A Hybrid Approach to Network Robustness Optimization using Edge Rewiring and Edge Addition

James Paterson

Submitted in partial fulfillment

of the requirements for the degree of:

Masters of Science

Department of Computer Science

Faculty of Mathematics and Science

Brock University

St Catharines, Ontario

Abstract

Networks are ubiquitous in the modern world. From computer and telecommunication networks to road networks and power grids, networks make up many crucial pieces of infrastructure that we interact with on a daily basis. These networks can be subjected to damage from many different sources, both random and targeted. If one of these networks receives too much damage, it may be rendered inoperable, which can have disastrous consequences. For this reason, it is in the best interests of those responsible for these networks to ensure that they are highly robust to failure. Since it is not usually feasible to rebuild most existing networks from scratch to make them more resilient, it is necessary to have an approach that can modify an existing network to make it more robust to failure.

Previous work has established several methods of accomplishing this task, including edge rewiring and edge addition. Both of these methods can be very useful for optimizing network robustness, but each comes with its own set of limitations. This thesis proposes a new hybrid approach to network robustness optimization that combines both of these approaches. Four edge rewiring based metaheuristic approaches were modified to incorporate one of three different edge addition strategies. A comparative study was performed on these new hybrid optimizers, comparing them to each other and to the vanilla edge rewiring only approach on both synthetic and real world networks. Experiments showed that this new hybrid approach to network robustness optimization leads to much more highly robust networks than an edge rewiring only approach.

Acknowledgements

I would like to acknowledge and thank the following people and organizations for their help and support:

- My supervisor, *Dr. Beatrice Ombuki-Berman*, for her fantastic assistance and insights. Thank you for your support throughout my research, and for encouraging me to pursue my Masters degree in the first place.

- *Dr. Brian Ross* for his assistance while deciding on my topic, and his detailed feedback on my work.

- The members of my examination committee for reviewing this work and providing their questions and comments.

- My parents, *Steve* and *Erin* for their love and support throughout this whole process.

- Brock University and the Government of Ontario for providing funding to help support my research.

J. P.

Contents

List of Figures

List of Tables

List of Algorithms

1 Introduction

This thesis explores a new hybrid method for optimizing a network's robustness to failure. This hybrid approach is used by a metaheuristic to optimize a target network's robustness. This new approach combines two existing approaches - optimization through edge rewiring and optimization through edge addition - to create highly robust networks.

Many real-world systems can be modelled as networks. From computer networks and power grids to social networks and air travel networks, these networks are found in nearly every aspect of our lives. Many of these networks form critical parts of our infrastructure and would have disastrous consequences if they were rendered nonfunctional. In a poorly designed network, the failure of even a small number of network components can cause huge damage to the overall network structure. A network's resilience to failure is known as "robustness".

Ideally, networks should be designed from the ground up to be robust to failure, and this approach has been examined [1]. However, many existing networks could benefit from enhanced robustness as well. It is possible to make an existing network more robust by making changes to the network structure. Network robustness can be improved by changing the pattern of connections, or edges, within the network graph. These changes can be made by reorganizing the existing edges within a network into a pattern that can more easily withstand damage to the network structure. This is the essence of the problem examined in this thesis - to find a way to modify the structure of a target network in order to make it more robust to failure.

Determining which structural modifications should be made to a network is not a trivial problem. The target network may have many hundreds or thousands of vertices and edges, and determining the exact organization of these components to make a network maximally robust is not feasible. For this reason, we will use metaheuristic optimization to find network structures which, while not strictly the "best" structure possible, are among the best we can find in a reasonable amount of time. While the existing approaches have seen some success, there is still room for further improvement. In order to build off of these previous approaches, it is necessary to know which metaheuristic performs best. For this reason, a comparative study will be performed as part of this thesis.

In [2] and [3], Schneider *et al.* make use of an edge rewiring approach in order to optimize the robustness of scale-free networks, which are networks with power law degree distributions [4]. One goal of their work is to make the networks more robust while maintaining this scale-free property. To this end, Schneider *et al.* make use of an operation known as an "edge swap" which could swap the endpoints of two edges [3]. This operation does not change the number of edges in the network or the degrees of affected vertices, so will not change the degree distribution, thus maintaining the scale-free property of the network. In their work, Schneider *et al.* also introduced a measure known as R value in order to measure the robustness of a network [3]. Using a simple hill climbing algorithm, Schneider *et al.* performed experiments to optimize the robustness of two different types of synthetic network as well as two real world networks [2]. They found that their approach led to networks that were significantly more robust than the starting networks.

Furthermore, Schneider *et al.* observe that the optimized networks exhibited an "onion-like" structure, with a group of highly connected core vertices in the centre surrounded by several shells of lower degree vertices [2]. Each of these shells contain vertices all with the same degree, with degree decreasing with distance from the central core cluster. In networks with this structure, pairs of vertices with the same degree will usually have a path between them that does not pass through any higher degree vertices [2]. This is very useful for defending against targeted attacks on the network that prioritize vertex degree since it ensures that the network still has many viable paths between low degree vertices after the high degree vertices have been removed.

In [5], Buesser *et al.* expand upon Schneider *et al.*'s [2, 3] work, making use of a simulated annealing algorithm for the robustness optimization. Working with the same restriction of maintaining the scale-free property of the networks, Buesser *et al.* also make use of the edge swap operation and the same R value robustness measurement [5]. Their experiments show that the use of simulated annealing leads to significantly higher robustness than the hill climber used in [2]. In [6], Paterson and Ombuki-Berman make use of the great deluge algorithm to optimize network robustness as well as the edge swap operation and R value from [2]. Experiments on synthetic networks of varying sizes show that the great deluge algorithm outperforms simulated annealing in optimizing network robustness [6].

Zhou *et al.* [7] tackle this same problem through the use of a memetic algorithm using a genetic algorithm with a local search. They introduce a novel crossover method specifically designed to preserve the scale-free property of networks in the same way as the edge swap operation from [2]. The local search operator that performs a number of edge swaps in such a way as to promote the onion-like structure found in highly robust networks [7]. This is done by prioritizing swaps that increase the assortativity of the network, a metric that is known to be correlated with the onion-like structure [8]. Experiments with both synthetic and real world networks show the effectiveness of this technique [7].

Other existing work has covered other robustness optimizing algorithms, robustness optimization for other types of networks, differing attack strategies, or other measurements of robustness. Louzada *et al.* [9] and Liang *et al.* [10] examine heuristic "smart rewiring" approaches that prioritize the rewiring of edges to promote certain network properties. Qiu *et al.* optimize robustness in wireless sensor networks using both a heuristic hill climbing algorithm [11] and a memetic algorithm

[12]. In [13], Zeng *et al.* examine networks subjected to attack on both vertices and edges, rather than just vertices. Wang *et al.* consider edge attacks in [14], focusing on networks where damage could result in cascading failures. Zhou *et al.* [15] look at optimizing networks subjected to multiple different targeted attacks strategies at the same time, using an evolutionary algorithm to optimize them. Liu *et al.*'s [16] look at ways to optimize networks for a different type of robustness measurement known as community robustness using a memetic algorithm.

One limit of these techniques is that they are only able to move around the existing edges within the network. This provides an upper limit on the robustness of the network, since even a network that has been made maximally robust through this edge rewiring method can be made even more robust by adding new edges. The addition of new edges to a network can also be used on its own in order to made a network more robust.

Cao *et al.* [17] use an edge addition approach to robustness optimization, focusing on a particular type of network failure that can occur in flow networks called cascading failure. This type of failure happens when the destruction of some part of the network causes the flow within the network to be diverted in such a way that it overloads the capacity of some other section of the network. This causes that secondary section to fail, which makes the flow divert yet again, potentially causing a chain reaction of failures [17]. The authors use three different strategies for determining how to add edges to the network: random linking strategy, high-betweenness linking strategy, and low-polarization linking strategy. Their work does not look at robustness in terms of a measure like the R value. They instead look at how much of the network remains connected after a cascade failure [17]. Of the three linking strategies tested, low-polarization linking gave the best results [17].

Other work on robustness optimization by edge addition includes Zhang *et al.* [18]. They examine a slightly different problem known as the network redundancy problem, where a network is made more robust by adding redundant multiedges to vulnerable points in a network. This is done through the use of a genetic algorithm, measuring robustness with a measure called natural connectivity [18]. Jiang *et al.* [19] use four different edge addition methods to optimize network robustness and Bai *et al.* [20] optimize robustness using an edge adding strategy that takes multiple local and global network properties into account when selecting new edges to add. Ji *et al.* [21] and Kazawa *et al.* [22] look at edge addition based robustness optimization in interdependent and multiplex networks, and Zhang *et al.* [23] examine the problem in wireless sensor networks. Ma *et al.* [24] also use edge addition, optimizing for a robustness measure known as transport efficiency.

These edge addition methods can be useful in improving network robustness, but they do come with a cost. In many real networks, the cost of adding entirely new edges can be significantly greater than moving existing edges [3]. The new connection itself may come with high costs and modifications to the vertices to support the additional connections may be even greater. For example, adding new flights out of an airport may require building additional gates to handle the increased traffic. For this reason, edge addition methods on their own may not always be the more economical way to improve a network's robustness.

Other methods for optimizing network robustness that do not use edge rewiring or edge addition have also been proposed. In [25], Parshani *et al.* look at reducing cascade failures in interdependent networks by, perhaps counterintuitively, removing dependency edges. The removal of these edges would prevent more vertices failing when their dependent vertices failed, lessening the chain reaction effect of cascade failures. Schneider *et al.* [26] examine this approach as well, looking at ways to more carefully select edges for removal to prevent cascade failures while keeping as much of the original network intact as possible. Bernstein *et al.* [27] consider geographically correlated failures in real world power grid networks. They examine ways to limit the effects of cascade failures in these networks by manipulating the supply and demand of vertices in the network during the cascade, ending the cascade early by putting the network back into a stable state. Wu *et al.* [28] have an interesting approach involving hiding information about the network's structure from attackers. Since targeted attacks rely on network properties such as vertex degree, obfuscating the true values of these properties can make it difficult to perform optimal attacks. This approach can make a network more robust against such attacks without making any changes to its structure.

This thesis proposes a novel hybrid approach to network robustness optimization that makes use of a combination of the edge addition and edge rewiring approaches. The goal of this work is to create a method of robustness optimization that retains the cost efficiency of the edge rewiring method, while including some limited amount of edge addition to further enhance the target network's robustness. The target network will have its structure modified through a series of edge swap operations and one of three different edge addition strategies. The proposed hybrid approach is incorporated within the simulated annealing [5], great deluge [6], and memetic algorithm [7] metaheuristics described above. Additionally, it will also be incorporated into a modification of Zhou's [7] memetic algorithm without the local search operator. These metaheuristics will be combined with two of the linking strategies for adding new edges described by Cao *et al.* [17]. A new linking strategy known as the increased assortativity linking strategy is also proposed, based on the findings in [8]. This new hybrid approach is then examined on both synthetic [29] and real world networks [30] to determine which combination of metaheuristic and linking strategy is most useful. An empirical study to compare these various approaches was carried out.

The experiments performed as part of this thesis show that this hybrid approach to network robustness optimization can be very effective. Networks optimized through this process show significantly higher levels of robustness compared to networks optimized using only edge swaps. This higher level of robustness is also sometimes accompanied by a disproportional increase in the number of edges within the network, indicating that this approach may not be the most cost efficient way to increase a network's robustness in some circumstances. This cost efficiency proves to not be an issue for the real world power grid network [30] that was tested, because this network saw a large increase in robustness with a disproportionately small number of new edges. The comparative study shows that there is no definitive best metaheuristic amongst the four methods tested, and different algorithms performed best in different circumstances. Testing also reveals that the newly proposed increased assortativity linking strategy is the best of the three tested methods for performing edge addition.

The remainder of this thesis is structured as follows. Chapter 2 provides background information about complex networks, network robustness, robustness measures, and the metaheuristics used. Chapter 3 describes the edge swap operation and linking strategies, as well as giving implementation details about the metaheuristics. Chapter 4 describes the experimental setup and the data used for the experiments. Chapter 5 shows the results of these experiments and provides discussion about these results. Chapter 6 presents this thesis' conclusions and provides ideas for avenues of future work.

2 Background Information

This chapter presents background information needed to understand the topic of this thesis. It will define several important graph theory terms, explain complex networks, describe how robustness can be measured, and outline the metaheuristics used.

2.1 Graphs

A graph is a mathematical structure representing the relationships between a number of objects. Graphs are commonly used as a mathematical representation of real-world networks. A graph G is defined as $G = (V, E)$ where V is a set of vertices (nodes) and E is a set of edges (connections, links) between vertices [4]. Each edge $e \in E$ is defined as $e = (u, v); u, v \in V$.

Directed/undirected: In a directed graph, edges have a defined direction, indicating a one-way relationship between vertices [4]. For a directed edge $e_{dir} = (u, v)$, the pair of vertices (u, v) is an ordered pair, indicating that there is an edge originated at u and terminating at v. The edges in an undirected graph do not have a defined direction, so all connections in an undirected graph are bidirectional [4]. For an undirected edge $e_{undir} = (u, v)$, the order of vertices in the pair doesn't matter since the edge represents a two-way connection.

Degree: The degree of a vertex in a graph is the number of edges which connect to that vertex [4]. In a directed graph, vertices have both an out-degree and an in-degree. The out-degree is the number of outgoing edges from a vertex, and the in-degree is the number of incoming edges.

Degree distribution: The degree distribution of a graph is a probability distribution of the degrees of a graph's vertices. This distribution shows how often vertices of particular degrees occur within the graph.

Incident: A vertex and edge are said to be incident if the vertex is one of the edge's endpoints [4]. Specifically, for an edge $e = (v_1, v_2)$ and a vertex v, e and v are incident if $v = v_1$ or $v = v_2$.

Path: A path is a sequence of edges $(v_0, v_1), (v_1, v_2), ..., (v_{n-1}, v_n)$ that connect two vertices v_0 and v_n [4]. If a path exists between two vertices it is possible to travel between them by following the edges of the path. Multiple paths can exist between the same two vertices.

Shortest path: The shortest path between two vertices is the path between them with the smallest number of edges. There may be multiple shortest paths if they all contain the same number of edges.

Connected/disconnected: A graph is said to be connected if, for every pair of vertices u and v, there is a path that connects u and v. If this is not the case, the graph is said to be disconnected [4].

Subgraph: A subgraph $G' = (V', E')$ is a graph formed from a subset of the vertex and edge sets of a graph $G = (V, E)$. E' must only include edges whose endpoints are in V' [4].

Induced subgraph: An induced subgraph is a subgraph which includes all edges from the original graph G whose endpoints are present in the subgraph's vertex set V' [4].

Connected component: A connected component is an induced subgraph $G' = (V', E')$ of a graph $G = (V, E)$ which is connected and whose vertices have no edge in E that connects them with a vertex not present in V' [4].

Largest connected component: The largest connected component (LCC) of a graph is the component with the largest number of vertices in its vertex set.

Fully connected: A graph is said to be fully connected if every vertex has an edge to every other vertex [4].

Assortativity: Assortativity is a measure of how frequently vertices in a network connect to vertices with the same or similar degree. A highly assortative network will have a large portion of its vertices connected to other vertices of similar degree [31].

2.2 Complex Networks

Complex networks are a class of network where the structure of the network is neither entirely random or entirely ordered [4]. These networks may exhibit properties such as a high clustering coefficient, a hierarchical structure, and other non-trivial structural properties. Many real-world networks are classified as complex.

The interesting properties present in these networks are often the result of some underlying process within the real-world system the network is modelling [4]. For instance, an internet social network is likely to develop a highly clustered structure. This is because an individual using the network is more likely to add a person as a friend if they already share a friend in common.

2.3 Attacking Networks

Network failure is usually modelled in terms of attacks on the network [2, 5, 6, 7, 32]. These attacks can represent both intentional targeted attacks on the network, as well as the random failure of edges or vertices. In either case, attacks are modelled by selecting an edge or vertex, then removing it from the network. In the case that a vertex is removed, all incident edges are removed along with it. The order in which edge and/or vertices are selected for removal is known as the attack sequence. This thesis will only be considering attacks on the vertices of networks.

In random attacks, the edge or vertex to be removed at any step in the attack sequence is selected in some stochastic manner without any consideration of the structure of the network [32]. This type of attack can be used to model damage to the network caused by unpredictable failures. For example, a storm knocking a tree down onto a power line can be modelled as a random attack on a power grid network.

In a targeted attack, the edge or vertex to be removed is chosen according to some metric with the intention of causing as much damage to the network as possible [32]. These types of attacks can be used to model more intentional, malicious attacks on the network. An example of this type of attack is if a hacker targets an important server to disrupt a computer network.

Targeted attacks can be performed in many different ways, depending on which metric is used to select targets. These metrics are usually some kind of measure of the importance of the edge or vertex, with the most important edges or vertices being prioritized [33]. One simple attack sequence for vertex-based attacks is to simply attack vertices in descending order of their degrees [34].

A common variation of this degree-based attack tactic is known as high degree adaptive (HDA) attack [2]. In the regular degree-based attack, the attack sequence is determined at the beginning of the attack based on the initial degree of the vertices. However, as vertices are removed during the attack, the degrees of the other vertices can change. This means that as the attack progresses, the "most important" vertices can change. In HDA, the selected target vertex is the vertex with the highest degree at the current step of the attack sequence, so the "most important" vertex is always the one selected. HDA is the attack method used throughout this thesis.

2.4 Measuring Robustness

In order to make a network more robust through some optimization process, we first need some way of quantifying how robust a network is to failure. Some possible metrics for network robustness are described below.

2.4.1 R Value

One common robustness measure is the R value [3]. R measure robustness as the average size of the network's largest connected component (LCC) over the course of the attack procedure. R is defined as:

$$R = \frac{1}{N-1} \sum_{Q=0}^{N} s(Q),$$

where $s(Q)$ is the fraction of nodes remaining in the LCC after Q vertices are removed.

In essence, the R value measures how quickly a network breaks apart as it is damaged [2]. This makes it a particularly good robustness measure for any network whose performance is mainly dependent on maintaining connectivity. Low value of R indicate that a network begins to break apart earlier on during the attack sequence or breaks apart more quickly, while higher values indicate that the network remains connected for longer or breaks apart more gradually. The R value can have values between 0 and 0.5, where an R value of 0 indicates a fully disconnected network and an R value of 0.5 indicating a fully connected network [2].

One important characteristic of the R value is its dependence on the attack sequence being used. R doesn't just consider the static structure of the network before any attacks are performed. It measures how the network breaks down as it loses vertices [3]. This means that R actually measures how robust a network is against a particular type of attack. If the attack sequence is changed then the network will break down in a different way, giving a different value for R.

2.4.2 E Value

Another robustness measure that has seen some use is the E value [35]. E is defined as:

$$E = \frac{1}{N(N-1)} \sum_{i \neq j} d_{ij}^{-1},$$

where d_{ij} is the shortest distance between vertices i and j in the network. When there is no path between vertices i and j, d_{ij}^{-1} will be zero.

E measures the efficiency of the communication between any two vertices. Efficiency can be an important measure for networks where the compactness of the network is more important that connectivity on its own. One heavy limitation of this measure is the fact that it is static as it only considers the network at once point in time [6]. This may not give a full picture of the vulnerability of the network. For instance, in a star network, there is a single central vertex to which every other vertex in the network is connected. This type of network is very efficient, since every vertex is at most distance 2 from every other vertex. However, the network is not very robust, since removing that one central vertex will completely disconnect the network.

To that end, a variant of the E value exists, known as $IntE$ [36]. This is defined in a similar method to the R value, measuring the average E value over the course of the attack sequence:

$$IntE = 1/N \sum_{Q=1}^{N} E(\frac{Q}{N})$$

where Q is the number of vertices removed at a particular step in the attack sequence. This variation gives a more accurate picture of a network's robustness, and retains the useful feature of the R value where the attack sequence used affects the robustness score. Similar to R, the $IntE$ value can measure how quickly a network breaks apart. This is due to the fact that disconnected vertices contribute nothing towards the E value for that step. This ability in combined with a measure of the overall compactness of the network through the attack, giving a much more useful robustness measure than the static E value [36].

2.5 Metaheuristics

Optimization is the process of finding the best element of some set. The definition of "best" will vary from problem to problem, but usually involves finding the minimum or maximum possible value for some function. The objective of an optimization problem is to find some state or set of inputs that gives an optimal value to such a function. Each state or set of inputs is known as a candidate solution to the problem, with the set of candidate solutions being known as the search space [37].

A method of exploring the search space is needed in order to discover solutions to these problems. In simple problems, this can be as straightforward as systematically checking each candidate solution within the search space. In more complex problems, the search space may be too large to examine every possible solution in a reasonable amount of time. In this case, we will require the use of a heuristic search procedure. Heuristic searches attempt to find solutions that are better than the initial state while not necessarily being the "best" [37]. These searches will only examine some fraction of the search space and give back a solution that is close to optimal, while not being truly optimal.

Two broad categories of search are local searches and population-based searches. In a local search, the algorithm only examines a single solution at a time, selecting the next solution to examine based on the current selection [37]. The set of solutions that can be reached from the currently examined solution is known as that solution's neighbourhood. The exact specification of the neighbourhood is problem dependent.

Population based searches work much differently. In this type of search, an entire group of candidate solutions, known as the population, is examined at the same time [37]. Exploration is performed in the search space by switching out some or all of this population of solutions, according to algorithm-specific methods.

Several heuristic optimization techniques are presented below. Each is described as a maximization problem, but works equally well for minimization.

2.5.1 Hill Climbing

One simple heuristic optimization technique is known as hill climbing [2, 37]. Hill climbing (HC) is a local search procedure - it only considers a single candidate solution at a time. The hill climbing algorithm starts by examining some initial solution S with a fitness score $f(S)$. The algorithm then considers the neighbourhood of S and selects a neighbouring solution S'. If $f(S') > f(S)$, S' becomes the new S and the algorithm continues to the next iteration. Otherwise, a different neighbour is considered. If all the neighbours of S are checked and none of them are better than S, the algorithm terminates and returns S as the best solution found.

Hill climbing can be effective in convex search spaces, where it is guaranteed to return an optimal solution. In other cases, it can easily get stuck in local optima [5].

2.5.2 Simulated Annealing

Simulated annealing (SA) is another local search procedure that improves upon the functionality of hill climbing [5, 37]. It is inspired by the process of annealing in metallurgy, where a piece of metal is heated up then slowly cooled in order to make it easier to work with. In SA, a new “temperature” parameter T is introduced. This parameter is used to help the algorithm avoid getting stuck in local optima the way hill climbing does [37].

The procedure for simulated annealing is generally the same as hill climbing, with one main exception. In hill climbing, the neighbouring solution S' is only accepted if $f(S') > f(S)$. Simulated annealing however, does not outright reject these worse solutions. The SA algorithm has a chance of accepting worse solutions with a probability proportional to T and inversely proportional to the difference in fitness between the two states [5]. This gives a chance for the algorithm to move out of local optima and have the potential of finding better solutions elsewhere in the search space.

An important part of simulated annealing is that the temperature parameter T changes over time. Similar to metallurgical annealing, the temperature starts of high and is allowed to gradually lower over time. The rate at which T is reduced during the search is governed by the cooling schedule [5]. The variable value of T plays a large part in the function of the SA algorithm. The high value of T early on in the search makes the algorithm more likely to accept locally bad solutions, enabling a higher degree of exploration in the beginning of the search. As T is lowered, the algorithm begins approaching a basic hill climbing search, allowing it to converge to a single solution [5].

2.5.3 Great Deluge

The great deluge algorithm (GD)[38] is a local search technique that takes a different approach to avoiding local optima. This algorithm is inspired by the idea of a person standing on an island that is slowly being submerged by a flood caused by a torrential rain. The person can walk anywhere on the island so long as they stay above the waterline. The GD algorithm works by introducing a new parameter called L, or the "water level". This water level parameter is gradually raised over the course of the search according to another parameter called "rain rate", or ΔL.

GD works a bit differently than hill climbing and simulated annealing. Rather than comparing the fitness of S' to the fitness of S to determine whether to accept the new solution, GD compares $f(S')$ to the water level L. If $f(S') > L$, the solution is accepted [38]. As the search continues and the water level continues to rise, the algorithm gets more picky about what solutions it accepts. Once no neighbour state has a fitness score greater than L, the algorithm is terminated.

2.5.4 Genetic Algorithm

Genetic algorithms (GA) are a type of population based evolutionary algorithm [39]. GAs find solutions to problems by mimicking the process of evolution by natural selection in biology. In biological evolution, each creature has a unique DNA sequence that stores genetic information about the creature. Creatures can pass on their genetic material by producing offspring, and random mutations can introduce variation into DNA. Creatures are also subject to selection pressures from their environment. These selection pressures ensure that individuals which are more fit for their environment are more likely to survive long enough to pass on their genes.

Genetic algorithms make use of a simulated approximation of this process. Candidate solutions in the search space are encoded by a chromosome in imitation of DNA. GA chromosomes are often just lists of numbers which can be used to reconstruct the solution, but other options are possible as well. The "genetic information" of individual candidate solutions within a GA population can be combined together to produce offspring solutions using a process known as crossover. Variation can also be introduced to chromosomes via a mutation operator. There are many different crossover and mutation methods, many of which are problem specific. Selection pressure is introduced through a selection operator. The selection operator determines which candidate solutions in the population will be used for crossover and mutation, and will favour those with higher fitness values.

2.5.5 Memetic Algorithm

Memetic algorithms (MAs) are variation of GAs inspired by Dawkin's idea of a meme [40]. Memes are units of cultural information, and are often described as the cultural equivalent to genes. Both memes and genes represent discrete pieces of information, and can be modified, combined, and passed down through a population. The main differences between the two is in the type of information they describe and the means by which they are transmitted. Memes can be though of as ideas, and are transmitted through speech, text, or other means of human communication. Genes, on the other hand, are information that describe how to construct cells and regulate biological processes, and are transmitted through biological reproduction. One other important difference between genes and memes is that, in nature, genes are not modified in any intentional way. The genetic information is only changed through random mutation and through random recombination with other individuals. Memes, on the other hand, are able to be deliberately refined and modified by the individuals that pass them on [41].

In order to mimic this process, memetic algorithms combine a genetic algorithm with a heuristic local search procedure [41]. This allows them to augment the evolutionary process of the GA with a method of locally refining individuals during the evolution process.

3 Metaheuristics for Robustness Optimization

This chapter provides implementation details for the four different metaheuristics used and the network operations used by these algorithms.

3.1 Network Operations

All four metaheuristics make use of two main network operations - edge swap and edge addition. This subsection describes the edge swap operation and three different methods of edge addition.

3.1.1 Edge Swap

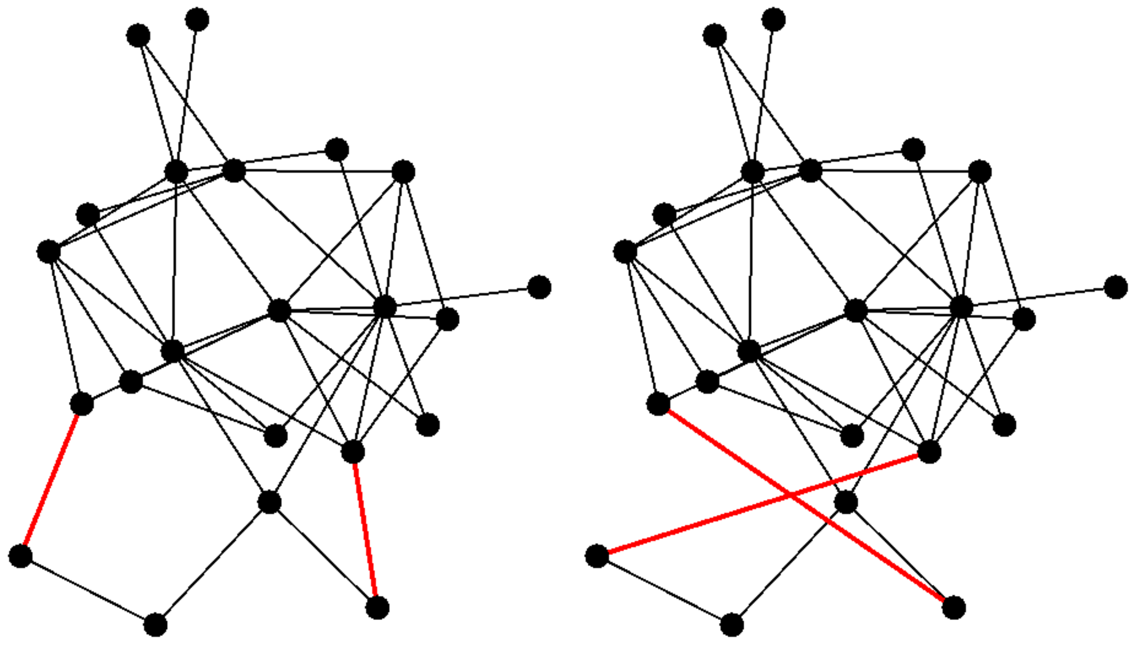


Figure 1: Example of edge swap operation

The edge swap operation [3] can be used to make small modifications to a network as a means to explore this problem's search space. An edge swap involves selecting two edges from the network, removing them, and adding two new edges with the original edges' endpoints swapped. If we select edges (v_1, v_2) and (v_3, v_4), we will remove these edges from the network and add in two new edges: (v_1, v_4) and (v_2, v_3) [3]. A visual example of this operation can be seen in Figure 1 and the algorithm is detailed in Algorithm 1.

Algorithm 1 Edge swap algorithm

```
function EdgeSwap($g$)
    $Candidates_{e1} \leftarrow E$

    repeat
        $e_1 \leftarrow$ RandomElement($Candidates_{e1}$)                    ▷ $e_1 = (v_1, v_2)$

        $Candidates_{e2} \leftarrow E$
        $Candidates_{e2} \leftarrow Candidates_{e2} - e_1$

        for all $e'_2$ in $Candidates_{e2}$ do                             ▷ $e'_2 = (v_3, v_4)$
            if SharesEndpoint($e_1, e'_2$) then
                $Candidates_{e2} \leftarrow Candidates_{e2} - e'_2$
            end if

            if EdgeExists($v_1, v_4$) or EdgeExists($v_2, v_3$) then
                $Candidates_{e2} \leftarrow Candidates_{e2} - e'_2$
            end if
        end for

        if !IsEmpty($Candidates_{e2}$) then
            $e_2 \leftarrow$ RandomElement($Candidates_{e2}$)
            return $e_1, e_2$
        else
            $Candidates_{e1} \leftarrow Candidates_{e1} - e_1$
        end if
    until IsEmpty($Candidates_{e1}$)

    return $Null$                                                ▷ No valid edge swap
end function
```

The selection of edges for this operation should be done carefully. If both edges share an endpoint, then the swap can result in a vertex with a self-referential loop. This is undesirable in many contexts. Additionally, if the edges (v_1, v_4) and/or (v_2, v_3) already exist, performing this operation can result in the formation of a multi-edge, which is also usually undesirable. The algorithm described in Algorithm 1 avoids these problems by excluding edges from consideration if swapping them would violate one of these conditions.

An important property of the edge swap is that is preserves the degree of all affected nodes [3]. This can be important if preserving the network's overall degree distribution is required, as in [2, 5, 6, 7].

For the purposes of this problem, we explore the search space by means of random edge swaps.

3.1.2 Edge Addition

The edge addition operation can be used to add brand new edges into a network. An important component of performing edge addition is the selection of which new edge should be added to the network. If the edge is added in the right spot, it can lead to a more robust network. Adding a new edge will never decrease the robustness of a network, but adding one in the wrong spot can result in no change to robustness.

Another way to explore the problem's search space is by adding brand new edges into the network. In this thesis, edge addition will be used in concert with the edge swaps described in Section 3.1.1. At each step in an optimization algorithm, we will decide, at random, whether to perform an edge swap or add a new edge. The probability of performing an edge addition instead of a swap is governed by the parameter p_{add}, which can have any value between 0 and 1.

Different strategies for edge addition can be used. This thesis makes use of the random linking strategy and high betweenness linking strategy from [17] and proposes a new linking strategy called increased assortativity strategy. The low polarization linking strategy from [17] is not used due to slow performance in preliminary tests.

Random Linking Strategy Random linking strategy (RLS)[17] is the simplest method of edge addition. This strategy involves selecting a pair of vertices with no existing edge between them via uniform random selection, then adding that edge to the network.

High Betweenness Linking Strategy The high betweenness linking strategy (HBS)[17] prioritizes connecting vertices which have a high betweenness value. Betweenness measures a vertex's importance within a network by measuring the number of paths that connect through it. In this thesis, a vertex is considered to have "high betweenness" if its betweenness score is greater than the average betweenness of all vertices in the network. This process selects from among these high betweenness vertices uniformly randomly.

Increased Assortativity Linking Strategy This thesis proposes a new edge addition heuristic known as increased assortativity linking strategy, which prioritizes the increase of this value with each new edge created. This strategy is motivated by the high robustness "onion-like" networks described by Schneider *et al.* [2]. These onion-like networks consist of shells of vertices of decreasing degree, with a high level of inter-connectivity within each layer. Tanizawa *et al.* [8] showed that highly assortative networks may have some of these same properties. The hope is that prioritizing new connections that increase the assortativity of a network, we can more easily achieve a high robustness network.

For each iteration of IAS, we will only form a connection between two vertices if the difference in their current degrees is less than or equal to the threshold parameter δ_d.

3.2 Metaheuristics Implementation Details

This section describes the four different metaheuristics examined in this thesis. Pseudocode and implementation details are provided.

Algorithm 2 Simulated annealing algorithm implementation

```
function SAOptimize($g_{init}$)
    $g* \leftarrow g_{init}$
    $T \leftarrow T_{max}$

    repeat
        $NoImprove \leftarrow 0$

        repeat
            $x \leftarrow$ Random(0,1)
            if $x < p_{add}$ then
                $g \leftarrow$ EdgeAdd($g*$)
            else
                $g \leftarrow$ EdgeSwap($g*$)
            end if

            if R($g$) > R($g*$) then
                $g* \leftarrow g$
            else
                $y \leftarrow$ Random(0,1)
                if $y < \text{EXP}(-\Delta R/T)$ then
                    $g* \leftarrow g$
                end if
            end if

            if $LastR <$ R($g*$) then
                $LastR \leftarrow$ R($g*$)
                $NoImprove \leftarrow 0$
            else
                $NoImprove++$
            end if
        until $NoImprove > MaxNoImprove$

        $T \leftarrow T * (1 - T_{decay})$
    until $T < T_{max}$
    return $g*$
end function
```

The general approach of each of these optimizers involves making a number of edge swaps and edge additions to a target network, using the metaheuristic to determine where and when these operations should be performed. This is equivalent to a search through a search space of network structures, with the neighbourhood of any given state defined as the set of networks that can be created by performing one edge swap or edge addition. The fitness value used for this search is the R value [3] robustness measure. By the end of the optimization, the result network should will be similar to, but more robust than, the initial target network.

3.2.1 Simulated Annealing

In the simulated annealing algorithm, we perform a number of iterations of search space exploration. At each iteration, we select randomly between performing an edge swap and an edge addition, as defined by the p_{add} parameter. If the swap or addition improves the robustness of the network, we keep the change. If it does not, we have a probability of keeping the change equal to:

$$p = e^{\frac{R_{new} - R_{old}}{T}}$$

where R_{old} is the R value before the swap or addition, R_{new} is the R value after the swap or addition, and T is the temperature parameter [5].

T is initialized to the value of the parameter T_{max}. Every time a change is made that improves the network's robustness, the temperature parameter T is modified using:

$$T_{i+1} = T_i * (1 - T_{decay})$$

where T_{decay} is the temperature decay parameter. Once T has been reduced below the minimum temperature T_{min}, the process terminates. The process also terminates if a sufficient number of iterations passes in a row without seeing improvements to the robustness value.

Algorithm 2 shows how this algorithm is implemented for this problem. We perform a number of iterations, decreasing T whenever too many consecutive iterations pass without seeing improvement to the network's R value. In each iteration, we randomly select between performing an edge addition or an edge swap then compare the R value of the network after the modification to the R value before the modification. If the modification led to a more robust network, the modification is kept. Otherwise, we have a chance to accept the modification based on the probability described above, otherwise the modification is not carried forward to the next iteration.

3.2.2 Great Deluge

In the great deluge algorithm, network modifications are accepted if they result in a network with a robustness value greater than a threshold value L, also called the water level [38]. Each iteration of the algorithm in which the modification is accepted, this threshold is increased by the rain rate, Δ_L. The threshold and the rain rate are initialized to a percentage of the initial network's robustness value. This percentage is controlled by the parameters $L_{\%}$ (for the threshold) and $\Delta_{L\%}$ (for the rain rate). The process terminates when a sufficient number of iterations passes in a row without finding a network structure with a robustness value greater than the threshold.

Algorithm 3 shows how this optimization works. We perform a number of iterations where we randomly select between performing an edge addition or an edge swap. We then compare the R value of the modified network to the current threshold L, retaining the modification if the R value is above this threshold. If too many iterations pass without finding a network with an R value greater than L, the algorithm is terminated.

Algorithm 3 Great deluge algorithm implementation

```
function GDOptimize($g_{init}$)
    $g* \leftarrow g_{init}$
    $L \leftarrow L_{\%} *$ R($g*$)
    $\Delta_L \leftarrow \Delta_{L\%} *$ R($g*$)
    $NoImprove \leftarrow 0$

    repeat
        $x \leftarrow$ Random(0,1)
        if $x < p_{add}$ then
            $g \leftarrow$ EdgeAdd($g*$)
        else
            $g \leftarrow$ EdgeSwap($g*$)
        end if

        if R($g$) $> L$ then
            $g* \leftarrow g$
            $L \leftarrow L + \Delta_L$
            $NoImprove \leftarrow 0$
        else
            $NoImprove + +$
        end if
    until $NoImprove > MaxNoImprove$
    return $g*$
end function
```

3.2.3 Genetic Algorithm

The genetic algorithm uses a population based evolutionary algorithm to optimize a network's robustness. The representation used for this GA is simply the graph of the network [7]. A numerical encoding of the network is unnecessary since the crossover and mutation operations (as well as the local search operator in the memetic algorithm variation) are executed on the network structure itself. An additional encoding/decoding step would be redundant.

The GA's population is initialized by creating copies of the initial graph and then performing a number of random edge swap operations on each copy [7]. These swaps are made totally randomly, without regard for whether they increase or decrease the network's robustness. This gives us our initial population.

The crossover and mutation operators are described below, as is the memetic algorithm variation of this GA and its local search operator.

The full GA algorithm is given in Algorithm 4. We begin by initializing the population, then evolve this population through a fixed number of generations. In each generation, we begin by selecting two parent chromosomes for the crossover operator using tournament selection. In tournament selection, k chromosomes are chosen from the population and the most fit chromosome from this group of k is selected. We then have a chance of performing crossover with these parents, based on the crossover rate. If crossover is performed, the newly produced child chromosomes are added to a new population set. Otherwise, the selected parent chromosomes are added to this set. Once the new population set is full, each chromosome has a chance of mutation being applied to it. If mutation is applied to a chromosome, we randomly select between performing an edge addition or performing the regular mutation operator. Finally, the new population is set as the current population and we continue to the next generation.

Crossover The crossover operation used in this algorithm is a custom crossover devised by Zhou *et al.* [7]. Since they were working with scale-free networks, the operator was designed in such a way as to preserve the degree distribution of the child networks, similar to the edge swap operation.

It begins by copying the parent chromosomes, G_{p1} and G_{p2}, into the child chromosomes G_{c1} and G_{c2}. We then obtain the following sets of vertices:

$V_i^{G_{c1}} = \{j|e_{ij} \in E^{G_{c1}}\}$

$V_i^{G_{c2}} = \{j|e_{ij} \in E^{G_{c2}}\}$

$\bar{V}_i^{G_{c1}} = V_i^{G_{c1}} - (V_i^{G_{c1}} \cap V_i^{G_{c2}})$

$\bar{V}_i^{G_{c2}} = V_i^{G_{c2}} - (V_i^{G_{c1}} \cap V_i^{G_{c2}})$

Algorithm 4 Genetic algorithm implementation

```
function GAOptimize($g_{init}$)
    $Pop \leftarrow$ PopInit($g_{init}$)
    $Gen \leftarrow 1$

    repeat
        $NewPop \leftarrow \emptyset$

        repeat
            $g_{p1}, g_{p2} \leftarrow$ TournamentSelection($k$)
            $g_{c1}, g_{c2} \leftarrow$ Crossover($g_{p1}, g_{p2}, CrossRate$)
            $NewPop \leftarrow NewPop + g_{c1}, g_{c2}$
        until IsFull($NewPop$)

        for all $g$ in $NewPop$ do
            $x \leftarrow$ Random(0,1)
            if $x < MutRate$ then
                $y \leftarrow$ Random(0,1)
                if $y < p_{add}$ then
                    $g \leftarrow$ EdgeAdd($g$)
                else
                    $g \leftarrow$ Mutation($g$)
                end if
            end if
        end for

        $Pop \leftarrow NewPop$
        $Gen{+}{+}$
    until $Gen > MaxGens$
end function
```

$V_i^{G_{c1}}$ and $V_i^{G_{c2}}$ are the sets of all vertices adjacent to vertex i in each child chromosome, and $\bar{V}_i^{G_{c1}}$ and $\bar{V}_i^{G_{c1}}$ are these same sets but with all of the common vertices stripped out.

We then iterate through each vertex $j \in \bar{V}_i^{G_{c1}}$ and select a vertex $k \in \bar{V}_i^{G_{c2}}$ that has not been used. We remove the edge e_{ij} from G_{c1} and the edge e_{ik} from G_{c2}, then add the edges e_{ik} into G_{c1} and e_{ij} into G_{c2}. Doing this changes the degree of vertices j and k in both child graphs, so we need to make a correction in order to preserve the degrees of all vertices.

In G_{c1}, we randomly select another edge e_{kl} connected to vertex k and remove it. Then we add another new edge e_{jl}. This restores the original degree of both vertices in G_c1. We then must perform a similar operation in G_{c2}. We then repeat this entire process for every vertex i in the network graph. This process results in two child networks that have some of the structural properties of each of their parents.

Mutation The mutation operator used in this genetic algorithm is simply performing a series of random edge swaps, similar to initialization process.

The edge addition is implemented as part of the mutation operation. Whenever mutation is performed, there is a chance that an edge addition is performed instead of the regular mutation operation. Like in the GD and SA algorithms, the probability of performing edge addition is governed by the p_{add} parameter.

Memetic Algorithm The memetic algorithm optimizer works in mostly the same way as the GA, but uses a heuristic local search procedure in place of the mutation operator [7]. This local search prioritizes edge swaps that increase the assortativity of the network. Highly robustness and high assortative have been shown to be correlated [8], so this heuristic may result in better improvements to the network's robustness.

This heuristic works by only performing edge swaps between pairs of edges e_{ij} and e_{jk} if the following inequality holds:

$$|d_i - d_l| + |d_j - d_k| < \alpha \times (|d_i - d_j| + |d_k - d_l|)$$

where d_i, d_j, d_k, and d_l are the degrees of nodes i, j, k, and l. The parameter α is used to control the strength of the preference for increased assortativity [7].

This heuristic edge swap is performed multiple times in a row, similar to the GA's mutation operator. Also similar to the GA, edge addition may be performed instead of the local search with a probability based on the parameter p_{add}.

4 Data and Experimental Setup

This chapter describes the setup for the experiments as well as the data used for testing. Additionally, the two main metrics of performance are described.

4.1 Data

Both synthetic and real world networks will be used for the experiments. The bulk of the experiments will be performed using synthetic networks generated using the Barabási-Albert model, and a second set of experiments will be performed on a real-world network.

4.1.1 Barabási-Albert Model

The Barabási-Albert (BA) model [29] is a graph model used to generate scale-free networks. The model generates networks starting from an initial cluster of m_0 fully connected vertices, then adds new vertices one at a time using a preferential attachment mechanism. When a new vertex is added, it is connected to m existing vertices in the network, prioritizing vertices with a high degree. This process is continued until the desired number of vertices, n has been added. The preferential attachment mechanism ensures that there is a small number of high degree hub vertices and progressively higher numbers of vertices with smaller degrees, as expected in a scale-free network.

The probability p_i that a newly added vertex will be connected to vertex i is given as:

$$p_i = \frac{d_i}{\sum_j d_j}$$

where d_i is the degree of vertex i and j is the number of vertices in the network at the current iteration of the construction algorithm.

4.1.2 Power Grid Network

The real-world network examined in this thesis is the power grid network known as the 1138-bus network [30]. It has 1138 vertices and originally had 2596 edges. The network included a large number of self referential edges, which were removed since they cause issues with some of the metaheuristic implementations. This left the network with 1458 edges. The focus of this work is on optimization of simple graphs (networks with no multi-edges or self-referential loops), and these implementations were not designed to work with non-simple graphs. For this reason, the simplest approach was to transform the network into a simple graph.

While this modification to the network did not affect its initial robustness, it may have some effect on the optimization process. The removal of the self-referential links may influence where edge addition tends to get performed, particularly for the HBS and IAS addition strategies. This is due to the change in network properties such as betweenness and vertex degree caused by the removal.

4.2 Parameter Optimization

Parameter optimization was performed on the GD, SA, and GA optimizers to determine the ideal set of parameters to use for further tests. Each of these parameter optimization experiments were performed on Barabasi-Albert networks with 100 vertices. 30 runs were performed for each parameter value, and results were analyzed using ANOVA and pairwise t-tests.

Tables 1, 2, and 3 display the different values tested for each parameter of each optimizer.

Table 1: Parameters used for testing the SA optimizer

Parameter	**Tested values**
T_0	0.1, 0.3, 0.6, 1.0, 1.5, 2.0, 3.0
T_{min}	0.0001, 0.0005, 0.001, 0.005, 0.01, 0.05, 0.1, 0.15
T_{decay}	0.005, 0.01, 0.05, 0.1, 0.15, 0.2, 0.25

Table 2: Parameters used for testing the GD optimizer

Parameter	**Tested values**
$L_{\%}$	0.3, 0.5, 0.7, 0.9, 1.0, 1.2
$\Delta_{L\%}$	0.0001, 0.0005, 0.001, 0.005, 0.01, 0.02
i_{max}	10, 20, 40, 60, 80, 100

Table 3: Parameters used for testing the GA/MA optimizers

Parameter	**Tested values**
gen_{max}	200, 400, 600, 800, 1000, 1200, 1400, 1600, 1800, 2000, 2500, 3000, 4000, 5000, 6000, 7000
$popSize$	5, 10, 15, 20, 25, 30, 50, 75, 100, 125, 150, 175, 200
R_C	0.1, 0.3, 0.5, 0.7, 0.9, 1.0
R_M	0.0, 0.1, 0.3, 0.5, 0.7, 0.9

4.2.1 Static Parameters

Some parameters are kept constant throughout all experiments. These parameters, along with their values, are listed in Table 4.

4.3 Main Experimental Setup

The main set of experiments will determine the effectiveness of incorporating edge addition into the regular edge rewiring optimization procedure. In order to compare the performance of this variation to the vanilla problem, a set of runs will be performed for each of the four optimization procedures (SA, GD, GA, MA) using only the vanilla edge rewiring method. Runs will be performed using BA networks of 100, 200, and 500 vertices.

Once the vanilla rewiring experiments are complete, a set of runs will be completed

Table 4: Static parameters

Name	Description	Value
δ_d	IAS similarity threshold	3
SA - $NoImprove$	Number of consecutive iterations with no change in R for SA optimizer	80
k	GA/MA tournament size	3
$MutateAmount$	Maximum number of edge swaps performed in GA mutation/MA local search	4
α	MA local search heuristic strength	0.9
m	Number of edges added for each new vertex in BA model	3
m_0	Number of vertices in initial fully connected cluster for BA model	3

with varying amounts of edge addition. Tests will be performed with 5%, 10%, 15%, 20%, 25%, and 30% edge addition for each of the 4 optimizers on each of the 3 network sizes.

Each experiment will have 30 runs, and edge addition will be performed using random linking strategy (RLS). Comparisons will be performed between both the ΔR value and RE value described below. Statistical significance will be determined using ANOVA and pairwise t-tests with a level of significance of $\alpha = 0.05$.

The experiments described above will be repeated with two other edge addition strategies: high betweenness linking strategy (HBS) and increased assortativity linking strategy (IAS). The best performing addition chance will be determined for each addition strategies, and the different strategies will then be compared to each other.

Experiments will also be performed on the power grid network. Tests will be performed with all 4 metaheuristic optimization methods using both the vanilla approach and the hybrid approach. The hybrid approach will be compared to the vanilla approach for each optimizer and then the best version of the optimizers will be compared against each other to determine the most effective method for increasing the network's robustness.

4.4 Performance Measures

Two different performance measures will be used to measure the effectiveness of the robustness optimization; they are described in this section.

4.4.1 ΔR

One of these performance measures is ΔR. This measure is calculated using the following formula:

$$\Delta R = \frac{R}{R_0}$$

where R is the R value at the end of the optimization and R_0 is the initial R value of the target network. This measure gives an indication of how much the robustness of the target network changes, independent of any other changes to the network. It also allows easy comparison between networks with different starting R values.

4.4.2 RE Value

The other value used to measure the performance of the optimization is RE value. This value measures how much the target network's robustness changes relative to how many new edges are added during the optimization.

Any network can be made maximally robust by continuously adding edges to it until it is fully connected. This is obviously an impractical approach for improving the robustness of any real-world network. Another problem that arises when we try to optimize a real-world network's robustness through edge addition is cost. Any change to a network's structure will incur some cost, but edge additions can be especially costly in many situations. They carry not only the cost of adding the connection itself, but may also carry costs associated with modifying the affected vertices to handle a larger number of connections.

It likely is not a good idea to add 20 new edges to a network for a 5% increase in robustness when we could get that same 5% increase by adding one single edge to a different spot. The RE value allows us to measure the difference between cases like these, as well as compare these kinds of results to optimized networks that had no new edges added. RE value is defined as follows:

$$RE = \frac{R}{R_0} \div \frac{|E|}{|E_0|}$$

where R is the current network robustness as measured by the R value, R_0 is the robustness of the initial network, $|E|$ is the number of edges currently in the network, and $|E_0|$ is the number of edges in the initial network.

The RE value can be used to compare the relative effectiveness of optimization done with different amounts of edge addition. A useful property of this measure is that it is equal to ΔR when the number of edges in the target network does not change (i.e. $p_add = 0$).

5 Results & Analysis

This chapter presents the results of the experiments described in Section 4 and the statistical analysis of these results.

5.1 Parameter Optimization

This section presents the results of the parameter optimization experiments for the simulated annealing, great deluge, and genetic algorithms. Additionally, the parameters used in the main set of experiments were selected based on these results and are presented here.

5.1.1 Simulated Annealing Parameters

ANOVA and pairwise t-tests showed that the different values for T_0 generally performed equally well. The one exception was $T_0 = 0.1$, which performed significantly worse than the others. Table 5 and Figure 2 present the results of this experiment. This indicates that the choice of starting temperature for the the SA algorithm was not very important, so long as a very low value was not used.

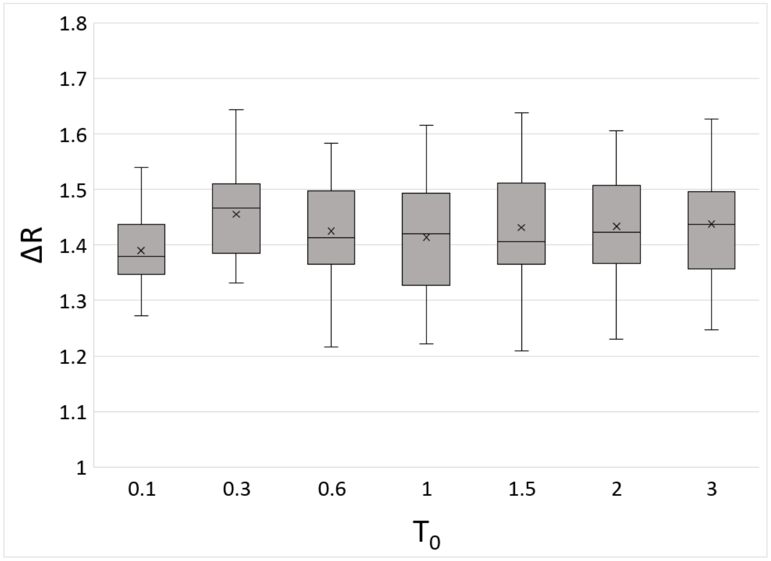


Figure 2: SA parameter optimization results for T_0

Table 5: Average R value for different values of T_0 in the SA optimizer

$\boldsymbol{T_0}$	$\boldsymbol{\Delta R}$
0.1	1.39
0.3	1.46
0.6	1.42
1.0	1.41
1.5	1.43
2.0	1.43
3.0	1.44

As shown in Table 6 and Figure 3, experiments with T_{min} showed no significant difference between the different tested values. Similar results were seen with T_{decay}, with all tested values having statistically similar results. This can be seen in Table 7 and Figure 4. These results indicate that the choice of these two parameters is not very important for the algorithm's performance.

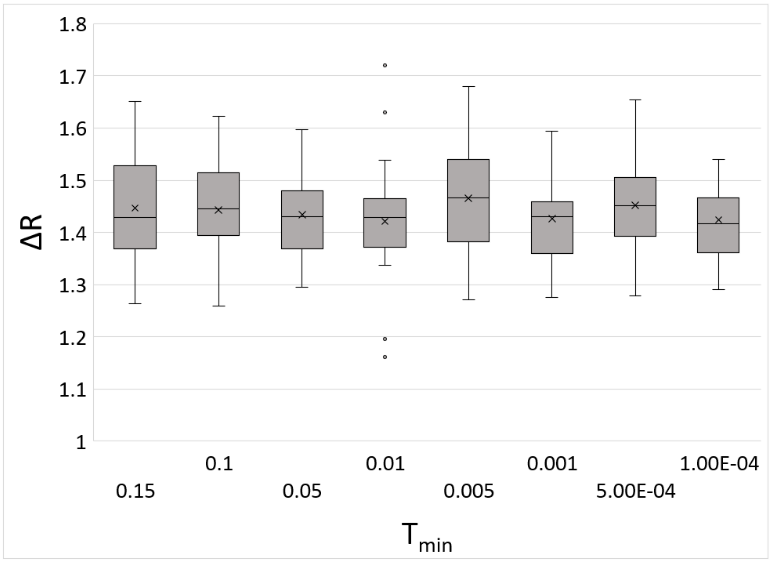


Figure 3: SA parameter optimization results for T_{min}

Table 6: Average R value for different values of T_{min} in the SA optimizer

T_{min}	ΔR
0.0001	1.42
0.0005	1.45
0.001	1.43
0.005	1.47
0.01	1.42
0.05	1.43
0.1	1.44
0.15	1.45

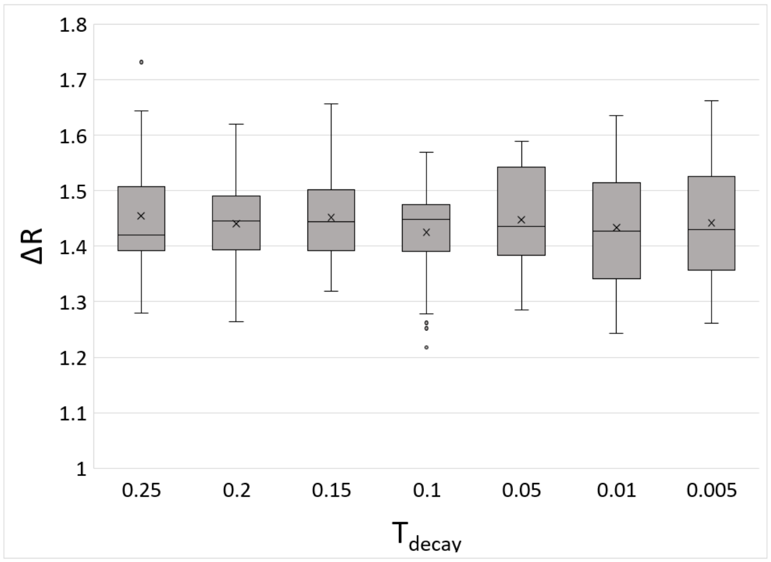


Figure 4: SA parameter optimization results for T_{decay}

Table 7: Average R value for different values of T_{decay} in the SA optimizer

T_{decay}	ΔR
0.005	1.44
0.01	1.43
0.05	1.45
0.1	1.42
0.15	1.45
0.2	1.44
0.25	1.45

Based on the results of these experiments, a T_0 value of 1.0, a T_{min} value of 0.01, and a T_{decay} value of 0.15 were selected for use in the remainder of experiments.

5.1.2 Great Deluge Parameters

Table 8 and Figure 5 show the results of the experiments for the parameter $L_{\%}$. These experiments show that the initial threshold level ($L_{\%}$) does not have a large impact on optimizer performance, so long as the value is not set too high. All tested values except 1.2 led to no significant difference in performance compared to each other. It is notable that a $L_{\%}$ of 1.2 led to no improvement to R value at all, as evidenced by the ΔR being exactly 1. This is a consequence of the way the great deluge algorithm works. If the initial threshold is set significantly above the initial fitness value, there may not be a neighbour of the initial problem state which gives a fitness value greater than this threshold. In this case, the algorithm cannot make any improvements and will be stuck in the initial state.

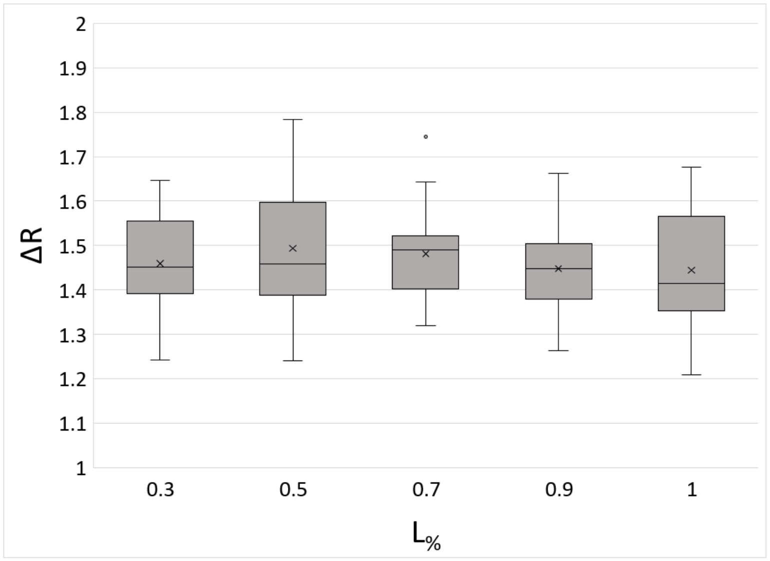


Figure 5: GD parameter optimization results for $L_{\%}$

As seen in Table 9 and Figure 6, the experiments showed that lower values for $\Delta_{L\%}$ are more beneficial. The values of 0.0001 and 0.0005 both performed significantly

Table 8: Average R value for different values of $L_{\%}$ in the GD optimizer

$\boldsymbol{L_{\%}}$	$\boldsymbol{\Delta R}$
0.3	1.46
0.5	1.49
0.7	1.48
0.9	1.45
1.0	1.44
1.2	1.00

better than the other tested values, and about as well as each other. Low values for this parameter mean that the will more gradually approach a solution.

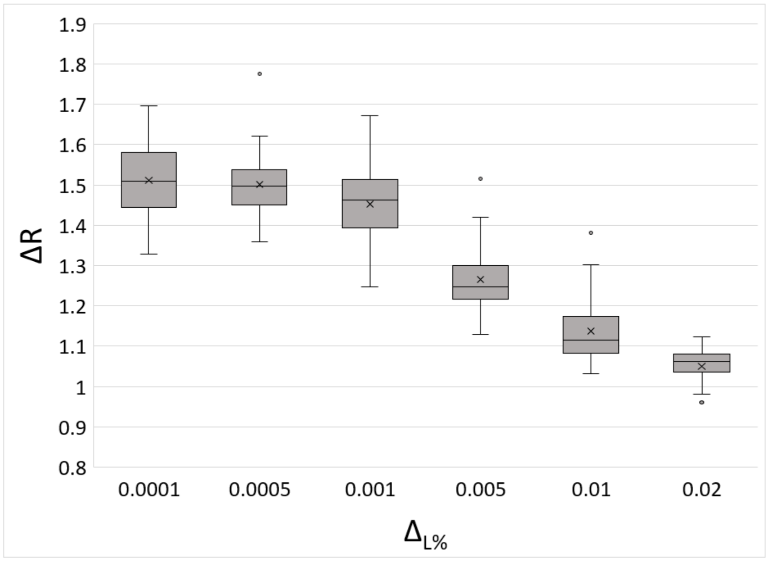


Figure 6: GD parameter optimization results for $\Delta_{L\%}$

Finally, larger values for i_{max} led to better results than lower values, as shown in Table 10 and Figure 7. The i_{max} values of 80 and 100 outperformed all other values and performed equally well as each other. Larger values for this parameter gives the algorithm more time to explore the search space around states in difficult parts of the search space, so the algorithm will be less likely to get stuck there as a local optimum.

Based on the results of these experiments, a $L_{\%}$ value of 0.5, a $\Delta_{L\%}$ value of 0.0005, and a i_{max} value of 100 were selected for the remaining experiments.

Table 9: Average R value for different values of $\Delta_{L\%}$ in the GD optimizer

$\boldsymbol{\Delta_{L\%}}$	$\boldsymbol{\Delta R}$
0.0001	1.51
0.0005	1.50
0.001	1.45
0.005	1.27
0.01	1.38
0.02	1.05

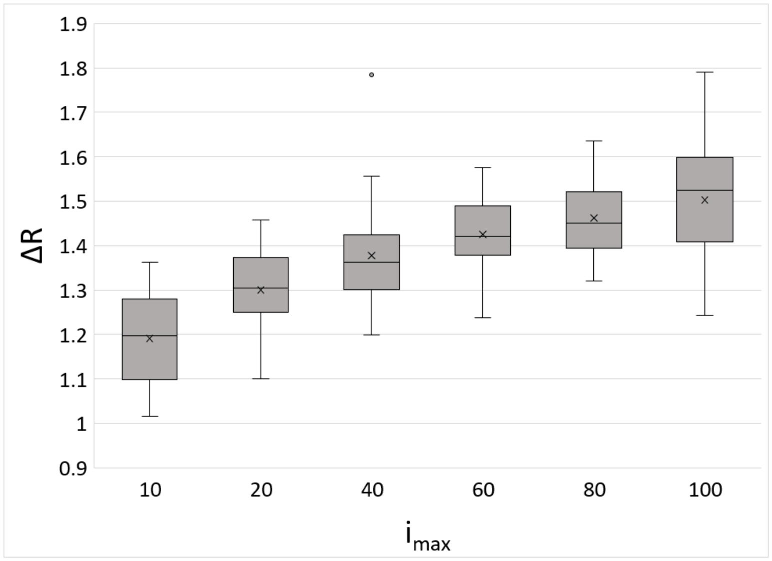


Figure 7: GD parameter optimization results for i_{max}

Table 10: Average R value for different values of i_{max} in the GD optimizer

$\boldsymbol{i_{max}}$	$\boldsymbol{\Delta R}$
10	1.19
20	1.30
40	1.38
60	1.42
80	1.46
100	1.50

5.1.3 Genetic Algorithm Parameters

The experiments showed that larger values for the max number of generations are generally better than lower values. These results can be seen in Table 11 and Figure 8. The ΔR for 7000 generations was higher than for most other values, though the ΔR was not statistically different for 3000 and 6000 generations. Since 3000 generations has equivalent performance to 7000 generations but uses less than half the time, that is the value that will be used in the rest of the experiments.

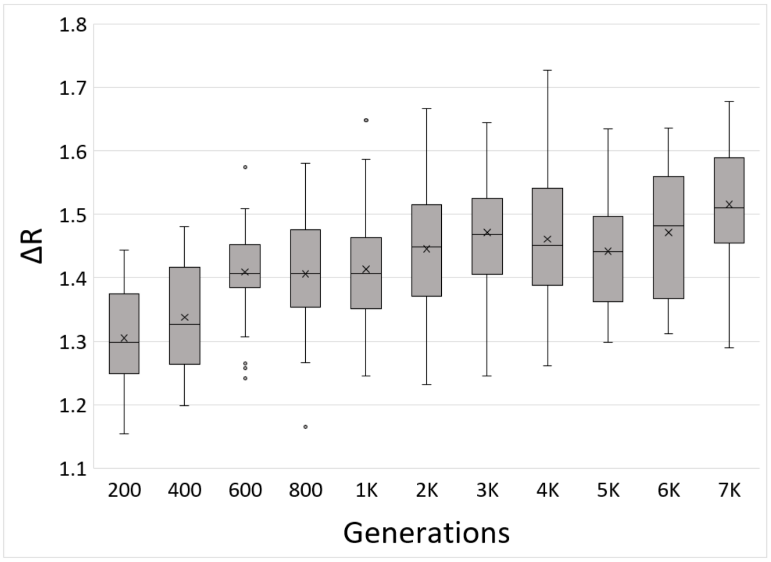


Figure 8: GA parameter optimization results for gen_{max}

Table 11: Average R value for different values of gen_{max} in the GA/MA optimizers

gen_{max}	ΔR	gen_{max}	ΔR
200	1.30	1800	1.44
400	1.34	2000	1.46
600	1.41	2500	1.46
800	1.41	3000	1.48
1000	1.42	4000	1.46
1200	1.40	5000	1.44
1400	1.42	6000	1.47
1600	1.44	7000	1.52

Table 12 and Figure 9 show the results of experiments on population size for the GA. These results show that lower population sizes outperformed larger sizes. Specifically, population sizes between 5 and 25 performed about equivalently, and sizes larger than 25 get progressively worse. These best performing population sizes are unusually small for a GA, but are consistent with the size used by Zhou *et al.* [7], who used a population size of 10 in their work.

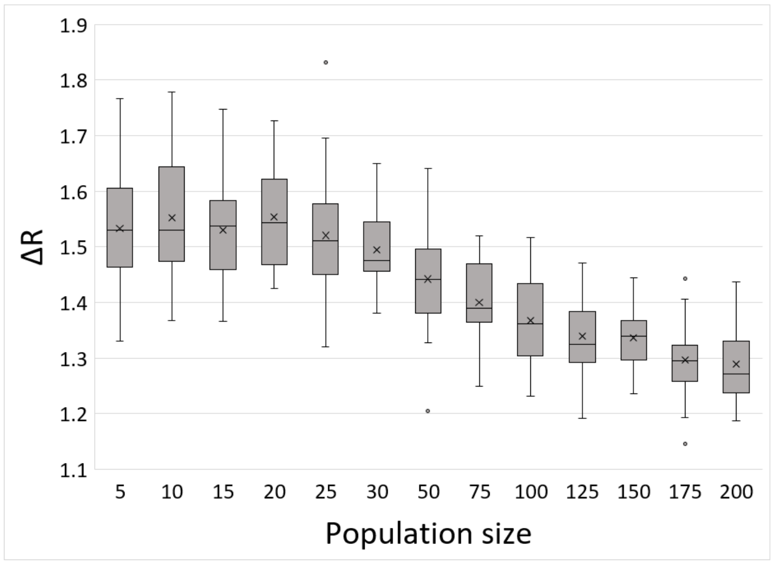


Figure 9: GA parameter optimization results for *popSize*

Table 12: Average R value for different values of *popSize* in the GA/MA optimizers

popSize	**ΔR**	***popSize***	**ΔR**
5	1.53	75	1.40
10	1.55	100	1.37
15	1.53	125	1.34
20	1.55	150	1.36
25	1.52	175	1.30
30	1.49	200	1.29
50	1.44		

Crossover and mutation rate were tested together, and there was no single best pair of values. Results are displayed in Table 13. It is notable that having a mutation rate of 0 or 0.1 led to significantly worse results that higher mutation rates and a crossover rate of 1 also led to worse results.

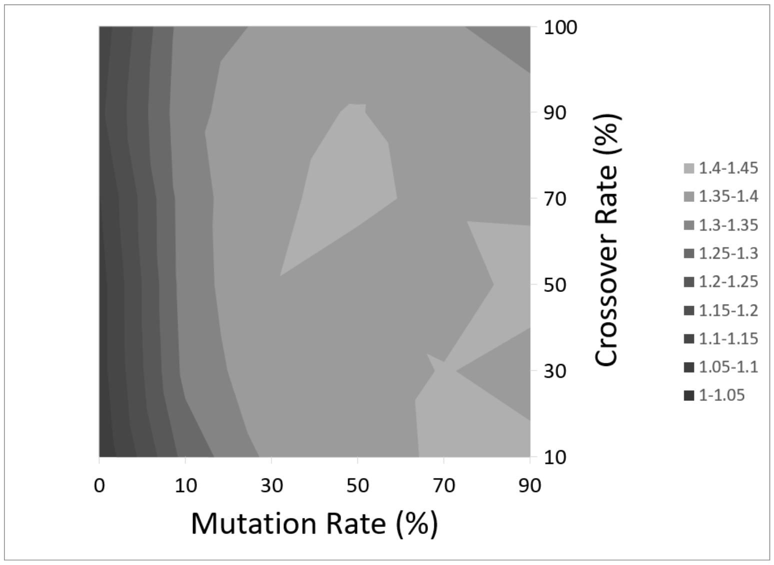


Figure 10: GA parameter optimization results for crossover and mutation rate

Table 13: Average R value for different values of crossover and mutation rate in the GA/MA optimizers

		Crossover rate					
		0.1	**0.3**	**0.5**	**0.7**	**0.9**	**1.0**
Mutation Rate	**0.0**	1.06	1.08	1.08	1.10	1.14	1.12
	0.1	1.27	1.32	1.32	1.33	1.34	1.33
	0.3	1.36	1.39	1.40	1.40	1.38	1.36
	0.5	1.39	1.37	1.38	1.41	1.40	1.35
	0.7	1.40	1.40	1.37	1.39	1.35	1.36
	0.9	1.42	1.38	1.42	1.39	1.37	1.33

Based on the results of these experiments, a *popSize* value of 10, a gen_{max} value of 300, a R_C of 0.7, and a R_M of 0.3 were selected for use in future experiments.

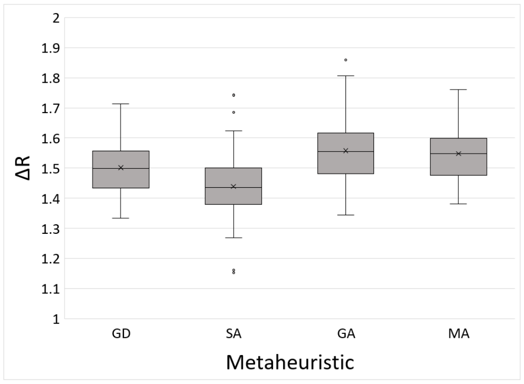
5.2 Vanilla Metaheuristic Results

This section presents results demonstrating the performance of the vanilla edge rewiring method. All four metaheuristics are compared on BA networks of 100, 200, and 500 vertices. The results are summarized in Table 14 and Figure 11. In the table, bolded results in a column indicate the best optimizer for that network size.

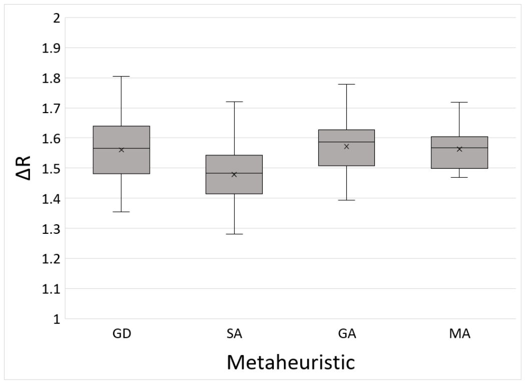
Table 14: Average ΔR value for vanilla edge rewiring optimizers.

Optimizer	**100 vertices**	**200 vertices**	**500 vertices**
Great Deluge	1.50	**1.56**	**1.49**
Simulated Annealing	1.44	1.48	**1.50**
Genetic Algorithm	**1.56**	**1.57**	**1.50**
Memetic Algorithm	**1.55**	**1.56**	**1.53**

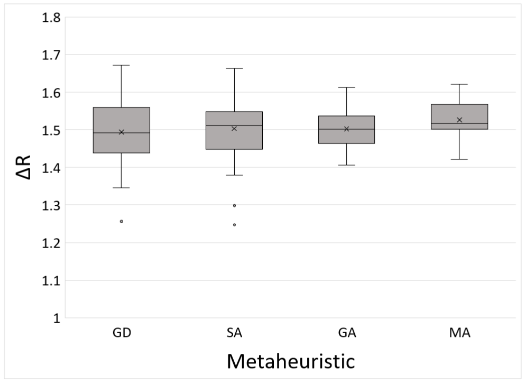
On the 100 vertex networks, the genetic and memetic algorithms had the best results, with the great deluge algorithm coming in second. Simulated annealing performed the worst of the four metaheuristics. For the 200 vertex networks, the GA, MA, and GD optimizers all performed about equivalently while the SA optimizer performed significantly worse. All four optimizers performed equivalently on the 500 vertex networks.



(a) 100 vertices



(b) 200 vertices



(c) 500 vertices

Figure 11: Performance of vanilla edge rewiring optimizers

These results indicate that the GA and MA optimizers perform well across all network sizes, while the SA optimizer performs poorly on small networks. For larger networks, all four optimizers performed similarly.

5.3 Hybrid Approach Results

This section presents the results of experiments with the hybrid approach. Each of the three linking strategies for edge addition are examined to determine the best ratio of edge addition to edge rewiring for that method. The three strategies are then compared to each other to determine which of them performs best.

5.3.1 Random Linking Strategy Results

Tables 15, 16, 17, and 18 show the effect of different amounts of random linking strategy (RLS) addition in the hybrid method. Each table shows both the ΔR and RE values for each of the three network sizes. The bold values show the statistically best result(s) in that column of the table.

Table 15: Great Deluge hybrid optimizer with RLS

	100 vertex		**200 vertex**		**500 vertex**	
p_{add}	$\boldsymbol{\Delta R}$	$\boldsymbol{RE}$	$\boldsymbol{\Delta R}$	$\boldsymbol{RE}$	$\boldsymbol{\Delta R}$	$\boldsymbol{RE}$
0.05	1.87	**1.26**	1.84	**1.48**	1.69	**1.54**
0.10	2.06	0.99	1.99	1.30	1.78	1.50
0.15	2.19	0.80	2.14	1.16	1.88	1.45
0.20	**2.29**	0.66	2.19	1.02	1.96	1.39
0.25	**2.31**	0.57	**2.32**	0.91	**2.04**	1.33
0.30	**2.36**	0.49	**2.39**	0.82	**2.09**	1.27

These results show that larger add chances lead to much larger ΔR values compared to low add chances. Conversely, large add chances result in smaller RE values compared to lower add chances. This is not always the case, however. On the 500 vertex network, the GA and MA optimizers did not have large reductions in RE as

Table 16: Simulated Annealing hybrid optimizer with RLS

	100 vertex		**200 vertex**		**500 vertex**	
p_{add}	$\boldsymbol{\Delta R}$	$\boldsymbol{RE}$	$\boldsymbol{\Delta R}$	$\boldsymbol{RE}$	$\boldsymbol{\Delta R}$	$\boldsymbol{RE}$
0.05	1.56	**1.41**	1.61	**1.47**	1.63	**1.53**
0.10	1.64	1.37	1.69	**1.45**	1.71	**1.51**
0.15	1.65	1.31	1.74	1.44	1.75	**1.49**
0.20	1.74	1.29	1.76	1.40	1.78	1.47
0.25	**1.85**	1.28	**1.79**	1.36	**1.81**	1.46
0.30	**1.81**	1.21	**1.84**	1.35	**1.85**	1.45

Table 17: Genetic Algorithm hybrid optimizer with RLS

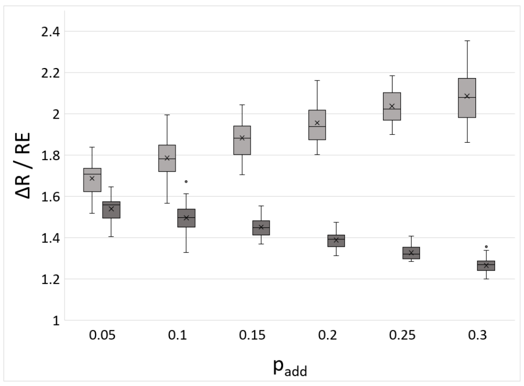
	100 vertex		**200 vertex**		**500 vertex**	
p_{add}	$\boldsymbol{\Delta R}$	$\boldsymbol{RE}$	$\boldsymbol{\Delta R}$	$\boldsymbol{RE}$	$\boldsymbol{\Delta R}$	$\boldsymbol{RE}$
0.05	1.65	**1.48**	1.67	**1.55**	1.58	**1.51**
0.10	1.76	**1.47**	1.75	**1.55**	1.63	**1.51**
0.15	1.79	1.44	1.75	1.50	1.67	**1.52**
0.20	**1.82**	1.38	1.79	1.48	**1.70**	**1.51**
0.25	**1.85**	1.37	**1.83**	1.49	**1.71**	1.50
0.30	**1.83**	1.33	**1.85**	1.46	**1.72**	1.49

the add chance increased, with the value staying relatively similar for most values. This can be seen clearly in Figure 12 for the largest network size, where both the GA and MA have relatively constant RE as p_{add} increases.

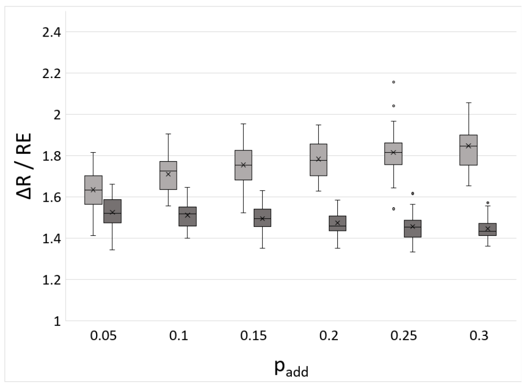
The great deluge algorithm seems to benefit the most from large add chances, with the highest ΔR among the four optimizers. However, GD also sees the lowest RE value for large add chances, with many tests giving RE values less than 1. This is visually demonstrated in Figure 12, with ΔR rising sharply with increasing p_{add} and RE falling sharply. Simulated annealing follows this same pattern, but with more moderate changes to both ΔR and RE compared to GD.

Table 18: Memetic Algorithm hybrid optimizer with RLS

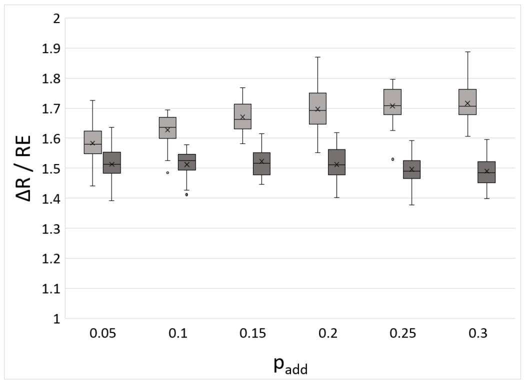
	100 vertex		**200 vertex**		**500 vertex**	
p_{add}	$\boldsymbol{\Delta R}$	$\boldsymbol{RE}$	$\boldsymbol{\Delta R}$	$\boldsymbol{RE}$	$\boldsymbol{\Delta R}$	$\boldsymbol{RE}$
0.05	1.69	**1.53**	1.67	**1.56**	1.58	**1.52**
0.10	1.75	1.48	1.75	**1.56**	1.64	**1.52**
0.15	1.78	1.43	1.78	1.51	1.67	**1.52**
0.20	**1.84**	1.41	1.82	1.52	1.70	**1.51**
0.25	**1.87**	1.41	1.83	1.49	**1.72**	**1.51**
0.30	**1.88**	1.34	**1.90**	1.50	**1.76**	**1.51**



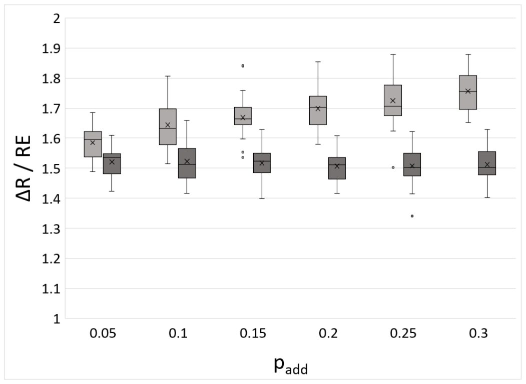
(a) Great Deluge



(b) Simulated Annealing



(c) Genetic Algorithm



(d) Memetic Algorithm

Figure 12: ΔR and RE value for different p_{add} values with RLS on 500 vertex networks. ΔR shown in light grey, RE shown in dark grey.

5.3.2 High Betweenness Linking Strategy Results

Tables 19, 20, 21, 22 show the results of the hybrid experiments with HBS addition. These experiments had some different results compared to the RLS experiments. It can still be seen that the RE value gets lower for larger add chances, as with RLS. However, this strategy performed differently in terms of ΔR. Some of the experiments showed a slight preference against a small add chance, but the results were mostly similar across different values.

Table 19: Great Deluge hybrid optimizer with HBS

	100 vertex		**200 vertex**		**500 vertex**	
p_{add}	$\boldsymbol{\Delta R}$	$\boldsymbol{RE}$	$\boldsymbol{\Delta R}$	$\boldsymbol{RE}$	$\boldsymbol{\Delta R}$	$\boldsymbol{RE}$
0.05	1.65	**1.12**	1.62	**1.32**	1.55	**1.43**
0.10	**1.69**	0.88	**1.70**	1.17	**1.58**	1.37
0.15	1.67	0.72	**1.74**	1.02	1.57	1.28
0.20	**1.71**	0.61	**1.73**	0.91	**1.59**	1.22
0.25	**1.73**	0.54	**1.74**	0.82	**1.59**	1.16
0.30	**1.70**	0.48	**1.73**	0.75	**1.62**	1.12

Table 20: Simulated Annealing hybrid optimizer with HBS

	100 vertex		**200 vertex**		**500 vertex**	
p_{add}	$\boldsymbol{\Delta R}$	$\boldsymbol{RE}$	$\boldsymbol{\Delta R}$	$\boldsymbol{RE}$	$\boldsymbol{\Delta R}$	$\boldsymbol{RE}$
0.05	**1.49**	**1.31**	**1.52**	**1.37**	1.55	**1.42**
0.10	**1.48**	1.16	**1.52**	1.24	**1.56**	1.33
0.15	**1.48**	1.02	**1.57**	1.13	**1.61**	1.26
0.20	**1.51**	0.94	**1.56**	1.07	**1.58**	1.19
0.25	**1.49**	0.86	**1.55**	0.96	**1.58**	1.12
0.30	**1.52**	0.76	**1.57**	0.90	**1.60**	1.04

Comparing the results from HBS shown in Figure 13 to the earlier results with RLS (Figure 12), it is clear that high amounts of HBS is much less useful for optiming robustness than large amount of RLS. The ΔR barely changes across different values of p_{add} while the RE value drops off with higher p_{add} values. This means that there is no benefit to using high levels of HBS, since doing so will not lead to more robust networks while also incurring additional costs related to adding more new edges.

Table 21: Genetic Algorithm hybrid optimizer with HBS

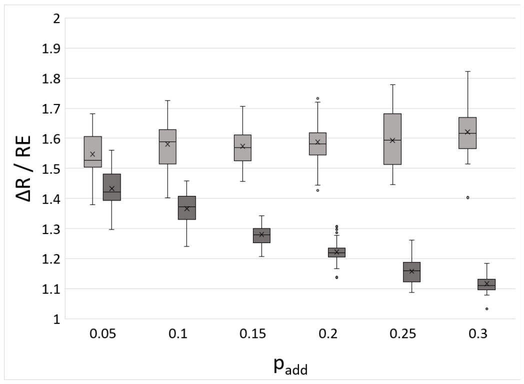
	100 vertex		**200 vertex**		**500 vertex**	
p_{add}	$\boldsymbol{\Delta R}$	$\boldsymbol{RE}$	$\boldsymbol{\Delta R}$	$\boldsymbol{RE}$	$\boldsymbol{\Delta R}$	$\boldsymbol{RE}$
0.05	1.60	**1.44**	1.57	**1.48**	**1.53**	**1.48**
0.10	**1.61**	1.29	**1.61**	1.44	**1.54**	**1.46**
0.15	**1.64**	1.21	**1.62**	1.37	**1.54**	1.43
0.20	**1.66**	1.14	**1.61**	1.33	**1.53**	1.40
0.25	**1.66**	1.08	**1.61**	1.29	**1.54**	1.38
0.30	**1.63**	1.00	**1.63**	1.27	**1.54**	1.37

Table 22: Memetic Algorithm hybrid optimizer with HBS

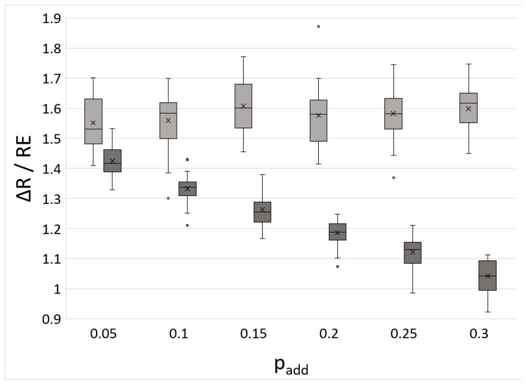
	100 vertex		**200 vertex**		**500 vertex**	
p_{add}	$\boldsymbol{\Delta R}$	$\boldsymbol{RE}$	$\boldsymbol{\Delta R}$	$\boldsymbol{RE}$	$\boldsymbol{\Delta R}$	$\boldsymbol{RE}$
0.05	**1.59**	**1.45**	**1.60**	**1.51**	**1.53**	**1.48**
0.10	1.57	1.30	**1.60**	1.44	**1.54**	**1.46**
0.15	**1.61**	1.22	**1.62**	1.40	**1.52**	1.41
0.20	**1.62**	1.14	**1.62**	1.34	**1.54**	1.51
0.25	**1.62**	1.12	**1.63**	1.29	**1.54**	1.38
0.30	**1.64**	1.03	**1.63**	1.27	**1.54**	1.36

This result is likely to to the way that HBS addition works. HBS prioritizes pairs of vertices that have a high betweenness value. It therefore has considerably less randomness compared to RLS. HBS may also be not very effective against a degree-based attack since it will continuously add edges to a subset of the network's vertices, increasing their degree and moving them up in the attack sequence. Adding even more edges to already high degree vertices will not have a large impact on the network's robustness since these vertices are the first to be destroyed during the attack.

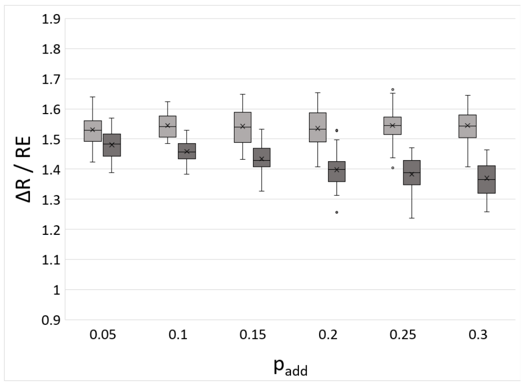
Similar to RLS, GD seems to benefit the most from high add chance in terms of ΔR, and has the worst RE values for high add chances among the four optimizers. It is notable that simulated annealing also experiences very low RE values as the add chance increases under HBS edge addition.



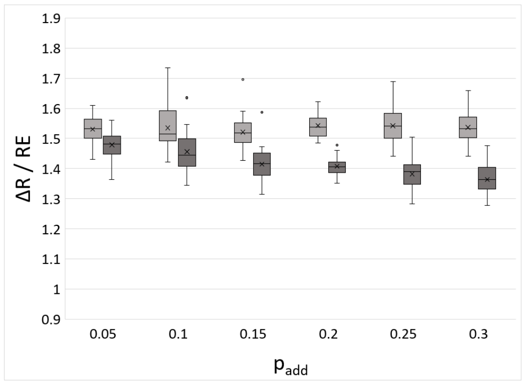
(a) Great Deluge



(b) Simulated Annealing



(c) Genetic Algorithm



(d) Memetic Algorithm

Figure 13: ΔR and RE value for different p_{add} values with HBS on 500 vertex networks. ΔR shown in light grey, RE shown in dark grey.

5.3.3 Increased Assortativity Linking Strategy Results

Tables 23, 24, 25, 26 show the results of the hybrid experiments with IAS addition. The results of these experiments have similarities to the experiments with RLS, in that higher add chances typically lead to higher ΔR and lower RE. The GA and MA optimizers substantially benefited from high add chance, with larger add chances leading to better RE than low add chances while also seeing large increases in ΔR. As with the other two edge addition strategies, GD seems to benefit the most from high add chance in terms of ΔR and has the worst RE for high add chances among the different optimizers.

Table 23: Great Deluge hybrid optimizer with IAS

	100 vertex		**200 vertex**		**500 vertex**	
p_{add}	$\boldsymbol{\Delta R}$	$\boldsymbol{RE}$	$\boldsymbol{\Delta R}$	$\boldsymbol{RE}$	$\boldsymbol{\Delta R}$	$\boldsymbol{RE}$
0.05	1.91	**1.27**	1.81	**1.46**	1.69	**1.54**
0.10	2.05	0.99	2.06	1.33	1.82	1.52
15	2.22	0.81	2.12	1.16	1.93	1.48
0.20	**2.30**	0.66	2.24	1.02	2.03	1.43
0.25	**2.30**	0.57	**2.34**	0.92	2.07	1.35
0.30	**2.39**	0.49	**2.37**	0.82	**2.18**	1.30

Table 24: Simulated Annealing hybrid optimizer with IAS

	100 vertex		**200 vertex**		**500 vertex**	
p_{add}	$\boldsymbol{\Delta R}$	$\boldsymbol{RE}$	$\boldsymbol{\Delta R}$	$\boldsymbol{RE}$	$\boldsymbol{\Delta R}$	$\boldsymbol{RE}$
0.05	1.59	**1.46**	1.62	**1.50**	1.65	**1.54**
0.10	1.66	**1.43**	1.70	**1.49**	1.73	**1.54**
0.15	1.73	1.42	1.78	**1.50**	1.77	**1.52**
0.20	1.75	1.36	**1.80**	1.47	**1.83**	**1.54**
0.25	**1.80**	1.38	**1.82**	1.45	**1.84**	1.52
0.30	**1.81**	1.35	**1.84**	1.46	**1.85**	1.51

Table 25: Genetic Algorithm hybrid optimizer with IAS

	100 vertex		**200 vertex**		**500 vertex**	
p_{add}	$\boldsymbol{\Delta R}$	$\boldsymbol{RE}$	$\boldsymbol{\Delta R}$	$\boldsymbol{RE}$	$\boldsymbol{\Delta R}$	$\boldsymbol{RE}$
0.05	1.69	**1.53**	1.66	**1.55**	1.57	1.50
0.10	1.75	**1.50**	1.75	**1.56**	1.64	**1.53**
0.15	1.83	1.47	1.78	**1.55**	1.70	**1.54**
0.20	1.81	1.43	**1.82**	**1.54**	1.71	**1.53**
0.25	**1.90**	1.45	**1.84**	1.53	**1.75**	**1.54**
0.30	**1.85**	1.40	**1.86**	1.51	**1.77**	**1.53**

Table 26: Memetic Algorithm hybrid optimizer with IAS

	100 vertex		**200 vertex**		**500 vertex**	
p_{add}	$\boldsymbol{\Delta R}$	$\boldsymbol{RE}$	$\boldsymbol{\Delta R}$	$\boldsymbol{RE}$	$\boldsymbol{\Delta R}$	$\boldsymbol{RE}$
0.05	1.71	**1.56**	1.66	**1.55**	1.60	1.53
0.10	1.77	1.50	1.75	**1.56**	1.65	1.53
0.15	1.80	1.48	**1.80**	**1.55**	1.70	**1.54**
0.20	**1.89**	1.50	**1.82**	**1.55**	**1.77**	**1.56**
0.25	**1.86**	1.45	**1.83**	**1.51**	**1.77**	**1.55**
0.30	**1.87**	1.41	**1.86**	1.51	**1.79**	**1.54**

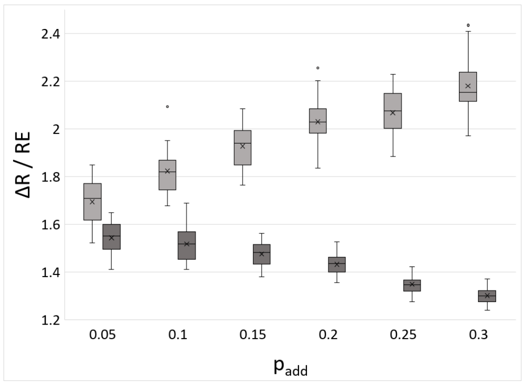
Figure 14 shows a visual representation of the performance of IAS with the 500 vertex networks. These charts show similarities with RLS: GD's ΔR increases greatly with increased p_{add} while RE drops off quickly and GA and MA see fairly consistent RE and increasing ΔR with high p_{add}. SA seems some slight differences, with a more consistent RE value with higher p_{add}, while still seeing increasing ΔR.

5.3.4 Comparison of Linking Strategies

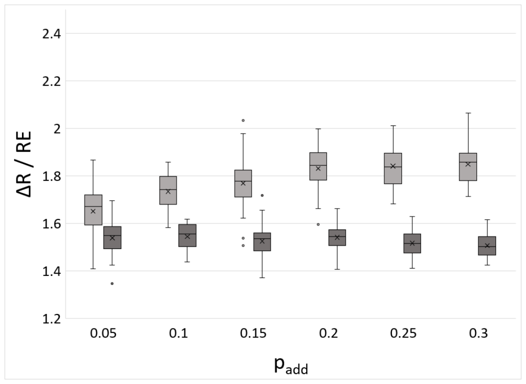
This subsection will compare the performance of these three linking strategies against each other and against the vanilla edge rewiring method for each metaheuristic. The linking strategies will be compared using the best value of the p_{add} parameter, as determined in the previous subsections. Specifically, p_{add} will be set to the value used in the test with the highest ΔR among the set of tests with the best RE value for that combination of linking strategy, metaheuristic, and network size. The p_{add} being used will be specified as each metaheuristic is discussed.

Table 27 shows a comparison of linking strategies for the great deluge algorithm. For this optimizer, the best performing tests were found to be those with $p_{add} = 0.05$ across all linking strategies and network sizes. The bolded values in the table show the best result in that column.

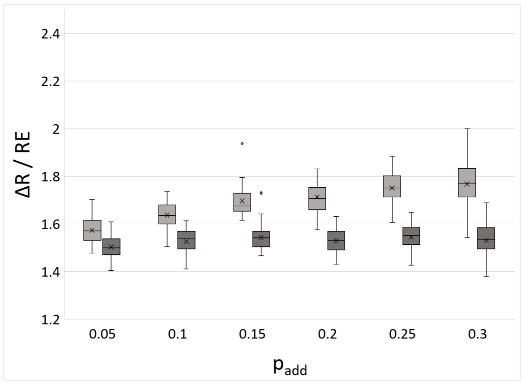
The inclusion of edge addition led to significant increases in ΔR over the vanilla method, regardless of which strategy was used. The RLS and IAS addition strategies



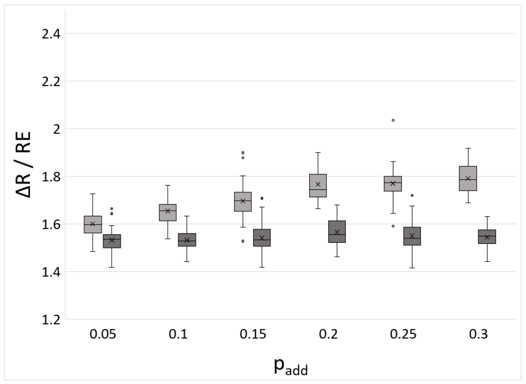
(a) Great Deluge



(b) Simulated Annealing



(c) Genetic Algorithm



(d) Memetic Algorithm

Figure 14: ΔR and RE value for different p_{add} values with IAS on 500 vertex networks. ΔR shown in light grey, RE shown in dark grey.

has the best ΔR across all sizes of network. HBS did not have as high of a ΔR as the other two edge addition strategies, but it was still better than no edge addition.

In terms of RE value, the vanilla method gave better results than any of the edge addition strategies for the 100 and 200 vertex networks. However, in the 500 vertex network, RLS and IAS had a significantly better RE value than the vanilla approach. HBS had the worst RE value across all network sizes.

These results indicate that the inclusion of edge addition, especially using the RLS and IAS strategies, led to greatly increased robustness of the target network, but this improvement came at the cost of a disproportionate number of new edges added to the network in most cases. The RE results for the 500 vertex network may hint that larger networks can benefit more from the inclusion of edge addition than

Table 27: Great Deluge - comparison of edge addition strategies

	100 vertex		**200 vertex**		**500 vertex**	
Addition Strategy	$\boldsymbol{\Delta R}$	$\boldsymbol{RE}$	$\boldsymbol{\Delta R}$	$\boldsymbol{RE}$	$\boldsymbol{\Delta R}$	$\boldsymbol{RE}$
None (vanilla)	1.50	**1.50**	1.56	**1.56**	1.49	1.49
RLS	**1.87**	1.26	**1.84**	1.48	**1.69**	**1.54**
HBS	1.65	1.12	1.62	1.32	1.55	1.43
IAS	**1.91**	1.27	**1.81**	1.46	**1.69**	**1.54**

small networks. These results also show that HBS is not a very effective method for performing edge addition within this hybrid approach. While it did lead to some small improvements to robustness, these were were accompanied by a disproportionate increase in the number of edges within the network, and HBS was outperformed by both other linking strategies.

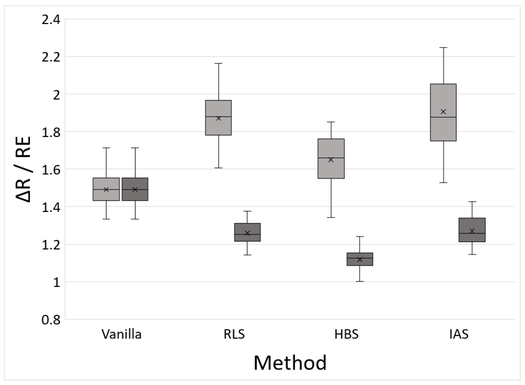
Table 28: Simulated Annealing - comparison of edge addition strategies

	100 vertex		**200 vertex**		**500 vertex**	
Addition Strategy	$\boldsymbol{\Delta R}$	$\boldsymbol{RE}$	$\boldsymbol{\Delta R}$	$\boldsymbol{RE}$	$\boldsymbol{\Delta R}$	$\boldsymbol{RE}$
None (vanilla)	1.44	**1.44**	1.48	**1.48**	1.50	1.50
RLS	1.56	1.41	1.69	1.45	1.75	1.49
HBS	1.49	1.31	1.52	1.37	1.55	1.42
IAS	**1.66**	**1.43**	**1.78**	**1.50**	**1.83**	**1.54**

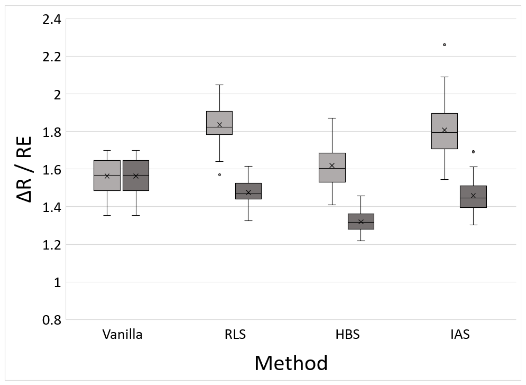
The best test results for the simulated annealing optimizer under different additions strategies are shown in Table 28. In the 100 vertex network, the 10% test for IAS is used. In the 200 vertex network, the 10% test for RLS and the 15% test for IAS are used. In the 500 vertex network, the 15% test for RLS and the 20% test for IAS are used. The 5% test is used in all other cases.

The SA operator with IAS performed extremely well. The best ΔR was obtained from this edge addition method for all network sizes. RLS came in second, and HBS a distant third barely better than the vanilla method.

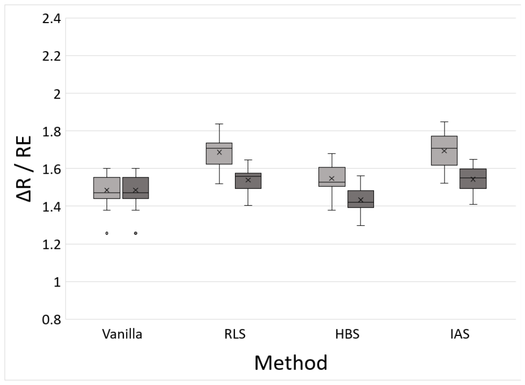
In terms of RE, IAS had results that were statistically similar to the vanilla method for the 100 and 200 vertex networks. For the 500 vertex network, IAS had a



(a) 100 vertices



(b) 200 vertices

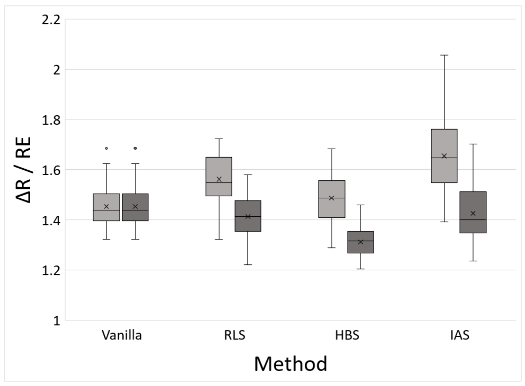


(c) 500 vertices

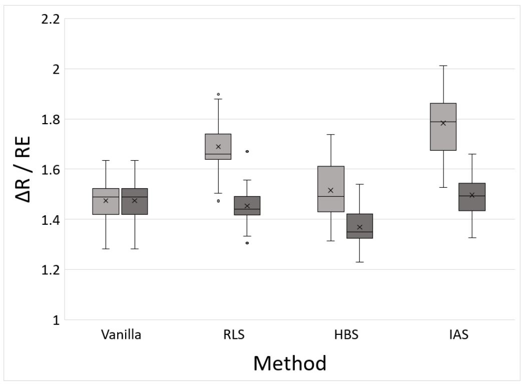
Figure 15: Comparison of three linking strategies and vanilla method for GD. ΔR shown in light grey, RE shown in dark grey.

significantly better RE value than the vanilla method. RLS did not perform as well with lower RE values than the vanilla method for the 100 and 200 vertex networks, and equal RE for the 500 vertex network. HBS was the worst performing, coming in last across all network sizes for RE.

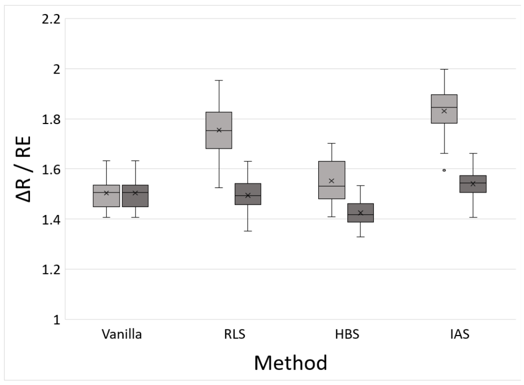
These results indicate that IAS was particularly effective for the simulated annealing optimizer. Its inclusion led to significantly increased robustness of the target network when compared to the vanilla method, and this increase in robustness was done without incorporating a disproportionate amount of new edges to the network. RLS was also fairly effective. While it was not quite as efficient with its edge addition, it did still lead to significant increases in robustness compared to the vanilla approach. HBS continues to be disappointing, falling far behind the other two linking strategies. Though there was some small increase in robustness with HBS, this was once again accompanied by a disproportionate increase in the number of edges.



(a) 100 vertices



(b) 200 vertices



(c) 500 vertices

Figure 16: Comparison of three linking strategies and vanilla method for SA. ΔR shown in light grey, RE shown in dark grey.

Table 29 shows a comparison of test results for different addition strategies for the GA optimizer. In the 100 vertex network, the 10% tests for RLS and IAS were used. In the 200 vertex network, the 10% test for RLS and the 20% test for IAS were used. For the 500 vertex network, the 20% RLS and 30% IAS tests were used. The 5% test was used for all network sizes for the HBS addition strategy.

For all network sizes, the IAS addition strategy outperformed the vanilla method in terms of ΔR. In the 100 vertex network tests, RLS had equivalent performance to IAS for ΔR, but IAS outperformed RLS for the other sizes. RLS still gave better results than the vanilla method for these larger networks. HBS performed much worse

Table 29: Genetic Algorithm - comparison of edge addition strategies

	100 vertex		**200 vertex**		**500 vertex**	
Addition Strategy	$\boldsymbol{\Delta R}$	$\boldsymbol{RE}$	$\boldsymbol{\Delta R}$	$\boldsymbol{RE}$	$\boldsymbol{\Delta R}$	$\boldsymbol{RE}$
None (vanilla)	1.56	**1.56**	1.57	**1.57**	1.50	1.50
RLS	**1.76**	1.47	1.75	**1.55**	1.70	**1.51**
HBS	1.60	1.44	1.57	1.48	1.53	1.48
IAS	**1.75**	1.50	**1.82**	**1.54**	**1.77**	**1.53**

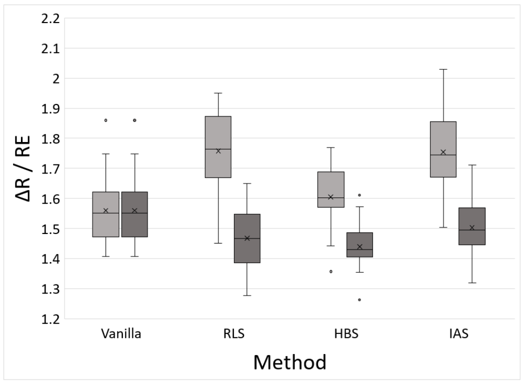
than the other two linking strategies once again, and did not even perform better than vanilla in the 200 vertex network.

The results in terms of RE were not as clear cut. For the 100 vertex network, the vanilla method had a better RE value than any of the addition methods. For the 200 vertex network, RLS and IAS performed equivalently to the vanilla method in terms of RE, and RLS and IAS were the best performing on the 500 vertex network tests. HBS had low RE values across all network sizes.

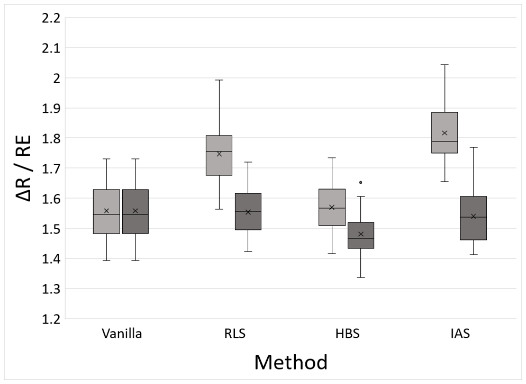
These results show that, like with the other optimizers, the inclusion of RLS or IAS edge addition leads to much better improvements in network robustness compared to the vanilla edge rewiring method. However, this robustness may come with a disproportionate amount of edge addition, particularly in smaller networks. The results hint that in large networks, the inclusion of edge addition may be a more efficient way of improving network robustness. As with the GD and SA optimizers, HBS is still the worst performing linking strategy.

Table 30 shows a comparison between the different edge addition methods for the MA optimizer. For the 100 vertex network, an add chance of 5% was used for all three edge addition methods. For the 200 vertex network, an add chance of 10% was used for RLS, 5% for HBS, and 25% for IAS. For the 500 vertex network, an add chance of 30% was used for RLS and IAS, and 10% for HBS.

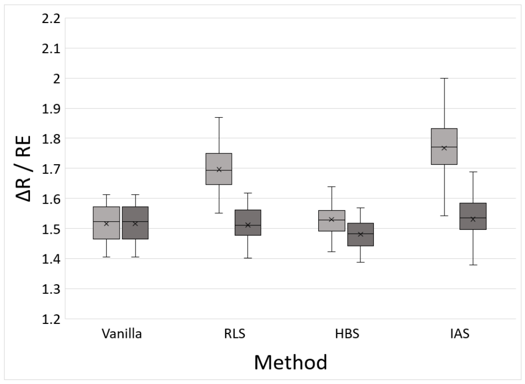
IAS proved to be very good amongst all network sizes, having the best ΔR value for each network size. RLS tied IAS for ΔR in the smallest network size, and came in



(a) 100 vertices



(b) 200 vertices



(c) 500 vertices

Figure 17: Comparison of three linking strategies and vanilla method for GA. ΔR shown in light grey, RE shown in dark grey.

second in the larger networks, though still had significantly better ΔR than the vanilla approach. HBS was the worst performing again, being barely better than the vanilla method on the 100 and 200 vertex networks in ΔR and performing equivalently to it in the 500 vertex network.

IAS and RLS did similarly well in terms of RE. Both strategies tied with the vanilla method for the 100 vertex network, RLS alone tied vanilla for the 200 vertex network, and IAS tied vanilla for the 500 vertex network. HBS had very poor RE once again, falling short of the vanilla method in all network sizes.

These results are consistent with the results for the other optimization methods, where the inclusion of RLS or IAS edge addition leads to great increases in ΔR compared to the vanilla method, but similar values for RE. This indicates a large

Table 30: Memetic Algorithm - comparison of edge addition strategies

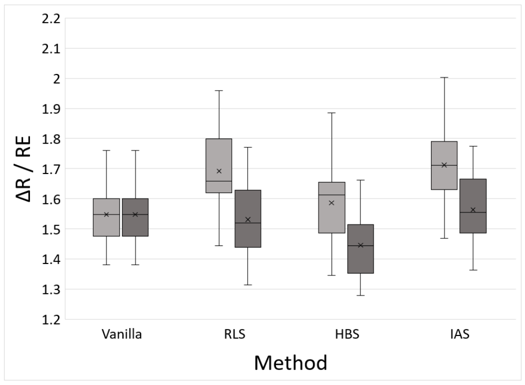
	100 vertex		**200 vertex**		**500 vertex**	
Addition Strategy	$\boldsymbol{\Delta R}$	$\boldsymbol{RE}$	$\boldsymbol{\Delta R}$	$\boldsymbol{RE}$	$\boldsymbol{\Delta R}$	$\boldsymbol{RE}$
None (vanilla)	1.55	**1.55**	1.56	**1.56**	1.53	**1.53**
RLS	**1.69**	**1.53**	1.75	**1.56**	1.76	1.51
HBS	1.59	1.45	1.60	1.51	1.54	1.46
IAS	**1.71**	**1.56**	**1.83**	1.51	**1.79**	**1.54**

increase in the robustness of the network, but with a proportional increase in the number of edges present.

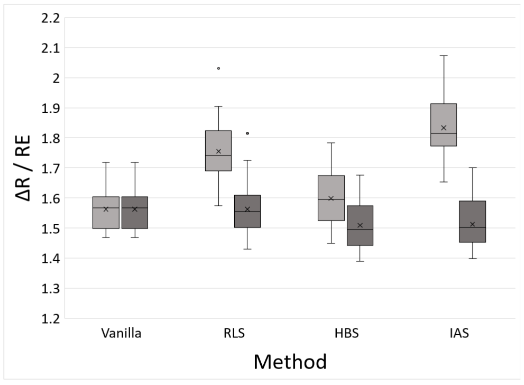
These experiments show that overall, the inclusion of edge addition leads to significantly greater improvements to network robustness compared to the vanilla edge rewiring method. This means that networks which have their structure optimized using a hybrid edge rewiring/edge addition method will be significantly more resilient to attacks and damage.

This increased robustness can come with a cost however. In some situations, the inclusion of edge addition into the optimization procedure can lead to an increase in the number of edges in the network that is disproportionate to the increase in robustness. This means that the modifications necessary to get a network to that high level of robustness may be more costly (in terms of time, money, etc.) in a real world network compared to the vanilla edge rewiring method. This is not always the case, however. In fact, for the largest tested network size, the inclusion of edge addition was strictly better to the vanilla method in terms of both ΔR and RE. This may indicate the hybrid approach to network robustness may be a more efficient way of improving the robustness of larger networks.

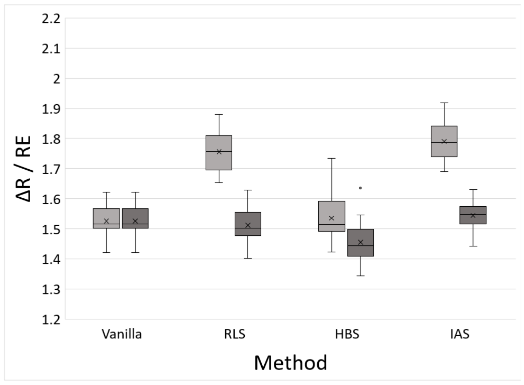
Of the three tested linking strategies, IAS performed consistently well across all experiments. RLS had equivalent performance to IAS in some cases, but overall did not perform quite as well. HBS performed very poorly across the board, as using this strategy in the hybrid approach has no substantial benefits over the vanilla approach.



(a) 100 vertices



(b) 200 vertices



(c) 500 vertices

Figure 18: Comparison of three linking strategies and vanilla method for MA. ΔR shown in light grey, RE shown in dark grey.

5.4 Metaheuristic Comparison

Table 31 shows a comparison between the four metaheuristic optimizers using the hybrid edge rewiring/addition method with IAS addition. IAS addition was used for this comparison because it was consistently good across all tests. This table uses the same add chances as the IAS results presented in the previous section.

This table shows that under the hybrid approach, there was no definitive best optimization method for all experiments. In the 100 vertex network, the GD optimizer had the best ΔR value but the worst RE value, while the MA optimizer had the best RE and was tied for the second best ΔR with the GA optimizer. This indicates that, for a small network, the GD optimizer will give the best overall robustness, but the MA optimizer will be best for increasing robustness without greatly increasing the

Table 31: Comparison of optimizer performance with hybrid approach.

	100 vertex		**200 vertex**		**500 vertex**	
Optimizer	$\boldsymbol{\Delta R}$	$\boldsymbol{RE}$	$\boldsymbol{\Delta R}$	$\boldsymbol{RE}$	$\boldsymbol{\Delta R}$	$\boldsymbol{RE}$
Great Deluge	**1.91**	1.27	**1.81**	1.46	1.69	**1.54**
Simulated Annealing	1.66	1.43	**1.78**	1.50	**1.83**	**1.54**
Genetic Algorithm	1.75	1.50	**1.82**	**1.54**	1.77	**1.53**
Memetic Algorithm	1.71	**1.56**	**1.83**	**1.51**	**1.79**	**1.54**

size of the network.

In the 200 vertex network, all 4 optimization methods had statistically equivalent ΔR values, while the GA and MA optimizers slightly beat out the GD and SA optimizers in terms of RE. In the largest tested network size, all 4 optimization methods had similar RE values, while the SA and MA optimizers had the best ΔR.

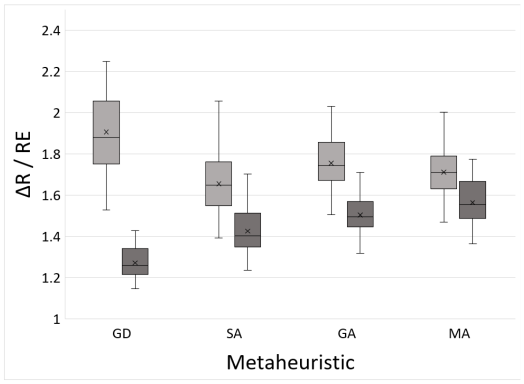
Overall, these results indicate that the MA optimizer was consistently good across all tested network sizes, with the GA optimizer being fairly close in performance. The GD optimizer can lead to great increases in robustness for small networks, at the cost of a disproportionate increase in network size. The SA optimizer is also fairly effective for larger networks, being either tied for first or a close second to the MA optimizer.

5.5 Real World Network

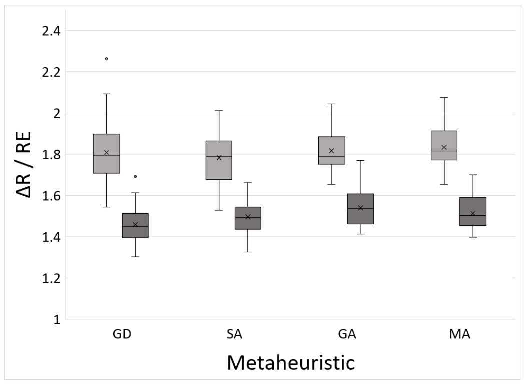
Table 32 shows the effect that the inclusion of edge addition had on the optimization of the robustness of a real world network. For all tests, the hybrid approach used 5% IAS addition.

This table shows that for all four optimization methods, the hybrid approach led to very significant improvements in both ΔR and RE compared to the vanilla approach. The GD and SA optimizers, in particular, had an enormous improvement with the hybrid approach, while the GA and MA optimizers saw more moderate improvement.

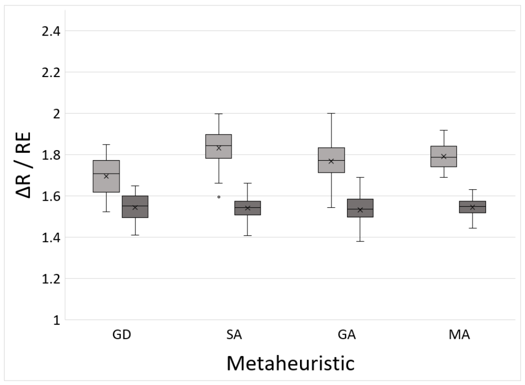
It is notable that these values for ΔR and RE are much higher than the results from the previous sections. This is largely due to the network being tested here



(a) 100 vertices



(b) 200 vertices



(c) 500 vertices

Figure 19: Comparison of the four metaheuristics using the hybrid approach with IAS edge addition. ΔR shown in light grey, RE shown in dark grey.

starting off with a much lower R value, leaving a lot more room for improvement compared to the synthetic networks.

Table 33 shows a comparison among the four optimizers on the real world network, all using the hybrid approach with 5% IAS. Of these, the GD optimizer was the best performing, with the SA optimizer coming in second. The GA and MA optimizers were tied for last with very similar results.

These experiments show that the inclusion of a hybrid approach can be extremely effective for improving the robustness of real world networks. One possible reason for this could be that the tested network was relatively sparse, so including additional edges had a more significant effect than it would on a denser network.

It is notable that the best performing optimizer on the real world network was the

Table 32: Hybrid vs. vanilla method on real world network.

Optimizer	**Method**	$\boldsymbol{\Delta R}$	$\boldsymbol{RE}$
Great Deluge	Vanilla	2.73	2.73
	Hybrid	**4.83**	**3.46**
Simulated Annealing	Vanilla	2.67	2.67
	Hybrid	**4.05**	**3.34**
Genetic Algorithm	Vanilla	2.38	2.38
	Hybrid	**2.96**	**2.67**
Memetic Algorithm	Vanilla	2.37	2.37
	Hybrid	**2.98**	**2.67**

Table 33: Comparison of optimizers for real network using hybrid approach.

Optimizer	$\boldsymbol{\Delta R}$	$\boldsymbol{RE}$
Great Deluge	**4.83**	**3.46**
Simulated Annealing	4.05	3.34
Genetic Algorithm	2.96	2.67
Memetic Algorithm	2.98	2.67

great deluge algorithm, while this metaheuristic did not perform nearly as well on the synthetic networks. One reason for this may be due to the GD optimizer performing a larger amount of edge addition relative to the other metaheuristics. In the real world network, the GD optimizer saw a 40% increase in the number of edges in the network, compared to 21% for SA, 11% for GA, and 12% for MA. Since the initial network for this test was so sparse, more edge addition may have been particularly effective at increasing its robustness. Since the synthetic networks were more dense, they did not always benefit as much from larger amounts of edge addition and in many cases this extra edge addition may have resulted in too many unnecessary new edges.

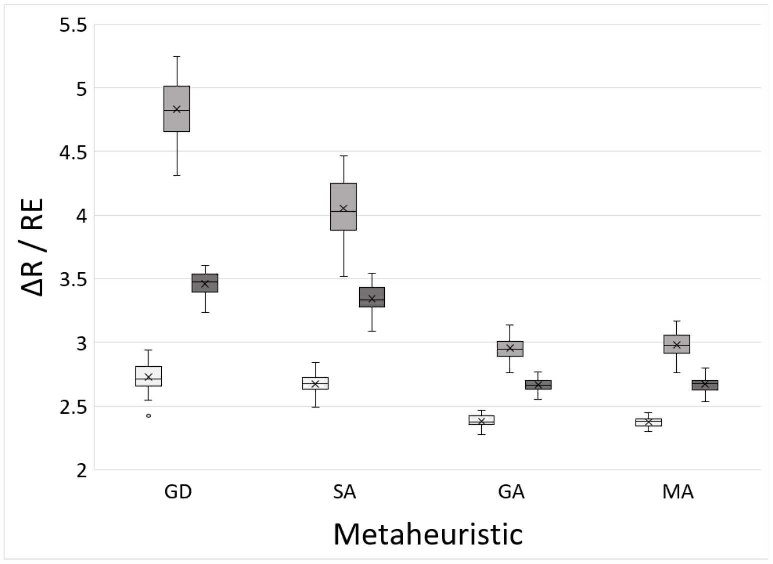


Figure 20: Comparison of the four metaheuristics on the power grid network using both the vanilla and hybrid methods. The ΔR / RE for the vanilla method is shown in the lightest grey, the ΔR for the hybrid method is shown in the middle grey, and the RE for the hybrid method is shown in dark grey.

6 Conclusion & Future Work

The use of a hybrid edge rewiring/edge addition approach to the problem of network robustness optimization has some advantages to the vanilla edge-rewiring-only method. In all experiments, the inclusion of edge addition led to networks with larger increases in R value than networks optimized using the vanilla method. In some cases, this increase in robustness also coincided with an increase in RE value, indicating that the change in robustness was disproportionately high compared to the number of new edges added to the network.

Three different methods of edge addition were tested, and the best performing method was the increased assortativity linking strategy (IAS). This method of edge addition prioritized the addition of links that increase the assortativity of the network, helping to promote the onion-like structure that is often found in highly robust networks.

Experiments performed on a real world power grid network showed that the hybrid approach performed significantly better than the vanilla approach in terms of both the flat increase in R value and in RE value. This result shows that the hybrid approach may have valuable applications to increasing the stability of real world networks.

Of the four tested metaheuristic optimization methods, all performed well in some circumstances. In the synthetic network tests, the memetic and genetic algorithms were the best performing overall. For the real world network, the great deluge algorithm performed best while the MA and GA optimizers did not perform as well. These results indicate that the best metaheuristic to use for the optimization of a network may be situational.

There are a multitude of avenues for expansion of this topic in future work. One area that requires more exploration is in testing a larger variety of networks, both real and synthetic. The experiments performed in this thesis on the power grid network hinted that low density networks may benefit more from the inclusion of edge addition than higher density networks do. Further experimentation will be required to test this hypothesis. Other graph properties such as degree distribution or clustering coefficient may also influence the effectiveness of this hybrid approach. The application of this hybrid approach to other types of networks may also prove to be useful. The real world network tested in this thesis had to be modified since the implementation was only designed to work with simple graphs. This is not an ideal approach when looking to optimize a real network, so modifications allowing the optimization of non-simple networks are worth exploring.

Another avenue of future research is the use of alternate methods of measuring robustness that prioritize the efficient addition of edges to the network. This thesis made use of the RE value as a post-optimization measurement of performance. Using RE value or a similar measure as the fitness value during optimization may influence which edges are selected for addition, potentially leading to more robust networks.

More complex mechanisms for determining how and when edge addition should be performed may also be an interesting expansion on this topic. This thesis used a simple probability at each iteration of the optimization algorithm to determine if a new edge should be added or a step of the vanilla optimization should be performed. Other methods such as a dynamically changing probability to add an edge may be worth exploring.

References

[1] Zhi-Xi Wu and Petter Holme. Onion structure and network robustness. *Physical Review E*, 84(2):026106, 2011.

[2] Hans J Herrmann, Christian M Schneider, André A Moreira, José S Andrade Jr, and Shlomo Havlin. Onion-like network topology enhances robustness against malicious attacks. *Journal of Statistical Mechanics: Theory and Experiment*, 2011(01):P01027, 2011.

[3] Christian M Schneider, André A Moreira, José S Andrade, Shlomo Havlin, and Hans J Herrmann. Mitigation of malicious attacks on networks. *Proceedings of the National Academy of Sciences*, 108(10):3838–3841, 2011.

[4] Mark Newman. *Networks*. Oxford University Press, 2018.

[5] Pierre Buesser, Fabio Daolio, and Marco Tomassini. Optimizing the robustness of scale-free networks with simulated annealing. In *International Conference on Adaptive and Natural Computing Algorithms*, pages 167–176. Springer, 2011.

[6] James Paterson and Beatrice Ombuki-Berman. Optimizing scale-free network robustness with the great deluge algorithm. In *International Conference on Industrial, Engineering and Other Applications of Applied Intelligent Systems*, pages 434–446. Springer, 2018.

[7] Mingxing Zhou and Jing Liu. A memetic algorithm for enhancing the robustness of scale-free networks against malicious attacks. *Physica A: Statistical Mechanics and its Applications*, 410:131–143, 2014.

[8] Toshihiro Tanizawa, Shlomo Havlin, and H Eugene Stanley. Robustness of onionlike correlated networks against targeted attacks. *Physical Review E*, 85(4):046109, 2012.

[9] Vitor HP Louzada, Fabio Daolio, Hans J Herrmann, and Marco Tomassini. Smart rewiring for network robustness. *Journal of Complex Networks*, 1(2):150–159, 2013.

[10] Bai Liang, Xiao Yan-Dong, Hou Lv-Lin, and Lao Song-Yang. Smart rewiring: Improving network robustness faster. *Chinese Physics Letters*, 32(7):078901, 2015.

[11] Tie Qiu, Aoyang Zhao, Feng Xia, Weisheng Si, and Dapeng Oliver Wu. Rose: Robustness strategy for scale-free wireless sensor networks. *IEEE/ACM Transactions on Networking*, 25(5):2944–2959, 2017.

[12] Tie Qiu, Jie Liu, Weisheng Si, and Dapeng Oliver Wu. Robustness optimization scheme with multi-population co-evolution for scale-free wireless sensor networks. *IEEE/ACM Transactions on Networking*, 27(3):1028–1042, 2019.

[13] An Zeng and Weiping Liu. Enhancing network robustness against malicious attacks. *Physical Review E*, 85(6):066130, 2012.

[14] Shuai Wang and Jing Liu. Enhancing the robustness of complex networks against edge-based-attack cascading failures. In *2017 IEEE Congress on Evolutionary Computation (CEC)*, pages 23–28. IEEE, 2017.

[15] Mingxing Zhou and Jing Liu. A two-phase multiobjective evolutionary algorithm for enhancing the robustness of scale-free networks against multiple malicious attacks. *IEEE Transactions on Cybernetics*, 47(2):539–552, 2016.

[16] Wenfeng Liu, Maoguo Gong, Shanfeng Wang, and Lijia Ma. A two-level learning strategy based memetic algorithm for enhancing community robustness of networks. *Information Sciences*, 422:290–304, 2018.

[17] Xian-Bin Cao, Chen Hong, Wen-Bo Du, and Jun Zhang. Improving the network robustness against cascading failures by adding links. *Chaos, Solitons & Fractals*, 57:35–40, 2013.

[18] Xiaoke Zhang, Jun Wu, Cuiying Duan, Michael TM Emmerich, and Thomas Bäck. Towards robustness optimization of complex networks based on redundancy backup. In *2015 IEEE Congress on Evolutionary Computation (CEC)*, pages 2820–2826. IEEE, 2015.

[19] Zhongyuan Jiang, Mangui Liang, and Dongchao Guo. Enhancing network performance by edge addition. *International Journal of Modern Physics C*, 22(11):1211–1226, 2011.

[20] Yiguang Bai, Sanyang Liu, and Zhaohui Zhang. Effective hybrid link-adding strategy to enhance network transport efficiency for scale-free networks. *International Journal of Modern Physics C*, 28(08):1750107, 2017.

[21] Xingpei Ji, Bo Wang, Dichen Liu, Guo Chen, Fei Tang, Daqian Wei, and Lian Tu. Improving interdependent networks robustness by adding connectivity links. *Physica A: Statistical Mechanics and its Applications*, 444:9–19, 2016.

[22] Yui Kazawa and Sho Tsugawa. Effectiveness of link-addition strategies for improving the robustness of both multiplex and interdependent networks. *Physica A: Statistical Mechanics and its Applications*, page 123586, 2019.

[23] Zhaohui Zhang, Sanyang Liu, Yanqi Yang, and Yiguang Bai. A link-adding strategy for improving robustness and traffic capacity in large-scale wireless sensor networks. *Cluster Computing*, 22(3):7687–7694, 2019.

[24] Jinlong Ma, Weizhan Han, Qing Guo, Zhenyong Wang, and Shuai Zhang. A link-adding strategy for transport efficiency of complex networks. *International Journal of Modern Physics C*, 27(05):1650054, 2016.

[25] Roni Parshani, Sergey V Buldyrev, and Shlomo Havlin. Interdependent networks: Reducing the coupling strength leads to a change from a first to second order percolation transition. *Physical Review Letters*, 105(4):048701, 2010.

[26] Christian M Schneider, Nuri Yazdani, Nuno AM Araújo, Shlomo Havlin, and Hans J Herrmann. Towards designing robust coupled networks. *Scientific Reports*, 3:1969, 2013.

[27] Andrey Bernstein, Daniel Bienstock, David Hay, Meric Uzunoglu, and Gil Zussman. Power grid vulnerability to geographically correlated failures—analysis and control implications. In *IEEE INFOCOM 2014-IEEE Conference on Computer Communications*, pages 2634–2642. IEEE, 2014.

[28] Jun Wu, Suo-Yi Tan, Zhong Liu, Yue-Jin Tan, and Xin Lu. Enhancing structural robustness of scale-free networks by information disturbance. *Scientific Reports*, 7(1):1–13, 2017.

[29] Réka Albert and Albert-László Barabási. Statistical mechanics of complex networks. *Reviews of modern physics*, 74(1):47, 2002.

[30] Ryan A. Rossi and Nesreen K. Ahmed. The network data repository with interactive graph analytics and visualization. In *Proceedings of the Twenty-Ninth AAAI Conference on Artificial Intelligence*, 2015.

[31] Mark EJ Newman. Mixing patterns in networks. *Physical Review E*, 67(2):026126, 2003.

[32] Réka Albert, Hawoong Jeong, and Albert-László Barabási. Error and attack tolerance of complex networks. *Nature*, 406(6794):378–382, 2000.

[33] Swami Iyer, Timothy Killingback, Bala Sundaram, and Zhen Wang. Attack robustness and centrality of complex networks. *PloS One*, 8(4), 2013.

[34] Petter Holme, Beom Jun Kim, Chang No Yoon, and Seung Kee Han. Attack vulnerability of complex networks. *Physical Review E*, 65(5):056109, 2002.

[35] Vito Latora and Massimo Marchiori. Efficient behavior of small-world networks. *Physical Review Letters*, 87(19):198701, 2001.

[36] Vitor HP Louzada, Fabio Daolio, Hans J Herrmann, and Marco Tomassini. Generating robust and efficient networks under targeted attacks. In *Propagation Phenomena in Real World Networks*, pages 215–224. Springer, 2015.

[37] Stuart J Russell and Peter Norvig. *Artificial intelligence: a modern approach*. Pearson Education Limited, 2016.

[38] Gunter Dueck. New optimization heuristics: The great deluge algorithm and the record-to-record travel. *Journal of Computational Physics*, 104(1):86–92, 1993.

[39] John Henry Holland et al. *Adaptation in natural and artificial systems: an introductory analysis with applications to biology, control, and artificial intelligence*. MIT press, 1992.

[40] Richard Dawkins. *The selfish gene*. Oxford University Press, 2016.

[41] Pablo Moscato. On evolution, search, optimization, genetic algorithms and martial arts: Towards memetic algorithms. *Caltech Concurrent Computation Program, C3P Report*, 826:1989, 1989.