and 1K random graphs are required to be *uniformly* random. This means that, for the 0K graphs, every graph of a particular size and mean degree is equally likely to be produced. For the 1K graphs, every graph of a particular size and degree distribution is equally likely to be produced. Additionally, since all the networks are modelled as simple graphs, the two random graphs are also required to be simple (i.e. undirected, unweighted, no self loops, and no more than one edge between any two vertices).

It is easy to generate 0K simple graphs uniformly at random. The Erdős-Rényi random graph model discussed in Section 3.2.1 has this capability, as does the random rewiring model discussed in Section 3.3.1. Generating the 1K random graphs on the other hand is considerably more challenging, and there is an extensive literature on the topic.

Sections 3.2 and 3.3 review the literature on several models for generating 1K random graphs, while also evaluating how well those models satisfy the uniform and simple graph criteria. The comparison is partly experimental, and partly based on the known properties on the models.

3.1.1 Comparing random graph models: methodology

The models discussed in Section 3.2 attempt to generate random graphs by sequentially adding vertices and edges, starting with an empty graph. The main concern with these models (for this thesis) is whether or not they generate 1K graphs *uniformly* at random.

It could be the case that most of these models generate an approximately uniform random sample of graphs, in which case it may not matter very much which model is used. If this is the case, then there will be very little topological variation between large samples of graphs generated by different models. This possibility is explored experimentally.

The methodology for the experiments in Section 3.2 is as follows. From each model, a sample of 100 random graphs with power-law degree distributions is generated. In order to reduce the number of variables, all the graphs have 1000 vertices and a mean degree of 8. The mean degree should be interpreted here as a measure of density; due the highly skewed nature of complex network degree distributions, the mean degree does not accurately indicate the expected degree.

For models where the power-law exponent can be adjusted, 6 samples are generated with different power-law exponents ranging from 2.0 to 3.0 in increments of 0.2. Two metrics are then computed and averaged over all the random graphs of each kind. By plotting the results with 95% confidence intervals it becomes clear that there is significant topological variation across the random graphs generated by different models. The null hypothesis (that the models are approximately uniform) can therefore be rejected.

The metrics used are the assortativity coefficient, and the clustering coefficient.

Both of these metrics are computationally easy to compute, and capture topology that is mostly independent of the degree distribution. The assortativity coefficient can be thought of as a summary statistic of the joint-degree distribution, and similarly the clustering coefficient can be regarded as a summary statistic of the 3K distribution (Mahadevan et al., 2006a).

If one accepts the claim of Mahadevan et al. that the 3K distribution captures all the key topological properties associated with complex networks, then assortativity and the clustering coefficient are the right metrics for these experiments. It is the view of the author that they are superior to the distance-based metrics that are more commonly employed in this area (by Gkantsidis et al. (2003) for example). The problem with distance based metrics (such as average distance and diameter) is that they are hard to relate to the local topology of the network.

For simple uniform 0K random graphs, the assortativity coefficient must be 0. Since every edge is independent, there will be no correlations between the degrees. Thus, the assortativity coefficient could be used as a simple test of randomness. This technique does not apply to simple uniform 1K random graphs. The combination of the degree distribution constraint, and the simple graph constraint, creates a bias towards disassortativity. This is because there are many more low degree vertices than hubs, and the simple graph constraint disallows duplicate edges between hubs, so the hubs must be mostly connected to vertices of dissimilar degree.

The randomizing models discussed in Section 3.3 start with a source graph, then rewire the edges randomly. These models are known in theory to generate uniform samples of random graphs. A potential drawback of rewiring is that, in the case of 1K random graphs, there is no known upper bound to the number of rewiring operations that may be required to ensure a uniform distribution. However, recent theoretical and experimental evidence suggests that the required number of rewiring operations is $O(n)$ (Ray et al., 2012). Independent experiments presented in Section 3.3 present further evidence of this claim, based on measurements of the assortativity coefficient, a metric not previously used for this purpose in the literature.

The truly uniform 1K random graphs from Section 3.3 can be compared with the generative models from Section 3.2 to evaluate how close to uniform those models really are. The comparison is done by computing for the randomized graphs the same metrics that were computed for the graphs from the generative models. It is found that there is a significant difference, so it can be concluded that none of the generative models can approximate uniform sampling, and therefore they are not suitable for use in the experiments that follow in the later chapters of this thesis.

3.2 Generative models

The generative models construct random graphs by starting with an empty graph, and adding vertices and edges until the relevant constraints are satisfied. Of these models, the Erdős-Rényi model (Section 3.2.1) generates 0K random graphs, while the others generate (or are thought to generate) 1K random graphs.

The main purpose of this section is to compare several models for 1K random graphs with each other, in terms of the assortativity and clustering coefficients, as described in Section 3.1.1. If all the models generate topologically similar graphs, then one can assume that the details of how a random graph is generated do not greatly affect the topological properties of the graphs, and hence any of the models are safe to use for generating uniform samples of 1K random graphs.

It turns out that this is not the case, and in fact the different models generate significantly different sorts of graphs. Further experiments in Section 3.3 reveal that none of the models presented in this section generate uniformly random graphs.

Of the models that generate 1K graphs, the Configuration model and the Havel-Hakimi algorithm both take a degree sequence as a parameter. The preferential attachment model discussed in Section 3.2.4 only generates graphs with power-law degree distributions.

In order to use models that require a degree sequence parameter, it is necessary to generate a sequence of degrees that matches the desired power-law distribution. For a given exponent γ and scale factor α, the formula is $d_i = i^{-1/(\gamma-1)}\alpha$ where d_i is the degree for vertex i.

As stated in Section 3.1, it is necessary to match the mean degree of the random graphs with the mean degree of the empirical graphs with which they are to be compared. Here, mean degree is being used as a measure of density. In a power-law setting it cannot be interpreted as the expected degree of a vertex. Mean degree ($\bar{d}$) is related to α and γ by the following equation:

$$\bar{d} = \frac{\sum_{i=1}^{n} d_i}{n} = \frac{\sum_{i=1}^{n} i^{-1/(\gamma-1)}\alpha}{n} \tag{3.1}$$

The full equation for a power-law degree sequence is derived by rearranging equation 3.1 for α and substituting it back into the power-law equation:

$$d_i = \frac{n\bar{d}}{\sum_{j=1}^{n} j^{-1/(\gamma-1)}} \cdot i^{-1/(\gamma-1)} \tag{3.2}$$

3.2.1 Erdős-Rényi random graphs

The Erdős-Rényi (ER) random graph model was introduced in a series of papers starting with Erdős and Rényi (1960). The model has two parameters: the number of vertices

n, and the number of edges m. All *multigraphs* with n vertices and m edges are equally likely to occur.

The ER model begins with n vertices and no edges. m edges are added one at a time. For each edge, two endpoints are selected independently and uniformly at random. An edge is then added between the two endpoints. This procedure can be modified to avoid generating graphs with self loops by selecting two *distinct* endpoints at random. The generated graphs will still be uniformly random.

The ER model can be easily adjusted to generate only simple graphs. Two distinct endpoints are selected at random, as with the original ER model. If there is no edge between the endpoints, a new edge is created. Otherwise no edge is created. The procedure continues until all m edges have been created. The edges are still independent of each other, so this modified model generates all simple graphs uniformly at random.

The original purpose of the ER model was to prove the existence or not of graphs with certain properties, and also to characterise the likelihood of generating graphs with certain properties. This is done by studying how the probability of generating a graph with the desired properties changes as $n \to \infty$. If the probability approaches 1, the property holds almost surely; if the probability approaches 0, almost no random graph has that property; otherwise it can be concluded that the desired property holds for at least some graphs. An in-depth discussion of this probabilistic method can be found in textbooks such as Diestel (2000).

Two well-known properties of random graphs that are of particular relevance to complex networks are the presence of a giant component, and the existence of short paths. Suppose that the parameter m, the number of edges, is a function of n, the number of vertices. If m grows faster than $\frac{(n-1)\log(n)}{2}$, then the ER model generates graphs that are almost surely connected (every pair of vertices can be connected by a path) (Diestel, 2000). Physicists refer to this property as a *phase transition* from a low density phase characterised by many small components, to a high density phase where there is a *giant component* of size $O(n)$ (Newman, 2003).

The existence of short paths in random graphs follows from the same argument presented in Section 2.2.4. Since the edges are independent, $d^x \approx n$ where d is the mean degree and x is the mean path length. Hence, $x = \frac{log(n)}{log(d)}$. If the mean degree is fixed, the average path length will remain small even as the random graphs get large. This simple argument comes from (Newman, 2003).

Other than these two properties, random graphs are very different from complex networks. Random graphs have Poisson degree distributions, which are centred about a mean, as opposed to the highly skewed degree distributions seen in complex networks (Newman, 2003). Since every edge is independent, there are no correlations in the degrees of adjacent vertices, and therefore no assortativity. For similar reasons, random graphs do not exhibit clustering.

3.2.2 Configuration model

The configuration model operates on a similar principle to the Erdős-Rényi model. This time there is only one parameter, a degree sequence (which also determines the number of vertices n and the mean degree). The procedure begins with a graph of n vertices and no edges. In order to maintain the degree sequence constraint, the configuration model assigns a number of "half-edges" to each vertex v, equal to the degree of v. Edges are then added one-by-one between randomly selected pairs of half-edges, in a manner analogous to the Erdős-Rényi model. In this way, all *multigraphs* with the desired degree sequence are generated uniformly at random (Molloy and Reed, 1995).

For each edge in the configuration model, the two endpoints are drawn randomly from the same probability distribution, so the expected assortativity coefficient is 0. For a power-law degree distribution, the behaviour of the clustering coefficient depends on the power-law exponent γ (Newman, 2003). When $\gamma < \frac{7}{3}$, the clustering coefficient increases with the size of the graph. In a sufficiently large graph it can even exceed 1 (this is only possible in multigraphs). For other values of γ, the clustering coefficient tends to 0 as the number of vertices increases, as in Erdős-Rényi random graphs. However, the rate of decrease is much slower, so unlike the Erdős-Rényi model, power-law configuration model graphs typically have significant clustering.

The main difficulty with the configuration model is that it generates multigraphs, whereas the methodology described in Section 3.1 requires simple graphs. Unlike for the Erdős-Rényi model, the configuration model cannot easily be modified to generate simple graphs uniformly at random.

A naïve way would be to select a pair of distinct half-edges at random, and attempt to add an edge between them. If this is not possible, select a different random pair of half-edges. It turns out that this scheme does not generate graphs uniformly. Furthermore, it may get stuck in a situation where the degree sequence is not yet realised (there are still unconnected half-edges), but no new edges can be added without violating the simple graph constraint.

The problem is that in a random simple graph with a specified degree sequence, there are dependencies between the edges. This is easy to see by considering the size of the space of edges that could be added. Suppose there are three vertices for example: u has 2 half-edges, v and w have 1. If the edge uv or uw is added, then there is one possibility for the next edge. However, if vw is added, then there are no possibilities for the next edge. Thus, the available choices for later edges depend non-trivially on which edges were added earlier in the process.

Projected configuration model

Another possible solution to the problem of modifying the configuration model to generate simple graphs is to generate multigraphs, then remove any self-loops or multiple edges that may have occurred. This operation will be referred to as *projection* to a simple graph. Deleting edges will change the degree sequence, and is likely to affect the uniformity of the generated graphs. However, assuming that the generated graphs are already close to being simple, it is possible that this process could come very close to generating simple graphs uniformly at random.

To see the effects of the projection operation on the topology of the graphs generated by the projected configuration model, six samples of 100 configuration model graphs were generated with power-law exponents ranging from 2.0 to 3.0 in increments of 0.2 as described in Section 3.1.1. Each of these graphs was then projected to a simple graph. Metrics were then computed for all the graphs, and the means and confidence intervals computed as per Section 3.1.1.

The basic form of the configuration model, which allows self-loops, was used for these experiments. The experiments were later repeated using the self-loop-free version of the configuration model, and the results (not reported) were similar.

Since the projection operation affects the degree distribution, it was necessary to confirm that the effect is not too large. Power-law exponents, and p-values were computed according to the method of Clauset et al. (2009). Recall from Section 2.2.1 that this p-value is the probability that a graph has a power-law degree distribution (as opposed to some other degree distribution). A value of 0.10 is considered consistent with the power-law hypothesis.

The results of this first experiment are reported in Figure 3.1. Note that although the confidence intervals are plotted for every sample of graphs, most of the intervals are too small to be seen. This lack of variation is entirely to be expected. The original graphs all had exactly the same degree distribution, so any variation must be due to errors introduced by the projection operation.

Figure 3.1 shows that the degree distribution is indeed affected by the projection operation, and that the graphs with smaller power-law exponents are most affected. Clearly, most of the multiple edges and self-loops will be incident on the hubs, so the projection operation mainly affects the heavy tail of the degree distribution. By reducing the degrees of those vertices, the weight of the tail is decreased. A lower power-law exponent indicates a heavier tail, hence the effect of the projection operation increasing the power-law exponent.

The fact that the projection operation has a greater effect when the power-law exponent is smaller is a pattern that will be seen throughout this section. The cause of the effect is that a heavier tail must also be shorter (since the area under a probability

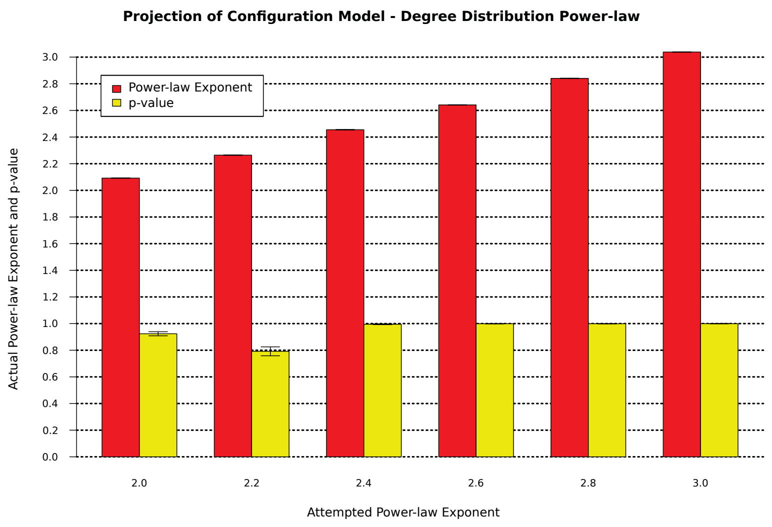

Figure 3.1: Power-law exponents and p-values after the projection operation, for a range of values of the power-law exponent. A p-value greater than 0.10 is considered consistent with the power-law hypothesis. Each bar is the average over 100 random graphs, with 95% confidence intervals. All graphs have mean degree 8.

distribution function is always 1). This implies a higher proportion of multiple edges and self loops on a smaller number of hubs, leading to the greater effect of the projection.

Although the projection operation affects the degree distribution of the graphs, the important properties of skewness and the heavy tail are preserved. Therefore, the effects on the degree distribution due to the projection operation are judged to be acceptable for the purposes of the experimental methodology outlined in Section 2.2.7.

It is well known that configuration model graphs of sufficient density have a giant component of size $O(n)$, and as with Erdős-Rényi random graphs, there is a phase transition (Molloy and Reed, 1995). Obviously, this property is preserved by the projection operation, since the removal of self loops and multiple edges cannot increase the number of components. However, as a matter of interest, the sizes of the largest components were computed, and the results are reported in Figure 3.2.

Having determined that the projection operation does not overly compromise the power-law degree distribution, the next step is to measure the assortativity and clustering coefficients, as outlined in Section 3.1.1. These two metrics were computed on both the plain configuration model random graphs, and on the projected graphs.

If the assortativity and clustering coefficients are preserved, as well as the degree distribution, then that could be taken as evidence that the projection operation has only a small effect on the uniformity of the samples. Such a result would provide a convenient control with which to compare the other models. Unfortunately, it turns out that the projection operation has a very significant effect on both assortativity and

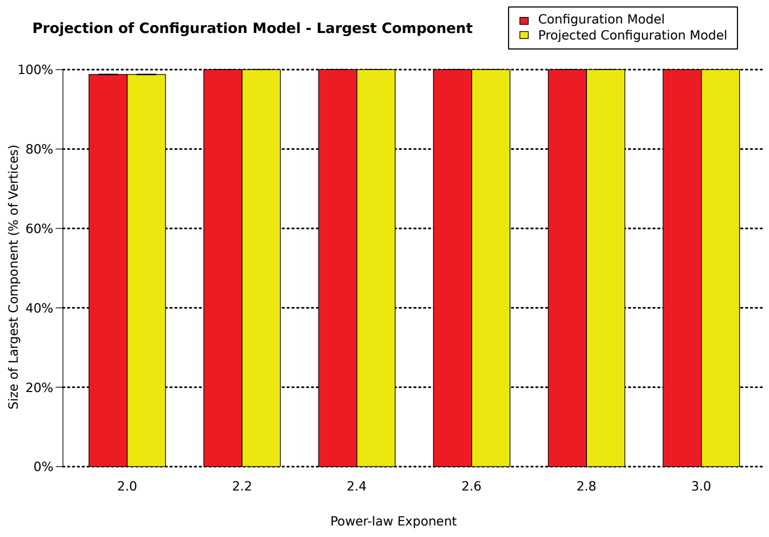

Figure 3.2: Percentage of vertices in the largest component of random graphs from the configuration model, before and after the projection operation, for a range of values of the power-law exponent. Each bar is the average over 100 random graphs, with a 95% confidence interval. All graphs have mean degree 8.

clustering.

Figure 3.3 compares the assortativity coefficients of configuration model graphs before and after the projection operation, for a range of power-law exponents. As can be seen from the error bars (which represent 95% confidence intervals), the configuration model generates graphs with no degree correlations. However, significant negative correlations appear following the projection.

The cause of the disassortativity is once again the concentration of the self-loops and multiple edges mainly on the hubs. Let an assortative edge be one that connects vertices of similar degree and a disassortative edge be one that connects vertices of dissimilar degree. Since the self-loops and multiple edges are mostly between vertices of similar (high) degree, they are mostly assortative. Overall, the configuration model graphs are neither assortative nor disassortative, so removing a large number of mainly assortative edges results in an overall disassortative graph.

The clustering results are reported in Figure 3.4. This time, the clustering is high in the configuration model graphs (as expected), but significantly lower following the projection operation. In this case the effect is due to the way multiple edges and self loops count towards the clustering coefficient.

Recall from Section 2.2.4 that the clustering coefficient is the ratio of triangles to pairs of adjacent edges. In a multigraph, there are more ways to form a triangle. For example, any combination of three edges between the same pair of vertices forms a

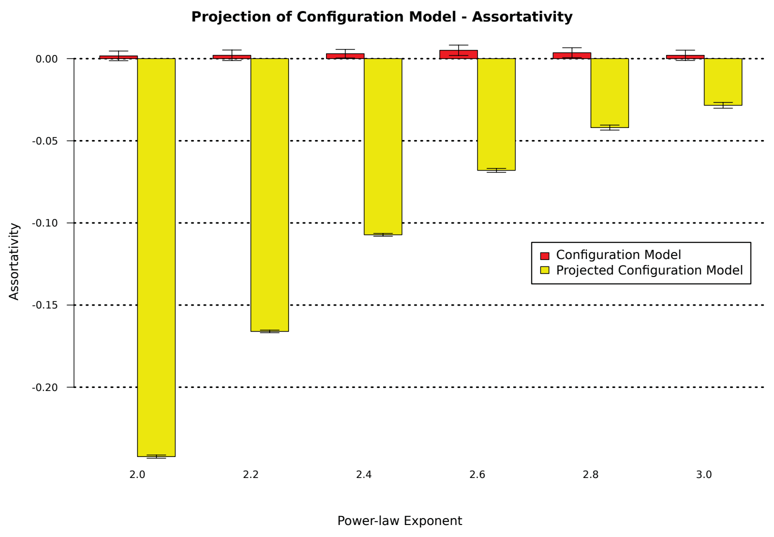

Figure 3.3: Assortativity of random graphs from the configuration model, before and after the projection operation, for a range of values of the power-law exponent. Each bar is the average over 100 random graphs, with a 95% confidence interval. All graphs were generated with mean degree 8. The assortativity coefficient ranges from 1 to -1.

triangle, any combination of three self loops on the same vertex forms a triangle, and triangles can be formed from combinations of self loops and multiple edges. Since the multiple edges and self-loops are mainly concentrated on a relatively small number of vertices, these multigraph triangles are highly likely to occur. This is why the clustering coefficient is so much smaller following the projection operation.

To summarize, the projection operation has a small, but appreciable, effect on the degree distribution. However, it has a large effect on both the assortativity coefficient and the clustering coefficient. This implies that the topological structure of the graph is changed significantly following the projection operation. It is not possible to draw any conclusions from these experiments as to whether or not the projected configuration model generates all simple graphs with the desired degree sequence uniformly at random. Further experiments presented in this chapter will show that the projected configuration model does not in fact generate graphs uniformly.

3.2.3 Havel-Hakimi procedure

A naïve modification to the configuration model to generate simple graphs with exactly the desired degree sequence works as follows. Normally, two half edges are chosen at random and a new edge is created between them. Instead, choose two half edges from the set of pairs that can be joined without violating the simple graph constraint. As explained in Section 3.2.2, this procedure can get stuck and fail to realise the desired

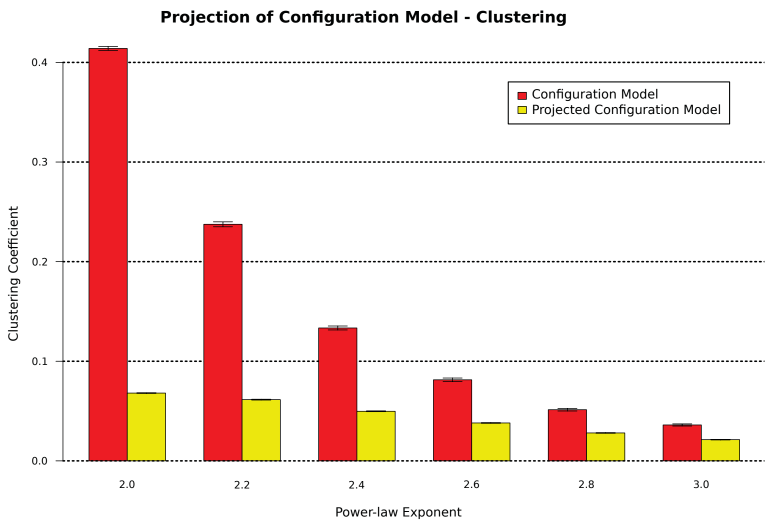

Figure 3.4: Clustering coefficient of random graphs from the configuration model, before and after the projection operation, for a range of values of the power-law exponent. Each bar is the average over 100 random graphs, with a 95% confidence interval. All graphs were generated with mean degree 8. The clustering coefficient ranges from 0 to 1 in simple graphs, although it can be greater than 1 in multigraphs.

degree sequence, should the wrong edges be added early in the process.

A more sophisticated algorithm due to Havel and Hakimi (Havel, 1955; Hakimi, 1962) is capable of constructing simple graphs that realise a desired degree sequence. This algorithm is based on the (Erdős-Gallai) condition, a necessary and sufficient condition for a degree sequence to be realisable as a simple graph (Erdős and Gallai, 1960). For all $k \leq n$:

$$\sum_{i=1}^{k} d_i \leq k(k-1) + \sum_{i=k+1}^{n} \min(k, d_i) \tag{3.3}$$

Where $d_1, d_2, \cdots, d_n$ is the degree sequence, ordered from highest degree to lowest degree. This equation states that the total degree of the k highest degree vertices is less than the maximum number of edges within the k highest degree vertices, plus the maximum number of connections available to vertices outside the k highest degree vertices. The necessity of the condition is obvious, the sufficiency is more involved.

The Havel-Hakimi algorithm begins as with the configuration model, by assigning each initially isolated vertex a degree from the degree sequence (which must satisfy equation 3.3). The *residual degree*, the difference between the intended and actual degrees, is tracked per vertex.

First a vertex v is selected, this is the *select* phase. Edges are then added between v and other vertices until the residual degree of v is 0, this is the *satisfy* phase. Following the satisfy phase, the procedure is repeated on the subgraph excluding v, until every vertex has residual degree 0. The strategies for selecting v, and selecting the vertices to connect to v, may be chosen so that equation 3.3 holds for every iteration of the algorithm.

In order to investigate the applicability of this algorithm for generating uniformly random samples of simple graphs (with a particular degree sequence), a range of strategies were compared according to the methodology outlined in Section 3.1.1. For each strategy, a large sample of random graphs was generated, then the means of the assortativity and clustering coefficients were computed per sample and compared.

Each strategy is a combination of a select strategy (for the select phase), and a satisfy strategy (for the satisfy phase). The full strategy is written as *select/satisfy*, for example, largest first/random.

For the select phase, the following strategies were tested:

- *Largest first.* Select the vertex with the highest residual degree
- *Smallest first.* Select the vertex with the lowest residual degree
- *Random.* Select a vertex of positive residual degree uniformly at random.
- *Proportional.* As with *random*, but select the vertex proportionally to the residual degrees rather than uniformly at random. Thus, a vertex with twice the residual degree is twice as likely to be selected.

For the satisfy phase, the following strategies were tested:

- *Random.* Connected the selected vertex to vertices chosen uniformly at random with positive residual degree.
- *Proportional.* As with *random*, but this time the vertices are selected proportionally to the residual degrees rather than uniformly at random.
- *Largest first.* Connect the selected vertex to the vertices of highest residual degrees.

Note that only strategies that use ‘largest-first’ for the satisfy phase are able to guarantee that equation 3.3 holds for every iteration of the algorithm. The other strategies are included in the interests of generating a wider variety of random graphs.

Although every combination of select and satisfy strategies was tested, results are reported only for the following strategies: largest-first / random, largest first / proportional, smallest-first / random, smallest-first / proportional, random / largest-first, random / proportional, proportional / proportional. These strategies give a good overview of the full range of results.

Before comparing the assortativity and clustering coefficients, a simple experiment was performed to check how well the different strategies realise the intended degree sequence. This is necessary because not all of the combinations of select/satisfy are guaranteed to maintain the Erdős-Gallai condition (equation 3.3).

A sample of 100 graphs was generated for every strategy, and for a range of power-law exponents ranging from 2.0 to 3.0 in increments of 0.2. The number of missing edges per graph, i.e. the number of edges that could not be inserted without violating the simple graph constraint, was computed for every graph. The mean for each sample is plotted with a 95% confidence interval in Figure 3.5.

Notice in Figure 3.5 that all the strategies fail to realise the degree sequence when the power-law exponent is 2.0. Since some of these strategies are guaranteed to realise any realisable degree sequence, it can be concluded that equation 3.2 does not generate a realisable degree sequence when the power-law exponent is 2.0. All the strategies perform better as the power-law exponent is increased. This suggests that power-laws with low exponents place tighter constraints on how simple graphs can be constructed.

The results for the assortativity and clustering coefficients are shown respectively in Figures 3.6 and 3.7. It can be seen that each strategy produces a class of graphs with uniquely (although sometimes subtly) different topological properties. This is most apparent when comparing strategies across a range of power-law exponents. If one were to use only a single power-law exponent (2.6), then all of the “satisfy proportional” strategies would appear to give identical results.

All of the strategies generate graphs that are more disassortative (or more assortative) than those generated by the projected configuration model (compare Figure 3.3). The smallest-first/proportional strategy comes close, probably because the “proportional” part mimics how the configuration model picks vertices to connect (proportionally to the residual degree), and the “smallest-first” part mimics the effect of the projection (since it leads to a graph with missing edges).

Comparing with Figure 3.4, it can be seen that all the strategies produce graphs with much higher clustering coefficients than the projected configuration model. Thus, the conclusion is that the strategy used to construct random simple graphs has a significant effect on the topology of the graphs generated. Therefore, when uniformly random graphs are required, it is essential to confirm that the intended procedure for generating those graphs really does meet the uniform requirement.

3.2.4 Preferential attachment

The Barabási-Albert (BA) preferential attachment model (Barabási and Albert, 1999) was introduced to provide a plausible explanation for how power-law distributions arise in complex networks. As the first successful model of its kind, it has been enormously

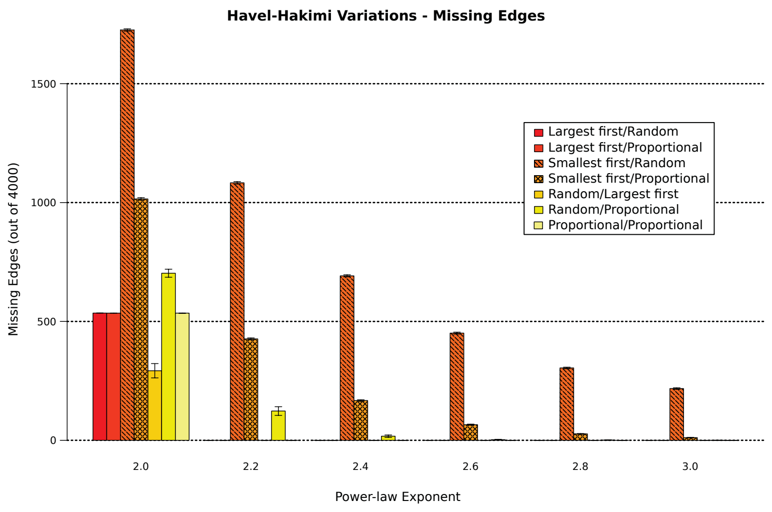

Figure 3.5: Number of missing edges in Havel-Hakimi random graphs with a power-law degree sequence. Each bar is the mean of 100 graphs, with a 95% confidence interval. Each graph has 1000 vertices and is intended to have 4000 edges.

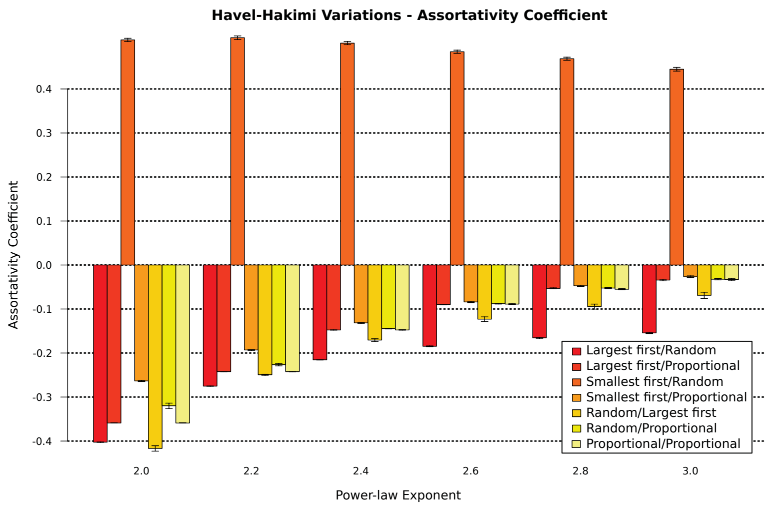

Figure 3.6: Assortativity coefficients of Havel-Hakimi random graphs with a power-law degree sequence. Each bar is the mean of 100 graphs, with a 95% confidence interval. Each graph has 1000 vertices, and a mean degree of approximately 8.

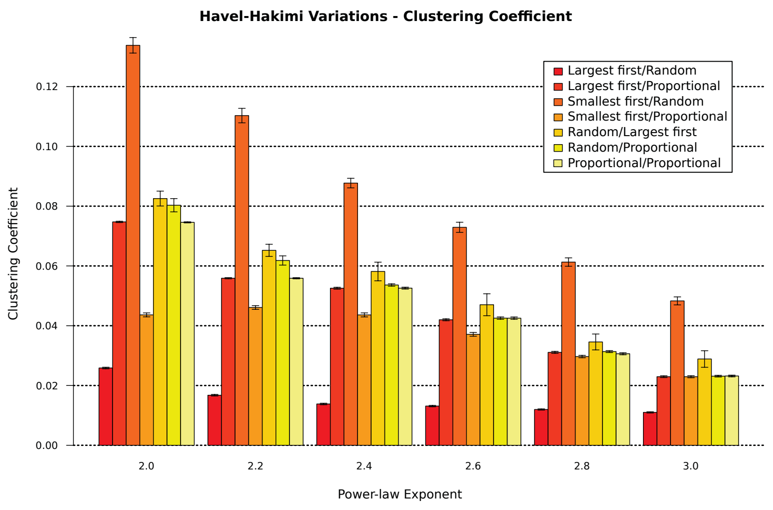

Figure 3.7: Clustering coefficients of Havel-Hakimi random graphs with a power-law degree sequence. Each bar is the mean of 100 graphs, with a 95% confidence interval. Each graph has 1000 vertices, and a mean degree of approximately 8.

influential in the field of complex networks.

The core idea of the BA model is to model the growth of the network over time. The growth of a BA network is governed by simple rules, and the power-law degree distribution arises as an emergent property. This is different to more traditional random graph models, such as the configuration model, which only consider the topology of the network at one fixed point in time.

The BA process proceeds in discrete timesteps. At each timestep, a new vertex v is added to the graph. The new vertex v is connected to m other vertices already in the graph. The vertices that v connects to are chosen at random proportional to their degree, according to the equation $\Pi(i) = d(i)/\sum_{j \neq i} d(j)$ where $\Pi(i)$ is the probability that v connects to i and $d(i)$ is the degree of i. Hence, the new vertices are “preferentially” attached to existing vertices of high degree. A core of $m_0 \geq m$ vertices is required to start the process, and it matters how this core is chosen (Bollobás and Riordan, 2003).

The degree distribution of a large BA network is a power-law distribution with exponent of 3 (Barabási and Albert, 1999). However, there are numerous extensions to the model which can account for a range of exponents, such as that of Dorogovtsev and Mendes (2000) for example; several variations are surveyed and analysed by Bollobás and Riordan (2003). The density is determined by the parameter m. Since m edges are added for each vertex, the mean degree approaches $2m$ for large numbers of vertices.

The BA model is problematic as a null model in empirical studies, although it is frequently misused for this purpose. As explained in Section 3.1, a suitable null model must generate every graph with the desired degree distribution uniformly at random. It is easy to see that the BA model does not meet this criteria by considering the degeneracy of the generated graphs (see Section 2.2.5 for a definition of degeneracy). The degeneracy of the generated part of a BA graph (excluding the initial core), is exactly m. Since a range of values of degeneracy are possible with any fixed power-law degree distribution, the BA model cannot generate the full range of possible topologies. Even within the power-law graphs with exponent 3 and degeneracy m, there is no reason to suppose that every graph is generated uniformly at random, and this was never the intended purpose of the model.

In order to demonstrate the difficulties with using the BA model as a null model, a series of experiments were performed. A range of values of m from 3 to 9 in increments of 1 were used, since m is the only parameter of the model that can be adjusted. The value of $m = 4$ corresponds to a mean degree of 8, which is comparable to the rest of the graphs in this chapter. For each value of m, a sample of 100 graphs were generated. As with the rest of the experiments in this chapter, each graph has 1000 vertices. For the seed graphs (from which BA networks grow), cliques of m vertices were used.

Measurements of the power-law fit are presented in Figure 3.8. These were computed using the method of Clauset et al. (2009), as discussed in Section 2.2.1. Notice that the power-law exponents are consistently lower than the expected value of 3, and the p values are high, but not 1. This is partly because the networks are too small for the power-law degree distribution to fully emerge. It also highlights another problem with the BA model: it cannot be used to precisely match a target degree distribution because the degree distribution is an emergent property from a stochastic process.

The results for assortativity and clustering are shown in Figures 3.9 and 3.10 respectively. The assortativity coefficient (for $m = 4$) is much lower than for the other models tested, although the clustering coefficient is approximately comparable to the Havel-Hakimi variations with similar power-law exponents.

3.3 Randomizing models

The models discussed in this section begin with a graph that already has a desired property. For 0K random graphs, that will be a graph with a particular mean-degree; for 1K random graphs, a graph with a particular degree sequence. The edges of the starting graph are then *rewired* in such a way that the mean degree (for 0K random graphs), or the degree-sequence (for 1K random graphs) is preserved, but the graphs are uniformly random in all other respects.

The 0K random rewiring algorithm presented in Section 3.3.1 is almost trivially simple (although care must be taken in the implementation, as will be seen in Section 3.3.1). The 1K random rewiring algorithm presented in Section 3.3.2 is based on

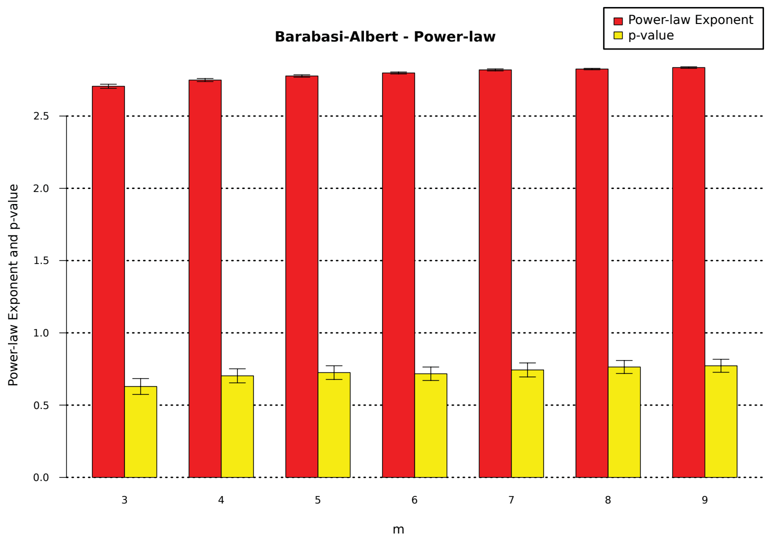

Figure 3.8: Power-law exponents and p-values of degree sequences in the Barabási-Albert preferential attachment model, for a range of values of m. Each bar represents the average over 100 graphs, with 95% confidence intervals. Each graph has 1000 vertices. p-values range from 0 to 1 and anything over 0.10 is considered consistent with the power-law hypothesis.

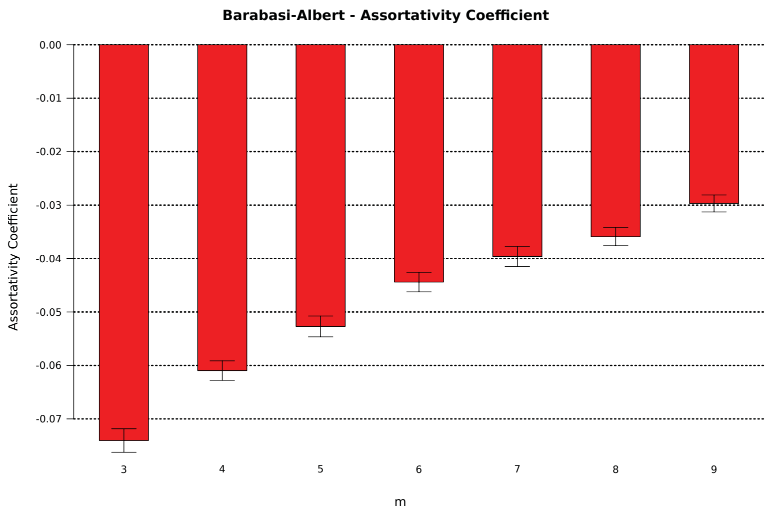

Figure 3.9: Assortativity coefficients of graphs generated by the Barabási-Albert preferential attachment process, for a range of values of m. Each bar represents the average over 100 graphs, with 95% confidence intervals. Each graph has 1000 vertices. The assortativity coefficient ranges from -1 to 1.

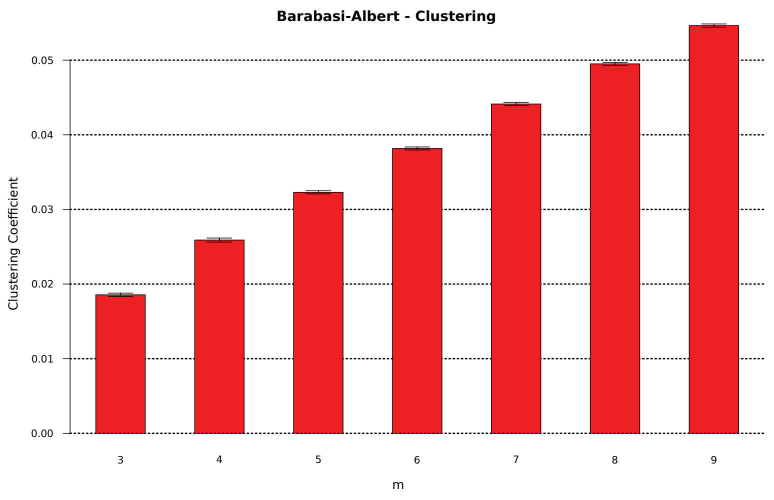

Figure 3.10: Clustering coefficients of graphs generated by the Barabási-Albert preferential attachment process, for a range of values of m. Each bar represents the average over 100 graphs, with 95% confidence intervals. Each graph has 1000 vertices. The clustering coefficient ranges from 0 to 1.

a *degree-preserving* rewiring operation. The correctness of the 1K rewiring algorithm depends on a theorem, for which a proof is presented in Chapter 4. Both algorithms are known to produce uniform samples of 0K and 1K graphs respectively.

The 'targeted rewiring' model presented in Section 3.3.3 is included because it can be used to generate graphs with properties that would otherwise be difficult to achieve. Such graphs are used in Section 3.3.2 in order to evaluate how quickly the 1K random rewiring procedure converges on a uniform distribution. Targeted rewiring can also be used to produce random graphs with power-law degree sequences, but there is no evidence that those graphs are uniformly random.

As with the generative models in Section 3.2, the assortativity and clustering coefficients were computed on graphs sampled from the 0K and 1K randomization models. The results can be directly compared with the results from Section 3.2, leading to the conclusion that none of the generative models achieve uniform sampling.

3.3.1 0K randomization

This model begins with a graph with n vertices and m edges. A sequence of graphs are generated by applying the following operation: one of the m edges e is selected uniformly at random, then two distinct endpoints v_1 and v_2 are chosen randomly from the n vertices. If there is no edge v_1v_2, then e is removed and replaced with v_1v_2. Otherwise the procedure is repeated until a feasible rewiring operation is found.

Once enough graphs have been generated, they constitute a uniformly random sample from the space of all simple graphs with the same n and m as the original graph.

This is easy to see by analogy with the Erdős-Rényi model. Typically, a large number of graphs must be generated before the sampling converges on uniformity. A smaller sample can be obtained by sampling from the larger sample.

It is important that *edges* be chosen at random. An implementer might be tempted to pick “random” edges by choosing a vertex at random, then picking a neighbour at random. This is how rewiring is performed in many small-world models (Watts and Strogatz (1998) for example), where it is acceptable only because there is no requirement to converge on uniformly random graphs.

To show that this is not valid, consider a wheel graph as a counter-example. One vertex is designed the hub, the others form the perimeter. The perimeter vertices are connected in a cycle, and all the perimeter vertices are connected to the hub. Half of the edges are on the perimeter, and the other half are spokes (connecting the hub to the perimeter). In a large wheel graph, if one were to pick a vertex at random, it would almost certainly be a perimeter vertex. If one were to then pick a neighbour at random, there would be a $\frac{2}{3}$ chance of a perimeter edge (since there are two neighbours on the perimeter), and a $\frac{1}{3}$ chance of a spoke. So this is not a valid way to pick edges uniformly at random.

Another important concern with random rewiring is: how many rewiring operations are necessary to compute a uniformly random sample. Clearly, the maximum number of rewiring operations required to transform one graph into any other (with the same n and m), is m. Thus, at least that many rewiring operations are required. The author is not aware of any theoretical or experimental results for the 0K rewiring model, so, it is assumed by analogy with the 1K model that $O(m)$ rewiring operations are sufficient, and the constant is 10 to 30.

This thesis uses the random rewiring model instead of the Erdős-Rényi model, purely as a matter of implementation convenience. In retrospect, the Erdős-Rényi model would have been a better choice as it generates a single random graph faster, and that graph is guaranteed to be uniformly random, whereas the rewiring model requires a large number of rewiring operations to converge on uniformity.

3.3.2 Degree preserving rewiring

The *degree-preserving rewiring* operation preserves the degrees of all vertices, and hence preserves the degree sequence of the original graph. Two distinct edges are chosen, and for each edge, one of the two endpoints is chosen; the endpoints of the edges are then swapped. Although there are four combinations of endpoints, in a simple graph there are only two distinct ways to rewire a pair of edges. Figure 3.11 demonstrates both possible rewirings.

Any graph can be transformed into any other (with the same degree sequence), by

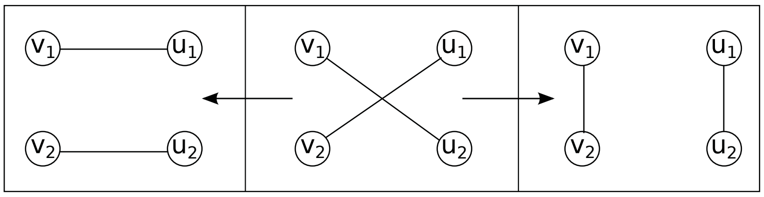

Figure 3.11: The degree-preserving rewiring operation. The centre figure represents the original edges, the other two figures represent the two possible rewirings.

applying enough degree preserving rewiring operations. The first proof of this theorem was given by Taylor (1980). An alternative proof is given in Chapter 4.

The 1K rewiring model introduced by Gkantsidis et al. (2003) operates similarly to the 0K rewiring model. A sequence of graphs are generated by applying rewiring operations. First, two edges are chosen independently and uniformly at random, then for each edge an endpoint is chosen randomly. If it is possible to swap the two edges without violating the simple graph constraint, then the next graph is generated by executing the rewiring operation. Otherwise the next graph is the same as the previous graph, this is called a *hold* operation. Once the sequence of graphs is long enough, it converges on a uniform sample from the space of all graphs with the same degree sequence as the original graph.

The hold operation is essential to achieving uniform sampling (Artzy-Randrup and Stone, 2005). This is because the number of graphs that can be reached by a single degree-preserving rewiring operation depends on the graph in question, and thus some graphs are more likely than others to show up during a sequence of rewirings. Artzy-Randrup and Stone suggest that the sequence of graphs converges very slowly to uniform sampling when using the hold operation, and they propose a more complicated algorithm. However, this is based on an assumption that most attempted rewiring operations fail, which is not the case on realistic undirected complex networks.

As with 0K randomization, an important concern is how many rewiring operations are necessary that the final graph is independent of it's starting point. As proved in Chapter 4, up to $m-1$ rewiring operations may be required to transform the initial graph into any other graph with the same degree-sequence. To sample from the entire space uniformly, at least this many rewiring operations will be necessary. Experimental and theoretical evidence suggests that $O(m)$ rewiring operations are sufficient (Ray et al., 2012; Gkantsidis et al., 2003) to sample uniformly from the entire space of graphs with the desired degree distribution, and that the constant is typically 10 to 30 (Ray et al., 2012).

In order to further validate the results of Gkantsidis et al. and Ray et al., two experiments were performed. Each experiment begins with 200 power-law graphs generated by the Havel-Hakimi algorithm described in Section 3.2.3. These graphs are arbitrarily divided into two groups of 100. One group is rewired to have a high assortativity or clustering coefficient, using the algorithm described in Section 3.3.3. The other group is rewired to have a low assortativity or clustering coefficient. These graphs are used as starting points for 1K rewiring.

From each starting point, a single graph is produced by performing $10m$, $100m$, and $1000m$ rewiring operations (including holds). Thus, for each group of 100 graphs, there is a sample of $10m$ rewired graphs, $100m$ rewired graphs, and $1000m$ rewired graphs. The assortativity coefficients and clustering coefficients are computed on each sample and compared to the values in the original graphs. This experimental design allows one to see how many rewiring operations are required for two topologically distinct samples to converge.

The results for the assortativity coefficient are shown in Figure 3.12, and for the clustering coefficient in Figure 3.13. The values of the coefficients become indistinguishable after only $10m$ rewiring operations, and do not change thereafter. The conclusion then is that $10m$ rewiring operations are sufficient to generate a uniform sample. This is consistent with the results of Gkantsidis et al. (2003) and Ray et al. (2012).

These experiments differ in their methodology from those presented in Gkantsidis et al. (2003) and Ray et al. (2012), which use mostly or exclusively distance-based metrics, which are poor at capturing topology as discussed in Section 3.1.1. The experiments of Gkantsidis et al. and Ray et al. generate their samples from a single staring graph, and compute the metrics at regular intervals as the samples are generated. Both sets of experiments compare the running means of the metrics as they are computed, and when the means no longer change, it is deemed that enough operations have been performed.

This methodology has the potential problem that the generated sample may appear uniform, when in fact it only represents a relatively small neighbourhood of graphs clustered around the starting graph. By using multiple starting points, the likelihood of this scenario is lessened.

Since it is known that the 1K rewiring process generates uniform samples of random graphs, it is now possible to empirically estimate the mean assortativity and clustering coefficients. The experiment is similar to those used in Section 3.2. A sample of 100 random graphs is generated for a range of power-law exponents from 2 to 3 in increments of 0.2, with each graph having 1000 vertices and 4000 edges. The starting graphs are generated by the Havel-Hakimi algorithm from Section 3.2.3, using the largest-first/random strategy. This is repeated for $10m$, $100m$, and $1000m$ rewiring operations, to ensure that the convergence is not affected by the power-law exponent.

The means and confidence intervals for the assortativity and clustering coefficients are presented in Figures 3.14 and 3.15 respectively. Comparing with the results from

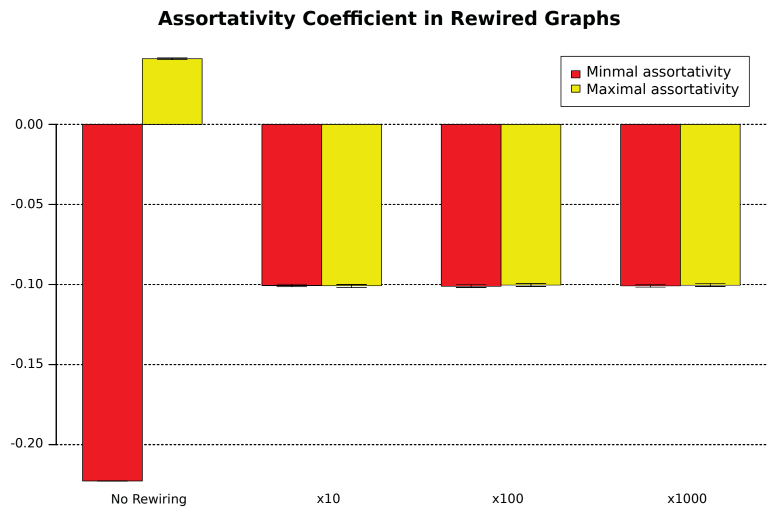

Figure 3.12: Assortativity coefficients of random graphs, following 1K randomization. The starting graphs are generated by the Havel-Hakimi algorithm using the largest-first/random strategy, with a power-law exponent of 2.6. They are then randomly rewired to have either maximal or minimal assortativity (see Section 3.3.3). Each bar represents the mean of 100 graphs, with 95% confidence intervals. Each graph has 1000 vertices and 4000 edges.

Section 3.2 yields strong evidence that none of the generative models discussed in that section generate uniform samples of random graphs.

By comparing Figures 3.14 and 3.15 with those for the configuration model (Figures 3.3 and 3.4), it can be seen that the simple graph constraint has a large impact on the topology of the graphs. The mechanisms behind the negative assortativity and low clustering were explained in Section 3.2.2. Of all the generative models, the projected configuration model is qualitatively the most similar to a 1K random graph model, but there are quantitative differences.

3.3.3 Targeted rewiring

The randomization strategies discussed so far start with a graph, then apply a number of rewiring operations that preserve some desired topological property while randomizing the graph with respect to all other properties. Targeted rewiring operates in the other direction. The rewiring operations are selected in a biased fashion so as to eventually create some structure that was not present in the original graph.

Figure 3.16 shows the results of an attempt to use a targeted rewiring procedure to generate power-law graphs. The initial graphs are Erdős-Rényi random graphs with 1000 vertices and 4000 edges. The procedure for rewiring the initial graphs to have power-law degree sequences is as follows:

A degree sequence s of length 1000 is constructed using equation 3.2 so that $\sum_i s_i =$

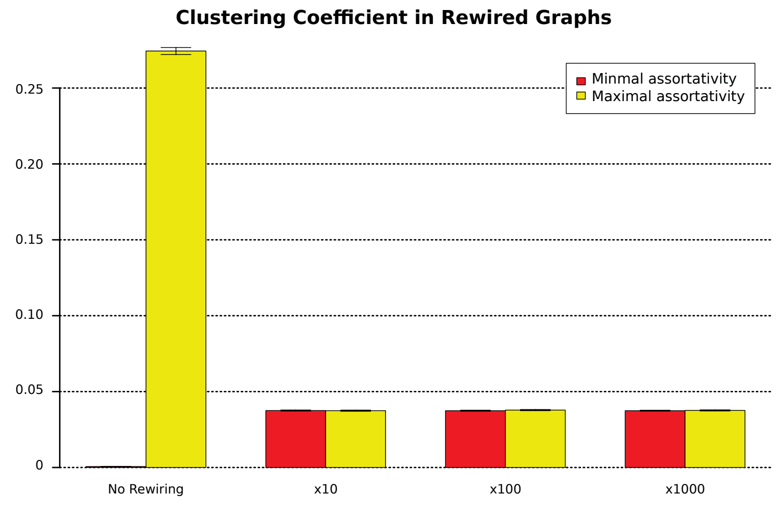

Figure 3.13: Clustering coefficients of random graphs, following 1K randomization. The starting graphs are generated by the Havel-Hakimi algorithm using the largest-first/random strategy, with a power-law exponent of 2.6. They are then randomly rewired to have either maximal or minimal assortativity (see Section 3.3.3). Each bar represents the mean of 100 graphs, with 95% confidence intervals. Each graph has 1000 vertices and 4000 edges.

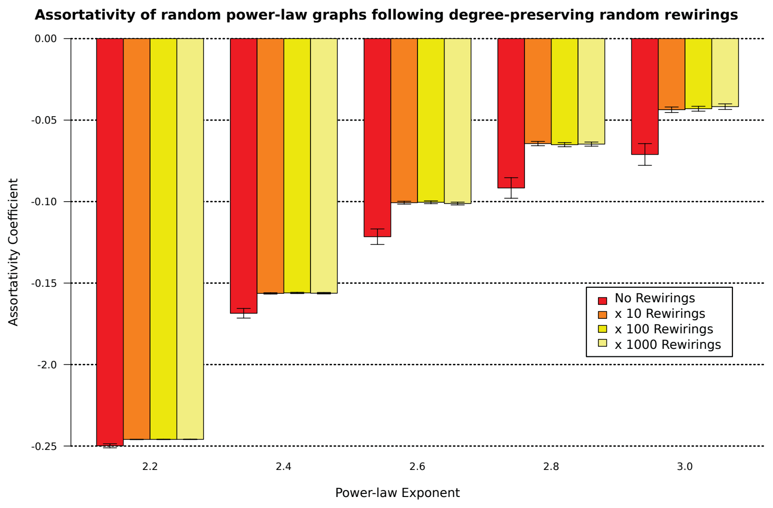

Figure 3.14: Assortativity coefficients of random graphs, following 1K randomization The starting graphs are generated by the Havel-Hakimi algorithm using the largest-first/random strategy. Each bar represents the mean of 100 graphs, with 95% confidence intervals. Each graph has 1000 vertices and 4000 edges.

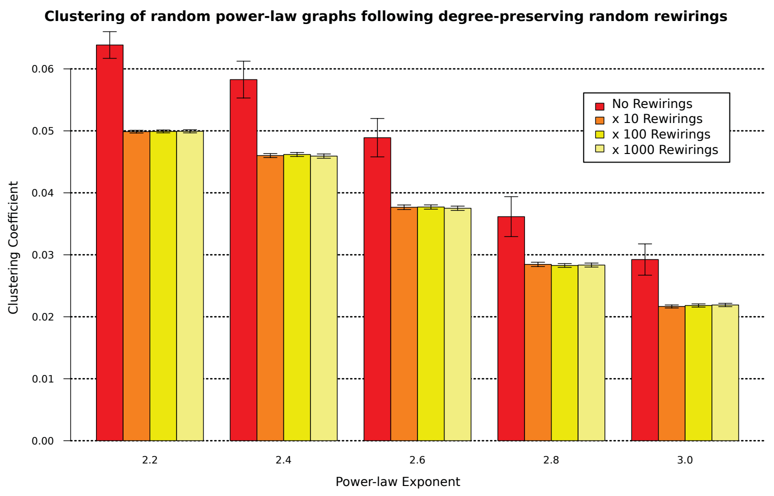

Figure 3.15: Clustering coefficients of random graphs, following 1K randomization. The starting graphs are generated by the Havel-Hakimi algorithm using the largest-first/random strategy. Each bar represents the mean of 100 graphs, with 95% confidence intervals. Each graph has 1000 vertices and 4000 edges.

$\sum_i d_i$ where d_i is the degree of vertex i in any one of the initial graphs. The rewiring operation is as follows: choose an edge e with endpoints $\{i_1, i_2\}$ at random, and a vertex j also at random. If $d_{i_1} \geq s_{i_1}$ then rewire $\{i_1, i_2\} \to \{j, i_2\}$. If $d_{i_2} \geq s_{i_2}$ then rewire $\{i_1, i_2\} \to \{i_1, j\}$. Otherwise, do no rewiring. This procedure is discussed in Lewis (2009), where it is shown to converge rapidly on the desired degree sequence. The central idea is to lower the degrees of vertices where the degree is too high. Since the total degree never changes, this must also increase the degrees of some other vertices, which will become the hubs.

The procedure was used to create graphs with a range of power-law exponents from 2.0 to 3.0, 100 graphs were generated for each power-law exponent. The power-law exponent and p-value were measure as explained in Section 2.2.1. The p-value indicates the probability that the data are indeed drawn from a power-law distribution, and any value greater than 0.10 is considered compatible with the power-law hypothesis. Figure 3.16 reports the mean values of the power-law exponents and p-values for each attempted power-law exponent, with 95% confidence intervals.

As can be seen in Figure 3.16, the targeted rewiring procedure generates graphs that match the intended power-law degree sequence closely, but not exactly. The procedure also appears to work better for smaller power-law exponents with the exception of 2.0, which, as discussed previously in Section 3.2.3, is not actually a realisable degree sequence.

Since the degree-preserving rewiring scheme described in Section 3.3.2 is known to converge on a uniform distribution, and there are no such results for the targeted

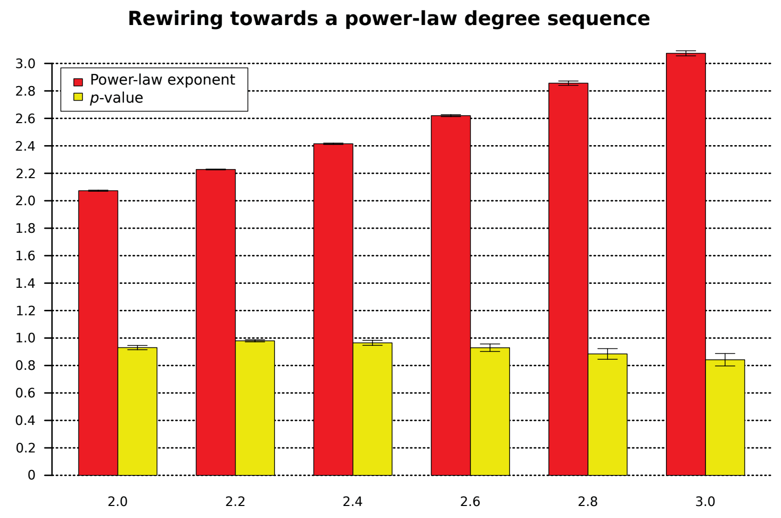

Figure 3.16: Power-law exponents and p-values of Erdős-Rényi random graphs that have been rewired using the targeted rewiring procedure, in an attempt to create power-law degree distributions in those networks. The p-value ranges from 0 to 1, and any value above 0.1 is considered consistent with a power-law degree distribution.

rewiring procedure, targeted rewiring was not investigated further as a method of generating random power-law graphs.

A similar targeted rewiring procedure was used to generate graphs with maximal and minimal assortativity as starting points for some of the experiments in Section 3.3.2. To generate those graphs, random degree-preserving rewirings were performed in batches of 10. After every 10 rewiring operations, the assortativity of the entire network was measured. Then, if it had moved away from the intended assortativity, the entire batch of rewiring operations was rejected and rolled-back. It is easy to see that this procedure will find a graph with maximal or minimal assortativity, while maintaining the degree sequence of the original graph.

3.4 Summary

A methodology to be used by the experiments presented in later chapters of this thesis was described in Section 3.1. This methodology requires the generation of 0K and 1K simple random graphs. 0K simple random graphs are drawn uniformly at random from all the simple graphs that have a particular size and mean degree. 1K simple random graphs are drawn uniformly at random from all the simple graphs that have a particular degree-sequence. 0K random graphs are easy to generate using a variety of well-known techniques. The generation of 1K simple random graphs is much harder.

In Section 3.2, a comparison was made between several random graphs models, popular in the literature, that may be able to generate 1K random graphs. It was found that the topological properties of the 1K graphs differed qualitatively between the different random graph models. This reinforces one of the main ideas of this chapter: that is is essential that random graphs be generated in a truly uniform manner.

The rewiring models reviewed in Section 3.3.2 are known to be capable of generating 1K random graphs, and they are more convenient from the point of view of the methodology of Section 3.1. For these reasons, the random rewiring models are used throughout the remainder of this thesis whenever random graphs are required.

By comparing the topologies of the randomized graphs (Section 3.3.2) to the generated graphs (Section 3.2), it can be seen that none of the generative models achieve uniform sampling, although the projected configuration model (Section 3.2.2) comes the closest. However, this does not imply that it is impossible to directly generate uniformly random graphs.

There is an algorithm due to Bayati et al. (2007) which could be adapted for power-law degree distributions, and that approximates uniform sampling to an arbitrary factor. The Bayati et al. algorithm was not considered for this thesis as it presents several practical difficulties. It is slow for complex networks (the running time is $O(nd_{max})$ where d_{max} is the maximum degree, which is $O(n)$ for complex networks). The algorithm is also difficult to implement and is not widely used in the complex networks literature.

The correctness of the 1K rewiring model (Section 3.3.2) relies on a theorem which is presented in the following Chapter.

Chapter 4

Proofs of rewiring theorems

Section 3.3.2 discussed an algorithm for uniformly sampling from the space of all simple graphs that have a particular degree sequence. The algorithm begins with a graph that is known to have the desired degree sequence, then a series of random degree-preserving rewiring operations are applied to the starting graph. Each rewiring operation creates a new graph for the sample, and when the sample becomes large enough it becomes approximately uniform.

The degree-preserving rewiring operation is illustrated in Figure 3.11. Essentially, two distinct and non-adjacent edges are selected, and for each edge one of the two endpoints is selected, then the endpoints are swapped. In this chapter, the degree-preserving rewiring operation is usually referred to as an *edge swap* operation, for the sake of brevity.

In order to see that the rewiring approach samples from the *entire* space of graphs with the desired degree sequence, it is necessary to show that any two graphs in that space can be linked by a chain of rewiring operations. More formally, the rewiring approach relies on the following theorem:

Theorem 1. Pick two arbitrary simple graphs G_1 and G_N with the same degree sequence. Then there is a sequence of edge swap operations that transforms G_1 into G_N such that every edge swap preserves the simple graph constraint.

This theorem was first proved by Taylor (1980). Taylor's paper also includes a proof for the following related theorem, which is important if one wants to take a sample with the *additional* constraint that all the graphs are connected:

Theorem 2. Pick two arbitrary connected simple graphs G_1 and G_N with the same degree sequence. Then there is a sequence of edge swap operations that transforms G_1 into G_N such that every edge swap preserves the connectedness and simple graph constraints.

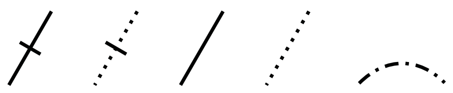

Figure 4.1: From left to right: An element of $R(G_1)$ (or $R(C)$ for some alternating cycle C); an element of $I(G_1)$ (or $I(C)$ for some alternating cycle C); an edge in G_1 but not in $R(G_1)$; an edge not in G_1 or $I(G_1)$; an alternating path.

Taylor's proof of Theorem 1 operates by induction on the degree sequence. He shows that G_1 and G_N can be both be rewired to a common target graph. To do this he shows how G_1 and G_N can be transformed into G'_1 and G'_N such that the highest degree vertex v_1 has the same neighbourhood in both G'_1 and G'_N. Then v_1 can be removed from G'_1 and G'_N, and by induction there is a sequence of edge swaps between $G'_1 \setminus \{v_1\}$ and $G'_N \setminus \{v_1\}$.

The proof of Theorem 2 is more complicated. Essentially, Taylor shows that if an edge that needs to be rewired would cut the graph into subgraphs A and B, then there must be a cycle in one of those subgraphs. An edge on the cycle can then be swapped with another edge to cross the boundary between A and B. This does not disconnect the graph because the 'borrowed' edge lies on a cycle.

This chapter presents an independent proof of both theorems. The proof operates by directly constructing the required sequences of edge swap operations to transform G_1 to G_N without violating the relevant constraints.

Since the required sequences of edge swaps are directly constructed, it is possible to place an upper bound on the number of edge swaps required to transform between two graphs. For simple graphs the bound is $|E| - 1$, where E is the edge set of G_1. In the connected case, the bound is $2|E| - 1$ since it may be necessary to use extra edge swap operations to rewire some edges.

4.1 Preliminaries

The goal is to find a sequence of edge swap operations that transforms an arbitrary simple graph $G_1 = (V, E_1)$ into another arbitrary simple graph $G_N = (V, E_N)$ so that the degree sequence of G_1 matches the degree sequence of G_N, and no edge swap ever violates the simple graph constraint. Every edge swap creates a new graph, so there is a sequence of graphs $G_1, \ldots, G_i, \ldots G_N$ where each can be transformed to the next by a single edge swap.

To transform G_i to G_N, some edges must be removed, and some added. Let $R(G_i) \subseteq E_i$ be the set of edges to remove, and $I(G_i) \subseteq \bar{E}_i$ be the set of edges to add (where $\bar{E}_i$ is the set of edges in the complement graph of G_i). Since each edge swap creates a new graph, these two sets must be recomputed following every edge swap. Note also that

$|R(G_i)| = |I(G_i)|$ since every graph in the sequence has the same degree-sequence and therefore the same number of edges. Figure 4.1 shows the graphical notation used to illustrate the proof.

An *alternating path* is defined as a vertex-labelled path $(v_1, v_2, \cdots, v_n)$ where the edges alternate between elements of E_i and $\bar{E}_i$. All of the edges in an alternating path are required to be unique, but the path may cross over itself, i.e. the same vertex may appear multiple times in an alternating path.

If the alternating path begins and ends at the same vertex then it is an *alternating cycle*. It will be required to have even length, so that for every pair of vertices in the cycle, there are two disjoint alternating paths between them. If $E(C)$ is the set of edges along an alternating cycle C in G_i, then $R(C) = E(C) \cap E_i$, and $I(C) = E(C) \cap \bar{E}_i$. It is important to note that an alternating cycle according to this definition does not represent an actual cycle in the underlying graph.

For the connected case (Section 4.3), $K \subseteq E(G_i)$ is a *cutset* if $G_i \setminus K$ has more components than G_i. A *bridge* is a single-edge cutset. If K is a cutset, but no proper subset of K is a cutset, then K is a *minimum-cut*.

4.1.1 An example

As an example of how the required sequence of edge swap operations might be constructed in practice, consider Figure 4.2. The top line shows two non-isomorphic graphs, G_1 and G_N with the same degree sequence (3, 2, 2, 1, 1, 1). The graphs have been labelled such that no vertex changes degree between the two graphs.

From the labelling, two edge sets are computed: $I(G_1)$ (the set of edges that must be added to G_1 to get G_N) and $R(G_1)$ (the set of edges that must be removed from G_1 to get G_N). The graphical notation from Figure 4.1 is used to show the sets $I(G_1)$ and $R(G_1)$ in the second row of Figure 4.2.

Notice that the sets $I(G_1)$ and $R(G_1)$ in Figure 4.2 are not minimal. If a different labelling had been chosen, then smaller sets would have been possible. The point is that any labelling which preserves vertex degrees will suffice, and so finding $I(G_1)$ and $R(G_1)$ is not an obstacle to implementing an algorithm that computes the sequence of edge swaps.

Following the construction of $I(G_1)$ and $R(G_1)$, the next requirement is to find the alternating cycles. A general method of finding the alternating cycles is given in Section 4.2. For the graph in Figure 4.2, it is possible to find the following alternating cycle: $(v_1, v_2, v_4, v_5, v_2, v_6, v_4, v_3, v_1)$.

An alternating cycle is rewired by reducing it to a smaller alternating cycle, which can be rewired recursively. The required reductions are explained in Section 4.2.2.

Returning to the example graph, the original alternating cycle can be split into two

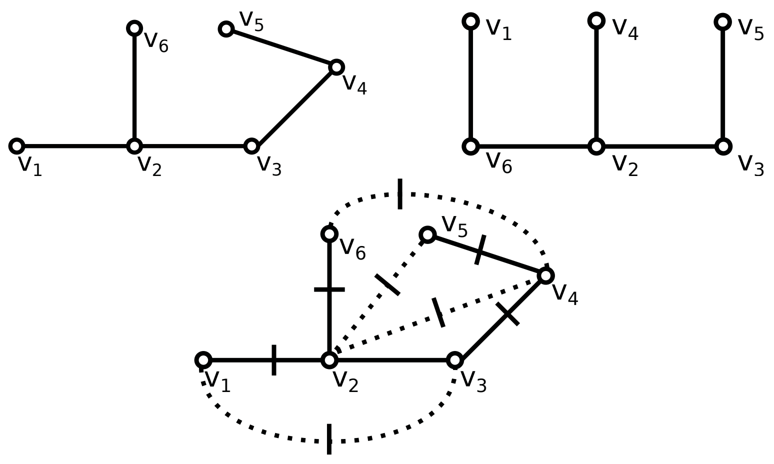

Figure 4.2: An example of how the sequence of edge swap operations is constructed in practice. *Top-left*: G_1. *Top-right*: A graph G_N with the same degree sequence as G_1, but not isomorphic to G_1. *Bottom*: Construction of the sets $I(G_1)$ (dotted) and $R(G_1)$ (solid), using the notation in Figure 4.1.

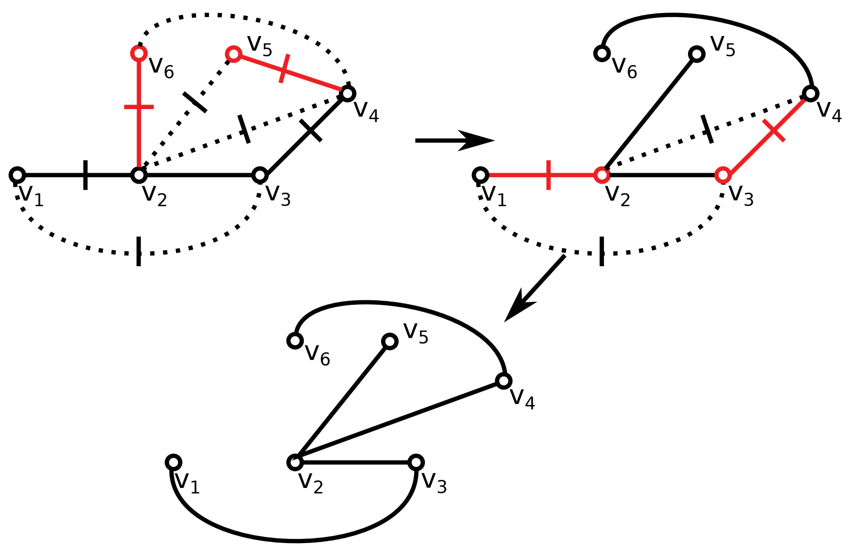

Figure 4.3: A sequence of three edge swap operations that rewires the graph G_1 (from Figure 4.2) to G_N (also from Figure 4.2). The edges and endpoints that are swapped are highlighted in red.

4-cycles $(v_2, v_6, v_4, v_5, v_2)$ and $(v_1, v_2, v_4, v_3, v_1)$ which are both trivial to rewire. The edge swap operations are as follows:

1. $\{e_{26}, e_{45}\} \rightarrow \{e_{25}, e_{46}\}$
2. $\{e_{12}, e_{43}\} \rightarrow \{e_{13}, e_{42}\}$

Thus, G_1 was rewired to G_N using two edge swap operations. The full sequence of edge swap operations is illustrated in Figure 4.3.

4.2 Proof for simple (unconnected) graphs

The essence of the proof is to choose an edge swap to apply to G_i such that $|R(G_{i+1})| < |R(G_i)|$ (and since $|R(G)| = |I(G)|$, $|I(G_{i+1})| < |I(G_i)|$). If such an edge swap can always be constructed, then G_1 can be rewired to G_N in a finite number of steps as required (since $R(G)$ is finite). The edge swap operations are found by selecting adjacent edges that lie on alternating cycles, as described in Section 4.2.2. The construction of the alternating cycles is described in the next section.

4.2.1 Alternating cycle construction

G_i and G_N have the same degree sequence, and so the vertices can be labelled such that no vertex changes degree when G_i is rewired. This implies that for every vertex v, $|N_v \cap R(G_i)| = |N_v \cap I(G_i)|$, where N_v is the set of edges incident on v. Now for every edge uv in $R(G_i) \cup I(G_i)$, an alternating path can be constructed as follows: The initial path is (u, v). Suppose $uv \in I(G_i)$, then there are edges $xu \in R(G_i)$ and $vy \in R(G_i)$, so the initial path can be extended to (x, u, v, y). If $uv \in R(G_i)$ then the path can be extended by a similar argument.

Since every alternating path can be extended in this way, and the set $I(G_i) \cup R(G_i)$ is finite, every alternating path will eventually loop back on itself and form an alternating cycle. The graph can be rewired by rewiring the alternating cycles individually.

4.2.2 Rewiring alternating cycles

An alternating cycle $C \in G_i$ can be rewired using the following recursive procedure. First, if the alternating cycle is of length four (the 4-cycle), then it can be rewired in one operation (Figure 4.4), reducing $R(C)$ by two.

Any k-cycle C where $k > 4$ can be *reduced* to a smaller cycle in the following way. Pick four distinct consecutive vertices (a, b, c, d). Let X be the rest of the cycle, so the full cycle is (a, b, c, d, X, a).

To see that it is always possible to choose four distinct consecutive vertices, note first that every three consecutive vertices (a, b, c) on an alternating cycle must be distinct. This is because, since (a, b, c) are part of an alternating cycle, there must be edges $ab \in I(C)$ and $bc \in R(C)$ (or $ab \in R(C)$ and $bc \in I(C)$), and $ab \neq bc$ since the sets $I(C)$ and $R(C)$ are disjoint. If four consecutive vertices (a, b, c, d) are selected, however, then it may be that $a = d$ as shown in Figure 4.5. In such cases, there is another vertex $e \notin \{a, b, c, d\}$ (also shown in Figure 4.5), and (b, c, d, e) form four distinct consecutive vertices as required.

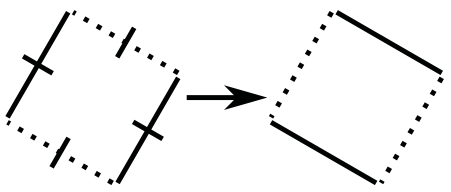

Figure 4.4: *Left*: The 4-cycle. *Right*: The 4-cycle following a single edge swap operation.

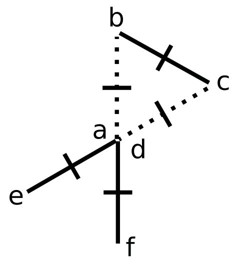

Figure 4.5: Four consecutive vertices (a, b, c, d) on an alternating cycle C where $a = d$. In this case, since C is an alternating cycle, there are two more vertices $e, f \notin \{a, b, c, d\}$ as shown in the figure. Now there are four distinct consecutive vertices (b, c, d, e) as required.

The remainder of this section assumes that $bc \in R(C)$. If $bc \in I(C)$ then the edge classes (I edges and R edges) are switched, but the cases are otherwise the same.

Now there are two cases (illustrated in Figure 4.6):

Case 1

There is an edge ad in G. Then there is a 4-cycle (a, b, c, d, a). After rewiring this 4-cycle, the original k-cycle becomes the $(k-2)$-cycle (a, d, X, a). Rewiring the 4-cycle reduces $R(C)$ by 1.

Case 2

There is no edge ad. Then there is a $(k-2)$-cycle (a, d, X, a). After recursively rewiring the $(k-2)$-cycle, there is a 4-cycle (a, b, c, d, a), which can be rewired trivially.

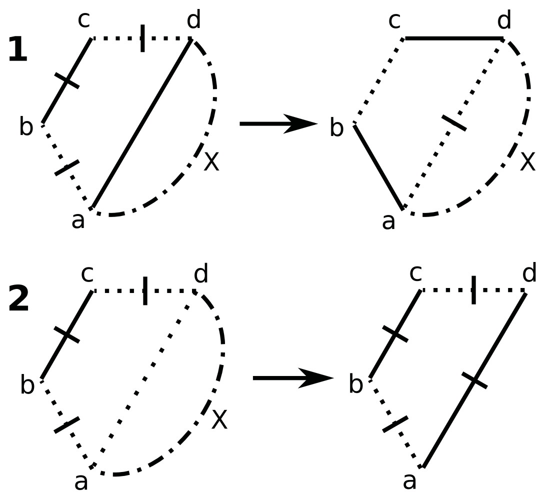

Figure 4.6:
Case 1: There is an edge ad, so rewire the 4-cycle (a, b, c, d, a) to get the reduced cycle (a, d, X, a)
Case 2: There is no edge ad, so recursively rewire the $(k-2)$-cycle (a, d, X, a) to get the 4-cycle (a, b, c, d, a)

Note that in both cases, an edge $e \notin E(C)$ is included in an edge swap operation. In Case 1, this edge becomes part of $I(C)$ for the reduced cycle; for Case 2 it becomes part of $R(C)$.

4.2.3 A special case

Case 2 from Section 4.2.2 attempts to rewire an alternating cycle (a, b, c, d, X, a) by identifying a smaller alternating cycle (a, d, X, a). However, it is possible that the edge $ad \in \bar{E}_i$ is included in the alternating path X, a case that requires special handling.

Assume that the edge ad occurs on every alternating path from a to d. If there were an alternating path that did not include ad, then the reduction could be carried out as usual using that path. If every alternating path from a to d includes ad, then $ad \in I(C)$ (in this example where $bc \in R(C)$) and there are four distinct vertices e, f, g, and h as shown in Figure 4.7. These vertices must be distinct, otherwise there would be an alternating path from a to d excluding ad. For the same reason, there cannot be any edge $ef \in I(C) \cup R(C)$.

Now, since $e \neq f$, there are two cases (illustrated in Figure 4.7):

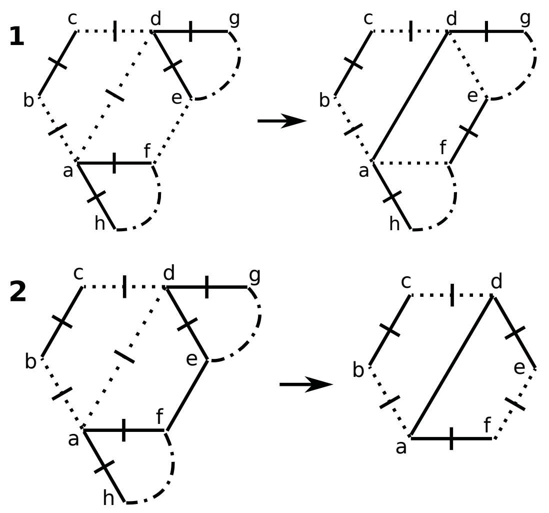

Figure 4.7: A k-cycle where the edge $ad \in I(C)$ occurs on every alternating path from a to d. This implies the situation illustrated where there are four distinct vertices e, f, g, and h.

Case 1: There is no edge ef so rewire the 4-cycle (a, d, e, f, a).
Case 2: There is an edge ef, so rewire the $(k-4)$-cycle $(a, d, g, \ldots, e, f, \ldots, h, a)$ recursively. This leaves the 6-cycle $a, b, c, d, e, f, a)$ to be rewired.

Case 1

There is no edge ef. Then there is a 4-cycle (a, d, e, f, a). After rewiring this 4-cycle, the original k-cycle becomes the $(k-2)$-cycle $(a, b, c, d, g, \ldots, e, f, \ldots, h, a)$. $R(C)$ is reduced by 1.

Case 2

There is an edge ef. Then there is a $(k-4)$-cycle $(a, d, g, \ldots, e, f, \ldots, h, a)$. After rewiring the $(k-4)$-cycle, there is a 6-cycle (a, b, c, d, e, f, a) which can be rewired using the reductions shown in Figure 4.6. Note that there is also a $(k-2)$-cycle $(a, b, c, d, g, \ldots, e, f, \ldots, h, a)$, but rewiring the $(k-4)$-cycle instead will be more convenient when the proof is extended to simple connected graphs in Section 4.3.

Every edge swap reduces $R(C)$ by 1 or 2 edges, and the last edge swap always reduces $R(C)$ by 2, so the alternating cycle is rewired using no more than $R(C) - 1$ edge swap operations.

This completes the proof of Theorem 1. The alternating cycle based proof can now be extended to the case of simple connected graphs, i.e. graphs with only one connected component. In this case, every edge swap operation is required to preserve the property that the graph is connected.

4.3 Extension to simple connected graphs

The reductions from Section 4.2 are of two kinds. Those that identify a 4-cycle to be rewired immediately, and those that identify a smaller alternating cycle to be rewired recursively. Case 1 from Section 4.2.2 and Case 1 from Section 4.2.3 are of the first kind. Case 2 from Section 4.2.2 and Case 2 from Section 4.2.3 are of the second kind.

The reductions that identify a smaller alternating cycle to be rewired recursively can be applied without disconnecting the graph. For the reductions that identify a 4-cycle to be rewired immediately, new cases are required since it is not necessarily possible to rewire a 4-cycle directly without disconnecting the graph. When it is necessary to rewire a 4-cycle C that disconnects the graph, an alternative 4-cycle C' will be constructed such that swapping the edges of $R(C')$ does not disconnect the graph.

C' is chosen so that $|R(G) \cup I(G)|$ is reduced by at least two when C' is rewired, so that the alternative edge swaps make the same progress towards G_N as the edge swaps from the simple case. However, there is a case where a sequence of four edge swaps may reduce $|R(G) \cup I(G)|$ by as little as 4 edges (Figure 4.11). This increases the upper-bound on the number of edge swap operations to $2|E| - 1$. (Since now the number of edge swaps required could be as high as $|R(G_1) \cup I(G_1)| - 1$ and $|R(G_1) \cup I(G_1)| \leq 2|E|$).

4.3.1 4-cycles

In the simple case (Section 4.2.2), it was always possible to rewire a 4-cycle with one edge swap. However, this edge swap may disconnect the graph as shown in Figure 4.8. In the case that C disconnects the graph, another cycle C' can be constructed such that there is at most one edge $e \in C' \setminus (R(G_i) \cup I(G_i))$. Thus, if C' is rewired, $|R(G_{i+1}) \cup I(G_{i+1})| \leq |R(G_i) \cup I(G_i)| - 2$, guaranteeing that every edge swap operation makes progress towards G_N. In the case that C' also cuts the graph, the C' construction can be applied iteratively to obtain a finite sequence of 4-cycles, where the last 4-cycle in the sequence does not cut the graph.

4.3.2 The C' construction

Assume that C, the 4-cycle in G_i to be rewired, has $R(C) \subseteq R(G_i)$ and there is at most one edge in $I(C) \setminus I(G_i)$. This assumption is assured by ordering the reductions as described in Section 4.3.3.

If $R(C)$ is a cutset in G_i, then there is an edge $xy \in I(G_i)$ that crosses the boundary created by the cutset, as shown in Figure 4.8. The edge xy is used to construct a new cycle, C', as shown in Figure 4.9. $R(C')$ and $I(C')$ follow the same assumptions made for C, so if $R(C')$ is also a cutset in G_i, then the C' construction can be applied again to construct a sequence of 4-cycles.

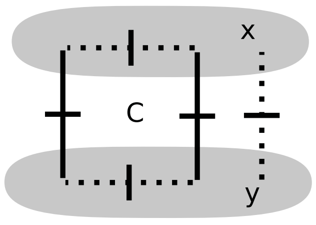

Figure 4.8: A 4-cycle in G_i that cuts the graph. If C were rewired using a single edge swap, then the graph would be separated into the two components indicated in grey. Since the target graph G_N is connected, there must be an edge $xy \in I(G_i)$. This edge is used to construct another 4-cycle that does not cut the graph (it is always possible to find such a cycle).

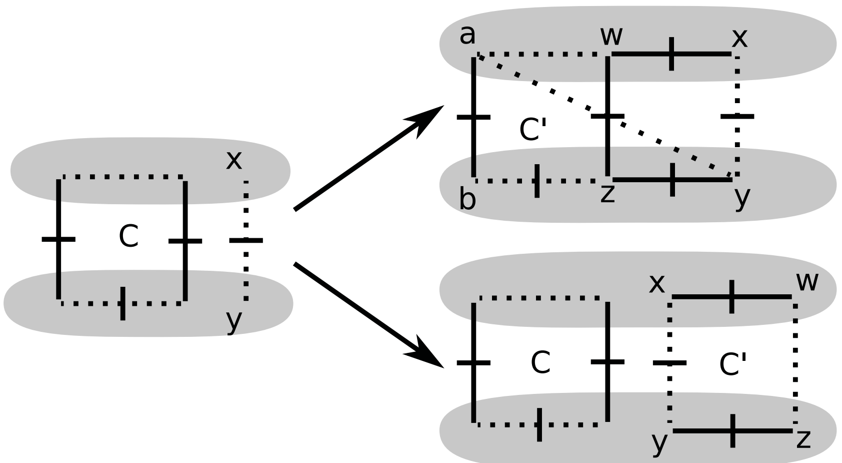

Figure 4.9: *Left*: C is a 4-cycle in G_i and $R(C)$ is a cutset, but G_N is connected so there must be an edge $xy \in I(G_i)$ that crosses the boundary created by $R(C)$. The alternating cycle construction from Section 4.2.1 finds the edges $xw \in R(G_i)$ and $yz \in R(G_i)$. In the case that $wz \in R(C)$ (*top*), there is a 4-cycle $C' = (a, b, z, y, a)$ which does not cut the graph (since $wz \notin R(C')$ crosses the same boundary). Otherwise there is no edge wz and $C' = (x, y, z, w, x)$ (*bottom*).

Suppose that neither C nor C' can be rewired without cutting the graph. It is now necessary to distinguish between a minimum-cut (neither edge in $R(C)$ is a bridge in G_i), and a *non-minimum-cut* where both edges are bridges in G_i. There are two cases:

Case 1

If C' is a minimum-cut, then so is C; the reason is shown in Figure 4.10. In this case, C and C' overlap as shown in Figure 4.11, and can be rewired by a sequence of four edge swaps. In the worst case, both C and C' may include one edge each from $\bar{E}_i \setminus I(G_i)$. Therefore, the four edge swap operations required to rewire C and C' may reduce $|R(G_i) \cup I(G_i)|$ by as little as 4 edges.

The bound of $|E|-1$ on the number of edge swaps for the simple case (Section 4.2) was based on the assumption that every edge swap reduces $|R(G_i) \cup I(G_i)|$ by at least 2. Hence, the upper bound for the connected case increases to $2|E|-1$ (it remains true that the last edge swap rewires two edges).

Case 2

If C' is not a minimum-cut, then repeat the C' construction to obtain a C''. This must always be possible when C' cuts the graph. Repeated application of the C' construction leads to a sequence of 4-cycles $C_1, \ldots, C_i, \ldots, C_n$.

A simple inductive argument illustrated in Figure 4.12 shows that every time the C' construction is applied, at least two new edges from the finite set $R(G_i) \cup I(G_i)$ are required, and therefore the sequence of 4-cycles is finite. Since the sequence is finite, it must be possible to rewire the last 4-cycle in the sequence without cutting the graph. Since every C' has at most one edge $e \notin R(G_i) \cup I(G_i)$, $|R(G_{i+1}) \cup I(G_{i+1})| \leq |R(G_i) \cup I(G_i)| - 2$, as required.

4.3.3 Ordering the reductions

The C' construction of Section 4.3.2 shows how to rewire a 4-cycle C in G_i that would cut the graph, but it assumes that all such cycles have $R(C) \subseteq R(G_i)$ and there is at most one edge in $I(C) \setminus I(G_i)$. These assumptions could be violated when edges from $E_i \setminus R(G_i)$ or $\bar{E}_i \setminus I(G_i)$ are included by the reductions in Sections 4.2.2 and 4.2.3. The solution is to order the reductions to avoid 4-cycles that do not satisfy the assumptions.

Assume that the cycle C includes at most one edge $xy \in E_i \setminus R(G_i) \cup \bar{E}_i \setminus I(G_i)$. This is certainly true prior to applying any reductions. If C is a 4-cycle, then it must be one of the two cases shown in Figure 4.13, and is handled as described in that figure.

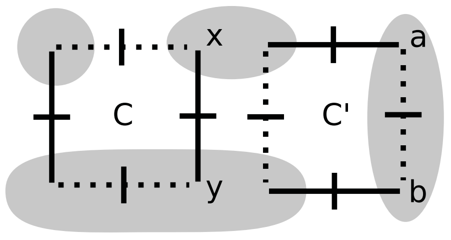

Figure 4.10: If C in G_i is not a minimum-cut, then C' cannot be a minimum-cut either. Suppose C' is a minimum-cut and C is not, then there must be a path from a to b in G_i. That implies that xy lies on a cycle in G_i, so C could have been rewired without cutting the graph, a contradiction.

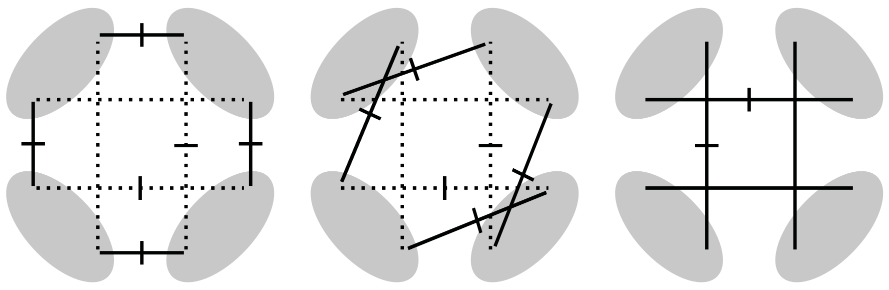

Figure 4.11: If C and C' are both minimum-cuts, then they overlap as shown. A sequence of four edge swaps rewires the two 4-cycles.

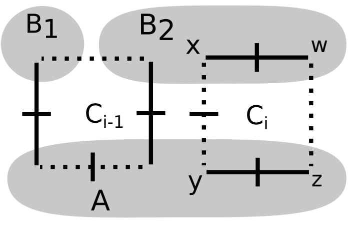

Figure 4.12: Two 4-cycles C_i and C_{i-1} in the sequence $C_1, \ldots, C_n$. Removing $R(C_{i-1})$ cuts the graph into the components A, B_1, and B_2 (or A and $B_1 \cup B_2$ if C_{i-1} is a minimum-cut). Assume by induction that all C_j for $j < i-1$ are in A (trivial for $i = 2$), then $xy \in I(G_1)$ and $xw \in R(G_1)$ do not occur in any $C_1, \ldots, C_{i-1}$

The reductions described in Sections 4.2.2 and 4.2.3 begin by choosing four distinct consecutive vertices (a, b, c, d) in the alternating cycle (a, b, c, d, X, a). If these vertices are chosen such that the (at most one) edge $xy \in E_i \setminus R(G_i) \cup \bar{E}_i \setminus I(G_i)$ is one of the edges ab, bc, or cd, then it is possible to apply the reduction without increasing the number of edges in the set $E_i \setminus R(G_i) \cup \bar{E}_i \setminus I(G_i)$.

The four distinct consecutive vertices (a, b, c, d) can be chosen so that $xy \in \{ab, bc, cd\}$ in the following way. First, let $b = x$ and $c = y$, then choose any a and d on the alternating cycle so that (a, b, c, d) are consecutive. Now if $a = d$, then there is another vertex $e \notin \{a, b, c, d\}$ as shown in Figure 4.5 and (b, c, d, e) are distinct and consecutive as required.

Now there are four cases: Cases 1 and 2 from Section 4.2.2, and Cases 1 and 2 from Section 4.2.3. Case 1 from Section 4.2.3 identifies a 4-cycle that always matches one of the two cases shown in Figure 4.13, so it can be handled as described in that figure.

Case 2 from Section 4.2.2 and Case 2 from Section 4.2.3 both identify a smaller alternating cycle (a, d, X, a). Since the smaller alternating cycle excludes the three edges ab, bc, and cd, it also excludes the edge xy. Thus, the reduced cycle still has at most one edge in the set $E_i \setminus R(G_i) \cup \bar{E}_i \setminus I(G_i)$.

Case 1 from Section 4.2.2 identifies a 4-cycle (a, b, c, d, a) to rewire. Since one of ab, bc, cd may be the edge xy, and since ad may also belong to the set $E_i \setminus R(G_i) \cup \bar{E}_i \setminus I(G_i)$, up to two edges in the 4-cycle may belong to the set $E_i \setminus R(G_i) \cup \bar{E}_i \setminus I(G_i)$. If the 4-cycle includes at least one edge from each of $R(G_i)$ and $I(G_i)$, then it is handled as shown in Figure 4.13 Case 2. Otherwise it is one of the four cases shown in Figure 4.14 and is handled as shown in that figure.

4.4 Bounds

For the simple graph case (Section 4.2), every edge swap operation reduces the length of an alternating cycle by at least 2 edges. The total length of all the alternating cycles in the initial graph is $|R(G_1) \cup I(G_1)|$, which is bounded above by $2|E|$ (since $|R(G_1)| \leq |E|$ and $|I(G_1)| = |R(G_1)|$). The last edge swap always rewires a 4-cycle, reducing $|R(G_{N-1}) \cup I(G_{N-1})|$ by 4. Thus, the upper bound on the number of edge swaps required for the simple case is $\frac{1}{2}(|I(G_1) \cup R(G_1)| - 2) \leq |E| - 1$.

For the connected case, essentially the same alternating cycle reductions are used. When it is not possible to rewire a 4-cycle C, an alternative 4-cycle C' is constructed such that there is at most one edge $e \in I(C') \setminus I(G_i)$. Therefore, rewiring C' reduces $|R(G_i) \cup I(G_i)|$ by 2 edges, making the same amount of progress as the reductions used in the simple case.

However, there is a case where a sequence of four edge swaps may reduce $|R(G) \cup I(G)|$ by as little as four edges. Now, up to $|R(G_1) \cup I(G_1)|$ edge swaps may be required

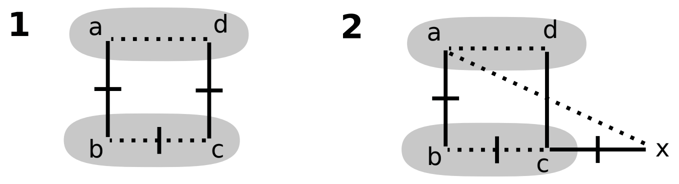

Figure 4.13: If a 4-cycle C in G_i that must be rewired includes at least one edge from each of $R(G_i)$ and $I(G_i)$, then it is one of the two cases shown in this figure. For each case, it is possible to construct a 4-cycle that satisfies the requirements to apply the C' construction described in Section 4.3.2.

Case 1: the cycle satisfies the requirements of Section 4.3.2 for the construction of an alternative cycle C'. Specifically, $R(C) \subseteq R(G_i)$ and there is at most one edge in $I(C) \setminus I(G_i)$.
Case 2: there is an edge $cd \in R(C) \setminus R(G_i)$ (note that the edge ad shown in the figure may or may not belong to $I(G_i)$ and this does not affect the following construction). An alternative 4-cycle is constructed as follows: due to the alternating cycle construction (Section 4.2.1) there must be an edge $xc \in R(G_i)$, but there cannot be an edge ax because $R(C)$ is a cutset in G_i. So the required 4-cycle is (a, b, c, x, a).

in the worst case. Since $|R(G_1) \cup I(G_1)| \leq 2|E|$, and the last edge swap rewires a 4-cycle (as in the connected case), the bound for the connected case becomes $2|E| - 1$.

Each edge swap can rewire at most 2 edges, and there are pairs of graphs which require every edge to be rewired, so at least $|E|/2$ edge swaps may be required in some cases. This suggests that the upper bounds of $|E| - 1$ and $2|E| - 1$ (for the simple and connected cases respectively) are close to optimal.

4.5 Summary

This chapter provides an alternative proof of Taylor's theorems about graph rewiring, which demonstrate that any graph can be rewired to any other of the same degree sequence, even when constrained to simple and connected graphs. The proof presented in this chapter directly constructs the required sequence of edge swap operations, as opposed to Taylor's proof which only proves the existence of such a sequence.

By directly constructing the edge swap sequences, it is possible to place an upper bound on the number of edge swap operations required to rewire between arbitrary graphs. The bound is $|E| - 1$ in the simple case, but $2|E| - 1$ for the connected case since some situations may required extra edge swaps. Note that there are arbitrary pairs of graphs for which $|E|/2$ edge swap operations may be required since each edge swap rewires two edges, and it may be necessary to rewire all edges. This suggests that the upper-bound of $|E| - 1$ is close to optimal.

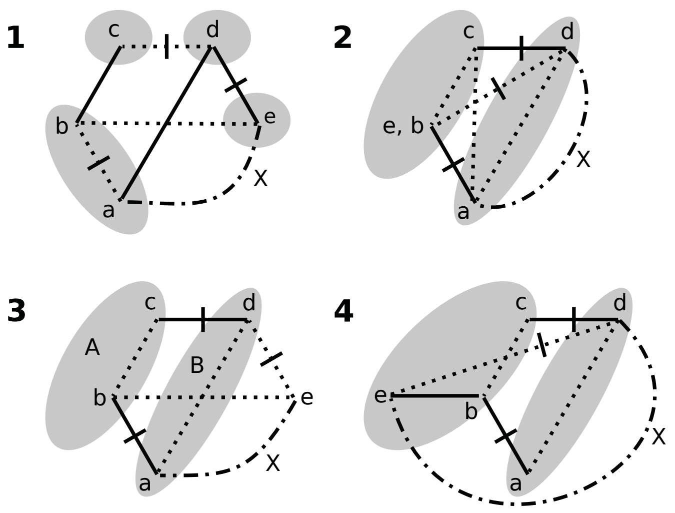

Figure 4.14: Case 1 from Figure 4.6 requires the rewiring of a 4-cycle $C = (a, b, c, d, a)$ in G_i, which may disconnect the graph. The reduction ordering described in Section 4.3.3 guarantees that there are at most two edges in C that are not in $R(G_i) \cup I(G_i)$. The following four cases are only required if $R(C) \cap R(G_i) = \emptyset$ or if $I(C) \cap I(G_i) = \emptyset$, otherwise Case 2 from Figure 4.13 would apply.

Case 1: $bc \in R(C) \setminus R(G_i)$. Since C is an alternating cycle, there is an edge $de \in R(C)$ as shown in the figure. Since $R(C)$ is a cutset, $e \neq b$ and there is no edge eb. Thus, (b, c, d, e, b) can be rewired without cutting the graph.
Case 2: $bc \in I(C) \setminus I(G_i)$. Since C is an alternating cycle, there is an edge $de \in I(C)$. Assume $e = b$. Then there is a 4-cycle (a, b, d, c, a) which can be rewired without cutting the graph. Rewiring (a, b, d, c, a) reduces $R(G_i)$ by 2, so this rewiring makes progress towards rewiring the entire graph.
Case 3: As for Case 2 except that $e \neq b$ and there is no edge eb. Then recursively rewire the alternating cycle (a, b, e, X, a) as shown in the figure. Note that the vertex e may be in the component labelled A or the component labelled B, and this does not make any difference in this case.
Case 4: As for Case 3 except that there is an edge eb. Since $R(C)$ is a cutset, e must be in the same component as b. Now the 4-cycle (e, b, c, d, e) can either be rewired directly (reducing the original alternating cycle as required), or if it cuts the graph, then it is handled as per Case 2 in Figure 4.13.

Chapter 5

The target set selection problem

The previous chapters have discussed the topological properties of complex networks (Chapter 2), and some of the random graphs models used to study those properties (Chapter 3). This chapter introduces two related problems concerning dynamic spreading processes on complex networks. The remainder of the thesis will aim to apply the theory from Chapters 2 and 3 to the problems described in this chapter. This will demonstrate how principles of network theory can be used to guide the design of algorithms optimised for complex networks.

There are numerous models for the simulation of how information may be dynamically transmitted and spread throughout complex networks, see Barrat et al. (2008) and Boccaletti et al. (2006) for a review of the better known models. In general, these models proceed in discrete timesteps. At each timestep the vertices are divided into those that are *active* and those that are *inactive*, where the active vertices are those that have been influenced by the spreading process. The spreading models considered in this chapter arise from the study of epidemiology (modelling the spread of infectious disease), rumour dynamics, and word-of-mouth marketing. This thesis is not concerned with the details of exactly what is being spread through the network, only with problems relating to the simulation procedures themselves.

Both of the problems presented in this chapter are concerned with finding a *target set*, a target set being the initial set of vertices from which some spreading process begins. The first problem, *minimum target set*, is discussed in Section 5.1. Here the goal is to find a minimum size target set from which influence will spread to the entire network. Minimum target set has been extensively studied in the context of a deterministic spreading process known as the *tipping model* or the *thresholds model* (see Ben-Zwi et al. (2009) and Nichterlein et al. (2010)). This version of the problem is the one used in this thesis.

The second problem, *maximum activation-set*, is discussed in Section 5.2. The goal of maximum activation set is to find a target set of size k which will lead to the

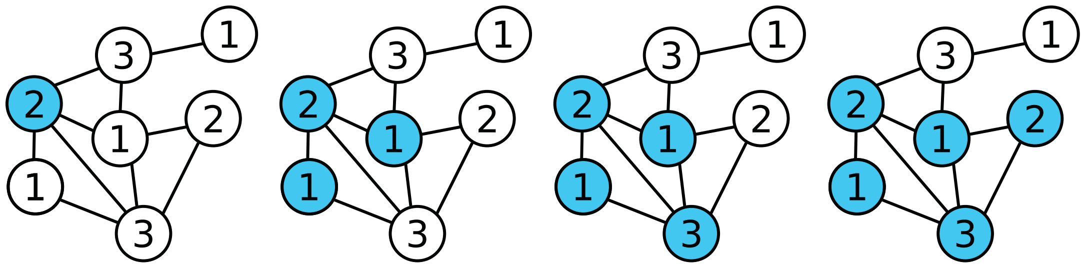

Figure 5.1: The tipping model. The numbers inside the vertices are the activation thresholds. Activated vertices are coloured blue, inactive vertices remain white. The initial target set (1 vertex) is shown on the left. To the right are shown the results of three successive activation rounds. No further activations are possible with this target set under these thresholds.

largest set of activated vertices (following many rounds of some spreading process). Maximum activation set has been studied under a wide range of very general spreading models (Kempe et al., 2005). Only one variation is considered in this thesis: Maximum activation set under the well-known SIR model (defined in Section 5.2). Justification for this particular choice is made in Section 5.2.

The main concern of this thesis is not the fine differences between different spreading processes, but rather how the theory of complex networks can inform the design of algorithms. Minimum target set and maximum activation set (under the SIR model) are chosen because the underlying spreading processes operate in a similar manner (they are both *progressive* as discussed in Sections 5.1 and 5.2), but one of the models (tipping) is deterministic while the other (SIR) is stochastic. This leads to interesting comparisons between what works in a deterministic setting and what works in a stochastic setting, without complicating the experiments with too many parameters.

5.1 Minimum target set

This problem concerns a deterministic spreading process known as the *tipping model*, introduced by Ben-Zwi et al. (2009) and based on a more general model studied by Kempe et al. (2003). In the tipping model (also known as the *thresholds model*), every vertex is in one of two states, *active* or *inactive*. Additionally, every vertex v has an integer threshold $t(v)$. Activation proceeds in rounds. During each round, for every vertex v the number of activated neighbours of v is compared to $t(v)$. If the number of activated neighbours $n(v)$ equals or exceeds $t(v)$, then v becomes activated. Figure 5.1 illustrates the process in a small graph.

The tipping model simulates a *progressive* process where active vertices remain active. Thus, the simulation reaches a point where no more vertices can change state, and this happens after no more than n rounds. This property allows one to easily simulate the cascade of activations in linear time. The algorithm is based on breadth-first search. Every active vertex is placed on a queue. Vertices are removed from the queue one by one and the neighbours are examined. Any neighbours that can be activated are marked as activated, then placed on the back of the queue. This continues until the queue is empty.

The problem is to find a minimum size target set (i.e. an initial set of vertices to activate) which will lead to a cascade of activations resulting in the activation of every vertex in the network. This is the same problem studied by Ben-Zwi et al. (2009). Since the sequence of activations is deterministic, it is possible to find a target set which is guaranteed to activate the entire network. In the case of a stochastic model, such as the SIR model used in Section 5.2, no such guarantee can be made. Hence a different but related problem (maximum activation set) is studied in that context.

An important issue to consider when using the tipping model is how the vertex thresholds are assigned. Due to the way the model is defined, any vertex with threshold 0 will be activated on the next round even if none of its neighbours are active. If $t(v) > d(v)$ (where $d(v)$ is the degree of v), then v can never be activated by its neighbours, and so it must belong to any minimum target set. If all the vertices have threshold 1 then the problem is trivial. Assuming the network is connected, any single vertex constitutes a target set for the entire network.

In practice, the thresholds are set in one of several ways (see for example Shakarian and Paulo (2012) or Nichterlein et al. (2010)). Under the *constant thresholds* scheme, $t(v) = \min(c, d(v))$ for some constant $c > 1$ and all vertices v. Under the *proportional thresholds* scheme, $t(v) = \max(1, r \cdot d(v))$ for some ratio $0 < r \leq 1$, and the thresholds are rounded to integers. A special case of proportional thresholds is *majority thresholds*, where $r = 0.5$. One further way to set the thresholds is probabilistically. This scheme is known as *linear thresholds* and is employed by Kempe et al. (2005). The thresholds are chosen uniformly at random from the interval $1 \leq t(v) \leq d(v)$ for each vertex v.

The experiments presented in this thesis compare the performance of algorithms under constant thresholds and under proportional thresholds. This ensures that the activation process remains deterministic, simplifying the experiments. If the thresholds were set randomly (using linear thresholds for example) then the size of a minimum target set could fluctuate significantly between experiments. By considering proportional thresholds with a range of values for r, it is possible to compare the effects of generally low thresholds with generally high thresholds. Constant thresholds, on the other hand, model a situation in complex networks where a few high degree vertices have proportionally much lower thresholds than the low degree vertices of the majority. Thus, contrasting the results for constant thresholds with the results for proportional thresholds hints at the effects of hubs on the cascade of activations.

An important special case is when $t(v) = d(v)$ for all vertices v. In this case, a minimum target set is equivalent to a minimum set of vertices C such that every edge is adjacent to a vertex in C. This is exactly equivalent to the well known NP-complete *minimum vertex cover* optimisation problem, discussed at length in Section 5.3.1. Due to this trivial reduction to vertex cover, the minimum target set selection problem is also NP-complete. Furthermore, minimum target set is known to be APX-hard (Chen et al., 2009). Thus, there is no polynomial time algorithm to solve general instances of the minimum target set problem, and there is no polynomial time approximation scheme.

Despite being NP-complete in general, it is possible to solve some restricted instances of minimum target set selection exactly. The case where $t(v) = 1$ for all v is a trivial example. However, polynomial time algorithms have also been devised for graphs with low treewidth (Ben-Zwi et al., 2009), graphs with small vertex covers, small cluster deletion sets, and small feedback edge sets (Nichterlein et al., 2010). These algorithms are discussed in detail in Section 5.3, including an evaluation of how well they perform on complex networks.

5.2 Maximum activation set

Whereas the minimum target set selection problem is concerned with finding a minimum size target set that activates the entire network, the maximum activation set problem seeks a size k target set (for some integer k) that activates the maximum number of vertices. The maximum activation set problem makes sense for a deterministic spreading process such as the tipping model discussed in Section 5.1, and also for stochastic spreading processes. For stochastic spreading processes, the goal is to find a target set which maximises the *expected* size of the activation set.

Like minimum target set selection, maximum activation set is NP-complete. In this case, though, there is a polynomial time approximation algorithm due to Kempe et al. (2005) with an approximation factor of $1 - 1/e - \epsilon$ where e is the base of the natural logarithm and $\epsilon > 0$ is an arbitrary real number.

Kempe et al. use a very general spreading model in their analysis, of which the well-known SIR model is a special case. The SIR model comes from epidemiology and was intended to model epidemic diseases, which spread rapidly through a population before dying out as people become immune. The SIR model predates the study of complex networks, and originally included an implicit assumption that the epidemic to be modelled occurs in a random network (similar to the Erdős-Rényi style networks described in Section 3.2.1). The model is easily adapted to complex networks, allowing for analytical comparisons between the random network and complex network cases (see Newman (2003), Boccaletti et al. (2006), Barrat et al. (2008)).

Under the SIR model, every vertex is in one of three states, (S)usceptible, (I)nfected, or (R)ecovered. Infected and Recovered are the activated states, Susceptible is the inactive state. Additionally, there is a parameter $0 \leq \beta \leq 1$ representing the infectivity of the disease (with 1 being highly infective and 0 being non-contagious). At each timestep, every Infected vertex infects each of its Susceptible neighbours with probability β. Then, the Infected vertices change to the Recovered state. Recovered vertices remain in the recovered state.

Like the tipping model, the SIR model is progressive, because the number of active (Infected or Recovered) vertices never decreases. Eventually, all the vertices are either Recovered or Susceptible, but there are no Infected vertices. Thus, it is possible to compute the expected size of an activation set using essentially the same algorithm as for the tipping model (described in Section 5.1). The expected size of an activation set is computed by running a large number of trial simulations of the SIR process, then computing the average size of the activation sets. The approximation algorithm of Kempe et al. (2005) relies on this method of computing the expected size of the activation set.

Although the maximum activation set is relevant for a variety of spreading models, this thesis only considers maximum activation set under the SIR model. This simplifies the experiments that follow in Chapter 7 by intentionally reducing the number of parameters.

5.3 Parametrization of minimum target set

This section concerns the polynomial time algorithms for minimal target set selection proposed by Ben-Zwi et al. (2009) and Nichterlein et al. (2010). Since the general problem is NP-complete (see Section 5.1), these algorithms are only applicable to restricted subsets of graphs. This section aims to determine whether or not the algorithms are applicable to typical complex network topologies.

The algorithms of Ben-Zwi et al. (2009) and Nichterlein et al. (2010) are based on the theory of parametrized complexity. Parametrized complexity is founded on the observation that many problems that are NP-complete in general can be solved in polynomial time in some restricted cases. One can often identify a *parameter* of the input that distinguishes between the intractable (NP-complete) cases, and the tractable cases (which permit polynomial time algorithms).

A suitable parametrization of the problem permits the design of *parametrized* algorithms, where the running time depends on both the parameter and the input size (as opposed to traditional algorithms where the analysis of the running time depends only on the input size). There is extensive literature on the subject of parametrized complexity, and many NP-complete problems are known to permit efficient parametrized solutions (Downey and Fellows, 2012).

The performance of a parametrized algorithm will be determined by how the running time of the algorithm depends on the parameter. A problem is said to be *fixed-parameter tractable* (FPT) if it permits an algorithm with running time $O(f(k) \cdot n^c)$ where k is the parameter, n is the input size (number of vertices or edges for graph problems), and c is a constant. f is an arbitrary function of k, and in the case of FPT parametrized versions of classically NP-complete problems, f grows exponentially or worse. Notice that if the parameter is regarded as a constant, then $f(k)$ contributes only a constant factor to the otherwise polynomial running time. Thus, a well designed FPT algorithm runs in polynomial time when the parameter is fixed, and the constant factor is small when parameter is sufficiently small.

The algorithm of Ben-Zwi et al. (2009) uses the *treewidth* of the graph as a parameter. Treewidth measures, in a sense, how tree-like a graph is (Bodlaender and Koster, 2008). A tree has treewidth 1, whereas the treewidth of a k vertex clique is $k - 1$. Treewidth is defined as the *width* of a minimum tree-decomposition. A tree-decomposition places the vertices of a graph into overlapping *bags* of connected vertices, so that for every edge in the graph there is a bag containing both the endpoints. The bags are connected together in a tree in such a way that for every vertex v, the subtree of bags containing v is connected. The width of a tree-decomposition is equal to the number of vertices in the largest bag less one, and a minimum tree-decomposition is one that minimises this width.

Algorithms based on treewidth, such as the Ben-zwi et al. algorithm, require a tree-decomposition of the input graph. Computing a minimum tree-decomposition is an NP-complete problem in itself. However it is often possible to find a small tree-decomposition using approximation algorithms. Section 5.3.4 considers the treewidth of the complex networks listed in Table 2.1, by computing upper and lower bounds.

Minimum target set selection can be solved trivially for trees, so if one assumes the problem is easier the closer the graph is to a tree, then treewidth is an obvious choice for the parameter. Ben-Zwi et al. (2009) give an algorithm for minimum target set selection parametrized by the treewidth of the graph w with running time $n^{O(w)}$. This algorithm is not FPT, as the degree of the polynomial in n depends on the parameter w, so it is only practical when the treewidth is extremely small. Ben-Zwi et al. (2009) additionally prove that there is no algorithm with running time $n^{o(\sqrt{w})}$ (unless $P = NP$), so their $n^{O(w)}$ algorithm is close to optimal.

The three parametrized versions of the problem considered by Nichterlein et al. (2010) are all FPT. They parametrize the problem by *vertex cover number* (τ) to get a running time of $O(2^{(2^\tau+1)\cdot\tau} \cdot m)$ where m is the number of edges. When parametrized by *cluster edge deletion number* (ξ), they get a running time of $O(4^\xi \cdot m + n^3)$ where n

is the number of vertices. Finally, when parametrized by *feedback edge set number* (f), the running time is $4^f \cdot n^{O(1)}$.

Nichterlein et al. claim that their FPT algorithms are suitable for complex networks, specifically social networks (Nichterlein et al., 2013). The remainder of this section investigates this claim by computing the values of the parameters on the complex networks listed in Table 2.1.

A vertex cover of a graph is a set of vertices C such that every edge is adjacent to at least one vertex in C. The vertex cover number τ of a graph is the size of a minimum vertex cover, and computing τ is well known NP-complete problem. Fortunately, vertex cover is also FPT when parametrized by the size of the vertex cover, i.e. if there is a small vertex cover then it can be found in polynomial time. Section 5.3.1 considers the vertex cover numbers of complex networks.

A cluster edge deletion set is a set of edges R such that for a graph with edge set E, $E \setminus R$ is a disjoint union of cliques, also known as a cluster graph. The cluster edge deletion number ξ is the minimum size of a cluster edge deletion set. As with treewidth and vertex cover, computing ξ is an NP-complete problem. Like vertex cover, it is also FPT when parametrized by the size of the cluster edge deletion set itself. Section 5.3.2 discusses the cluster edge deletion sets of complex networks and how they are computed.

Finally, a feedback edge set is a set of edges F so that for a graph with E edges, $E \setminus F$ is a cycle-free graph (a forest). The feedback edge set number f of a graph is the size of a minimum feedback edge set. If F is a minimum feedback edge set, then $T = E \setminus F$ is a maximum spanning forest, which can be computed using a straightforward breadth-first or depth-first search. The minimum feedback edge set, then, consists of all the edges that the breadth-first or depth-first search does not add to the spanning forest (i.e. $F = E \setminus T$). Note that minimum target set can be solved on a per-component basis, so one need only consider the feedback edge set number of the component with the most cycles. Section 5.3.3 discusses the feedback edge set number parameter in complex networks.

5.3.1 Vertex cover number

The parametrization of minimum target set by vertex cover number (Nichterlein et al., 2010) requires that the input graph have a small vertex cover. This section attempts to compute the sizes of minimum vertex covers in the networks listed in Table 2.1, the results are reported in Table 5.1. Note that since the computation of a minimum vertex cover is an NP-complete problem, it is not always possible to compute an exact solution. In these cases, upper and lower bounds are reported instead. Vertex cover is FPT when parametrized by the solution size k, so in many cases exact solutions can be found even on large graphs. Even when exact solutions are not possible, the FPT algorithms can be adapted to improve the upper and lower bounds.

The parametrized version of the vertex cover problem is to find a vertex cover of size no more than k for some parameter k. This can be solved in $O(1.2852^k + kn)$ time (Chen et al., 1999), although a simpler version of this algorithm is used in this section, with a running time of $O(2^k + kn)$. In order to find the minimum vertex cover using the FPT vertex cover algorithm it is necessary to trial multiple values of k. Although the algorithm constructs a vertex cover smaller than k, it does not necessarily construct the smallest such vertex cover.

The number of values of k that must be tested can be reduced by using a binary search strategy. Upper and lower bounds for the size of the minimum vertex cover are maintained, and k is selected to be in the middle of the range. If the parametrized algorithm finds a vertex cover, then the size of that vertex cover becomes the new upper bound, otherwise k becomes the new lower bound. Eventually the upper and lower bounds become equal, at which point the minimum vertex cover has been found. It is easy to see that only a logarithmic number of trials are required.

In order to set the initial upper and lower bounds, two heuristics are used. The first is based on graph matching. A *matching* of a graph $G = (V, E)$ is a set $M \subseteq E$ such that all $e \in M$ are mutually independent (not adjacent to each other). A vertex adjacent to an edge in M is said to be *matched*. Additionally, a matching is *maximal* if there is no $e \in E$ that is not adjacent to an edge in M, and a matching is *maximum* if it is the largest possible matching of the graph. Every maximum matching is maximal.

A maximal matching can be computed by a straightforward greedy approach (add random edges until no more edges can be added). A maximum matching can also be computed in polynomial time, but requires a more complicated approach (Diestel, 2000; Micali and Vazirani, 1980).

It is easy to see that, given a maximal matching of a graph, the union of all the endpoints of the matching is a vertex cover of the graph. Every edge that is not in the matching must have a matched endpoint since the matching is maximal, therefore every edge is covered as required. Furthermore, there can not be any vertex cover smaller than the number of edges in the matching. This is because at least one endpoint for every edge must be in the vertex cover. Thus, given a maximal matching M of a graph, the vertex cover number τ must lie in the range $|M| \leq \tau \leq 2|M|$.

In an effort to get a tighter upper bound, a simple heuristic greedy algorithm can be used. At each step, the algorithm puts the vertex of highest degree into the vertex cover, then removes all edges incident on that vertex. This continues until there are no more edges left in the graph, at which point the vertex cover is complete.

The classic FPT algorithm for vertex cover proceeds in two phases. The first phase, *kernelization*, reduces the size of the problem to a *kernel* no larger than $2k$ by applying a sequence of reductions in polynomial time. The second phase, the *depth-bounded search-tree*, searches the kernel for a vertex cover smaller than k. The kernelization phase is fast, since the running time does not depend on k. However, the running time of the search-tree phase is exponential in k, which makes it impractical unless k is small or the kernel is small.

Many reduction rules have been devised for the kernelization of vertex cover, see Abu-Khzam et al. (2004) and Chen et al. (1999). The rules are to be applied iteratively, until no further reduces are possible. The following rules have found to be highly effective in practice (Abu-Khzam et al., 2004):

Degree 0 : Isolated vertices are not part of any minimum vertex cover, so they can be removed from the graph.

Degree 1 : For a degree 1 vertex v with neighbour u, there is a minimum vertex cover that includes u and not v.

Degree 2 : For a degree 2 vertex v with neighbours u and w such that there is an edge uw, there is a minimum vertex cover that includes u and w, but not v. There is also a reduction rule for the case when u and w are not adjacent, but it is surprisingly difficult to implement efficiently.

Degree k : A vertex with degree greater than k must belong to any vertex cover smaller than k because otherwise all its neighbours would have to go in the vertex cover, resulting in a vertex cover larger than k.

Crown reduction : In a graph $G = (V, E)$, a crown consists of an independent set $I \subseteq V$, and a set $H = N(I)$ where $N(I)$ is the neighbour set of I and $I \neq \emptyset$. A crown must also satisfy the condition that in the bipartite graph induced by I and H, there is a matching such that every vertex in H is matched. For every crown, there is a minimum vertex cover that contains all the vertices of H and none of the vertices of I. A crown can be found and removed in *linear time* (Abu-Khzam et al., 2004).

Reduction by matching : In a graph $G = (V, E)$, two disjoint sets $C' \subseteq V$ and $V' \subseteq V$ can be computed in $O(\sqrt{|V|} \cdot |E|)$ time such that every $v \in C'$ is in a minimum vertex cover of G, and V' is the reduced problem instance. The algorithm works by finding a maximum matching in a graph derived from G. This reduction was first proposed by Chen et al. (1999), and is based on a theorem by Nemhauser and Trotter Jr (1975).

When a reduction rule adds a vertex to the vertex cover, k is reduced by one. Thus, the value of k for the kernelized graph is typically less than the original value of k.

Of the reduction rules, all but the degree k rule can be applied without having to first determine the value of k. This fact was exploited for this thesis when computing the vertex cover numbers reported in Table 5.1. Before computing upper and lower bounds and proceeding to run the full algorithm for various values of k, the graph is fully kernelized except for the degree k rule. The reduction by matching guarantees that the kernel is no larger than $2k$ (Chen et al., 1999), thus there is a lower bound of $l/2$ for the size of the kernel, where l is the size of the kernel (typically much smaller than the size of the network n).

Upper and lower bounds are computed for the kernelized graph by combining the $n/2$ lower bound, the upper and lower bounds from matching, and a greedy upper bound. The greedy upper bound is computed by iteratively adding the highest degree vertex to the vertex cover and removing it from the graph, until the graph is empty and therefore a vertex cover for the entire network has been computed.

Following computation of the bounds, the full FPT algorithm (including kernelization with the degree k rule and the search-tree phase) is run for various values of k until a minimum vertex cover of the kernel is found. To get a vertex cover of the original graph, it is necessary to include any vertices that were added to the vertex cover by the initial kernelization phase.

It has been noted (by Abu-Khzam et al. (2004) for instance) that the vertex cover kernelization performs much better in practice than the theory implies. One often finds a kernel smaller than $2k$, and in some cases the kernel is even smaller than k. In the case of complex networks, the reason is most likely due to the k-shell structure of these networks, discussed in Section 2.2.5. Repeated application of the degree 1 and 2 rules removes the 1 and 2 shells from the graph. Most vertices in complex networks belong to the low shells, so large numbers of vertices can removed by these simple reductions.

The version of the search-tree phase used in this thesis is based on the observation that for any vertex v, either v belongs to the minimum vertex cover, or the neighbours of v belong to the vertex cover. A search tree can be constructed by recursively branching on every vertex. Since the desired vertex cover is smaller than k, there is no need to search more than k levels into the tree, hence the running time is no worse than 2^k.

The overall running time of the search-tree phase can be improved by running the kernelization phase again between branches (Niedermeier and Rossmanith, 2000). Note that more sophisticated branching rules are possible, which can reduce the running time of the search tree phase from $O(2^k)$ to $O(1.2852^k)$ (Chen et al., 1999). This is still exponential, so unless k is small (or the kernel is very small), neither algorithm will be practical. Thus, it was not considered worthwhile attempting the more complicated search tree when computing Table 5.1.

Network	Size (vertices)	Minimum cover	
		Lower bound	Upper bound
Physicists 1	40,421	19,809 (49%)	24,412 (60.4%)
Physicists 2	34,546	15,868 (45.9%)	28,053 (81.2%)
Enron	36,692	13,334 (36.3%)	14,653 (39.9%)
Gnutella	10,876	4,348 (40.0%)	
Blogs	1,490	560 (37.6%)	
Internet	27,719	8,087 (29.2%)	
Neural	297	134 (45.1%)	227 (76.4%)
Metabolic	453	249 (55.0%)	

Table 5.1: Sizes of minimum vertex covers for the networks described in Section 2.1. For networks where an exact solution could not be computed, upper and lower bounds are reported. The numbers are also reported as a percentage of vertices.

Despite the powerful algorithmic techniques employed, it was not possible to compute exact vertex covers for all the networks listed in Table 2.1. This is largely because, as can be seen in Table 5.1, the vertex covers are not small enough for the FPT approach to guarantee reasonable running times. Consider for example the Neural network where k is lower bounded by 134. The search tree phase has running time $O(2^k)$ for the algorithm used, and $O(1.2852^k)$ for the optimal algorithm. Even using the optimal algorithm, $1.2852^{134} \approx 4 \times 10^{14}$. Note that the other networks where exact solutions could not be computed have far higher lower bounds for k than does the Neural network.

In the cases where it was possible to compute exact vertex cover numbers, this was due to the previously discussed “unreasonable effectiveness” of the kernelization phase. Where the kernel was too large, it was not possible to run the search-tree phase to completion, and hence only the upper and lower bounds are reported.

Although it was not possible to compute exact values for the vertex covers of all the networks, the computed lower-bounds are all very high, ranging from 29% to 55%. This translates to minimum vertex covers that are generally large in terms of the absolute number of vertices, and therefore unsuitable as a parameter for the minimum target set problem.

It should be noted that there are numerous other heuristic algorithms for computing a minimum vertex cover that could have been employed, such as hill-climbing and genetic algorithms. It was not considered worthwhile implementing such algorithms since, although they might permit smaller upper bounds, the lower bounds are already very high. Thus, the conclusions from this section would not be affected.

5.3.2 Cluster edge deletion number

Given a graph $G = (V, E)$, the *minimum cluster edge deletion* problem is to find a set $D \subseteq E$ such that $G \setminus D$ is a disjoint union of cliques, also known as a *cluster graph*. The cluster deletion number ξ is the size of a minimum cluster edge deletion set. As with vertex cover, this is an NP-complete problem.

Cluster edge deletion is FPT when parametrized by the solution size k, with the algorithm of Gramm et al. (2003) giving a running time of $O(1.77^k + |V|^3)$. If there is a cluster edge deletion set smaller than k, then the algorithm returns such a set (although it may not be the smallest possible), otherwise the algorithm fails. By running the algorithm for various values of k it is possible to home in on a *minimum* cluster edge deletion set.

The essential insight of Gramm et al. (2003) is to look for *conflict triples*. A conflict triple is formed by three vertices u, v, w where there are edges uv and vw, but no edge uw. Such triples cannot exist in a cluster graph. In cluster deletion, either one or both of uv and vw must be deleted. One can obtain a lower-bound to the cluster deletion number by simply counting the number of conflict triples, since at least one edge must be deleted to resolve each triple.

The algorithm of Gramm et al. (2003) proceeds in two phases, kernelization and a depth-bounded search-tree. The kernelization rules produce a kernel no larger than $2k^2 + k$, and this takes $O(|V|^3)$ time to compute. The kernelization rules given by Gramm et al. are for the more general *cluster edge editing* problem, where edges may be added as well as deleted to get a cluster graph. To ensure that the algorithm always makes progress, edges may be marked *permanent* to show that they cannot be later deleted, or *forbidden* to show where edges have already been deleted. The simplified rules for cluster edge deletion are as follows:

Rule 1.1 : For every pair of vertices $\{u, v\}$, if $|N(u) \cap N(v)| > k$ (where $N(v)$ is the set of vertices adjacent to the vertex v) and there is an edge uv, then mark uv permanent. If there is no edge uv then the instance has no solution.

Rule 1.2 : For every pair of vertices $\{u, v\}$, if $|(N(u) \cup N(v)) \setminus (N(u) \cap N(v))| > k$, mark uv forbidden. If there is an edge uv then add it to the deletion set.

Rule 1.3 : If there is a conflict between rules 1.1 and 1.2 ($|N(u) \cap N(v)| > k$ and $|(N(u) \cup N(v)) \setminus (N(u) \cap N(v))| > k$), then the instance has no solution.

Rule 2 : For every three vertices u, v, and w, if uv and uw are permanent, then so is vw. If there is no edge vw then the instance has no solution. If uv is permanent and uw is forbidden, then vw must also be forbidden. If there is an edge vw then add it to the deletion set.

Rule 3 : If a connected component forms a clique, it can be removed from the graph.

Although the cubic running time of this kernelization *is* polynomial, it remains highly problematic for large graphs (recall from Section 5.3.1 that the vertex cover kernelization was essentially linear). Furthermore, notice that kernelization rule 1 only applies when $k < |V|$ and rule 2 only applies following at least two applications of rule 1. As can be seen from the results in Table 5.2, $k > |V|$ in every network tested, so these kernelization rules cannot be directly applied to the complex networks considered in this thesis.

In addition to the kernelization rules (which guarantee that the kernel is no larger than $2k^2+k$), Gramm et al. also give three heuristic reduction rules. These degree-based rules do not provide any theoretical guarantees regarding the size of the kernel, but they are highly effective in practice for the same reasons as the degree-based reductions for vertex cover (Section 5.3.1). Each of the heuristic reduction rules operates on three adjacent vertices u, v, and w. The three rules are as follows:

Degree 1 : If u and v both have degree 1, and there is no edge uv, then there is a minimum cluster edge deletion set including uw.

Degree 2 : If the degree of u is 1, the degree of v is 2, and there are edges uv and vw, then there is a minimum cluster edge deletion set including vw.

Degree 3 : If both u and v have degree 3, w has degree 2, there are edges uv, uw, and vw; and two distinct vertices x and y with edges ux and vy, then there is a minimum cluster edge deletion set including ux and vy.

The depth-bounded search-tree phase constructs a search tree by recursively branching on the conflict triples. For each conflict triple, there are two possibilities: either the first edge is deleted, or the first edge is not deleted and only the second edge is deleted. The height of the tree is limited by the current value of the parameter k. By interleaving kernelization phases between levels of the search-tree, and employing some rules to avoid exploring branches that cannot lead to a solution, Gramm et al. are able to achieve their 1.77^k running time for the search-tree phase.

Values for the cluster edge deletion numbers of the networks listed in Table 2.1 were computed using ideas from Gramm et al. (2003), and the results are reported in Table 5.2. The procedure was as follows: first compute a kernel using the degree based reduction rules. Then get a lower bound for k by counting the non-overlapping conflict triples. Using the binary search inspired technique for choosing values of k (Section 5.3.1), attempt to find an exact solution using the search-tree phase and interleaving of the degree based reduction rules.

Network	Size (edges)	Minimum deletion set (% of edges)
Physicists 1	175,693	117,368 (66.8%) *
Physicists 2	420,877	272,475 (64.7%) *
Enron	183,831	159,292 (86.7%) *
Gnutella	39,994	35,328 (88.3%)
Blogs	16,715	16,047 (96.0%)
Internet	41,684	30,784 (73.9%)
Neural	2,148	1,189 (55.3%) *
Metabolic	2,025	1,682 (83.1%)

Table 5.2: Sizes of minimum cluster edge deletion sets for the networks described in Section 2.1. The * indicates graphs for which it was not possible to compute an exact solution. In those cases the reported numbers are lower bounds only.

It can be seen in Table 5.2 that the cluster edge deletion numbers are high in both absolute and relative terms. Even in cases where it was only possible to compute a lower bound, the bounds are all greater than 50% of the edges. This rules out the use of cluster edge deletion number as a parameter for minimum target set in complex networks.

5.3.3 Feedback edge set number

A minimum feedback edge set is a minimum set of edges that, when deleted, leave a cycle-free network, i.e. a tree. A minimum feedback edge set maximises the number of edges in the resulting tree. Thus, in a connected network (i.e. there is a path between every pair of vertices), removing any minimum feedback edge set results in a spanning tree with $n - 1$ edges. The number of edges in the minimum feedback edge set is then $m - (n - 1)$ where n is the number of vertices in the network and m is the number of edges.

There is a complication, however, in that a network may have more than one connected component. Therefore, it is necessary to find the connected components (with a breadth first search for example) in order to compute the minimum feedback edge set.

Although the complex networks tested are generally sparse, this still implies very large feedback edge sets. For this reason, the feedback edge set number was computed for the largest component only, since minimum target set can be solved per component. Table 5.3 reports these feedback edge set numbers for the networks listed in Table 2.1. Notice that even for the sparsest network (the Internet network), the feedback edge sets are far too large to allow for a practical parametrization of the minimum target set problem.

Network	size (edges)	Feedback edge set (% of edges)
Physicists 1	175,693	135,279 (77.0%)
Physicists 2	420,877	386,384 (91.8%)
Enron	183,831	147,116 (80.0%)
Gnutella	39,994	29,119 (72.8%)
Blogs	16,715	15,493 (92.7%)
Internet	41,684	15,767 (37.8%)
Neural	2,148	1,852 (86.2%)
Metabolic	2,025	1,573 (77.7%)

Table 5.3: Sizes of minimum feedback edge sets for the networks described in Section 2.1. The feedback edge set number reported is for the largest connected component, rather than the entire network, as explained in Section 5.3.3.

5.3.4 Treewidth

Ben-Zwi et al. (2009) prove that when parametrized by the treewidth w of the input, there is no algorithm for minimum target set selection with running time $n^{o(\sqrt{w})}$ or faster. Thus, treewidth is only useful as a parameter for large networks if it is very small. The treewidth based algorithm of Ben-Zwi et al. has a running time of $n^{O(w)}$, and the constant factor in the $O(w)$ part is greater than 1, so the running time is at least n^w. Therefore, even graphs with treewidth as small as 3 are likely to be intractable.

Computing the treewidth of a graph, like vertex cover number and cluster deletion number, is an NP-complete problem. Computing a tree decomposition is not known to be FPT, but there are close approximation algorithms. However, the approximation algorithms have slow running time, so heuristics are typically employed instead (Bodlaender and Koster, 2008). Note that a treewidth-based algorithm does not require the *minimum* tree decomposition, but merely a sufficiently small tree decomposition

Instead of attempting to compute an accurate approximation, loose upper and lower bounds are computed to confirm that the treewidth of complex networks is typically too large to permit a parametrized approach to minimum target set. For the lower bound, *degeneracy* is used. Degeneracy is related to the k-shell decomposition (Section 2.2.5). It is the highest k for which there is a non-empty k-shell. It can equivalently be defined as the maximum possible minimum degree of any induced subgraph of the original graph.

It is known $w(G) \geq \delta(G)$ where $w(G)$ is the treewidth of a graph G and $\delta(G)$ is the minimum degree of G (Bodlaender and Koster, 2008). It is also well-known that the treewidth of any induced subgraph can be no higher than the treewidth of the entire graph, or conversely, the treewidth of the entire graph must be at least as high as the treewidth of any induced subgraph. Thus, degeneracy (the highest possible minimum degree of any induced subgraph) is a lower-bound to the treewidth of the entire graph.

Network	size (vertices)	Treewidth	
		Lower bound	Upper bound
Physicists 1	40,421	29	3745
Physicists 2	34,546	30	9671
Enron	36,692	43	2247
Gnutella	10,876	7	2804
Blogs	1,490	36	258
Internet	27,719	14	105
Neural	297	10	84
Metabolic	453	10	38

Table 5.4: Upper and lower bounds to the treewidth of the networks described in Section 2.1. These numbers were computed as described in Section 5.3.4.

For upper bounds, the greedy-degree algorithm described in Bodlaender and Koster (2010) is used because it is fast, and is known to provide a reasonably tight upper bound in practice. This greedy algorithm constructs a tree-decomposition by constructing bags out of the lowest degree vertices first.

The results for treewidth upper and lower bounds are reported in Table 5.4. Although the treewidth of the networks is considerably smaller than the vertex cover number, the cluster deletion number, and the feedback edge set number, it is still a significant fraction of the number of vertices in the network. This mirrors a result of Gao (2009), which shows that for at least one class of complex network (Barabási-Albert random graphs), the treewidth is a linear function of the size of the network.

In conclusion, the treewidth parameter is too large in the networks tested to permit a parametrized solution to minimum target set. It is, however, small enough that it may be a plausible approach for other problems on complex networks.

5.4 Summary

This chapter introduces two problems relating to spreading processes on complex networks: minimal target set under the tipping model, and maximum activation set under the SIR model. Both these problems are NP-complete, and minimum target set is also APX-hard. It is for their computational difficulty, and the clear practical applications, that these two problems were chosen for study.

The two problems are explored in detail throughout the remaining two chapters of this thesis, where the goal is to apply the complex network theory discussed in Chapters 2 and 3 to the problems in an effort to obtain smaller solutions, or faster solutions, than possible with the currently known heuristic algorithms. It is hoped that this will lead to some insight into how complex network theory can be applied in general to algorithm design.

For the minimum target set problem there are a number of parametrized algorithms that are able to solve the problem in polynomial time on restricted classes of networks: those with small vertex covers, small cluster deletion numbers, small feedback edge sets, and extremely low treewidth. Section 5.3 employs state of the art algorithms to compute these quantities on the complex networks listed in Table 2.1 (although for feedback edge it is trivial to compute an exact solution). It is found that none of the parameters are small enough to permit the practical computation of exact solutions.

These results may appear to contradict Nichterlein et al. (2013), who claim that the parametrized target set algorithms can be realistically employed on several social networks. However, they must filter the edges of the networks in order for the parameters to be small enough. In fact, on empirical social network data (specifically their "DBLP conference" and "DBLP author" networks), they filter out over 99% of the edges before the parameters become small enough to be practical.

Although the parameters (vertex cover number, cluster deletion number, feedback edge set number) are very high in real networks, some FPT algorithms (such as the vertex cover algorithm) are known to be much faster in practice than their upper bounds would suggest. This is most likely not the case for for the algorithms described by Nichterlein et al. (2010).

Both the vertex cover based, and the cluster deletion set based algorithms for minimum target set selection operate by a brute force search on a set which is upper bounded in size by the relevant parameter. These sets are highly restrictive in their topology, and so it is unlikely that they are much smaller in practice than their upper bounds would suggest. The feedback edge set based algorithm includes a kernelization rule that removes degree 1 vertices. This may be effective on networks with a large 1-shell, but on it's own it is not sufficient to compute exact solutions unless the feedback edge set number is also small.

The following chapter discusses several known heuristics for minimum target set, and introduces a novel *distributed* heuristic. The possibility of combining heuristic and parametrized approaches is then explored. Chapter 7 then introduces the idea of augmenting the input graph with a small number of edges, so that a small target set can be found, even if there was no such target set in the original graph.

This is based on the assumption that it is possible to introduce new edges to many networks, which is reasonable for the social networks and technological networks that are the main focus of complex networks research. For example, an online social network could "recommend" new friends (as Facebook is known to do). For technological networks, it may be possible to build new communications links or (in the case of the Internet) establish new connections between ISPs.

Chapter 6

Heuristics for target set selection

The minimum target set selection problem introduced in Chapter 5 is NP-hard, inapproximable, and the known FPT parametrized algorithms are not directly applicable to complex networks. Thus, a heuristic approach is necessary.

This chapter introduces a range of heuristics for minimum target set selection in Section 6.1, some of which are informed by the properties of complex networks and others which are not. Section 6.2 introduces a novel heuristic for minimum target set which can be applied in a distributed manner. The sizes of the target sets obtained by each of the heuristic algorithms are compared on the networks listed in Table 2.1.

Finally, in Section 6.3, the possibility of combining heuristic and parametrized solutions is explored. Specifically, it is discovered that as hubs are progressively removed from complex networks, there is a rapid fall in the feedback edge set and vertex cover numbers of the networks. This suggests that a hybrid algorithm combining the heuristic "hubs first" approach of Section 6.1.1 with the FPT parametrized algorithms discussed in Chapter 5 may yield better solutions in general than purely heuristic solutions.

6.1 Greedy heuristics

This section describes three heuristic algorithms for computing a minimum target set. The hubs-first algorithm (Section 6.1.1) is inspired by the distinctive degree topology of complex networks. The marginal-gain algorithm (Section 6.1.2) is adapted from the optimal approximation algorithm for maximum activation set (discussed in Section 5.2). Although marginal-gain cannot provide an approximation guarantee for minimum target set, it is plausible that it could at least provide a small target set in practice. The Shakarian-Paulo-Reichman algorithm, independently discovered by Reichman (2012) and Shakarian and Paulo (2012), places vertices in the target set based on their thresholds; it was not intended for any network topology in particular.

These three heuristics are compared in Figures 6.1 and 6.2 (which also include the distributed heuristic introduced in Section 6.2). Target sets are computed using each of the heuristics for a range of networks and threshold assignments. The networks are the empirical complex networks listed in Table 2.1.

The thresholds are assigned in two different ways as discussed in Section 5.1: *proportional* thresholds are assigned as a fixed proportion of the degree for each vertex, *constant* thresholds are assigned as the minimum of a constant and the degree for each vertex. Figure 6.1 shows the results for proportional thresholds, Figure 6.2 shows the results for constant thresholds. The proportional thresholds are assigned from the range 0.1 - 0.9. For the constant thresholds, a mostly arbitrary range from 2 to 20 was chosen. Since most vertices in complex networks have low degree it makes sense to choose low numbers for constant thresholds (otherwise most vertices would have thresholds equal to their degree).

6.1.1 Hubs first

This algorithm is based on the observation that complex networks have strongly skewed, heavy-tailed degree distributions, as discussed in Section 2.3. This implies a relatively small number of very high degree vertices (hubs). These vertices would make good candidates for a target set on account of their high degree (they can directly influence more vertices than a similarly sized target set selected from low-degree vertices).

The hubs-first algorithm is as follows: the highest degree vertex v is added to the target set, then $A(v)$, the activation set from v, is computed. Note that $v \in A(v)$. There would be no benefit to adding a $u \in A(v)$ to the target set in addition to v, since u is always activated by v anyway. Thus, the next step is to remove $A(v)$ from the graph. It is then necessary to adjust the thresholds to account for the influence of the activated vertices that are removed. For every vertex $w \notin A(v)$ that is adjacent to vertices in $A(v)$, the threshold $t(v)$ is reduced by $|N_w \cap A(v)|$, where N_w is the set of vertices adjacent to w. The procedure is then repeated from the highest degree in the resulting graph until the graph is empty, at which point the target set activates the entire network.

This is essentially a quadratic time algorithm, since finding the highest degree vertex requires scanning the entire network. An improvement to $O(n \log n)$ time could be achieved by tracking vertex degrees in a heap, but in practice this optimisation is not necessary. The addition of each hub causes the network to become smaller, and it quickly shrinks to the point where there is little to be gained from a more sophisticated algorithm.

Experimental evaluation of hubs-first on the complex networks listed in Table 2.1 is reported in Figures 6.1 and 6.2 for proportional and constant thresholds respectively. The algorithm appears to perform well across all the thresholds ranges evaluated.

From Figures 6.1 and 6.2, it can be seen that the hubs first algorithm often produces a smaller target set than other, more complicated heuristics. This suggests that in networks with a highly-skewed and long-tailed degree distribution, the high degree vertices form a relatively small target set for the entire network. It is likely that the other heuristics also find target sets containing large numbers of high degree vertices. However, the other heuristics are able to include more lower-degree vertices which may be less optimal, hence why the hubs-first heuristic tends to produce the smallest target set.

6.1.2 Marginal gain

The marginal-gain algorithm (proposed by Kempe et al. (2005)) is based on the optimal approximation algorithm for maximum activation set. Maximum activation set (see Section 5.2) is a subtly different problem to minimum target set in that it seeks a target set of fixed size k which activates the largest possible proportion of the network. The existence of a target set of size k that activates the entire network does not imply the non-existence of a smaller target set that does the same.

For maximum activation set, the algorithm of Kempe et al. (2005) achieves an approximation bound of $1 - 1/e - \epsilon$ where e is the base of the natural logarithm. No such bound is possible for minimum target set (Chen, 2009), but it is plausible that this algorithm could compute almost optimal target sets in practice.

The marginal-gain algorithm, as stated by Kempe et al. (2005), was intended for stochastic activation models (such as SIR), but it is trivially adapted to the deterministic tipping model used for the minimum target set problem. The goal of the algorithm is to maximise the activation set, i.e. the set of vertices that will eventually become activated when the target set is activated. When a vertex is added to the target set, the extra vertices that become activated are referred to as the *marginal* activation set. The marginal gain algorithm adds vertices to the target set one by one, always choosing the vertex that maximises the marginal activation set.

Computing the activation set from a target set is a linear time operation. The naïve marginal-gain algorithm requires that this computation be made for every vertex in the graph, and for every vertex that gets placed in the target set, so the overall running time is cubic. This is problematic on large graphs. Improvements by Chen et al. (2009) reduce the running time to quadratic, but the Chen et al. version of the algorithm is only applicable to independent cascade models (such as SIR, but not the tipping model; see Kempe et al. (2005)).

In order to apply the marginal-gain algorithm to the complex networks listed in Table 2.1, an optimisation of the naïve algorithm was required. The activation sets $A(v)$ from all the vertices v were precomputed and stored in large bit-sets. The activation set A of the current target set T was also maintained at each step. For the set A and for all the sets $A(v)$, the neighbour sets $N(A)$ and $N(A(v))$ were also maintained. $N(A)$ is defined to be $\bigcup_{a \in A} N_a \setminus A$ where N_a is the set of vertices adjacent to the vertex a.

In the case that $A \cap A(v) = \emptyset$ and $N(A) \cap N(A(v)) = \emptyset$, the marginal gain from adding v to the target set is simply $A(v)$. Checking this condition takes linear time (intersection on bit-sets), but the constant is much smaller than always recomputing the entire activation set. If the condition is not satisfied then it is necessary to recompute the activation set from adding v to the target set, and the marginal gain will be $A(\{v\} \cup T) \setminus A$ where T is the target set so far. Since $A(\{v\} \cup T)$ is a subset of $A(\{v\} \cup T')$ whenever $T \subset T'$, adding v to the target set will activate every vertex in $A(\{v\} \cup T)$ on this round, and on all subsequent rounds. Therefore, every vertex in $A(\{v\} \cup T)$ excluding v can be permanently removed from the set of candidates to include in the target set.

The sizes of the target sets produced by the marginal-gain algorithm, as shown in Figures 6.1 and 6.2, are generally good. However, marginal-gain (cubic running time) is often beaten by heuristics with much better running time, such as the hubs-first algorithm.

6.1.3 The Shakarian-Paulo-Reichman algorithm

This heuristic algorithm for minimum target set was first introduced by Reichman (2012), who also proves that it finds a target set no larger than $\sum_{v \in V} \min\left(1, \frac{k}{d(v)+1}\right)$ where V is the vertex set of the graph, k is the *maximum* threshold, and $d(v)$ is the degree of the vertex v. The algorithm was independently discovered by Shakarian and Paulo (2012), who applied it to complex networks and found that in practice, it often finds a target set much smaller than Reichman's upper-bound.

The Shakarian-Paulo-Reichman (SPR) algorithm is as follows: for every vertex, compute the difference between the degree and the threshold $\delta(v) = d(v) - t(v)$. If $t(v) > d(v)$ or (equivalently) $\delta(v) < 0$, then v can never be activated by its neighbours, therefore v must belong to the minimum target set. The algorithm operates by successively removing vertices from the graph, causing the degrees of the remaining vertices to decrease. At each step, the vertex v with the smallest non-negative $\delta(v)$ is selected to be removed. The thresholds remain constant throughout this process, so eventually $\delta(v) < 0$ for all vertices v remaining in the graph. At this point, the vertices remaining in the graph are a target set for the original graph.

The asymptotic running time for the SPR algorithm is the same as for the hubs-first algorithm discussed in Section 6.1. The naïve implementation (used in this thesis) is quadratic, but an improvement to $O(n \log n)$ is possible by tracking $\delta(v)$ for each vertex in a heap. As with the hubs-first algorithm, the network gets smaller after every round of the algorithm, and so the runtime is much faster in practice than the upper bounds would suggest.

Both algorithms (SPR and hubs-first) are considerably faster than the cubic marginal-gain algorithm discussed in Section 6.1.2. As seen in Figures 6.1 and 6.2, the SPR algorithm usually produces a smaller target set than either hubs-first or marginal-gain, except for high (above 0.7) proportional or high constant thresholds.

6.1.4 Experimental comparison

Each of the three heuristic algorithms discussed in this section was used to compute a target set for each of the networks listed in Table 2.1. The results for proportional thresholds are reported in Figure 6.1 and the results for constant thresholds are reported in Figure 6.2. Figures 6.1 and 6.2 also include results for target sets computed using the distributed algorithm discussed in Section 6.2.

For the proportional thresholds, the hubs-first algorithm generally produces the smallest target set whether the thresholds are high or low. The marginal-gain algorithm performs similarly to the hubs-first algorithm, except when the thresholds are very high. This effect is especially apparent in the Neural and Metabolic networks. Finally, the Shakarian-Paulo-Reichman algorithm tends to produce larger target sets than either of the other two heuristics.

For constant thresholds, the results are somewhat different. Here, SPR algorithm performs well on the Blogs, Neural, and Metabolic networks, but still produces relatively large target sets for the other networks. As for proportional thresholds, the hubs-first and marginal-gain algorithms continue to produce similar results.

It is not surprising that the marginal gain algorithm performs comparatively worse when the thresholds are higher. Activating a single vertex is not sufficient to start a cascade of activations, unless that vertex had threshold 1. If all vertices have high thresholds (as under high proportional thresholds), then it may be necessary to activate a large number of vertices in order to start a cascade. Until these initial vertices have been activated, the marginal-gain algorithm is effectively blind (as the activation set from each vertex includes only the vertex itself).

Concerning the relatively poor performance of the SPR algorithm, especially for high proportional thresholds, consider that the algorithm removes vertices where the threshold is close to the degree. Under high proportional thresholds, all vertices have thresholds close to their degree. Thus, the SPR algorithm is likely to remove some of the high degree vertices, which would probably be better included in the target set (considering that the hubs-first heuristic produces the smallest target sets in these cases).

Of the three algorithms compared in this section, the hubs-first algorithm has the best performance across a range of network topologies and threshold values. The running time of the hubs-first algorithm was similar to the SPR algorithm in that both implementations were $O(n^2)$ (although the runtime is much faster in practice than this upper bound would suggest). The marginal-gain algorithm also produces relatively small target sets, but the running time ($O(n^3)$) is too slow for large networks.

6.2 A distributed heuristic algorithm

Conventional algorithms for networks consider the entire network as a single static structure. Distributed algorithms, on the other hand, view the vertices of the network as individual computing elements and the edges as communication links between the vertices. A distributed algorithm is executed concurrently on every vertex in the network. Vertices may send messages to their immediate neighbours, but any long-range messages must be routed via a sequence of directly connected vertices. This is a natural model of computation for several important classes of complex networks, including communications networks such as the Internet, and social networks.

This section considers a distributed version of the Shakarian-Paulo-Reichman algorithm (described in Section 6.1.3). As with the SPR algorithm, the distributed algorithm maintains $\delta(v) = d(v) - t(v)$ for each vertex. The key difference is that instead of removing the vertex with the smallest $\delta(v)$ in the network, every vertex v examines its 1-hop neighbourhood, and the vertex u with the lowest $\delta(u)$ in that neighbourhood is removed. This heuristic is illustrated in Figure 6.3. Note that many vertices may be removed in a single round when using this heuristic.

The distributed version proceeds in rounds, as with the conventional SPR algorithm, but there are several adaptations to the distributed environment. The modified procedure is run *concurrently* by every vertex in the graph; synchronization of the rounds is achieved by sending messages between the vertices. The SPR algorithm calls for vertices to be removed from the network between the rounds, which does not make sense in a distributed environment where the network cannot be modified. Instead, vertices are marked as being in a *removed* state.

In the distributed algorithm, every vertex is in one of three states: *removed*, *targeted*, or *undecided*. The *removed* state represents the vertices that would be removed from the network by the conventional SPR algorithm. The *targeted* state represents vertices v where $\delta(v) < 0$, i.e. vertices that must go into the target set. Vertices that are neither *removed* nor *targeted* are *undecided*, and every vertex begins in this state. This system requires a redefinition of $\delta(v)$ as $|U(v)| + |T(v)| - t(v)$ where $U(v)$ is the set of all *undecided* neighbours of v, $T(v)$ is the set of all *targeted* neighbours of v and $t(v)$ is the threshold of v as usual.

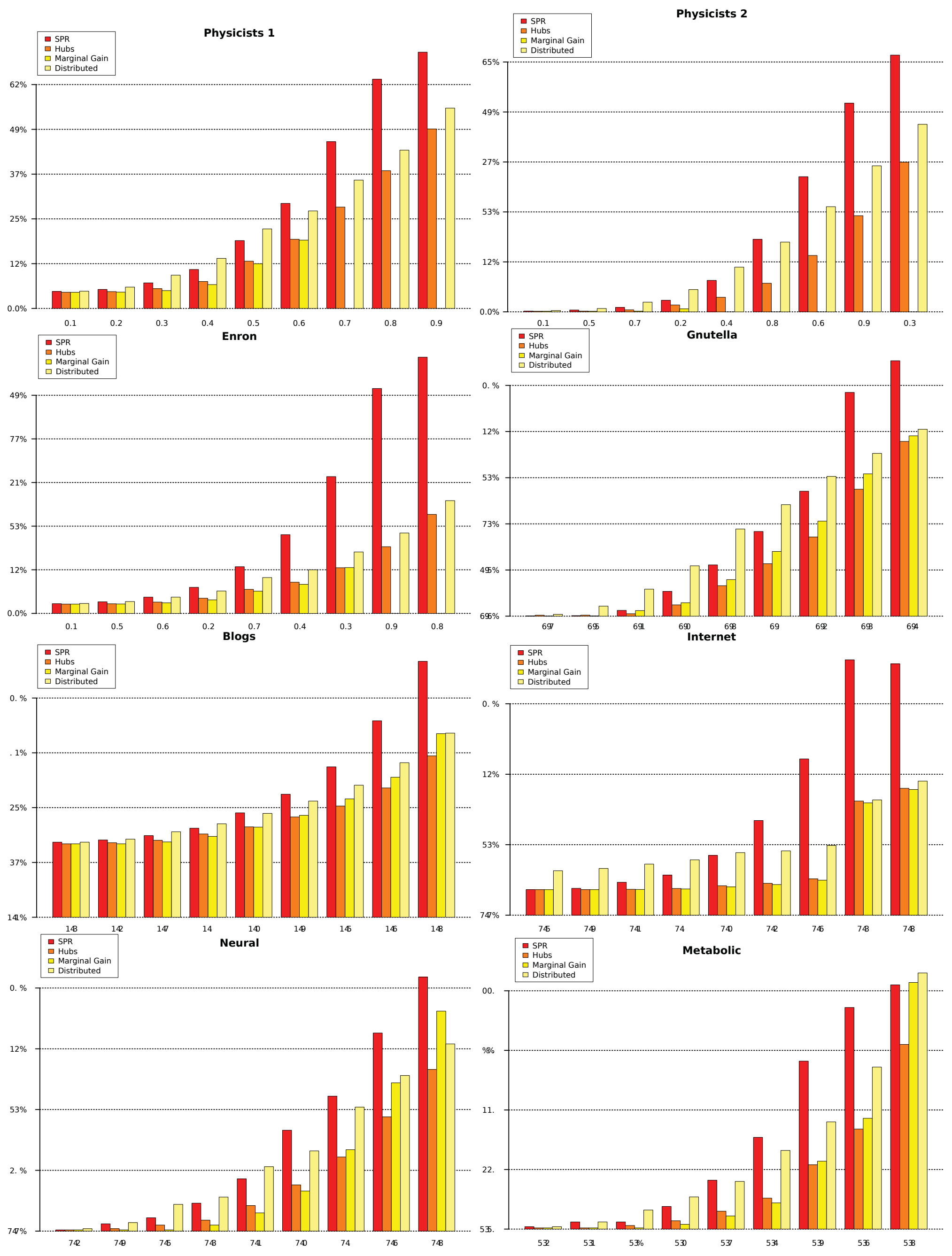

Figure 6.1: Comparison of heuristics for minimum target set selection with proportional thresholds from 0.1 to 0.9, in the networks listed in Table 2.1. SPR is the Shakarian-Paulo-Reichman algorithm (Section 6.1.3). Distributed is the distributed heuristic (Section 6.2). The sizes of the target sets are reported as percentages of the vertices in the networks.
On some large networks with high thresholds, the marginal-gain algorithm was terminated before it could compute a target set (as the runtime was too long). The missing data are for the Physicists 1 network (thresholds 0.7 to 0.9), for the Physicists 2 network (thresholds 0.5 to 0.9), and for the Enron network (thresholds 0.8 to 0.9).

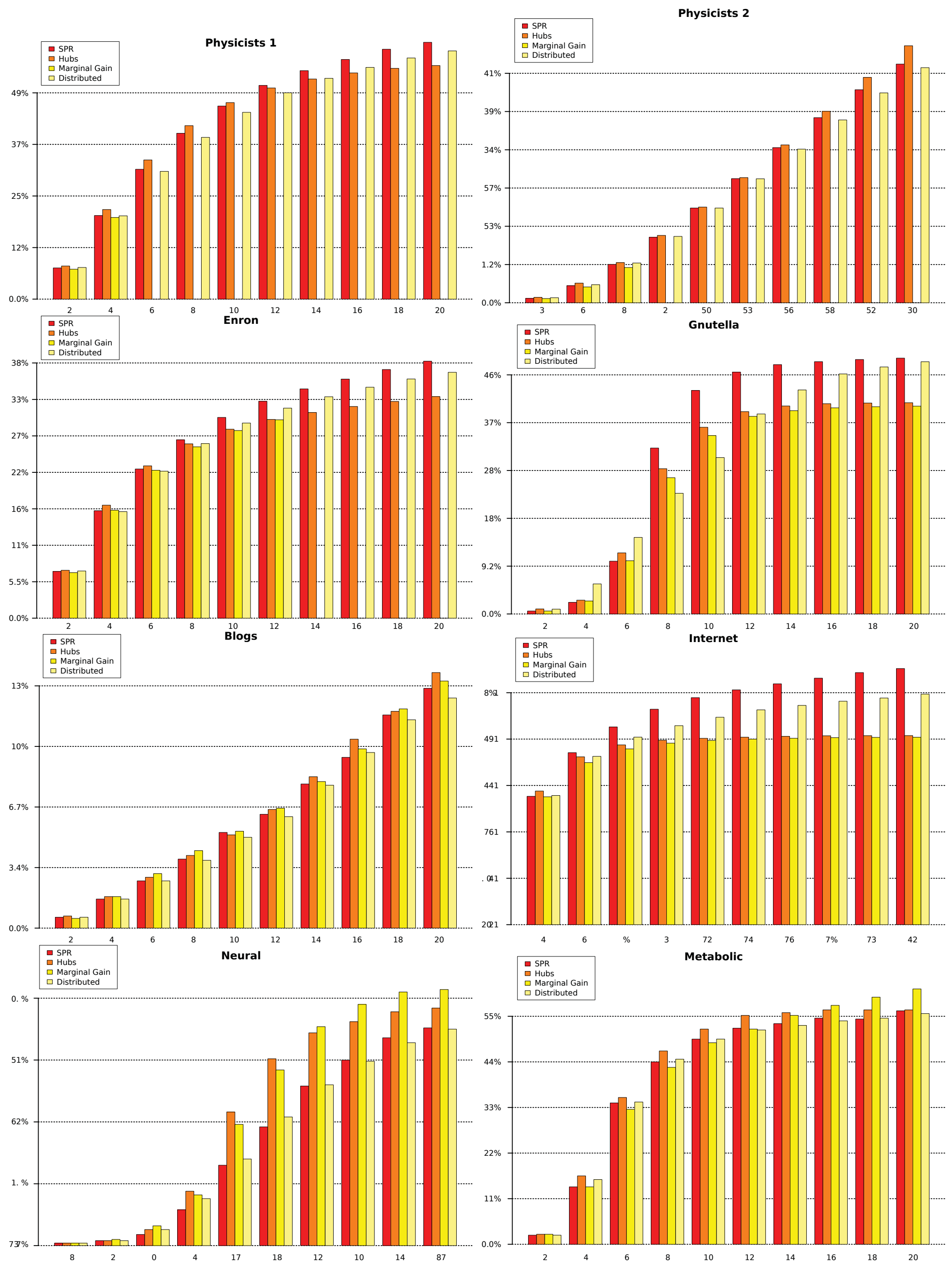

Figure 6.2: Comparison of heuristics for minimum target set selection with constant thresholds from 2 to 20, for the networks listed in Table 2.1. The sizes of the target sets are reported as percentages of the vertices in the networks.
On some large networks with high thresholds, the marginal-gain algorithm was terminated before it could compute a target set (as the runtime was too long). The missing data are for the Physicists 1 network (thresholds 6 to 20), for the Physicists 2 network (thresholds 8 to 20), and for the Enron network (thresholds 14 to 20).

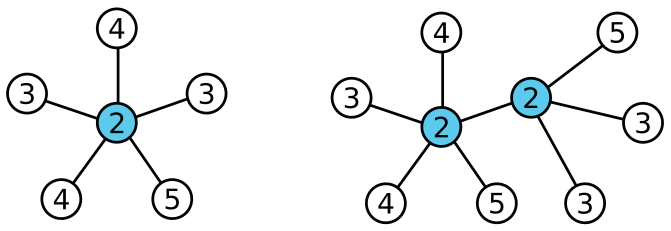

Figure 6.3: An illustration of the distributed heuristic. Each vertex v is labelled with $\delta(v)$. *Left*: The highlighted vertex will be removed during the next round, since it has a lower $\delta(v)$ than all its neighbours. *Right*: Both of the highlighted vertices have lower $\delta(v)$ than their neighbours, but only one of them can be removed during the next round since they are adjacent. Which vertex to remove is chosen by some arbitrary criteria.

During each round, every *undecided* vertex communicates with its immediate neighbours to determine whether it should transition to the *removed* or the *targeted* state, or whether it should remain *undecided*. Once a vertex transitions to the *removed* or *targeted* state it is no longer required to respond to messages from its neighbours, since these states are final. As with the original SPR algorithm, at least one vertex will transition to a final state on every round, so at most n rounds are required where n is the number of vertices in the network. The distributed algorithm is designed so that a large number of vertices may transition on each round, and in practice it is found that the required number of rounds is much less than n.

Each round of the distributed algorithm is composed of three communication phases:

Phase 1: Every *undecided* vertex v sends a message containing $\delta(v)$ asynchronously to all of its *undecided* neighbours. v then waits until it has received the same message from all its neighbours. If $\delta(v) < 0$, then v now transitions to the *targeted* state and no longer responds to messages from its neighbours. The neighbours of v can deduce that this transition has taken place.

Phase 2: Every vertex decides if it is a candidate for transition to the *removed* state. A vertex v is a candidate if $\delta(v) \leq \delta(u)$ for every vertex u adjacent to v. Every vertex v then sends a message to each of its neighbours telling them whether or not v is a candidate. v waits until it has received the same message from all its *undecided* neighbours.

Phase 3: If v is not a candidate, then it remains in the *undecided* phase. If v is a candidate, and none of the neighbours of v are candidates, then v transitions to the *removed* state. If v is a candidate, and at least one neighbour u of v is also a candidate, then only one of u or v may transition to the *removed* state. This ensures that the central principle of the SPR algorithm, that the removed vertices are always activated by the non-removed vertices, is maintained. This situation is illustrated in Figure 6.3

(*right*). If u and v are both candidates, then some kind of tie-breaker is required. For example, if every vertex can be assigned a unique id number, then the vertex with the lowest id could be chosen to transition to the *removed* state.

Once v has decided whether or not to transition to the *removed* state, it sends a message to all its neighbours informing them of the decision. v waits until it has received the same message from all its neighbours, and then updates its own $\delta(v)$ to reflect any neighbouring vertices that have transitioned to the *removed* state. If $\delta(v)$ is now negative, v will transition to the *targeted* state at the beginning of the next round.

At any one moment, the current round of the algorithm may differ between distant parts of the network. However, notice that no vertex enters *phase 2* of the algorithm until all of its neighbours have finished *phase 3* from the previous round. Hence, neighbouring vertices always agree on the current round. It is easy to see that there are no further race conditions or deadlocks.

The same algorithm can be stated in a sequential form: first, a list is made of every vertex that is a candidate for removal (a vertex v is a candidate if $\delta(v) < \delta(u)$ for all neighbours u of v). Then, for every candidate v with a neighbour u that is also a candidate, either v or u is removed from the list according to some tie-break criteria, as in the distributed version. All the vertices remaining in the list are removed from the network. $\delta(v)$ is recomputed for every vertex, and the process repeats in rounds until every remaining vertex has negative $\delta(v)$, as in the original Shakarian-Paulo-Reichman algorithm.

For the distributed version of the algorithm, the running time is linear since each round takes constant time per vertex (assuming bounded degree), and there are at most n rounds. The bounded degree assumption is reasonable here because most vertices in a complex network have low degree, and even the high degree hubs have much lower degree than n in large networks. The sequential version is quadratic since the entire network must be scanned to compute the candidates list. However, the constant is much smaller in practice than for the SPR algorithm, because multiple vertices are removed on each round (whereas the SPR algorithm removes exactly one vertex per round).

6.2.1 Experimental results

The distributed algorithm was implemented in both its distributed and sequential forms. The distributed implementation uses the Scala actors API (Haller and Odersky, 2007) to provide true parallel processing. The algorithm was run on each of the networks listed in Table 2.1, and for a range of proportional and constant thresholds. The results are shown in Figure 6.1 for proportional thresholds, and Figure 6.2 for constant thresholds. These figures also show the results for the heuristic algorithms discussed in

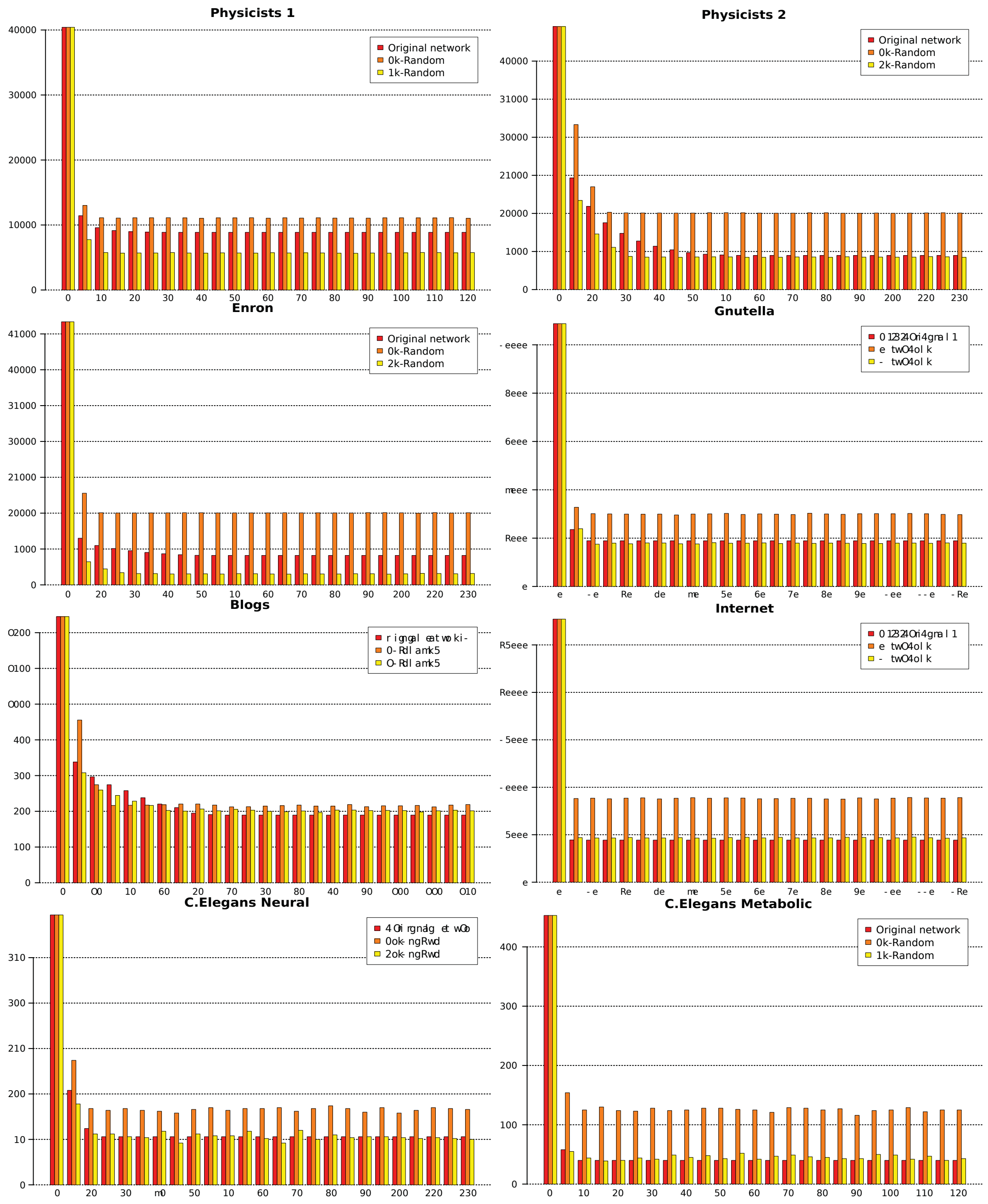

Figure 6.4: Sizes of computed target sets by the distributed algorithm described in Section 6.2. Results are shown for each of the networks listed in Table 2.1, and for $0K$ and $1K$ randomized versions of these graphs. The x-axis shows the limit on the number of rounds, the y-axis shows the size (in vertices) of the computed target set. The first bar in each chart shows the results for a limit of 0, which represents a situation where all the vertices are placed into the target set. Thus, the proportional size of the target set can be checked by comparing with the first bar in the chart.

Section 6.1, including the original Shakarian-Paulo-Reichman algorithm.

For low proportional thresholds (Figure 6.1), and constant thresholds (Figure 6.2), the performance of the distributed algorithm is comparable to the SPR algorithm in terms of the size of the computed target set. For the high proportional thresholds (> 0.5), the performance of the distributed algorithm is generally superior to SPR, and comparable to the other heuristics (hubs-first and marginal-gain). Overall, the distributed algorithm does not usually produce a smaller target set than the hubs-first algorithm, although in practice it may be faster if the number of rounds is significantly less than the size of the network.

These results are somewhat surprising, as the distributed algorithm does not have a global view of the network, unlike the conventional SPR algorithm. Note that the activation process was simulated from every target set to confirm that all the algorithms compared in Figures 6.1 and 6.2 do indeed produce legitimate target sets.

The main difference between the two algorithms (distributed and conventional SPR), is that the distributed algorithm can remove multiple vertices in a single round. If a vertex u has a large number of neighbours with small $\delta(v)$, and there are not too many edges between those neighbours (as is typical of hubs in complex networks), then the distributed algorithm may remove enough neighbours of u to make $\delta(u)$ negative, forcing u into the target set. Since u was a hub, it was a good candidate for this. The conventional SPR algorithm would be forced to remove the neighbours one-by-one, until $\delta(u) = 0$, at which point there is a risk that u will be removed from the graph, resulting in a larger target set.

A further concern for the performance of the distributed algorithm is the number of rounds it takes to compute a target set. If this number is a significant fraction of the number of vertices in the network (n), then there is little point in attempting a distributed approach (since the previously discussed sequential version of the distributed heuristic would be more efficient).

For each of the networks listed in Table 2.1, the distributed algorithm was run with a limited number of rounds. The algorithm was modified to terminate after the limit on the number of rounds was reached. Following termination, any *undecided* vertices were placed in the target set. These experiments were repeated on $0K$ and $1K$ random graphs (see Section 3.3). To reduce the number of parameters in the experiments, the threshold assignment was limited to proportional thresholds with a factor of 0.5. The results of the experiments are reported in Figure 6.4.

The results reported in Figure 6.4 show that the number of rounds required in practice is significantly less than n. For most of the networks tested, 5 rounds are sufficient to find a small target set. On other networks (Physicists2 and Blogs), no more than 50 rounds were required; very few considering the sizes of these networks

($34,546$ for Physicists2, $1,490$ for Blogs).

Figure 6.4 also shows the results for $0K$ and $1K$ randomized networks (as described in Section 3.3). For the $0K$ random networks, the minimal target set is reached after less than 20 rounds in all cases. This is much faster than in the original networks, or the $1K$ random networks. However, the computed target set is considerably larger in the $0K$ random networks in most cases.

The $1K$ random networks show essentially the same pattern as the original networks, so it can be concluded that the degree structure of complex networks plays a role in the ability of the distributed algorithm to find target sets that involve longer chains of activation than can be found in $0K$ random networks. However, there is additional structure in the complex networks that is also relevant to this effect.

6.3 Combining heuristics with parameters

This section describes a scheme whereby a heuristic algorithm (such as those discussed in Section 6.1) is combined with one of the parametrized algorithms introduced by Nichterlein et al. (2010). Recall from Section 5.3 that the algorithms of Nichterlein et al. cannot be applied directly to complex networks, as the values of the parameters are usually too high. Note, however, that the hubs-first and SPR algorithms described in Section 6.1 operate by progressively removing vertices from the network. This results in simpler networks as vertices are removed, and so it is likely that some of the parameters will also fall (feedback edge set number and vertex cover number for example).

This leads to a hybrid algorithm, where the heuristic algorithm is run until some network parameter becomes sufficiently small that an exact solution can be computed for the remainder of the problem using a parametrized algorithm. The possibility of such an algorithm is explored by considering the combination of the hubs-first algorithm (Section 6.1.1) with the parametrized algorithms using feedback edge set number (Section 5.3.3) and vertex cover number (Section 5.3.1).

The hubs-first heuristic is chosen as it removes the vertices with the greatest impact on the network first. The SPR algorithm operates in the opposite direction, tending to remove low degree vertices first. Thus, it is plausible that removing a small number of hubs could lead to a large reduction in one of the network parameters. This would allow the parametrized phase of the hybrid algorithm to solve the bulk of the problem exactly. If the SPR algorithm were used for the heuristic phase, it is likely that a large number of vertices would have to be removed before the parameters became small enough for the parametrized phase to take over.

The feedback edge set number and vertex cover number parameters are chosen because they are the most likely to decrease as high degree vertices are removed by the heuristic phase of the hybrid algorithm. Removing high degree vertices will tend to reduce the density of the network, directly leading to a decrease in the feedback edge set number. Vertex cover number is also related to sparsity: if there are fewer edges, then it may be that fewer vertices will be required to cover them.

The viability of this hybrid approach (hubs-first plus feedback edge set or vertex cover) was tested experimentally by progressively removing vertices from networks in order from highest to lowest degree. The two parameters feedback edge set number and vertex cover number were recomputed after removing each vertex. The parameters were computed only for the largest connected component of the network, rather than for the entire network. The reasoning for this will be explained in the following paragraphs. Since some of the networks are too large to compute an exact vertex cover number, an upper bound was computed instead, using the same technique as was used in Section 5.3.1.

A further concern when removing high degree vertices is that there is likely to be a dramatic increase in the number of components (and therefore a similarly dramatic decrease in the size of the largest component). In most complex networks, the majority of the vertices are connected in a single "giant" component of size $O(n)$ where n is the number of vertices in the network. It is well-known that as high degree vertices are removed from complex networks, the size of the giant component decreases rapidly, until at some point it breaks apart into a multitude of much smaller components and there is no longer a clearly identifiable giant component in the network (Newman, 2003). Newman and other physicists refer to this phenomenon as a phase transition from a dense state with a giant component to a sparse state.

If the largest component is small enough (≤ 30 vertices for example), then the remainder of the problem could be solved by brute force, and the parametrized approach would provide little benefit. For this reason, the size of the largest component was measured as hubs were removed, in addition to the feedback edge set number and vertex cover number. Thus, it is possible to determine if the parameters fall to a small size before the giant component breaks up.

A further consideration is that since the problem can be tackled on a component-by-component basis, it is sufficient to consider the values of the network parameters only in the largest component. The expected rapid decrease in the size of the largest component will contribute to the decreasing values of the other parameters. Hence, rather than measuring the parameters for the entire network, they are measured only for the largest component.

These experiments were run on the complex networks listed in Table 2.1, and on randomized versions of those networks according to the methodology described in Section 3.1. The results are reported in Figure 6.5 for feedback edge set number, Figure 6.6 for vertex cover number, and Figure 6.7 for the size of the largest component.

The results for feedback edge set number and vertex cover number are similar. The parameters fall rapidly at first and then more slowly. Depending on the network, the parameters become small after between 5% and 25% of the vertices have been removed. The size of the largest component (Figure 6.7) falls slowly at first, with a rapid decrease at some point which depends on the network, as predicted by the phase transition theory (Newman, 2003). Notice that this rapid decrease in the size of the largest component is not visible for the Physicists 2 network, suggesting that more vertices would have to be removed before the phase transition is reached (the charts only show the removal of up to 30% of the vertices).

There are clear qualitative differences in the behaviour of the original networks compared to the $0K$ and $1K$ randomized networks. The decrease in the feedback edge set number and vertex cover number parameters on $0K$ networks appears to be linear with respect to the number of vertices removed. This differs from the curve seen in the original and $1K$ randomized networks, with an initially rapid decrease followed by a longer period of slow decrease. Although the original networks and the $1K$ networks show the same shaped curve, the rate of decrease is faster in the original networks. This suggests that the behaviour observed in the original networks is largely due to the strongly skewed, heavy-tailed degree distributions of these networks, but that there is also another unidentified topological factor at work.

By comparing Figure 6.7 with Figures 6.5 and 6.6, it can be seen that the point at which the feedback edge set and vertex cover parameters become small is about the same point at which the size of the largest component becomes small. Therefore, the benefit of using a parametrized approach, rather than simple brute force, to solve the bulk of the problem is marginal.

These experiments looked at the values of the network parameters as hubs were removed. However, note that the hubs-first algorithm (Section 6.1.1) removes not only the highest degree vertex v on each round, but also $A(v)$ (the activation set of v). To further evaluate the viability of the combined hubs-first / FPT strategy, the experiments were repeated, removing $A(v)$ at each step rather than just v. Since a threshold assignment is necessary in order to compute $A(v)$, and to avoid increasing the number of variables in the experiment, majority thresholds were used (i.e. proportional thresholds with ratio $r = 0.5$).

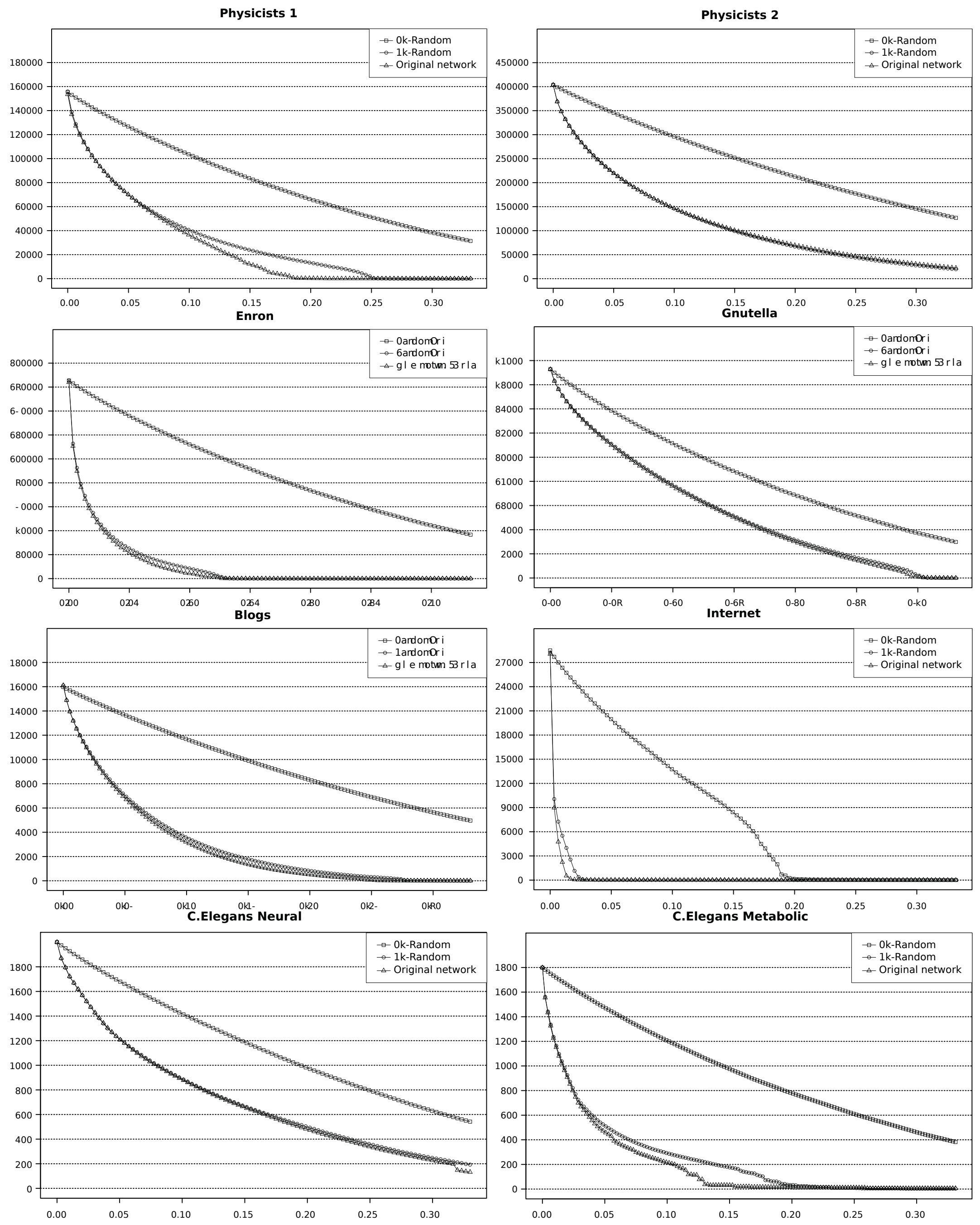

Figure 6.5: Sizes of minimum feedback edge sets in complex networks as vertices are removed from the networks in order from highest to lowest degree. The networks used are those listed in Table 2.1. The x-axis shows the proportion of vertices removed, the y-axis shows the size of the feedback edge set. The experiments were repeated on $0K$ and $1K$ random graphs (as per Section 3.1).

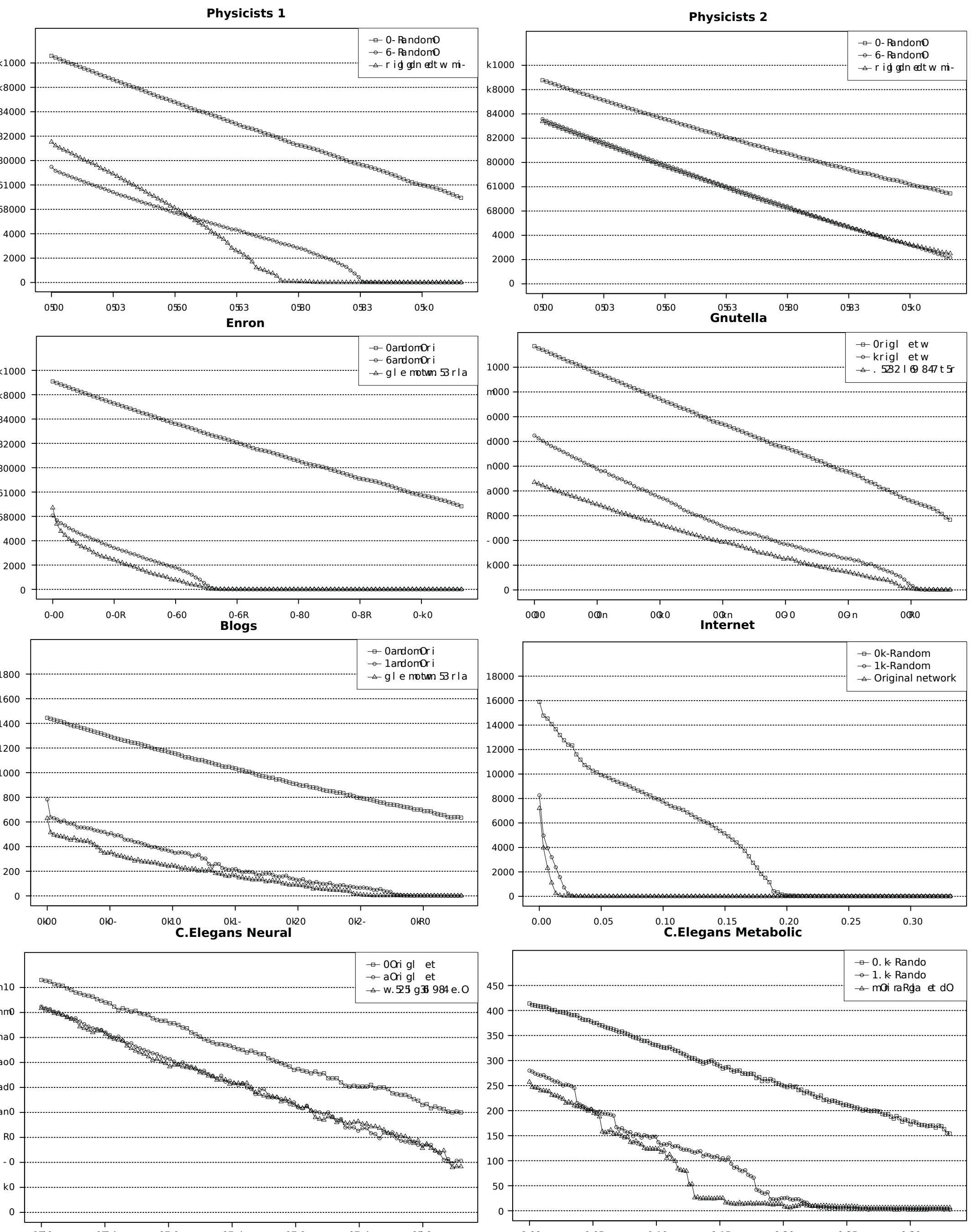

Figure 6.6: Upper bounds for minimum vertex covers in complex networks as vertices are removed in order from highest to lowest degree. The networks used are those listed in Table 2.1. The x-axis shows the proportion of vertices removed, the y-axis shows the vertex cover upper bound. The experiments were repeated on $0K$ and $1K$ random graphs (as per Section 3.1).

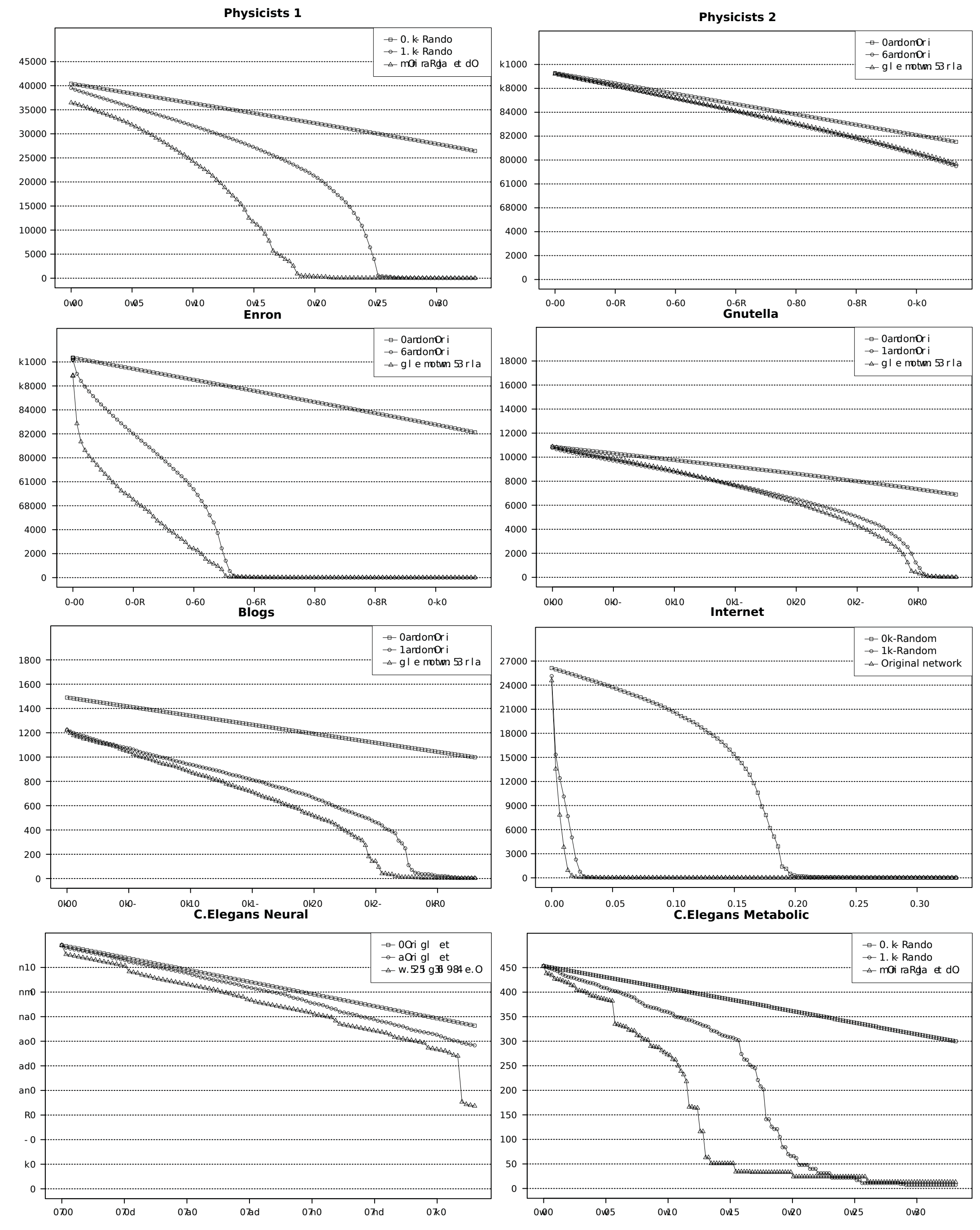

Figure 6.7: Size of the largest connected component in complex networks as vertices are removed in order from highest to lowest degree. The networks used are those listed in Table 2.1. The x-axis shows the proportion of vertices removed, the y-axis shows the number of vertices in the largest component. The experiments were repeated on $0K$ and $1K$ random graphs (as per Section 3.1).

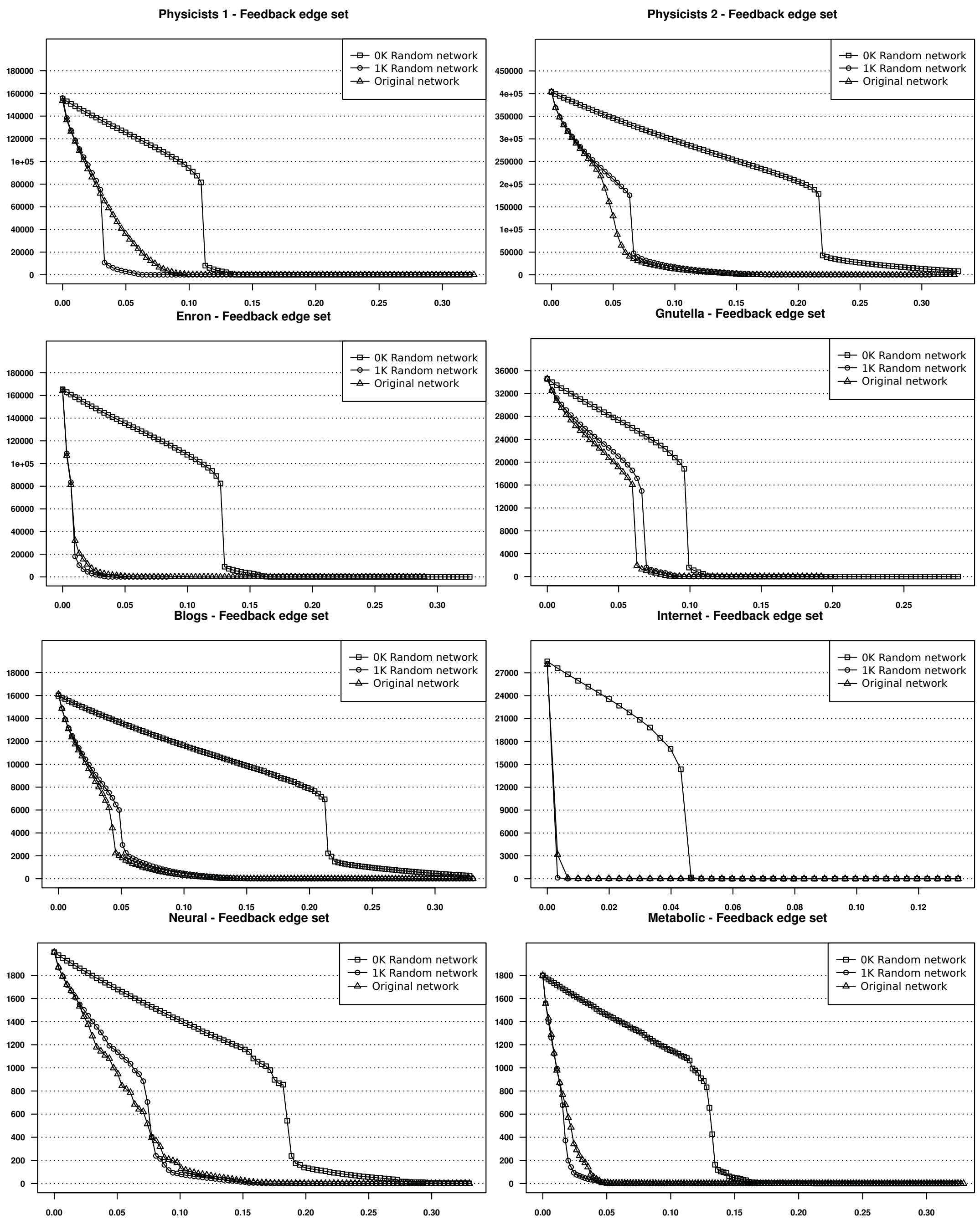

Figure 6.8: Sizes of minimum feedback edge sets in complex networks as hubs and activation sets are removed with majority thresholds ($r = 0.5$). The networks used are those listed in Table 2.1. The x-axis shows the proportion of vertices removed, the y-axis shows the size of the feedback edge set. The experiments were repeated on $0K$ and $1K$ random graphs (as per Section 3.1).

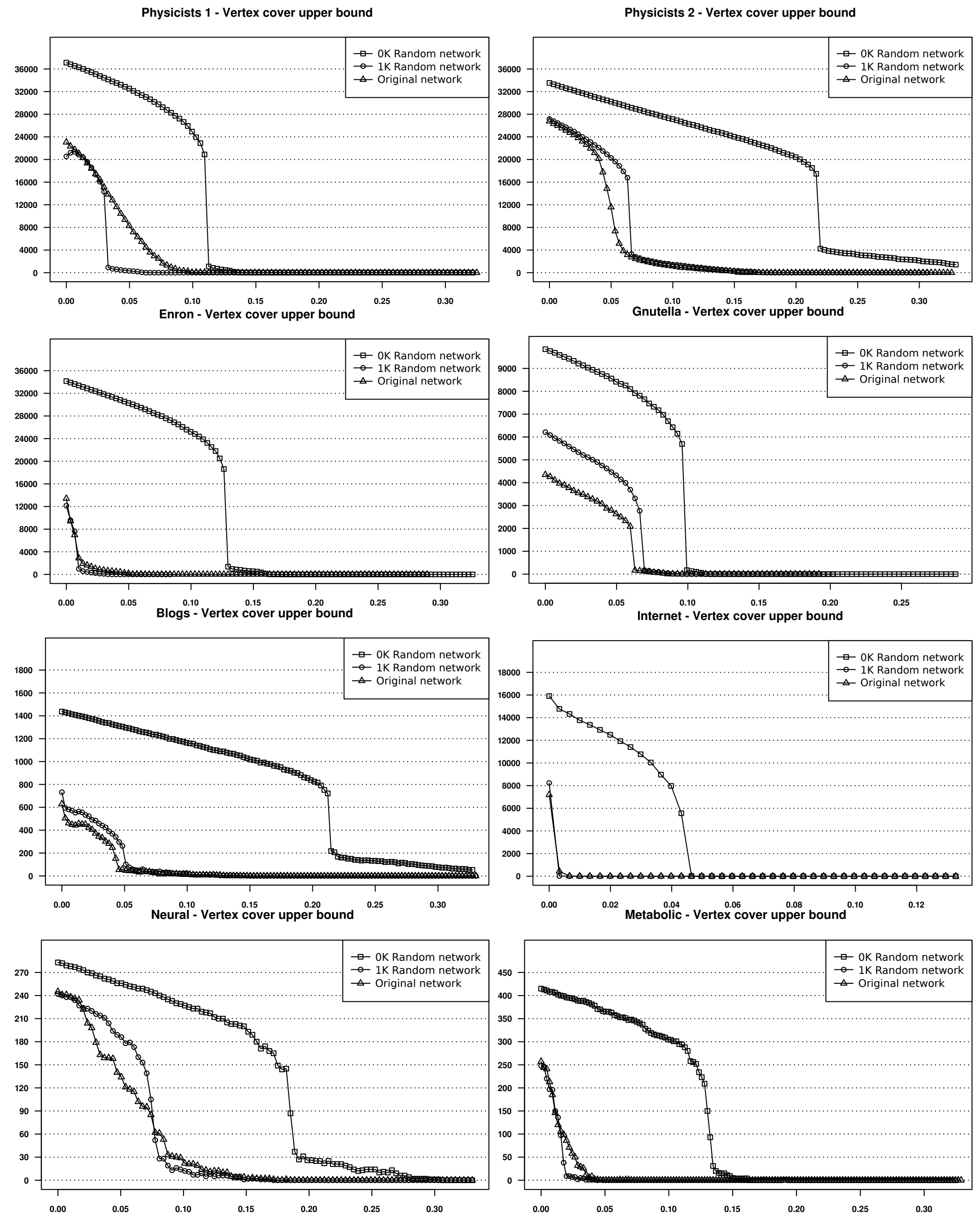

Figure 6.9: Upper bounds for minimum vertex covers in complex networks as hubs and activation sets are removed with majority thresholds ($r = 0.5$). The networks used are those listed in Table 2.1. The x-axis shows the proportion of vertices removed, the y-axis shows the vertex cover upper bound. The experiments were repeated on $0K$ and $1K$ random graphs (as per Section 3.1).

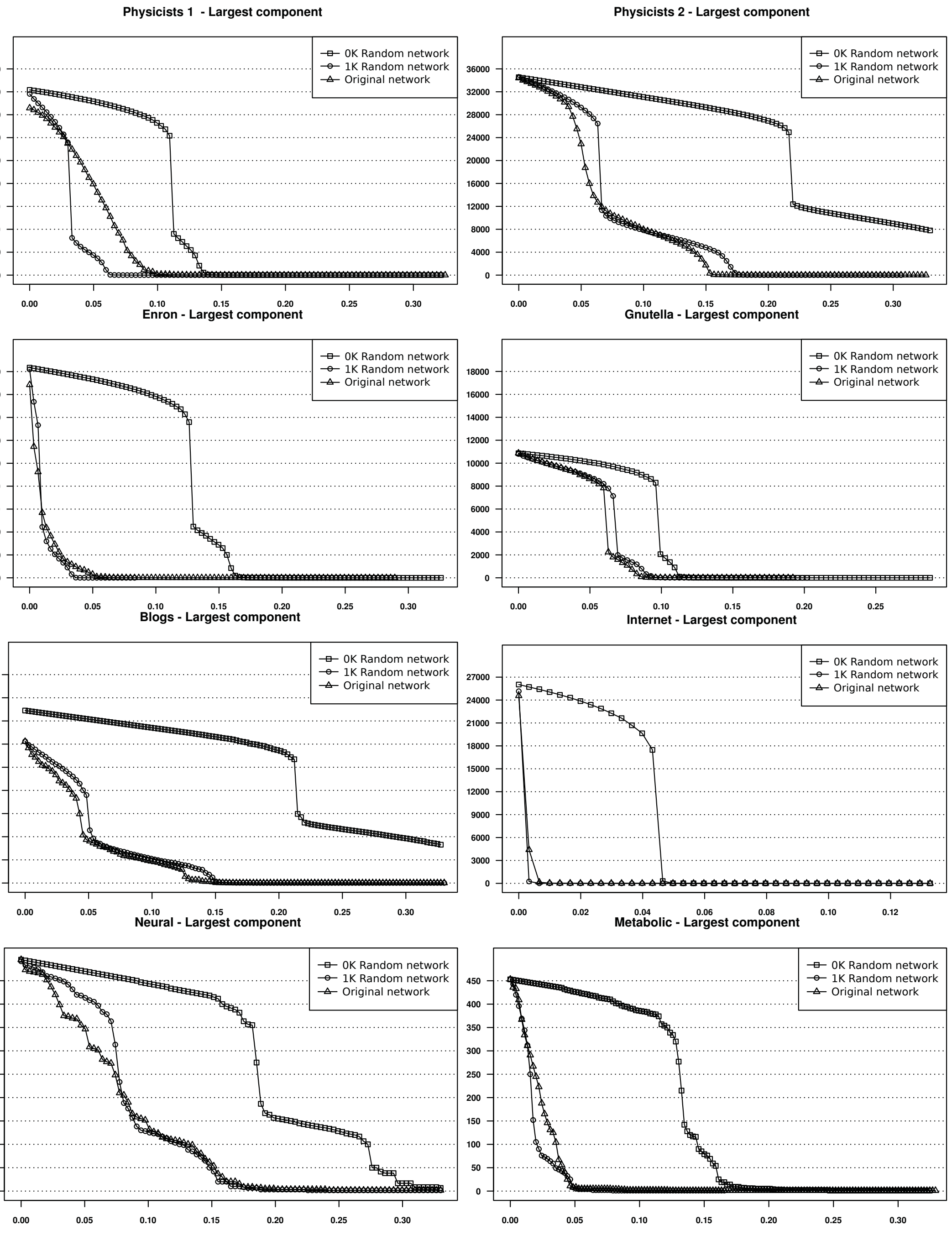

Figure 6.10: Size of the largest connected component in complex networks as hubs and activation sets are removed with majority thresholds ($r = 0.5$). The networks used are those listed in Table 2.1. The x-axis shows the proportion of vertices removed, the y-axis shows the number of vertices in the largest component. The experiments were repeated on $0K$ and $1K$ random graphs (as per Section 3.1).

The results for the experiment removing hubs and activation sets are shown in Figure 6.8 for feedback edge set number, Figure 6.9 for vertex cover number, and Figure 6.10 for the size of the largest component. All of the parameters fall much faster than they did in the original experiment where hubs only were removed. However, it is still true that the feedback edge set number and the size of the largest component become small at about the same point, and so the benefit of using an FPT algorithm over a brute force search remains marginal.

6.4 Summary

This chapter began by comparing three heuristics for minimum target set, known or adapted from the literature: hubs-first, the Shakarian-Paulo-Reichman algorithm, and the marginal-gain algorithm of Kempe et al. (2005). It was found that of these, the hubs-first algorithm has the best overall performance in terms of the size of the target set computed. This suggests that in typical complex networks, the hubs make a good target set for the entire network.

A novel heuristic was introduced in Section 6.2, which was designed to be applied in a distributed manner. A sequential version of the same algorithm was also presented. The distributed algorithm shows superior performance to SPR for high proportional thresholds, and comparable performance for lower thresholds. The running time is also considerably less than SPR in practice because the algorithm finishes after a very small number of rounds (SPR usually takes a number of rounds proportional to the number of vertices in the network, but the distributed algorithm completes within 50 rounds even on networks with tens of thousands of vertices).

Finally, an attempt was made to combine heuristic approaches to minimum target set with the parametrized algorithms discussed in Section 5.3. A combination of the hubs-first heuristic algorithm, and the feedback edge set number and vertex cover number parametrized algorithms was considered. An algorithm based on this hybrid approach would remove high degree vertices from the network, adding them to the target set, until one of the two parameters (feedback edge set number or vertex cover number) became small enough in the remainder of the network to permit an exact solution (to the remainder of the problem) by a parametrized algorithm.

The viability of this approach was tested experimentally by progressively removing hubs from the networks listed in Table 2.1, and measuring the relevant parameters (feedback edge set number and vertex cover number). The size of the largest component was also measured, as this is also expected to decrease as vertices are removed, and if the largest component is small then a brute force solution is practical.

It was found that all the parameters measured fall far more rapidly in the original complex networks than in randomized versions of those networks. However, the feedback edge set numbers and vertex cover numbers do not fall much more rapidly than the size of the largest component. This suggests that the benefit of using a parametrized algorithm for the final phase is marginal, because by the time a parametrized algorithm is practical, brute force is also practical. Hence, no attempt was made to implement the hybrid algorithm.

The next chapter continues to explore the idea of making relatively small modifications to a network, in order to achieve better performance for carefully designed heuristic algorithms. Rather than removing vertices, the following chapter will explore how edges can be *added* to a network to enable the computation of a smaller target set, or a larger activation set (in the case of the maximum activation set problem).

Chapter 7

Shrinking a target set by edge augmentation

The previous chapter considered a hybrid algorithm for minimum target set selection, where part of the problem is solved by a heuristic algorithm which operates by greedily removing high degree vertices (hubs) from the network (the hubs-first heuristic described in Section 6.1.1). It was found that the network parameters discussed in Section 5.3 fall rapidly as hubs are removed. Once these parameters become sufficiently small, the remainder of the problem can then be solved exactly using one of the parametrized algorithms.

One limitation of such an algorithm is that it would not necessarily find a significantly smaller target set than a purely heuristic algorithm. The number of hubs that must be removed to get small parameters is about 5% - 10% of the total number of vertices in the network, a large number considering the sizes of complex networks. Heuristic algorithms such as SPR (Section 6.1.3) can often find target sets of about this size (see Figure 6.1 as evidence), and the optimal vertices to include in a target set are not necessarily the hubs. If a smaller target set is required than what is computed by heuristics, then another approach will be necessary.

This chapter concerns an alternative approach to finding a small target set, in which the network is augmented with new edges to ensure that a small target set does indeed exist. This is done in such a way that the small target set is easily computed. The hope is that it will be possible to add a relatively small number of edges to the network in order to achieve a much smaller target set.

For the kinds of network where the target set problem is most relevant, i.e. social networks and communications networks, edge augmentation strategies could be easily applied (by encouraging new connections between users of an online social network, or by adding new links to a communications network for example).

The edge augmentation approach can be sensibly applied to both the minimum target set selection problem (described in Section 5.1), and the maximum activation set problem (described in Section 5.2). Section 7.1 of this chapter discusses three augmentation strategies for the minimum target set selection problem, and Section 7.2 discusses four augmentation strategies for the maximum activation set problem.

7.1 Augmentation for minimum target set selection

This section proposes three strategies for augmenting networks with new edges, so that the networks have small target sets that can be easily computed. This is done in the context of the minimum target set selection problem explained in Section 5.1, using the "tipping" model introduced in that chapter.

All three of the proposed strategies are based on the SPR (Shakarian-Paulo-Reichman) algorithm described in Section 6.1.3. Recall that the SPR algorithm operates by maintaining $\delta(v) = d(v) - t(v)$ for every vertex v where $d(v)$ is the degree of v and $t(v)$ is the threshold of v. At each step a vertex with minimum, but non-negative, $\delta(v)$ is removed from the network, until every vertex has negative $\delta(v)$. Note that $d(v)$ may change as vertices are removed from the network, but $t(v)$ remains constant. Vertices with negative $\delta(v)$ must be added to the target set because they cannot be activated by their neighbours (which have been removed from the graph).

This suggests a way to achieve a smaller target-set: in addition to removing vertices with small non-negative $\delta(v)$, add edges adjacent to vertices with negative $\delta(v)$. This will allow at least some vertices to maintain a non-negative $\delta(v)$, and thus avoid being placed in the target set. Three variations on this idea are considered:

Local: Following every round of the SPR algorithm, if $\delta(v)$ for any vertex v has become negative, then add edges between v and $NN(v)$ where $NN(v)$ is the set of vertices of distance no greater than 2 from v. Edges are added to v until $\delta(v) = 0$. v is connected to vertices $u \in NN(v)$ in order of increasing $\delta(u)$.

Long-range: As for the *local* strategy, except that v may be connected to any other vertex in the network. v is connected to vertices u in order of increasing $\delta(u)$. Eventually, this process produces a target set no larger than the highest threshold in the network.

Shrink: Run the SPR algorithm until all vertices have negative $\delta(v)$. Then select the vertex v with the largest (closest to 0) $\delta(v)$, and add enough edges adjacent to v to make $\delta(v) = 0$. Remove v from the target set and continue. As with the long-range strategy, this processes produces a target set no larger than the highest threshold in the original network when enough edges are added.

The *local* strategy is designed to model a situation where it is easier to add short-range edges. For example, in a social network it is likely to be easier to encourage edges between vertices that already share a common neighbour, than between arbitrary vertices. Restricting the extra edges to neighbours of neighbours reduces the effectiveness of this edge augmentation strategy, because when $\delta(v) < 0$, there may not be enough eligible edges to prevent v from going into the target set. The size of the effect can be seen by comparing the *local* and *long-range* strategies.

All three strategies place edges where they are most needed to prevent vertices from being forced into the target set, therefore, it is hypothesised that the addition of a relatively small number of edges will cause a large reduction in the size of the computed target set. The *local* strategy is likely to produce the smallest decrease in the size of the target set, since it is severely restricted in what edges it may add to the network. The *shrink* strategy is likely to be the most effective, since each edge added by that strategy contributes to the $\delta(v)$ of two vertices that would otherwise go into the target set. This is not necessarily the case for the long-range and short-range strategies.

7.1.1 Experimental results

The relative performance of the three edge augmentation strategies was compared experimentally. For each of the networks listed in Table 2.1, and for $0K$ and $1K$ randomized versions of those networks, target sets were computed after augmenting edges according to each of the three strategies (local, long-range, and shrink).

For each trial, an upper bound l was imposed on the number of edges that could be added. Then, each of the augmentation strategies was run, adding as many edges as possible up to l, and the size of the target set was computed. The method for computing the target set is based on the Shakarian-Paulo-Reichman algorithm and varies for each of the three strategies as previously described. For each network (including the random networks) 100 trials were performed with l ranging from 0 to m (the number of edges in the network). Each new trial begins with the original (not augmented) network, or in the case of the random networks, a freshly randomized copy of the original network.

To reduce the number of parameters in the experiments, only one threshold assignment was tested. All of the experiments used proportional thresholds with a ratio of 0.5 (i.e. the majority thresholds scheme). Under this scheme, the threshold t of a vertex v is computed as $t(v) = max(1, 0.5 \cdot d(v))$ rounded to the nearest integer, where $d(v)$ is the degree of v.

The results of these experiments are shown in Figure 7.1 for the local strategy, Figure 7.2 for the long range strategy, and Figure 7.3 for the shrink strategy.

Notice that for all the strategies used, the target set falls rapidly to a small (but non-zero) size. This is because under every strategy tested, the number of vertices between

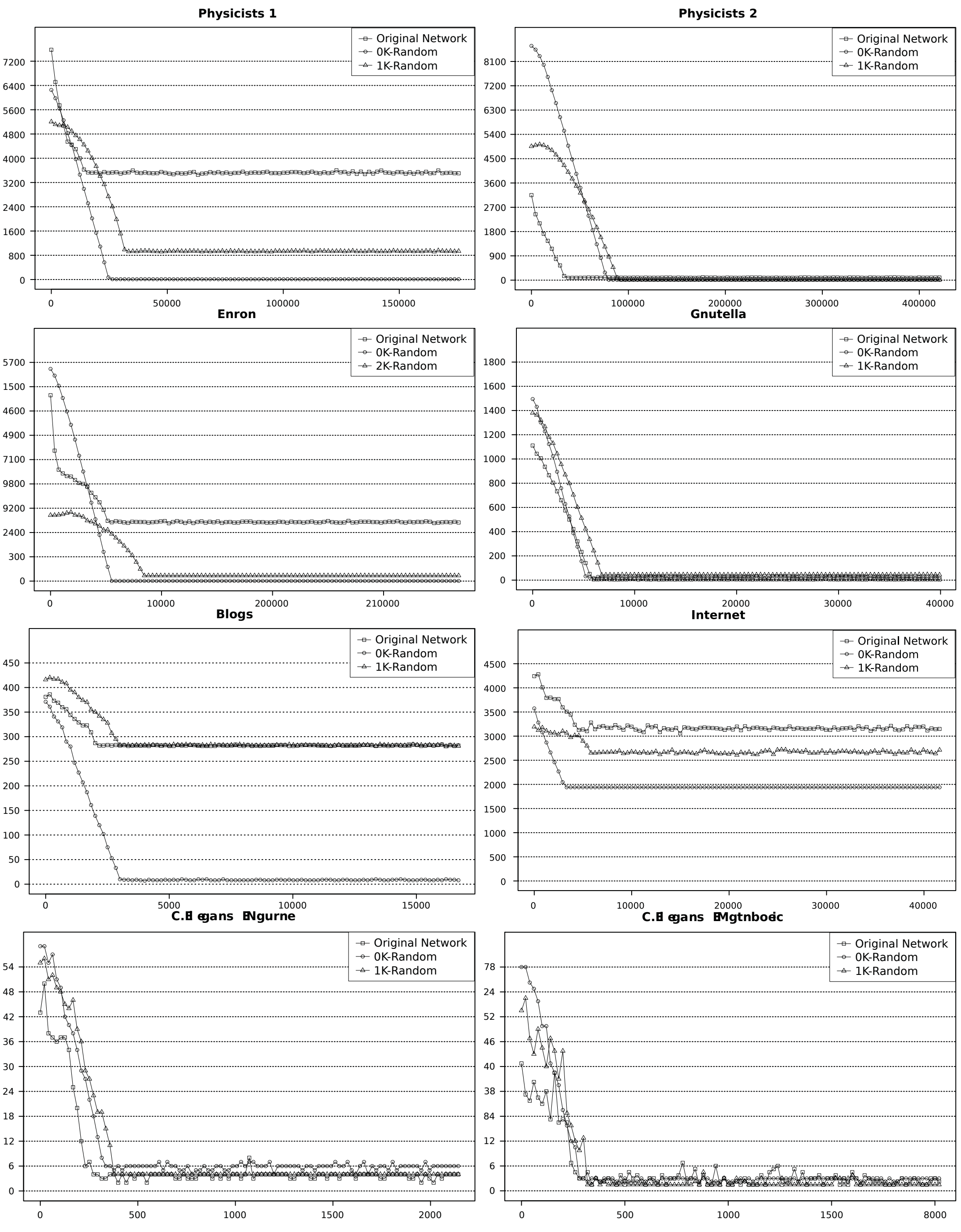

Figure 7.1: Plots of the computed target set sizes when edges are added according to the *local* strategy as described in Section 7.1, on the networks listed in Table 2.1 and random networks as described in Section 3.1. The target sets are computed by the SPR algorithm (Section 6.1.3). The x-axis shows the maximum number of edges that may be added, ranging from 0 to the number of edges in the network. The y axis shows the number of vertices in the computed target set.

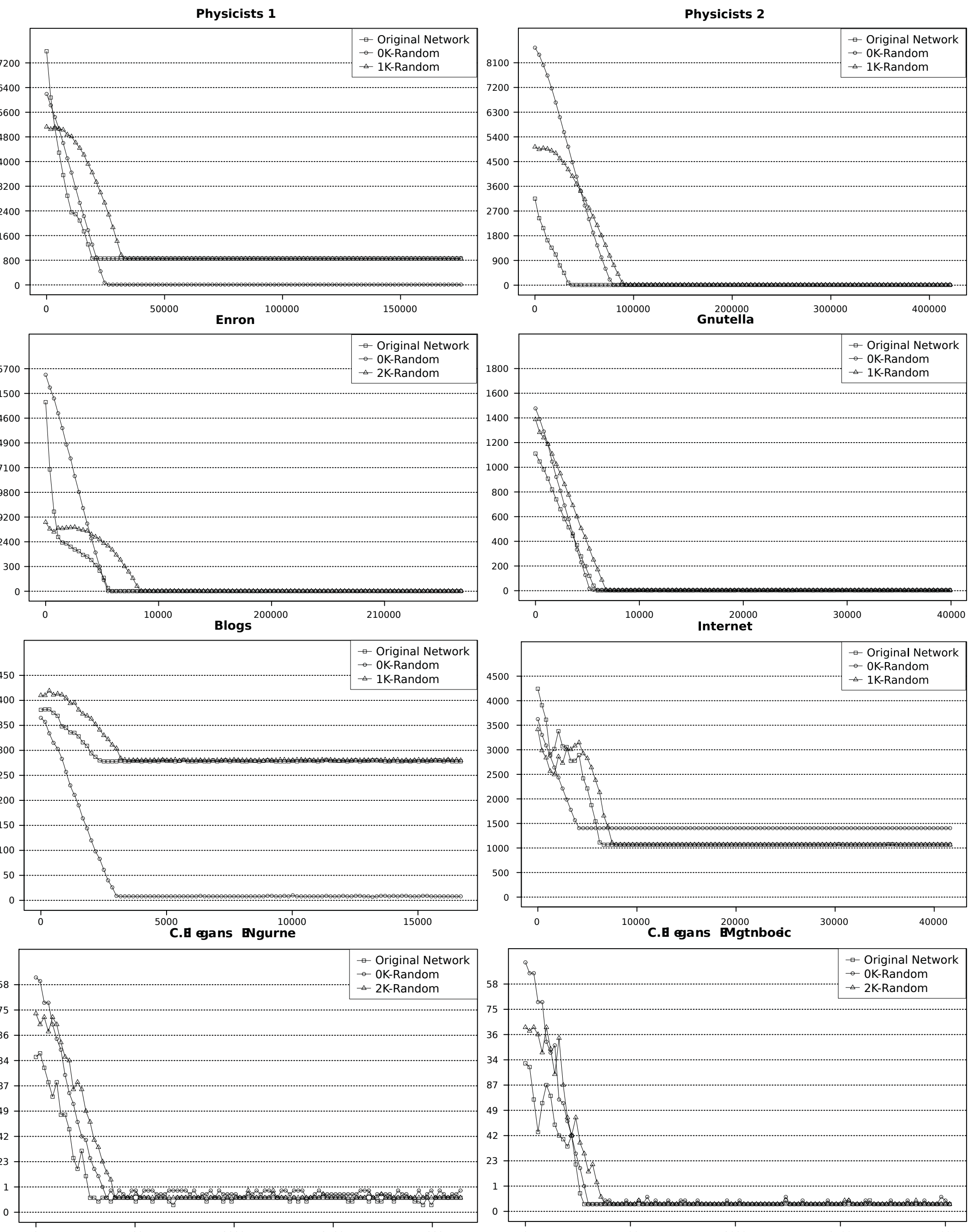

Figure 7.2: Plots of the computed target set sizes when edges are added according to the *long-range* strategy as described in Section 7.1, on the networks listed in Table 2.1 and random networks as described in Section 3.1. The target sets are computed by the SPR algorithm (Section 6.1.3). The x-axis shows the maximum number of edges that may be added, ranging from 0 to the number of edges in the network. The y axis shows the number of vertices in the computed target set.

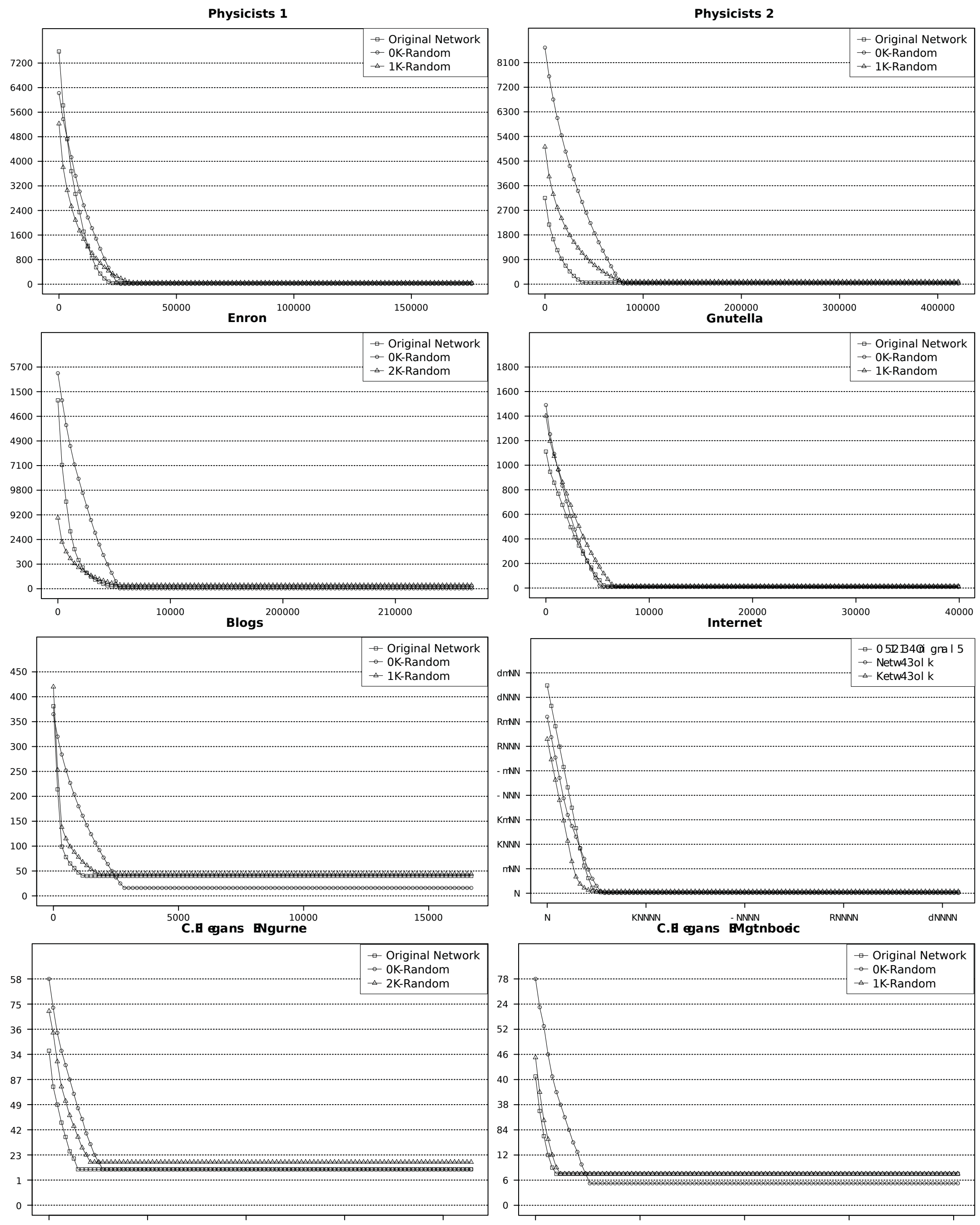

Figure 7.3: Plots of the computed target set sizes when edges are added according to the *shrink* strategy as described in Section 7.1. The networks are those listed in Table 2.1, and the random networks are as described in Section 3.1. The target sets are computed by the SPR algorithm (Section 6.1.3). The x-axis shows the maximum number of edges that may be added, ranging from 0 to the number of edges in the network. The y axis shows the number of vertices in the computed target set.

which edges may be added decreases as the processes continues, and eventually no more edges can be added. For the *local* and *long-range* strategies, edge augmentation is interleaved with the SPR algorithm, which removes vertices from the graph. Hence, the longer the process continues, the fewer vertices are available to connect with new edges. The *shrink* process explicitly removes a vertex from the network after each round of edge augmentation, and again this reduces the pool of new edges that could be added.

In all of the networks tested, and for every strategy used, the target set reaches its smallest size after the number of edges is increased by about 20%. However, in most networks the target set is close to this minimum after an increase in the number of edges by only $5\% - 10\%$. The *local* and *long-range* strategies take slightly longer to reach their minimums than the *shrink* strategy, and the *shrink* strategy generally produces the smallest target set. As seen by comparing Figures 7.1 and 7.2, the *local* strategy is usually significantly outperformed by the other two strategies in terms of the final size of the target set.

For the *shrink* strategy, the algorithm generally performs better on the original networks than on the $0K$ random networks; although the effect is sometimes weak. The behaviour of the $1K$ random networks is distinct from the original networks, and typically sits in between the original networks and the 0K random networks. This implies that there is another factor involved in the rapid decrease in the size of the computed target set, other than the degree distribution, although the degree distribution is clearly relevant.

For the *local* and *long-range* strategies, the rate of decrease of the size of the computed target set as edges are added is essentially the same for the original networks and all the random networks. However, there are significant differences in the sizes of the eventual target sets. This is likely due to the nature of these two algorithms that edges cannot be added adjacent to vertices that have already been removed. The placement of the early augmented edges can therefore have a large effect on the eventual size of the target set.

The placement of the augmented edges is largely determined by degree, hence for the *long-range* strategy the $1K$ random networks show similar behaviour to the original networks, whereas the $0K$ random graphs show different (typically smaller) minimal target sets.

For the *local* strategy, both the $0K$ and $1K$ random networks typically produce smaller final target sets that the original networks. This is most likely due to the *clustering* effect, described in Section 2.2.4. Complex networks are known to have relatively high clustering coefficients, which means that neighbours of neighbours are more likely to be connected than in random networks. This further reduces the number of places where edges may be added under the *local* strategy, leading to much smaller target sets for the randomized networks than for the original networks.

One final irregularity that can be seen in Figures 7.1 and 7.2 is that sometimes the size of the computed target set actually increases when edges are added. This is partly a random effect from using a different random graph for each data point. Notice, however, that the effect also shows up occasionally in the original networks (see for example Figure 7.2, 'C. Elegans Metabolic'). In such cases, the effect may be due to the Shakarian-Paulo-Reichman algorithm effectively choosing vertices based on degree, which is altered when edges are added. This could cause the algorithm to stray further from the optimal path.

7.2 Augmentation for maximum activation set

This section attempts to apply the edge augmentation strategies discussed in the previous section to the maximum activation set problem. Recall from Section 5.2 that the maximum activation set problem is to find a target set of size k (for some fixed k) that activates the maximum number of vertices in the network.

The version of the maximum activation set problem described in Section 5.2 is based on the SIR model (described in that chapter), where instead of every vertex having an activation threshold, there is a global infectivity constant β. At each step of the activation process, every vertex attempts to activate all of it's unactivated neighbours, and succeeds (for each vertex) with probability β.

Since this is a probabilistic process, the quantity to maximise is in fact the *expected* size of the activation set. The only known method for computing the expected size of the activation set $\sigma(A(T))$ from a target set T is by repeated trials of the activation process (Kempe et al., 2005). The expected size of the activation set is the mean of the activation sets computed in all the trials.

In order to compare strategies for edge augmentation, it is necessary to compute an approximately optimal target set in the augmented networks. The optimal approximation algorithm for maximum activation set, given by Kempe et al. (2005), is to add the next vertex that results in the largest increase in the expected size of the activation set. The approximation factor is $1 - 1/e - \epsilon$, where e is the base of the natural logarithm (Kempe et al., 2005). This is an $O(nmkN)$ algorithm, where n is the number of vertices in the network, m is the number of edges, N is the number of trials and k is the maximum size of the target set.

An optimisation of the Kempe et al. (2005) algorithm is offered by Chen et al. (2009). The optimised algorithm has a running time of $O(mkN)$, which is achieved by computing $\sigma(A(T \cup v))$ simultaneously for every vertex v (as opposed to the original algorithm where it is computed per vertex, each vertex costing $O(mkN)$ time to compute). This algorithm is referred to as the *optimised marginal gain algorithm*, and is used throughout the remainder of this chapter to compute the target sets.

The optimisation works as follows: for each trial, every edge in the network is either removed or kept with probability β (the infectivity constant). The activation set from any subset of vertices T can then computed as $\bigcup_{v \in T} C(v)$ where $C(v)$ is the connected component in the derived network that contains v. This is equivalent to computing the activation set from T by simulating the activation process, except that the expensive steps (computing the derived network and finding the connected components) need only be done once per trial.

The augmentation strategies explored in Section 7.1 cannot be directly applied to the maximum activation set problem under the SIR model (since they rely on the Shakarian-Paulo-Reichman algorithm, which is not applicable to the SIR model). Instead, the following modified strategies are used:

Neighbours: Edges are added between the neighbours of high degree vertices. This is similar to the *local* strategy for the minimum target set selection problem (Section 7.1). The justification is that in real-world networks (such as social networks), it is probably easier to add edges between vertices that are already close together (vertices that have short distances between them). High degree vertices are prioritised because they are more likely to become infected (due to their high degree), and so they can be expected to pass on the infection to a significant number of their neighbours.

Distance: Pick a vertex v at random, and compute the distances from v to every other vertex in the network (by breadth first search for example). For all vertices u such that distance(v, u) is maximal, the edge vu is added. The process is repeated with a new random v until the desired number of edges have been added to the network. Here the justification is that long range connections may allow the infection to spread to, and hence to seed parts of the network that would otherwise be too far away from the target set.

For the minimum target set selection problem, the most successful edge augmentation strategy operated by first computing a target set, then using that target set to guide the augmentation process. That strategy (the *shrink* strategy) is not directly applicable to the maximum activation set problem, where the goal is to maximise the activation set rather than to minimise the target set. However, two strategies based on this concept were considered:

Distance from target set: Compute a target set T using the optimised marginal gain algorithm. Use a breadth first search to compute the distances of all the vertices in the network from T. Add edges from randomly selected vertices $v \in T$ to vertices outside

T, in order from the farthest to the closest vertex, choosing randomly when multiple vertices are the same distance from T.

Dead-patches: This strategy attempts to connect the target set to patches of vertices that are otherwise unlikely to be activated (perhaps because they are relatively isolated from the rest of the network). First, the activation probability $p_a(v)$ is computed for every vertex v. This is done by performing a large number of trial activations (using the optimisation of Chen et al. (2009)), and counting how many times each vertex is activated. Then, the following statistic is computed for every vertex v: $D(v) = p_a(v) + \sum_{u \in N(v)} p_a(u)/|N(v)|$ where $N(v)$ is the set of vertices adjacent to v. Edges are added between randomly selected vertices in the target set T and vertices v outside T in order from lowest to highest $D(v)$, in the same fashion as for the *distance from target set* strategy.

For the last two strategies, where a target set is computed first and then used to guide the augmentation process, the target set is not recomputed following the edge augmentation. The goal of these strategies is to increase the expected size of the activation set from the same target set. This differs from the first two strategies, which aim to alter the structure of the network so that a well-chosen target set can achieve a larger activation set than would have been possible in the original network.

Another strategy, which might give better results, would have been to compute a target set using the marginal gain algorithm, then for each edge e compute the expected marginal gain in the activation set from adding e to the network. Then, add edges in order of decreasing marginal gain. This approach was not used because even if the marginal gain from each potential edge e is not recomputed between adding edges, there are still $O(n^2)$ edges that could be added (in a sparse network), and each one requires a large number of trial runs of the $O(n)$ activation process to compute the expected marginal gain. The overall cubic running time is not fast enough for such an algorithm to be practical.

The hypothesis is that adding a relatively small number of well-placed edges to a network will result in a much larger activation set from the same sized target set. This is evaluated experimentally.

7.2.1 Experimental evaluation

The four augmentation strategies for maximum activation set were compared experimentally. These experiments presented two difficulties over the experimental evaluation of the minimum target set selection problem in Section 7.1.1. Firstly, there is an extra variable for the maximum activation set problem: the size of the target set k. Secondly, the optimised marginal gain algorithm that is required to compute the expected sizes of the activation sets has a running time of $O(nNk)$, and is considerably slower in practice than the $O(n^2)$ Shakarian-Paulo-Reichman algorithm that was employed in for minimum target set selection.

These two challenges lead to a simplified experimental design. The infectivity constant β was fixed at a value that would ensure that a reasonably large target set would be required to activate the entire network. If the infectivity constant were excessively high, then there would be little improvement to be made by any augmentation strategy, except for extremely small target sets. Specifically, informal trials were used to choose the value $\beta = 0.05$ as meeting these requirements.

The size of the target set k was allowed to vary from 1 to 50, in increments of 5, so that it is possible to compare the relative effects of adding more vertices to the target set with augmenting edges. For each value of k, and for each of the augmentation strategies, a target set was computed, along with the expected size of the activation set from that target set. The augmentation strategies can then be compared with each other according to how large an activation set they produce.

The number of edges to add was also fixed, at 1000 edges. This number was chosen after preliminary trials suggested that the edge augmentation strategies would not be very effective for maximum activation set. If any of the edge augmentation strategies have the desired property that adding a small number of edges produces a much larger activation set, then adding such a large number of edges should produce a significant increase in the expected size of the activation set, especially on the smaller and sparser networks.

Since the networks used in these experiments (those listed in Table 2.1) cover a range of sizes and densities, the proportional number of edges to add effectively varies across the different networks. If the augmentation strategies behave as did those described in Section 7.1, then there should be a noticeable increase in the expected sizes of the activation sets even on the large networks.

Two additional "augmentation strategies" were included in the experiments as controls. The *none* strategy does not add any edges, it is used as a baseline with which to compare the effects of the other augmentation strategies. It also enables comparisons between the relative effects of augmenting edges and adding more vertices to the target set.

The *random* augmentation strategy adds edges between endpoints chosen uniformly at random (in such a way that the simple graph constraint is maintained). The inclusion of the random strategy allows one to conclude whether or not the other strategies are more or less effective than chance.

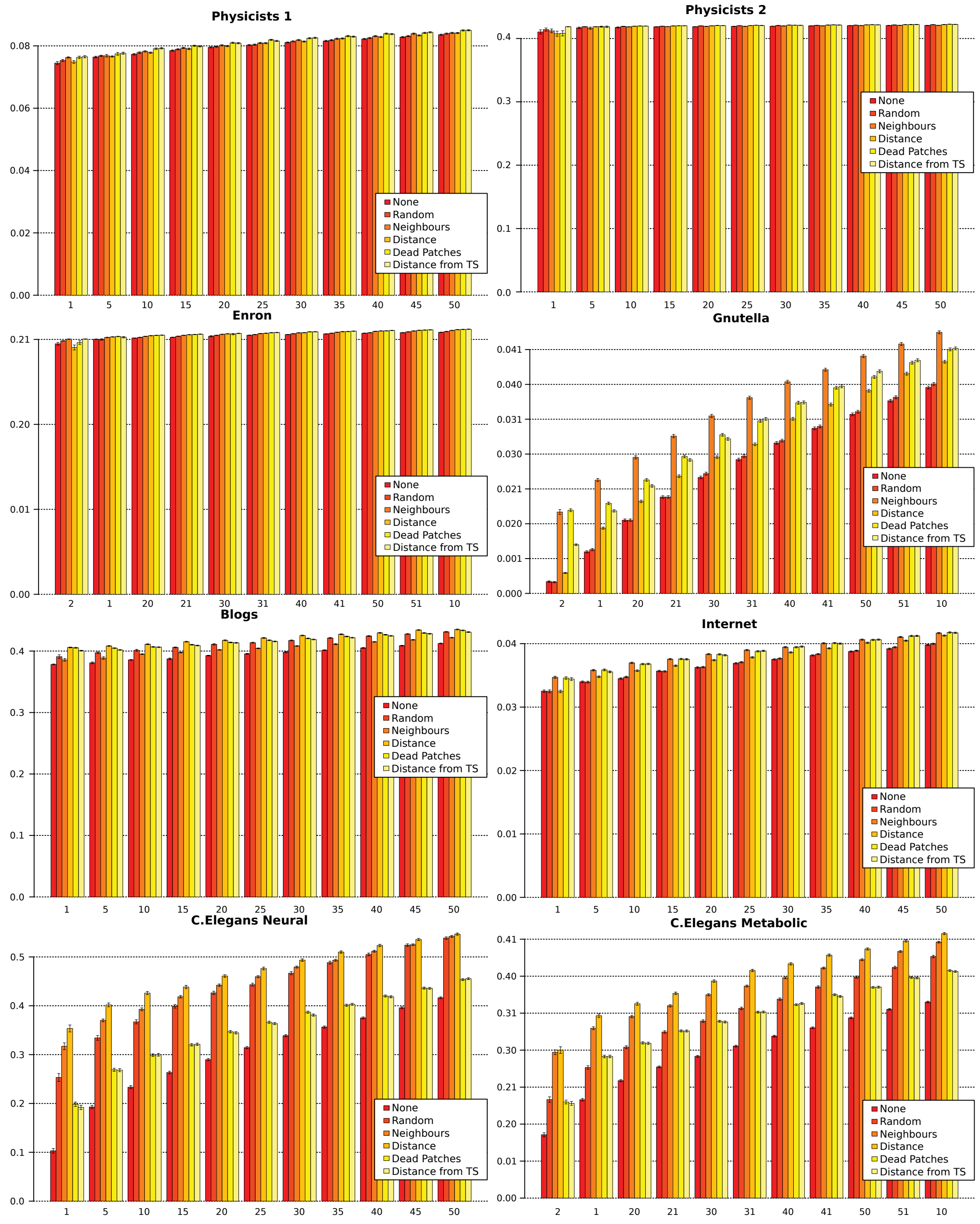

Figure 7.4: Expected size of activation set following several different edge augmentation strategies. The x-axis shows the size of the target set k, the y-axis shows the size of the activation set as a proportion of the vertices in the network. The expected size of the activation set is computed as the mean over 100 trial runs of the activation process, 95% confidence intervals indicate the level of variance over those trials. As explained in Section 7.2.1, the infectivity constant β is fixed at 0.05, and the number of edges to augment is fixed at 1000. The *none* and *random* strategies are controls.

7.2.2 Interpretation of results

The results from the experiments are presented in Figure 7.4. The first thing to note is that the effects of the edge augmentation strategies are related to the sizes and densities of the networks (refer to Table 2.1). This is because the number of edges to augment was fixed at 1000, so relatively more edges are augmented in the smaller and sparser networks.

The *random* strategy is usually outperformed by the distance-based strategies (neighbours, distance, dead-patches, and distance from target set), as expected. This indicates that the effects of those strategies are due to the informed placement of the edges, not merely due to the resulting increase in the density of the network. However, the activation sets following the distance based augmentation strategies are only marginally larger than following the random strategy.

By comparing the distance-based strategies with the "none" strategy for higher values of k (the size of the target set), one can compare the relative effects of adding edges versus adding vertices to the target set. Specifically, Figure 7.4 allows one to compare adding 5 vertices to the target set and adding 1000 edges to the network. Even though such a large number of edges are added, the edge augmentation strategies are only marginally more effective than adding more vertices to the target set.

The benefit of adding edges seems to be independent of the size of the target set. The Gnutella and Neural networks do show an interesting effect where the strategies that augment based on a pre-computed target set (dead-patches and distance from target set) are relatively more effective when k is small. This could be because with a smaller target set, the mean distance from the target set is higher and there are more likely to be "dead-patches" of vertices that don't become activated.

A final observation is that the neighbours strategy, which connects vertices of distance 2, is surprisingly effective. In the Gnutella network, neighbours is the most effective strategy, in the Internet, Enron, and Physicists 1 networks it performs equally as well as the most effective strategies. The only network where the neighbours strategy performs poorly relative to the other strategies is the blogs network. This hints that the clustering properties of complex networks (explained in Section 2.2.4), including social networks, are highly relevant to how well infections spread throughout the network. This is somewhat counter to intuition, which would suggest that it is mainly the short average distances of complex networks that allow a small target set to infect a large proportion of the network.

7.3 Summary

This chapter has explored an approach to the minimum target set selection and maximum activation set problems, where new edges are added to the network to enable the computation of a smaller target set (in the case of the minimum target set selection problem), or a larger activation set (in the case of the maximum activation set problem). The hope was that a much smaller target set (or a much larger activation set) could be achieved by adding a relatively small number of edges to a network. In the case of minimum target set selection, this would be analogous to the results of Section 6.3 where large reductions in important network parameters were achieved by removing relatively small sets of vertices.

For each problem, several different edge augmentation strategies were devised. Local strategies add edges between neighbours of neighbours, increasing the clustering coefficient of the network. This is based on the assumption that such short-range edges may be less expensive to add to a real network (a social network for example). The long-range strategies are permitted to add edges anywhere in the network, and were expected to perform better than the local strategies.

For the minimum target set selection problem (Section 7.1), three strategies were attempted. Both the local and long-range strategies produced a rapid decrease in the size of the target set, as desired. The local strategy was found to have a tendency to get "stuck", whereas the long-range strategy can continue for longer, ultimately producing a smaller target set than the local strategy. The most effective strategy was one where a target set was precomputed, then edges were added within the target set to reduce the number of vertices that must be targeted. This strategy (called *shrink*) was found to produce the fastest decrease in the size of the target set, and the smallest ultimate target set.

The results for the maximum activation set problem (Section 7.2) were less promising. Similar local and long-range strategies to those used for the minimum target set selection problem were employed (called *neighbours* and *distance*). Two additional long-range strategies were attempted (*dead-patches* and *distance from target set*), based on the idea of pre-computing a target set then augmenting the network so that that particular target set activates more of the network. These strategies were found to be generally less effective in increasing the size of the activation set than simply putting more vertices in the target set.

It was not possible to use the same experimental design for the maximum activation set problem as was used for the minimum target set selection problem. This is because maximum activation set has an extra parameter (size of the target set k), and it is more computationally expensive to compute a target set for maximum activation set when using the optimal algorithm. The methodology that was chosen in Section 7.2.1

is limited in that only very limited ranges for values of the parameters could be tested. In particular, the number of edges to augment was fixed at 1000, and the infectivity constant was fixed at $\beta = 0.05$.

Although the experiments for maximum activation set as described in Section 7.2.1 were essentially negative, it is not possible to rule out the possibility that edge augmentation might be effective for different combinations of parameters. However, it is the opinion of the author that the particular choices for the parameters were reasonable, and that it is unlikely that edge augmentation would be effective for this problem (even with higher β or larger target sets). The possibility remains, of course, that a more sophisticated edge augmentation strategy could produce similar results for maximum activation set as were achieved for minimum target set selection.

Chapter 8

Conclusions

This chapter begins with a summary of the work presented in this thesis. This is followed in Section 8.2 with a summary of the main findings of the research. Finally, Section 8.3 considers the limitations of this thesis, and several directions in which the research could be continued.

8.1 Summary

It is well-known that complex networks exhibit many kinds of non-random structure, as was summarised in Chapter 2. This thesis was mainly interested in how that structure can inform the design of algorithms operating on complex networks. A broad range of network datasets were selected, as described in Section 2.1. In order to focus on the structure that is common to complex networks, rather than the specifics of narrow classes of networks, it was decided to project every network to a simple graph. In this way, edge directions, edge weights, and self-loops were ignored, so all the networks could be treated identically.

The large sizes of typical complex network datasets present a major challenge to the design of algorithms for these networks. The networks used in this thesis (listed in Table 2.1) ranged from 297 to $40,421$ vertices. Many complex networks are considerably larger: see for example Ugander et al. (2011), who study the Facebook network which has millions of vertices.

When dealing with such large datasets, it is *not* sufficient for an algorithm to run in polynomial time in order for it to be tractable in a practical sense. For example, Table 2.2 shows the mean distances in the networks described in Section 2.1. To compute this quantity exactly, one must compute the shortest paths between all pairs of vertices, which takes $O(n^3)$ time. This was found to be too slow in practice, and instead a sampling approach was required. Experience with heuristic algorithms for minimum target set and maximum activation set (from Chapters 6 and 7) suggests that

$O(n^2)$ is the slowest running time that can be reasonably tolerated when processing large networks.

Two closely related NP-complete problems, the minimum target set selection problem and the maximum activation set problem were introduced in Chapter 5. These problems were used throughout the remainder of the thesis as a case study for how the structure of complex networks affects the performance of various algorithms.

Throughout Chapters 5 to 7, a variety of approaches to the minimum target set and maximum activation set problems were explored. The relative viability of each approach, and the relative performance of the various algorithms that were devised, was evaluated experimentally. These experiments consisted mainly of running the algorithms on a set of real-world complex network datasets, listed in Table 2.1. In order to ensure that the results were specific to complex network structures, and not merely to chance, it was necessary to include random graphs in the experimental methodology.

Generating random graphs is a surprisingly difficult problem, and this was the subject of Chapter 3. As explained in Section 3.1, the experimental methodology employed in this thesis requires two kinds of random graphs, which are named $0K$ and $1K$ random graphs as per Mahadevan et al. (2006a). According to the definitions used in this thesis, a $0K$ random graph is drawn uniformly at random from the space of all *simple* graphs having a particular size and density. A $1K$ random graph, on the other hand, is drawn uniformly at random from the space of all simple graphs having a particular *degree-sequence.*

The difficult case is that of the $1K$ random graphs. Chapter 3 lists two well-known procedures that generate $0K$ random graphs (The Erdős-Rényi procedure, and the random rewiring procedure). There is also a popular procedure for generating $1K$ random *multigraphs* (which may contain multiple edges and self-loops), the configuration model described in Section 3.2.2. Chapter 3 explored several methods used in the complex networks literature to generate $1K$ random simple graphs, including a generalisation of the configuration model.

It was found that only one method is able to reliably (and efficiently) produce $1K$ random graphs. This being the random degree-preserving rewiring algorithm of Gkantsidis et al. (2003), described in Section 3.3.2. This algorithm relies on a somewhat obscure theorem in graph theory, first proved by Taylor (1980). An alternative proof of the theorem is offered in Chapter 4. The alternative proof operates by directly constructing a sequence of rewiring operations, as opposed to the original proof which merely aims to prove the existence of such a sequence. By directly constructing the sequence of operations, it was possible to derive an upper bound which is probably close to optimal.

8.1.1 Target sets for complex networks

For the minimum target set selection problem, as explained in Section 5.3, there are several known algorithms that can solve this NP-complete problem in polynomial time on limited classes of networks. Specifically, minimum target set selection is *fixed-parameter-tractable* when parametrized by vertex cover number, cluster deletion number, and feedback edge set number (Nichterlein et al., 2010). The presence or absence of these structures (small vertex covers, small cluster deletion sets, and small feedback edge sets) in complex networks has not been well explored in the literature, so an attempt was made to compute the exact values of all these parameters on the complex networks listed in Table 2.1.

Since the computation of vertex cover number and cluster deletion number are both NP-complete, computing exact values for large complex networks required advanced FPT algorithms, as described in Section 5.3. In cases where it was not possible to compute the parameters exactly, upper and lower bounds were computed instead. The results of these computations show that all three parameters (vertex cover number, cluster deletion number, and feedback edge set number) are too large in real-world complex network datasets for the parametrized target set algorithms to apply.

Chapter 6 began an exploration of heuristic algorithms for minimum target set, since minimum target set is inapproximable (Chen, 2009), and the known parametrized algorithms cannot be directly applied. Experimental comparisons were made between an obvious "hubs-first" greedy strategy, a recent heuristic algorithm by Shakarian and Paulo (2012) and Reichman (2012), and a novel heuristic based on the SPR (Shakarian-Paulo-Reichman) algorithm that can be applied in a distributed manner. The distributed heuristic was found to produce smaller target sets than the conventional SPR algorithm for high proportional thresholds, and it was also the fastest of the algorithms even when applied sequentially. However, the hubs-first approach still produced the smallest target sets in most cases, suggesting that in typical complex network datasets, the high degree vertices form a good target set.

An attempt was made in Section 6.3 to combine heuristics with parametrized algorithms. It is well-known that progressively removing the highest degree vertices (the *hubs*) rapidly causes the network to become disconnected. This suggested that the hubs-first greedy heuristic could cause a decrease in some network parameters as hubs are removed. At some point, the parameters become small enough that a parametrized algorithm can be used to compute an exact solution for the remained of the network.

Experimental evaluation of this hypothesis showed that the feedback edge set and vertex cover number parameters fall rapidly as hubs are removed. However, the point at which the parameters become small enough to permit a parametrized solution is close to the point at which the largest connected component becomes small enough to permit a brute force solution. Hence, any benefits of using such a hybrid algorithm would be marginal.

The essential idea of the hybrid algorithm is that, due to the structure inherent in complex networks, it may be possible to make a relatively small modification to the network in order to achieve a large increase in the performance of some algorithm. Chapter 7 explores a further variation on this idea, where instead of removing vertices from the network, edges are added. The goal is to add a relatively small number of edges to a network, in order to achieve a much smaller target set (in the case of the minimum target set selection problem), or a much larger activation set (in the case of the maximum activation set problem).

The experiments reported in Chapter 7 revealed that using an edge augmentation scheme based on the SPR algorithm, it was indeed possible to add a relatively small number of edges to a network in order to achieve a much smaller target set. However, in the case of the maximum activation set problem, none of the edge augmentation schemes tested were found to show this effect.

The results for maximum activation set suggest that adding more vertices to the target set is more effective in increasing the size of the activation set than adding more edges to the network (for the maximum activation set problem). It must be noted however, that the experimental methodology used for maximum activation set was limited, and it may be that the desired effect could be achieved with more thorough experimentation.

8.2 Main findings

The original aims of this thesis, as listed in Section 1.3, were twofold: to attempt to identify previously unnoticed structure in complex networks, and to explore how the structure inherent in complex networks can be leveraged algorithmically.

An attempt was made to find new structure in complex networks that could be exploited algorithmically, but no such structure was be found. In particular, testing on a range of complex networks concluded that most complex networks do not have small vertex covers, small cluster edge deletion sets, or small feedback edge sets, all of which would be very useful properties if they were present.

In order to improve on existing heuristics for the target set problems, an approach was taken in which the network is modified in order to "strengthen" the structure that is already there. This lead to the main finding of this thesis: that is is often possible to make small modifications to the structure of complex networks, in order to achieve large gains in the performance of suitably tuned heuristic algorithms. The author considers this a positive result for the second of the aims of this thesis.

Experimental comparisons with random networks show that the results of this research are specific to complex networks, and must therefore be due to the unique topological features of these networks. Finding an algorithm to generate the random networks, one of the objectives listed in Section 1.3, proved to be more challenging than expected. This work ultimately lead to a novel proof of a theorem concerning the rewiring of graphs, which was presented in Chapter 4.

8.3 Future work

There are several directions in which this work could be continued. One such direction would be to continue searching for previously unnoticed structure in complex networks. However, it is the opinion of the author that one is unlikely to find any stronger structure in complex networks than what is already known. Experience suggests that the main topological structures in complex networks, the heavy-tailed degree distribution and the small-world effect, are only a weak form of structure, and that complex network topologies are in fact more random than the literature would suggest.

An obvious limitation of the experiments presented in this thesis is that they rely on a small sample of only eight network datasets. Relatively speaking, these datasets are of small and medium size. None have more than $50,000$ vertices, whereas the largest complex networks studied to date have millions of vertices. Furthermore, all the network datasets were projected down to simple undirected graphs, ignoring the potentially rich topologies created by directed edges, edge weights, and multiple edges. An important direction for future work, therefore, would be to scale up the experiments with more networks, and larger network datasets. This would make it practical to differentiate between directed and undirected networks, for example, and the results of the experiments may differ significantly between them.

Finally, the goal of this thesis was to find *general* techniques for designing algorithms to operate on complex networks. The algorithms developed, however, are particular to the target set problems which were used as a case study in this thesis. The next step in this research would be to attempt to apply the key concept, of making modifications to a network, to other network problems.

Bibliography

Faisal N. Abu-Khzam, Rebecca L. Collins, Michael R. Fellows, Michael A. Langston, W. Henry Suters, and Christopher T. Symons. Kernelization algorithms for the vertex cover problem: Theory and experiments. *ALENEX/ANALC*, 69, 2004.

Lada A. Adamic and Natalie Glance. The political blogosphere and the 2004 U.S. election: divided they blog. In *Proceedings of the 3rd international workshop on Link discovery*, LinkKDD '05, pages 36–43, New York, NY, USA, 2005. ACM. ISBN 1-59593-215-1. URL `http://doi.acm.org/10.1145/1134271.1134277`.

Lada A. Adamic, Rajan M. Lukose, Amit R. Puniyani, and Bernardo A. Huberman. Search in power-law networks. *Phys. Rev. E*, 64:046135, Sep 2001.

J Ignacio Alvarez-Hamelin, Luca Dall'Asta, Alain Barrat, and Alessandro Vespignani. Large scale networks fingerprinting and visualization using the k-core decomposition. In *Advances in neural information processing systems*, pages 41–50, 2005.

José Ignacio Alvarez-Hamelin, Luca Dall'Asta, Alain Barrat, Alessandro Vespignani, et al. K-core decomposition of internet graphs: hierarchies, self-similarity and measurement biases. *Networks and Heterogeneous Media*, 3, 2008.

Yael Artzy-Randrup and Lewi Stone. Generating uniformly distributed random networks. *Phys. Rev. E*, 72:056708, Nov 2005.

Albert-László Barabási and Réka Albert. Emergence of scaling in random networks. *Science*, 286:509–512, 1999.

Alain Barrat, Marc Barthelemy, and Alessandro Vespignani. *Dynamical processes on complex networks*, volume 1. Cambridge University Press Cambridge, 2008.

Mohsen Bayati, Jeong Kim, and Amin Saberi. A sequential algorithm for generating random graphs. In Moses Charikar, Klaus Jansen, Omer Reingold, and José Rolim, editors, *Approximation, Randomization, and Combinatorial Optimization. Algorithms and Techniques*, volume 4627 of *Lecture Notes in Computer Science*, pages 326–340. Springer Berlin / Heidelberg, 2007.

Oren Ben-Zwi, Danny Hermelin, Daniel Lokshtanov, and Ilan Newman. An exact almost optimal algorithm for target set selection in social networks. In *Proceedings of the 10th ACM conference on Electronic commerce*, EC '09, pages 355–362, New York, NY, USA, 2009. ACM. ISBN 978-1-60558-458-4.

S. Boccaletti, V. Latora, Y. Moreno, M. Chavez, and D. U. Hwang. Complex networks: Structure and dynamics. *Physics Reports*, 424:175–308, 2006. ISSN 0370-1573.

Hans L. Bodlaender and Arie M. C. A. Koster. Combinatorial optimization on graphs of bounded treewidth. *The Computer Journal*, 51:255–269, 2008.

Hans L. Bodlaender and Arie M.C.A. Koster. Treewidth computations I. Upper bounds. *Information and Computation*, 208:259 – 275, 2010. ISSN 0890-5401.

Béla Bollobás and Oliver Riordan. Mathematical results on scale-free random graphs. *Handbook of graphs and networks*, 1:34, 2003.

Ulrik Brandes. A faster algorithm for betweenness centrality. *The Journal of Mathematical Sociology*, 25:163–177, 2001.

Jianer Chen, Iyad Kanj, and Weijia Jia. Vertex cover: Further observations and further improvements. In Peter Widmayer, Gabriele Neyer, and Stephan Eidenbenz, editors, *Graph-Theoretic Concepts in Computer Science*, volume 1665 of *Lecture Notes in Computer Science*, pages 313–324. Springer Berlin / Heidelberg, 1999.

N. Chen. On the approximability of influence in social networks. *SIAM Journal on Discrete Mathematics*, 23:1400–1415, 2009.

Wei Chen, Yajun Wang, and Siyu Yang. Efficient influence maximization in social networks. In *Proceedings of the 15th ACM SIGKDD international conference on Knowledge discovery and data mining*, KDD '09, pages 199–208, New York, NY, USA, 2009. ACM. ISBN 978-1-60558-495-9.

Aaron Clauset, Cristopher Moore, and M. E. J. Newman. Hierarchical structure and the prediction of missing links in networks. *Nature*, 453:98–101, may 2008. ISSN 0028-0836.

Aaron Clauset, Cosma Rohilla Shalizi, and M. E. J. Newman. Power-law distributions in empirical data. *SIAM Review*, 51:661–703, 2009.

Reinhard Diestel. *Graph Theory.* Graduate Texts in Mathematics. Springer-Verlag, New York, 2 edition, 2000.

S. N. Dorogovtsev and J. F. F. Mendes. Evolution of networks with aging of sites. *Phys. Rev. E*, 62:1842–1845, Aug 2000.

S. N. Dorogovtsev, A. V. Goltsev, and J. F. F. Mendes. k-core organization of complex networks. *Phys. Rev. Lett.*, 96:040601, Feb 2006.

S. N. Dorogovtsev, A. V. Goltsev, and J. F. F. Mendes. Critical phenomena in complex networks. *Rev. Mod. Phys.*, 80:1275–1335, Oct 2008.

Rodney G. Downey and Michael R. Fellows. Fundamentals of parameterized complexity. *Undergraduate Texts in Computer Science, Springer-Verlag*, 2012.

Jordi Duch and Alex Arenas. Community detection in complex networks using extremal optimization. *Phys. Rev. E*, 72:027104, Aug 2005. URL `http://link.aps.org/doi/10.1103/PhysRevE.72.027104`.

P. Erdős and A Rényi. On the evolution of random graphs. In *Publication of the Mathematical Institute of the Hungarian Academy of Sciences*, pages 17–61, 1960.

Paul Erdős and T. Gallai. Graphs with prescribed degrees of vertices. *Mat. Lapok*, 11: 264–274, 1960.

Alessandro Ferrante, Gopal Pandurangan, and Kihong Park. On the hardness of optimization in power-law graphs. *Theoretical Computer Science*, 393:220 – 230, 2008. ISSN 0304-3975.

Yong Gao. Treewidth of Erdős-Rényi random graphs, random intersection graphs, and scale-free random graphs. Technical report, University of British Columbia, Aug 2009.

Johannes Gehrke, Paul Ginsparg, and Jon Kleinberg. Overview of the 2003 KDD Cup. *ACM SIGKDD Explorations Newsletter*, 5(2):149–151, 2003.

C. Gkantsidis, M. Mihail, and E. Zegura. The Markov chain simulation method for generating connected power law random graphs. *Proceedings of the Fifth Workshop on Algorithm Engineering and Experiments*, 111:16, 2003.

Kwang-Il Goh, Eulsik Oh, Hawoong Jeong, Byungnam Kahng, and Doochul Kim. Classification of scale-free networks. *Proceedings of the National Academy of Sciences*, 99:12583–12588, 2002.

Jens Gramm, Jiong Guo, Falk Hüffner, and Rolf Niedermeier. Graph-modeled data clustering: Fixed-parameter algorithms for clique generation. In Rossella Petreschi, Giuseppe Persiano, and Riccardo Silvestri, editors, *Algorithms and Complexity*, volume 2653 of *Lecture Notes in Computer Science*, pages 108–119. Springer Berlin Heidelberg, 2003. ISBN 978-3-540-40176-6.

S. Louis Hakimi. On realizability of a set of integers as degrees of the vertices of a linear graph. *Journal of the Society for Industrial & Applied Mathematics*, 10: 496–506, 1962.

Philipp Haller and Martin Odersky. Actors that unify threads and events. In *Coordination Models and Languages*, pages 171–190. Springer, 2007.

Vaclav Havel. A remark on the existence of finite graphs. *Casopis Pest. Mat*, 80:1253, 1955.

Young Hyun, Bradley Huffaker, Dan Andersen, Emile Aben, Matthew Luckie, K. C. Claffy, and Colleen Shannon. The IPv4 Routed /24 AS Links Dataset, 2009. URL `http://www.caida.org/data/active/ipv4_routed_topology_aslinks_dataset.xml`.

David Kempe, Jon Kleinberg, and Éva Tardos. Maximizing the spread of influence through a social network. In *Proceedings of the ninth ACM SIGKDD international conference on Knowledge discovery and data mining*, KDD '03, pages 137–146, New York, NY, USA, 2003. ACM. ISBN 1-58113-737-0.

David Kempe, Jon Kleinberg, and Éva Tardos. Influential nodes in a diffusion model for social networks. In Luís Caires, Giuseppe F. Italiano, Luís Monteiro, Catuscia Palamidessi, and Moti Yung, editors, *Automata, Languages and Programming*, volume 3580 of *Lecture Notes in Computer Science*, pages 1127–1138. Springer Berlin Heidelberg, 2005. ISBN 978-3-540-27580-0.

Maksim Kitsak, Lazaros K. Gallos, Shlomo Havlin, Fredrik Liljeros, Lev Muchnik, H. Eugene Stanley, and Hernán A. Makse. Identification of influential spreaders in complex networks. *Nature Physics*, 6:888–893, 2010.

Jon Kleinberg. Complex networks and decentralized search algorithms. In *Proceedings of the International Congress of Mathematicians (ICM)*, volume 3, pages 1019–1044, 2006.

Bryan Klimt and Yiming Yang. Introducing the Enron corpus. In *First conference on email and anti-spam (CEAS)*, 2004.

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*, KDD '05, pages 177–187, New York, NY, USA, 2005. ACM. ISBN 1-59593-135-X.

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

Jure Leskovec, Kevin J. Lang, Anirban Dasgupta, and Michael W. Mahoney. Community structure in large networks: Natural cluster sizes and the absence of large well-defined clusters. *Internet Mathematics*, 6:29–123, 2009.

Ted G. Lewis. *Network Science*. Wiley, 2009.

L. Li, D. Alderson, R. Tanaka, J. C. Doyle, and W. Willinger. Towards a theory of scale-free graphs: Definition, properties, and implications (extended version). Technical report, California Institute of Technology, Pasadena, CA, USA, Jan 2005.

Priya Mahadevan, Dmitri Krioukov, Kevin Fall, and Amin Vahdat. Systematic topology analysis and generation using degree correlations. *ACM SIGCOMM Computer Communication Review*, 36:135–146, 2006a.

Priya Mahadevan, Dmitri Krioukov, Marina Fomenkov, Xenofontas Dimitropoulos, K. C. Claffy, and Amin Vahdat. The internet as-level topology: three data sources and one definitive metric. *SIGCOMM Comput. Commun. Rev.*, 36:17–26, January 2006b. ISSN 0146-4833.

S. Maslov, K. Sneppen, and A. Zaliznyak. Detection of topological patterns in complex networks: correlation profile of the internet. *Physica A Statistical Mechanics and its Applications*, 333:529–540, feb 2004.

S. Micali and Vijay V. Vazirani. An $O(\sqrt{|V|} \cdot |E|)$ algorithm for finding maximum matching in general graphs. In *21st Annual Symposium on Foundations of Computer Science*, pages 17–27, 1980.

Stanley Milgram. The small world problem. *Psychology today*, 2:60–67, 1967.

Michael Molloy and Bruce Reed. A critical point for random graphs with a given degree sequence. *Random Structures & Algorithms*, 6:161–180, 1995. ISSN 1098-2418.

George L. Nemhauser and Leslie E. Trotter Jr. Vertex packings: structural properties and algorithms. *Mathematical Programming*, 8(1):232–248, 1975.

M. E. J. Newman. The structure of scientific collaboration networks. *Proceedings of the National Academy of Sciences*, 98(2):404–409, 2001. URL `http://www.pnas.org/content/98/2/404.abstract`.

M. E. J. Newman. Assortative mixing in networks. *Phys. Rev. Lett.*, 89:208701, Oct 2002.

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

M. E. J. Newman. Modularity and community structure in networks. *Proceedings of the National Academy of Sciences*, 103:8577–8582, 2006.

M. E. J. Newman and M. Girvan. Finding and evaluating community structure in networks. *Phys. Rev. E*, 69:026113, Feb 2004.

André Nichterlein, Rolf Niedermeier, Johannes Uhlmann, and Mathias Weller. On tractable cases of target set selection. In Otfried Cheong, Kyung-Yong Chwa, and Kunsoo Park, editors, *Algorithms and Computation*, volume 6506 of *Lecture Notes in Computer Science*, pages 378–389. Springer Berlin / Heidelberg, 2010.

André Nichterlein, Rolf Niedermeier, Johannes Uhlmann, and Mathias Weller. On tractable cases of target set selection. *Social Network Analysis and Mining*, 3:233–256, 2013. ISSN 1869-5450.

Rolf Niedermeier and Peter Rossmanith. A general method to speed up fixed-parameter-tractable algorithms. *Information Processing Letters*, 73:125–129, 2000.

Jaideep Ray, Ali Pinar, and C. Seshadhri. Are we there yet? When to stop a Markov chain while generating random graphs. In Anthony Bonato and Jeannette Janssen, editors, *Algorithms and Models for the Web Graph*, volume 7323 of *Lecture Notes in Computer Science*, pages 153–164. Springer Berlin Heidelberg, 2012. ISBN 978-3-642-30540-5.

Daniel Reichman. New bounds for contagious sets. *Discrete Mathematics*, 312:1812 – 1814, 2012. ISSN 0012-365X.

Matei Ripeanu, Ian Foster, and Adriana Iamnitchi. Mapping the gnutella network: Properties of large-scale peer-to-peer systems and implications for system design. *arXiv preprint cs/0209028*, 2002.

Paulo Shakarian and Damon Paulo. Large social networks can be targeted for viral marketing with small seed sets. In *Proceedings of the 2012 International Conference on Advances in Social Networks Analysis and Mining (ASONAM 2012)*, ASONAM '12, pages 1–8, Washington, DC, USA, 2012. IEEE Computer Society. ISBN 978-0-7695-4799-2.

Yilin Shen, Dung T. Nguyen, Ying Xuan, and My T. Thai. New techniques for approximating optimal substructure problems in power-law graphs. *Theoretical Computer Science*, 447:107–119, 2012. ISSN 0304-3975.

Isabelle Stanton and Ali Pinar. Constructing and sampling graphs with a prescribed joint degree distribution. *J. Exp. Algorithmics*, 17:3.5:3.1–3.5:3.25, sep 2012.

Richard Taylor. *Constrained switchings in graphs*, pages 314–336. Number 884 in Lecture Notes in Mathematics. Springer, 1980.

J. Ugander, B. Karrer, L. Backstrom, and C. Marlow. The anatomy of the Facebook social graph. *ArXiv e-prints*, Nov 2011.

Duncan J. Watts and Steven H. Strogatz. Collective dynamics of 'small-world' networks. *Nature*, 393:440–442, 1998.

J. G. White, E. Southgate, J. N. Thompson, and S. Brenner. The structure of the nervous system of the nematode C. Elegans. *Phil. Trans. R. Soc. London.*, 314: 1–340, 1986.

Appendix A

Description of software tools

The highly experimental nature of this thesis necessitated the development of a range of software tools in order to perform computations on large networks, and to coordinate the many computations that comprise each experiment. Due to the relatively large size of the networks considered in this thesis (hundreds to tens of thousands of vertices), efficiency was a major concern in the development of this software. In order to get the best performance out of modern computer architectures, it is essential to perform as many computations as possible in parallel. Thus, the software was architected in such a way as to exploit parallelism, while minimising the burden this inevitably places on the programmer.

Another major concern was the ease with which new experiments could be developed. It was found that the primary difficulty was with new code. Testing new algorithms, or new variations on old algorithms, usually requires new code. New code was found to be prone to failure due to unanticipated corner cases. Since an experiment may take a long time to complete (several days for example), the software was designed so that if an experiment failed, then it could be restarted from the point at which it failed. In this way, bugs could be fixed, and updated code deployed without necessarily having to restart a long experiment from the beginning.

This appendix begins with a high level overview of the architecture of the software that was developed for this thesis (Section A.1). Section A.2 explains the DSL (Domain-Specific Language) that was developed in order to coordinate the various computations that make up each experiment. Section A.3 describes some of the details of how the system was implemented. Finally, Section A.4 describes some limitations of the software, and some directions in which it could be further developed.

A.1 Architecture

The software follows a client-server architecture. The server component is responsible for running the computations that make up each experiment, and collating the results. The client component is a GUI (Graphical User Interface) which can be used to monitor and control the server. The client is connected to the server by a network socket, so the client and the server need not be running on the same physical machine. In fact, the intention is for the server component to run on a remote server, which can then be controlled by the client component.

Each experiment running on the server is represented by a `Job` object. The server creates a new directory for each job, and the outputs from the job are placed in that directory. Each `Job` consists of one or more `Task`s. The `Task`s represent the individual computations that go into each experiment. Computing the average distance of a network, for example, is a single `Task`. The code to perform a task is contained in an object called a `Tool`. The tools are similar in operation to UNIX processes: each tool reads in one or more files, processes the data, and produces one or more output streams which can be redirected to files, or to other `Tool`s.

The server includes a primitive plugin mechanism that permits ordinary OS processes to be used as `Tool`s. Thus, one can define a `Tool` in any programming language. This turned out to be useful for the later experiments in this thesis, where it was necessary to reimplement computationally expensive algorithms in the C programming language in order to achieve maximum performance. It also allowed scripts written in the R statistical programming language to be directly integrated into the experiments, automating the collation and presentation of experimental results.

The server maintains a log of every `Task` that is run. If the server is terminated during an experiment, this log can be used to restart the experiment with the task that was running when the termination occurred. The log also records any errors produced by `Task`s. This is useful for dealing with newly written `Tool`s which often contain bugs that only manifest on large inputs. If an experiment fails due to a buggy `Tool`, then it can be restarted (after deployment of the updated `Tool`), repeating only the parts that failed the last time around. The logging mechanism proved especially helpful for long experiments, which are more likely to fail due to their length and complexity.

Each `Job` on the server is specified using a DSL (explained in Section A.2). This DSL describes how the inputs and outputs of various `Tool`s are connected in order to produce the desired results. The DSL is designed in such a way that the server can deduce the data dependencies between the `Task`s that make up each `Job`. This allows the server to safely run `Task`s in parallel, without requiring any additional effort from the programmer.

A.2 Job specification DSL

The job specification DSL (Domain Specific Language) is used to specify the `Tasks` that make up a `Job`. A more general scripting language, such as UNIX shell scripts, could have been used for this purpose. The advantage of using a custom DSL is that it can be tailored to the task at hand, in this case specifying the `Job` in such a way that the `Tasks` can be run in parallel.

The DSL consists of symbol definitions, tool invocations, and three operators to encode the various ways `Tasks` are allowed to depend on each other. Symbol definitions are one of the following (where *`x`* is a symbol, *`value`* is a number or a string value, *`num`* is an integer or floating point number, and *`values`* is a comma separated list of *`value`*s:

- *`x`* `=` *`value`*
- *`x`* `=` *`num`* `to` *`num`* `by` *`num`*
- *`x`* `=` {*`values`*}

Symbol definition is regarded as a kind of trivial `Task`, with no inputs. Note that the second two symbol definitions define one symbol to have multiple values. In these cases, the symbol definition `Task` has multiple outputs.

Task invocations are as follows (where *`tool`* is the name of a `Tool`, and *`parameter-name`* is the name of a parameter to the `Tool`):

- *`tool parameter-name value, parameter-name value, ...`*

The number of inputs to, and outputs from a tool invocation `Task` depends on the tool. Some have no inputs (such as tools that generate graphs), some have one input, others have multiple inputs (such as tools for data collation).

The operators used to connect the tasks are:

- `|`, the “flat map” operator
- `>>`, `>>>`, the “tie” operators
- `;`, the alternatives operator

The flat map operator (*`task`* `|` *`task`*) gathers all the outputs from the `Task` on the left into a list. The elements of that list become inputs to the `Task` on the right, which is run once for each input in the list. This is called a flat map operation because the outputs from all invocations of the `Task` on the right are collated together into a single list. Thus, Cartesian products can be written like so:

- `x = 1 to 10 by 1 | y = 1 to 10 by 1 | ...`

The tie operator (*task* >> *task*) is similar to the flat map operator, except that it feeds all the outputs from the `Task` on the left into a *single* invocation of the `Task` on the right. This is used mainly for collation.

The alternatives operator (*task*; *task*) performs both the `Task` on left and the `Task` on the right independently of each other. This is used to specify tasks that do not depend on each other (and can therefore be run in parallel).

The scope of the tie operator (`>>`) does not extend over the alternatives operator. This is so that it is easy to define multiple independent sub-jobs ending in a collation step. A variation on the tie operator, `>>>` is used when it is necessary to extend the scope over alternatives. Additionally, parentheses can be used to control the scope of the various operators.

Since `Task`s can only be connected by these three simple operators, it is easy (in theory at least) to deduce which `Task`s may run in parallel, and where it is necessary to add synchronization. For the flat map operator, invocations of the `Task` on the right may begin as soon as outputs become available from the `Task` on left. Each invocation of the `Task` on the right is independent of the others, so they can all run in parallel. The tie operator, on the other hand, inhibits parallelism. The `Task` on the right cannot run until all its inputs are available.

In practice, a great deal of parallelism is possible with this simple DSL. In fact it is similar to, although not directly inspired by, Google's famous map-reduce architecture.

A.2.1 An example

The following is the specification that was used to generate Figure 2.3:

```
data | (
    k = "real" | kcore_distribution;
    k = "1k" | fast-r1k-rewire times 100 | kcore_distribution;
    k = "0k" | fast-r0k-rewire times 100 | kcore_distribution >>>
    lineplot title "k-core Distribution", xlabel "shell", ylabel "size"
)
```

The `data` tool outputs a predefined set of networks (in this case the networks listed in Table 2.1). The `kcore_distribution` tool computes the *k*-core distribution of a network, and the `fast-r`*x*`k-rewire` tools transform networks by randomly rewiring them as described in Chapter 3. The `lineplot` tool produces a line plot from its inputs. This code produces one line plot for each of the original networks, and they are assembled by hand to produce the final figure. (In the final version of this thesis, the lines representing the randomized graphs were removed as they were not sufficiently relevant to Chapter 2).

A.3 Implementation

The Scala programming language was selected for the implementation of the server and client components. A major advantage of the Scala programming language is that it is designed to compile for the JVM (Java Virtual Machine), which greatly eases deployment of the server code to a remote server. Scala is also a higher level programming language than Java (the standard JVM programming language), with a more powerful type system inspired by the functional programming language Ocaml. This makes it more convenient for the programmer, particularly when implementing complex software such as was required for this thesis.

Another useful feature of Scala is its Actor API for concurrency (Haller and Odersky, 2007). The Actor API models a concurrent system with many small objects ("actors") that communicate only be sending messages to each other. It is often easier to reason about actor based algorithms than traditional lock based algorithms. Another benefit is that Scala's implementation of actors does not require an OS thread for each actor, which means that many actors can run simultaneously without consuming too many system resources.

Scala proved to be an ideal language for implementing the `job` specification DSL. The functional style parser combinators library makes writing a parser almost as easy as writing a formal grammar, without the hassle of external tools (i.e. compiler compilers). Interpreting the syntax tree is made convenient by pattern matching. In an imperative style language such as C, or Java, implementing a DSL can be a major undertaking. In Scala, the parsing and interpretation parts are easy. The only difficulty was ensuring the right semantics for concurrency.

Initially, the `Tool` components were also developed in Scala. This presented some challenges. Most of the tools in this thesis implement graph algorithms, and so they require a graph library of some sort. It was found early on that the standard open-source graph libraries for the JVM do not scale past a few hundred vertices. Therefore, a custom graph library was developed. The main challenge is fitting the entire graph into memory, especially since some algorithms require multiple copies of the same graph.

The solution was to use an adjacency list to represent the graph, with a few modifications to improve the asymptotic performance. The standard adjacency list structure consists of a linked list of vertices, and for every vertex, a linked list of the neighbours of that vertex. Therefore, determining if two vertices are adjacent is an $O(n)$ operation. This is much too slow for large graphs.

The modified adjacency list uses a dynamic array to store the vertices, giving constant time vertex lookup, and amortized constant time vertex append. Instead of linked lists to store the vertices, sorted arrays are used. This gives logarithmic lookup time and high compactness, at the expense of making edge insertion more expensive. In practice, though, most vertices have low degree, so edge insertion is not a performance issue. Now it is an $O(\log(n))$ operation to determine if two vertices are adjacent.

This graph library was ported to the C programming language when it became necessary to re-implement the more computationally expensive algorithms in that language. Both the C, and the Scala versions of the library were extensively tested using automated unit tests. For the `Tool` implementations in Scala and C, informal testing was deemed sufficient.

A.4 Limitations and future work

The high-level architecture of this software system is not specific to network analysis. The same DSL and server architecture could easily be applied to other domains involving the transformation and collation of large amounts of data.

The main limitation of the software as it stands is that the concurrency was implemented for a shared memory machine. Modern high performance computing architectures are based on distributed computing, where communication between processes happens over network sockets. The solution to this limitation, clearly, is to migrate away from the Scala Actors API, which does not provide the flexibility required for a distributed environment. The concurrency part of the server could, for example, be reimplemented on top of the industry standard MPI (Message Passing Interface) API.