Weighted Graph Compression using Genetic Algorithms

Emilia Rutkowski

Submitted in partial fulfilment
of the requirements for the degree of

Master of Science

Department of Computer Science
Brock University
St. Catharines, Ontario

Abstract

Networks are a great way to present information. It is easy to see how different objects interact with one another, and the nature of their interaction. However, living in the technological era has led to a massive surge in data. Consequently, it is very common for networks/graphs to be large. When graphs get too large, the computational power and time to process these networks gets expensive and inefficient. This is common in areas such as bioinformatics, epidemic contact tracing, social networks, and many others. Graph compression is the process of merging nodes that are highly connected into one super-node, thus shrinking the graph. The goal of graph compression is to merge nodes while mitigating the amount of information lost during the compression process. Unweighted graphs are largely studied in this area. However, in this thesis, we extend the approaches to compress weighted graphs via genetic algorithms and analyse the compression from an epidemic point of view. It is seen that edge weights provide vital information for graph compression. Not only this, but having meaningful edge weights is important as different weights can lead to different results. Moreover, both the original edge weights and adjusted edge weights produce different results when compared to a widely used community detection algorithm, the Louvain Algorithm. However, the different results may be helpful to public health officials. Lastly, the NSGA-II algorithm was implemented. It was found that NSGA-II is more suitable as a pre-processing tool, in order to find a target compression that introduces a comfortable level of distortion, and then using the single-objective genetic algorithm to achieve an improved solution for the target.

Acknowledgements

I would like to send out an overwhelming and heartfelt thank you to everyone that supported and encouraged me throughout my graduate career. You motivated me at my lowest points to reach my full potential and I would not have gotten this far without each and every one the incredible individuals mentioned below.

Most importantly my graduate supervisor, Professor Sheridan Houghten, who guided my research for the past two years, and took me under her wings. Even though we did not end up going to Scotland or Poland together, we created some lasting memories. Your constant guidance, encouragement, and positivity is something to admire, and I hope to take it with me into my future endeavours. I cannot thank you enough for taking a chance on me and pushing me out of my comfort zone every once in a while.

A special thank you to our system administer, Professor Cale Fairchild, that had to deal with my constant requests to run experiments on the school's servers, and me taking up valuable CPU time to make paper deadlines.

I would also like to thank many computer science professors at Brock University, Professor Beatrice Ombuki-Berman who taught my first artificial intelligence class, and introduced me to evolutionary algorithms. I attribute this class to sparking my interest in evolutionary algorithms. Also a thank you to Professor Ke Qiu, and Dave Bockus for writing my recommendation letters, and seeing my potential.

Subsequently, I have to mention my appreciation for my family. My parents who encouraged me to go into Computer Science from the beginning, and always had confidence in me. My sister who was so proud of my accomplishments, and never doubted my abilities. My significant other that dealt with all the meltdowns, celebrations, nerves, and triumph along the way. Being able to lean on you during the lows and celebrate with you during the highs has strengthened our relationship.

Finally, a shout-out to Mahtay Cafe in downtown St.Catharines for being my office off campus. Having most of my research months coinciding with the SARS-CoV-2 lockdown, having a cafe to be able to go to in order to write has been a breath of fresh air. The change of scenery was paramount to my productivity to finish writing.

Contents

1 Introduction **1**

1.1 Structure of Thesis 2

2 Background **3**

2.1 Graph Theory 3

2.2 Graph Compression 4

2.3 Genetic Algorithms 5

2.4 Epidemics 6

2.4.1 Social Contact Networks 7

3 Literature Review **8**

3.1 Unweighted Graph Compression 8

3.2 Weighted Graph Compression 9

4 Methodology **12**

4.1 Representation 12

4.2 Initialization 13

4.3 Distance 13

4.4 Selection 14

4.5 Mutation 15

4.6 Crossover 15

4.7 Elitism 16

4.8 Fitness Function 16

4.8.1 First Fitness Function 17

4.8.2 Second Fitness Function 17

4.9 Data Sets 18

4.10 Adjusted Rand Index 19

5 Unweighted Graphs vs Weighted Graphs 21

5.1 Introduction 21

5.1.1 Unweighted Fitness Function 22

5.2 Experiment Details 22

5.3 Results and Discussion 22

5.3.1 Unweighted vs First Fitness Function 23

5.3.2 Unweighted vs Second Fitness Function 25

5.4 Conclusion 28

6 Fitness Functions 29

6.1 Introduction 29

6.2 Experiment Details 29

6.3 Results and Discussion 30

6.4 Conclusion 36

7 Original Weights vs Adjusted Weights 37

7.1 Introduction 37

7.2 Experiment Details 37

7.3 Results and Discussion 39

7.4 Conclusion 40

8 Graph Compression vs Community Detection 45

8.1 Introduction 45

8.2 Background 46

8.2.1 Community Detection 46

8.2.2 Louvain Algorithm 46

8.3 Experiment Details 47

8.4 Results and Discussion 47

8.5 Conclusion 50

9 NSGA-II 56

9.1 Introduction 56

9.2 Background 57

9.2.1 NSGA-II 57

9.2.2 Pareto Ranking 57

9.2.3 Crowding Distance 58

9.3 Methodology 58

9.3.1 Elitism 59
9.3.2 Representation 59
9.3.3 Crossover 59
9.3.4 Fitness Functions 60
9.4 Experiment Details 60
9.5 Results and Discussion 61
9.5.1 First Fitness Function 62
9.5.2 Second Fitness Function 65
9.5.3 One Point vs Two Point Crossover 65
9.6 Conclusion 67

10 Conclusion and Future Work **68**
10.1 Weighted vs Unweighted Graphs 68
10.2 First vs Second Fitness Function 68
10.3 Original vs Adjusted Weights 69
10.4 Community Detection 69
10.5 NSGA-II 70
10.6 Future Work 70

Bibliography **75**

Appendices **76**

A Original Graph 06-02 **76**

B Original Graph 04-28 **77**

C Original Graph 07-07 **78**

D Original Graph 07-15 **79**

E Mutation And Crossover Rates **80**

F Distance **84**

List of Tables

4.1 Summary of graph data 19

5.1 Summary of Experimental Parameters 23

5.2 Best fitness value obtained for each compression ratio for weighted graphs using the first fitness function (Section 4.8.1) and unweighted graphs using the unweighted fitness function (Section 5.1.1). 23

5.3 The Adjusted Rand Index between weighted graphs (using the First Fitness Function) and the non-weighted graphs 24

5.4 Best fitness value obtained for each compression ratio for weighted graphs using the first fitness function (Section 4.8.2) and unweighted graphs using the unweighted fitness function (Section 5.1.1). 27

5.5 The Adjusted Rand Index between weighted graphs (using the Second Fitness Function) and the non-weighted graphs 27

6.1 Summary of Experiment Parameters 30

6.2 Similarity of sets of super nodes in best results from fitness function 4.1 vs those from fitness function 4.2. 34

7.1 How weights change when using Equation 4.3 38

7.2 Summary of Experiment Parameters 38

7.3 Similarity of sets of super-nodes in the best results from using the original graphs vs those from using the adjusted graphs. 39

8.1 Adjusted Rand Index: Similarity of sets of super-nodes in best results from Compression vs those from Community Detection. 48

9.1 Example Solutions for Pareto Dominance 58

9.2 Pareto Ranks for Solutions in Table 9.1 58

9.3 Summary of Experiment Parameters 61

9.4 Best Result Given by a Single Objective GA and a Multi Objective GA At 40% Compression Ratio using the First Fitness Function . . . 62
9.5 Best Result Given by a Single Objective GA and a Multi Objective GA At 40% Compression Ratio using the Second Fitness Function . . 65

E.1 Symbol Representation 82

List of Figures

2.1 An example of merging nodes *a* and *b* to produce super-node *ab*. . . . 5
2.2 Genetic Algorithm Outline 6

4.1 An example of a chromosome 13
4.2 How merges can happen at a greater distance than specified. 14
4.3 Single Point Mutation Example 15
4.4 Parent Chromosomes for Two Point Crossover 16
4.5 Result of Two Point Crossover 16
4.6 An example of a possible merge 17

5.1 Weighted Graph 06-02, that was compressed 20% by the First Fitness Function 25
5.2 Unweighted Graph 06-02, that was compressed 20% 26
5.3 Weighted Graph 06-02, that was compressed 20% by the Second Fitness Function 28

6.1 Two nodes merging without any external edges 31
6.2 Two nodes merging with one external edge 31
6.3 Two merging nodes have edges, of the same weight, to the same external node. 32
6.4 Two merging nodes have edges, of varying weight, to the same external node. 32
6.5 Example of merge with equal variation between external nodes 33
6.6 Example of merge with differing variation between external nodes . . 33
6.7 Translating the Second Fitness Function to the First Fitness Function 35
6.8 Translating the First Fitness Function to the Second Fitness Function 35

7.1 Different merges with original weights vs adjusted weights using the First Fitness Function 40
7.2 06-02 with original weights, compressed 50% with First Fitness Function. 41

7.3 06-02 with adjusted weights, compressed 50% with First Fitness Function. . . . 42
7.4 06-02 with original weights, compressed 50% with Second Fitness Function. . . . 43
7.5 06-02 with adjusted weights, compressed 50% with Second Fitness Function. . . . 44

8.1 Louvain Communities compared to First Fitness Function 48
8.2 Louvain Communities compared to Second Fitness Function 49
8.3 $04 - 28_{adj}$, 50% compression, First Fitness Function 51
8.4 $04 - 28_{adj}$, 50% compression, Louvain Algorithm 52
8.5 06-02, 50% compression, First Fitness Function 53
8.6 06-02, 50% compression, Second Fitness Function 54
8.7 06-02, 50% compression, Louvain Algorithm 55

9.1 NSGA-II Chromosome Example 59
9.2 Parent Chromosomes for One Point Crossover 60
9.3 Result of One Point Crossover with index 3 60
9.4 Similarity Curves for all Graphs using the first fitness function 63
9.5 Similarity Curves for all Graphs using the second fitness function . . 64
9.6 The last generations when 06-02 was run with One Point and Two Point Crossover for 250 generations using the First Fitness Function . 66
9.7 The last generations when 06-02 was run with One Point and Two Point Crossover for 250 generations using the Second Fitness Function 67

A.1 Data set from “Infectious: Stay Away Event” Day 06-02 76

B.1 Data set from “Infectious: Stay Away Event” Day 04-28 77

C.1 Data set from “Infectious: Stay Away Event” Day 07-07 78

D.1 Data set from “Infectious: Stay Away Event” Day 07-15 79

E.1 The mean best fitness value with 95% confidence intervals for the First Fitness Function 4.8.1. See Table E.1 for symbol representation. . . . 81
E.2 The mean best fitness values with 95% confidence intervals for the Second Fitness Function 4.8.2. See Table E.1 for symbol representation. 83

F.1 The mean best fitness values with 95% confidence intervals for the First Fitness Function 4.8.1. Where solid symbols show the results for 10% compression, and the hallow symbols show the results for 20% compression. 85
F.2 The mean best fitness values with 95% confidence intervals for the Second Fitness Function 4.8.2.Where solid symbols show the results for 10% compression, and the hallow symbols show the results for 20% compression. 86

Chapter 1

Introduction

When there is a disease outbreak, the world must rally together to contain the disease. The individuals infected must be isolated in order for the disease to be contained and hopefully eradicated. The idea of isolation and quarantine goes back to the 14th century when repeated waves of plagues swept across Europe, and Viscount Bernabo of Reggio, Italy demanded that every person infected with the Plague be taken outside of the city to the fields to either die or recover [23]. While we no longer banish infected individuals to a field, the idea of quarantine is still a widely used and very effective strategy to contain a disease outbreak [41].

There is no denying that information is power. The more accurate information available to public health officials will result in a more effective and specific plan to contain the disease. This means that the only areas that will have to adhere to restrictions and physical distancing are those in high risk. One of the ways to get this information is through contact tracing. Areas that have been able to successfully apply contact tracing were able to control the outbreak more effectively [8] [32]. Information dealing with interactions between individuals are kept in a *contact network*, or weighted graph, where individuals are represented as nodes, the edges between them represent an interaction, and the edge weight represent the strength of their interactions. Note that in this thesis, we use the terms network and graph interchangeably.

Since the information era, data is not only being collected, but sought out by many companies [31]. The amount of data that has to be stored and analyzed, including data in large contact networks, is tremendous. However, when it comes to graphs, there is a solution that can help handle this difficulty - *compression*. In this thesis, we use genetic algorithms to compress four weighted networks that hold information pertaining to the strength of interactions between individuals. We will show that edge weights provide critical information for graph compression. Moreover, having meaningful edge weights is just as important since different weights lead to different compressions. Similarly, when we compare our genetic algorithm with a well known community detection algorithm, the sets of super-nodes/communities suggested are also different. Nevertheless, these different results may be helpful to public health officials. Lastly, we will show that the multiple objective genetic algorithm, NSGA-II, is more fitted as a pre-processing tool in order to find a target compression ratio that introduces a comfortable level of distortion before using the single objective genetic algorithm to obtain the final solution.

1.1 Structure of Thesis

This thesis is organized as follows. Chapter 2 contains all the background information needed in order to understand the basic principles that was used in the study. We follow that up by how the problem has been tackled in the past in Chapter 3, and the methodology and specific details about our implementation in Chapter 4. Subsequently we have all the different experiments run with our methodology, and discussions about them in Chapters 5, 6, 7, 8, and 9. Finally, we recap all the results and relate them to an epidemic viewpoint in Chapter 10 as well as give recommendations about how to take this research to the next level.

Chapter 2

Background

This section will give a brief introduction to topics that will be discussed and/or used in this thesis. We will first start with a summary of some main graph theory concepts. Next, the fundamental strategy of graph compression, and the technique used in this thesis, will be presented. This is followed by an overview of genetic algorithms, which we use in the study to compress the graphs. Finally, we discuss epidemics and contact networks as this study applies graph compression to epidemic modelling.

2.1 Graph Theory

A graph G is composed of a set of nodes (or vertices) V and a set of edges E where each edge connects two nodes. Edges are denoted by a pair of vertices (u, v) where the edge is between the nodes u and v, and the *size* of a graph G is the total number of edges $|E|$ in G. The *order* of a graph G is the number of nodes $|V|$ in G. The *degree* of a vertex is the number of edges connected to that vertex. Note that the sum of the degrees in a graph G is exactly two times the size of a graph. This is because an edge will only ever connect two nodes, and both of the nodes will take that edge into account in their degree. In *weighted graphs*, the edges have an associated weight that indicate the strength of the connection between those two nodes. The weight of an edge (u, v) is denoted by $w(u, v)$. When there are no weights associated, it is assumed that all edges have the same weight. *Annotated graphs* are when each node has an associated description. The annotation associated for the node u is denoted by $\lambda(u)$.

For every type of graph the following terms hold true. If there is an edge (u, v) then the nodes u and v are said to be *adjacent*. If two nodes are adjacent, they are also considered to be *neighbors*. The edge (u, v) is *incident* to the nodes u and v. The *degree* of a node, is the number of edges incident to it. A *path* between two nodes is the sequence of edges that connects them, with no node being visited more than once. The *distance* between two nodes is the length of the shortest path between them. A *complete graph* is a graph where there is an edge connecting each node to every other node to the graph. Information on graphs can be found in the books [39, 26].

2.2 Graph Compression

Graph compression is the act of merging nodes together to decrease the order of the graph. When two nodes are merged together, they form a *super-node*. Essentially this one *super-node* is the equivalent of the two original nodes. As a result, the graph's order decreases by one for each merge that occurs. The edges incident to the nodes that were merged need to be modified into *super-edges* since instead of pointing to the original nodes, they are now pointing to the super-node. During decompression, the super-node is deconstructed into its original nodes, and the super-edges incident to the super-node are now connected to all the nodes that composed the super-node. Therefore, when a graph G is compressed into G', and then subsequently decompressed into G'', there may be edges in G'' that do not exist in G - these are called *fake edges*. Fake edges are a form of distortion introduced to the graph due to the compression process [3]. It follows that every super-node when decompressed creates a complete sub-graph.

When compressing and decompressing a weighted graph, the weight of existing edges may also change. This would be another form of distortion introduced to the graph due to compression. When transforming the edges into super-edges, the weights of the edges are averaged out [38]. Note that if there is no edge present between two nodes in the graph, it is treated as an edge of weight zero. This is shown in Figure 2.1. As seen, when node a and b merge to form super-node ab a self-edge is created with weight $w(a, b)/1 = 1/1 = 1$ since only one edge can be present between two nodes. The neighbor node c has an edge that that connects to a, but not b. However, when the super-node ab is created, it now has to point to both nodes. So, the edge (a, c) is transformed into an edge (ab, c) with weight $(w(a, c) + w(b, c))/2 = (4 + 0)/2 = 4/2 = 2$. Therefore, when decompressed, there are now two edges - (a, c) and (b, c) both with weight two.

The ideal compression would be to make G'' equal to G, which is considered a lossless compression. If G'' is not equal to G, it is considered a lossy compression. However, a lossless compression is very unlikely. Consequently, we focus on minimiz-

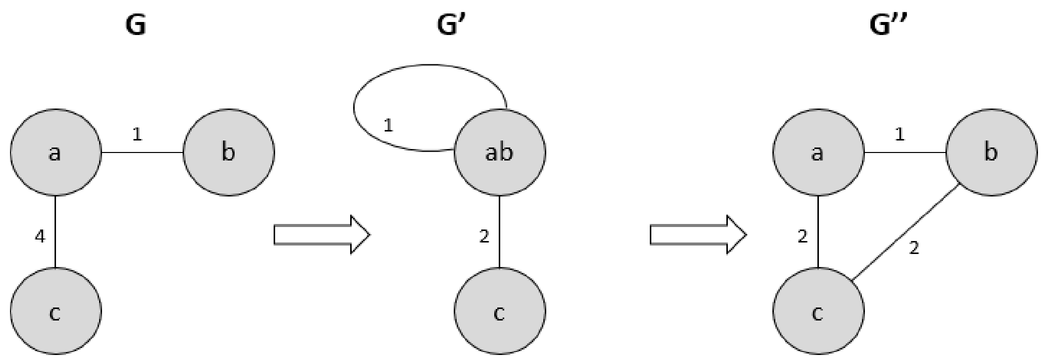


Figure 2.1: An example of merging nodes a and b to produce super-node ab.

ing the distortion due to compression by as much as possible.

2.3 Genetic Algorithms

A genetic algorithm (GA) is an algorithm based on Darwin's idea of "survival of the fittest". It is inspired by biological evolution. In a GA, like living organisms, there is a population made up of chromosomes. In biology these chromosomes make up a person or an animal, but in a GA these chromosomes represent a solution to the problem that the GA is trying to solve. Note that the genetic algorithm is an optimization algorithm. Genetic algorithms are used with NP-Hard problems when time is of importance. GAs may find an acceptable solution in a reasonable amount of time. In the current section, we will review the basic concepts of genetic algorithms found in [27], and a more specific description of the genetic algorithm used in these studies is found in Chapter 4.

Much like in biology, the population in the GAs evolves throughout generations by going through crossover and mutation. Crossover is when two chromosomes exchange genes to make child chromosomes. Mutation is when the chromosome undergoes a slight variation.

Throughout these generations, the chromosomes, or, more accurately, the solutions provided by the GA, should be getting better and better until the whole population converges. Convergence is when every chromosome in the population is almost the same. In the end, we should be left with a near optimal solution in a reasonable amount of time.

To track which chromosomes are better than others - and hence which should be given priority during evolution - each chromosome has to go through a fitness

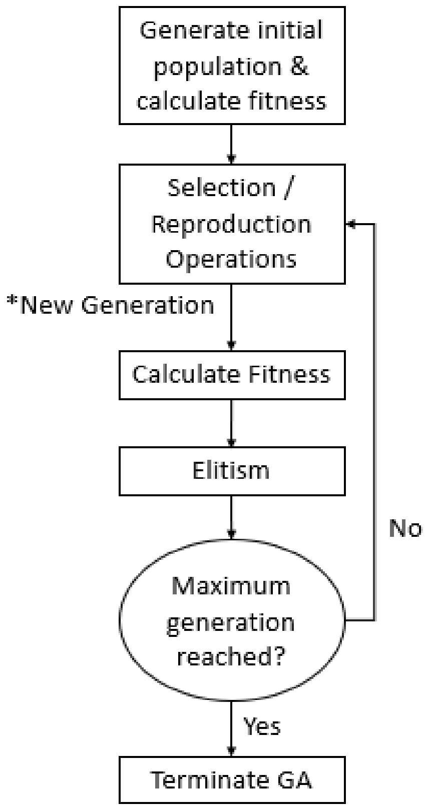

Generate initial population & calculate fitness
Selection / Reproduction Operations
*New Generation
Calculate Fitness
Elitism
No
Maximum generation reached?
Yes
Terminate GA

Figure 2.2: Genetic Algorithm Outline

function. This function will assign a fitness value to the chromosome, which gives insight into how good the proposed solution is. The entire flow of genetic algorithms can be seen in Figure 2.2.

2.4 Epidemics

This study will focus on graphs related to epidemics. Epidemics are defined as a widespread occurrence of a disease within a community. Public health officials need information and data when making choices about how to reduce the spread of such diseases to the masses. In an epidemic, the detection and containment of those with the disease, or most likely to have the disease, is of upmost importance to fighting the spread of the disease [42]. The recent SARS-CoV-2 pandemic has proved that those who were able to implement effective contact tracing were able to control the illness more successfully than those who did not [18, 32, 8, 11]. Epidemics are spread through social contact between people. Therefore, if we know who is in constant contact, we can predict where the disease will spread, and who may get affected.

Therefore, we will be using social contact networks (or graphs) in our study.

2.4.1 Social Contact Networks

A social contact network is essentially a weighted graph where the nodes represent individuals, the edges represent that the two individuals have a connection, and the weight of the edge is the strength of the relationship between the two individuals. The weights are an important part of social networks, as they essentially give us the risk shared between the individuals. The higher the weight, the more of a risk of transmission of the disease from one individual to another. For example, two individuals meeting for a short time, outside, with masks on have a much lower chance of infecting each other than two individuals who meet for hours, in an enclosed space, with no protective measures. We will be applying graph compression to networks such as these.

Chapter 3

Literature Review

In this chapter, papers exploring graph compression strategies are summarized and reviewed. Note that the compression that occurs when zipping a file on the computer is not sufficient as it has no meaning of the merges it is creating, and therefore, does not maintain important structure in the graphs. Here, we look at techniques that compress both unweighted and weights graphs, while aiming to maintain the structure. This serves as a literature review and will be where the inspiration for our study originates.

3.1 Unweighted Graph Compression

Graph compression is a problem that is widely researched. Many studies were performed on minimizing the size of a graph [3, 5]. In those studies, it was noticed that the algorithm would merge nodes far away from each other in the graph. [12] aimed to correct this, by enforcing a local merge. A node could only merge with a node that was at most x distance away, in terms of number of edges. The study looked at a distance of 3, 5, and 10. The study concluded that this restriction produces more meaningful compression, and reduced the overall distortion (fake edges created) for two out of the three graphs studied.

The same conclusion was found in [4], which applied compression to unweighted graphs to discover cities in role playing games. The distances of 3 and 5 were tested. In 11 out of the 12 experiments, the distance of 3 resulted in the same or a better compression result than a distance of 5.

In addition to a local merge, [3] also found that high crossover, and low mutation is better for compression. It was said that mutation is disruptive to the graphs structure. This study also had a different definition of similarity. The fitness function preferred nodes that had a high number of neighbors in common - with a bonus if the nodes were neighbors themselves.

Zakirov *et al.* introduced a Non-dominated Sorting Genetic Algorithm (more commonly known as NSGA-II) to graph compression in [43]. NSGA-II essentially attempts to optimize two fitness functions at the same time, and sorts the different solutions produced using Pareto ranking. The advantage of using NSGA-II over a generic GA is that at the end of the run, the algorithm gives an array of possible solutions at different compression rates. This way researchers are able to examine the connectivity of the graph more thoroughly as it gives a more natural view of the compression range.

Collins *et al.* also used NSGA-II in their study [5]. However, it was only used as a primitive test to pick a good compression rate. The NSGA-II algorithm took into consideration two factors: compression rate and similarity that was based on reducing the number of fake links created due to compression. Once a seemingly ideal compression rate was chosen, the study ran the generic GA at the specified ratio. The study found the solution found by the single objective GA was significantly better than the solution found by NSGA-II at the same compression rate.

3.2 Weighted Graph Compression

Weighted graph compression has more information related to each edge than an unweighted graph. We are no longer just considering if the two nodes are related or not. We are now interested in how similar/close the two nodes are. This is a huge obstacle to tackle when considering weighted graph compression. In addition to the creation of new edges, we also need to think about the addition/loss of some edge weights when creating super-edges. Becuase of this, there is more research concerning unweighted graphs, but the study of compressing weighted graphs is very important, and has been explored more recently.

Newman demonstrated how any weighted graph can be converted to an unweighted multi-graph [25]. A multi-graph is an unweighted graph that can have multiple edges between nodes. The number of edges between nodes can be viewed as the "weight", and this gives insight into how similar/closely connected the two nodes are. Through transforming weighted graphs to multi-graphs, Newman was able to apply concepts and techniques developed for unweighted graphs, with minor tweaks, to weighted graphs. This includes the community detection algorithm proposed by [10].

Toivonen *et al.* had described two forms of the compression problem in [38]: (1) simple and (2) generalized. Both take into account the weight of the edges in the original graph and the corresponding decompressed graph where the simple problem compares each edge individually and the generic problem compares all paths of length λ. Note that the simple problem is the general problem when $\lambda = 1$. The study found that weighted graphs can be compressed efficiently, and is a promising pre-processing step for large graphs. In this paper, we will focus on the simple problem.

Many algorithms to compress graphs are done in a pair-wise compression, meaning that one node is merged at a time. Khan *et al.* stated that this strategy can be very inefficient and suggested using a set-based algorithm instead to compress multiple nodes at once. [17] modifies the Set-Based Graph Summarization (SAGS) algorithm from [16] in order to apply it to weighted graphs. The process was essentially the same, with the only difference being that after finding candidate nodes, highly similar nodes were removed using post-pruning for edge weights. Afterwards, the edge weights were averaged out for the resulting sets.

Other researchers had a similar idea and used different methods to find sets of nodes to merge together. Serafino uses a tree to suggest merges [33]. The algorithm proposed is named MDT-COMPRESS. The algorithm computes the MDTree of the graph, and then merges the nodes from that tree in a bottom-up fashion. The resulting compressed graph is a simple graph - meaning that there are no self-loops. Therefore, the edge weights in each super-node are lost. In the results, MDT-COMPRESS proved to be more efficient, by requiring less time to run. The root mean square error (RMSE) was also lower than the other algorithms referenced in the paper. Finally an experiment was run to observe the performance of the compressed graph vs the original graph in a clustering problem. MDT-COMPRESS had the closest results to the uncompressed results than the other algorithms referenced in the paper.

Adler and Mitzenmacher propose an algorithm FIND-REFERENCE to compress directed weighted graphs by using an affinity graph and a minimum spanning tree [1]. FIND-REFERENCE attempts to find nodes that share multiple neighbors. An important property to note is that references cannot create a cycle. Therefore, if node s is a reference for node t which is a reference for node u, then node u cannot be a reference for node s. This algorithm first computes the affinity graph G_S for graph G, and the minimum directed spanning tree D for G_S. Subsequently, if node i points to node j in D, then those two nodes are compressed in G. In experiments, it is proved that the results obtained by FIND-REFERENCE are statistically significant to the results obtained by Huffman-based alternatives.

Finally, Liu *et al.* took it a step further and studied compressing weighted time-evolving networks [22]. This is more challenging due to the dimension of time. At each time interval, the edges or the edge weights can change. The algorithm proposed, named MaxErrComp, runs in a greedy fashion. A hierarchical cluster tree is created, and the algorithm gradually increases the clustering cutoff value to obtain clusters. These clusters are then merged, averaging their edge weights. Since the algorithm uses a cutoff value, the compression error is bounded at any time interval in the dynamic graph. This process was compared with a random algorithm, and a static compression algorithm that compressed each time interval of the dynamic graph individually. MaxErrorComp excelled during the experiments in the encoding cost, preserving shortest path weights, and compression quality.

Chapter 4

Methodology

In this chapter, we take a deeper look into the technique used to generate the sequence of merges used to compress the graphs. All parts of the genetic algorithm are explained in their own sections to give the reader a full understanding of the methodology used throughout this thesis.

4.1 Representation

The first obstacle to solve is how to represent a valid solution to the problem. In a genetic algorithm, the chromosome is the structure that contains the solution. Usually, the chromosome is a string or array. Our solution consists of pairs of nodes to be merged in order to compress the graph. It is important to note that each node in the graph has a unique ID from 0 to $N_G - 1$ in order to identify the nodes, where N_G is the number of nodes in the original graph.

Firstly, the chromosome's length depends on the compression rate, and the order of the graph. The compression rate, C, is defined as $1 - \frac{N_{G'}}{N_G}$, where $N_{G'}$ is the number of desired nodes in the compressed graph. The compression rate is an input parameter. Recall, that each merge decrements the order of the graph by one. Therefore, it follows that the length of the chromosome is equal to $N_G - N_{G'}$ as well as $N_G * C$, where N_G is the number of nodes in G and C is the compression rate. For example, if G has 100 nodes, and the compression rate is 5%, then the chromosome should contain $(100 * 0.05) = 5$ merges, and the compressed graph, G', will have a total of $(100 - 5) = 95$ nodes. Such a chromosome can be seen in Figure 4.1.

Secondly, since each merge requires two nodes, the chromosome should be able to contain two pieces of information for one merge. Therefore, we needed a two dimensional array as our chromosome where the corresponding indices contain the

root	7	20	32	8	56
offset	99	60	1	5	42

Figure 4.1: An example of a chromosome

two nodes for a single merge. The first dimension, which we will call the root array, holds the ID of the first node to be merged. This can be any node in the graph, and has the range of [0, N_G]. The second dimension, which we call the offset array, holds the offset of the second node ID from the node in the root array. This is how far away the second node ID is from the first node. Therefore, this dimension has a range from [1, $N_G - 1$], so that we do not merge the same two nodes together. An example of a chromosome is shown in Figure 4.1. In this example, if the graph that the chromosome is compressing consists of 100 nodes, the first merge would occur between node 7 and node $(7 + 99) \mod 100 = 6$.

4.2 Initialization

The initial population of chromosomes is randomly generated. For each index in the chromosome, both the root and the offset need to be determined. All the merges in this research will be a local merge, meaning that, nodes are only permitted to merge with nodes within a specified distance, d, also called the *maximum distance*. This is a user input parameter, and will be discussed more in Section 4.3. It is important to note, that edge weights are not taken into consideration when calculating distance. To initialize a chromosome, a random node is first selected. The node's ID is set as the value in the root array. Afterwards, a breadth first search is completed to gather all the neighboring nodes within the specified distance, d. One of these nodes is randomly selected and the offset for that node is calculated. The offset is equal to $n_1 - n_2 \mod N_G$, where n_1 is the node in the root array, and n_2 is the second node to be merged. This occurs for the full length of the chromosome.

4.3 Distance

Distance is the length of the shortest path between two nodes. The distance parameter determines the local nodes that a specific node is able to merge with. If the distance is set to 3, only nodes that are distance 3 away from the active node can merge with it. As mentioned earlier, distance is purely the number of edges between two nodes - edge weights are not taken into consideration when calculating distance. It was

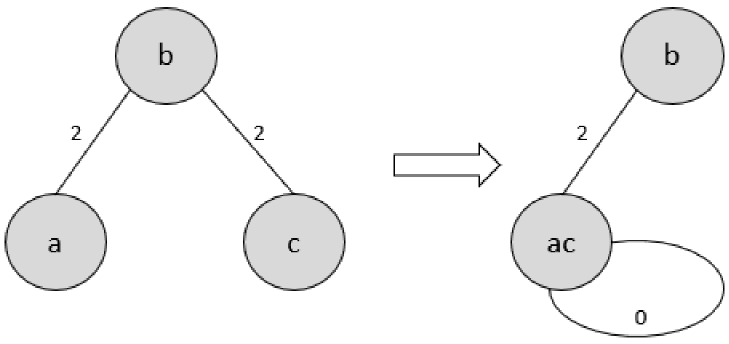


Figure 4.2: How merges can happen at a greater distance than specified.

noticed in [12] that localized merges perform better for compressing graphs. This means that, both the fitness value was lower in the experiments, and the localized merge allowed for more meaningful/sensible merges. Therefore, we set the distance very low. In Appendix F we see that a distance of one is best for the graphs we are considering. It is important to note that even though the distance is one, there is a slight chance that two nodes that are at a greater distance may merge. This is due to the crossover reproduction operation. Lets look at Figure 4.2. Here we see that node a and c ended up merging even though the distance between these two nodes is two. During the early formation of the chromosome, the nodes a and b must have merged. This would now make node a and c a distance one away from each other. Therefore, further down in the chromosome, node a and c can legally merge together. When this chromosome underwent crossover with another chromosome, the first merge (that merged node a and b) was lost, but the second merge that merged a and c, was kept. Therefore, we have a merge at a greater distance than explicitly allowed.

4.4 Selection

The GA utilizes tournament selection to designate two chromosomes from the population to undergo crossover, mutation, or duplication. We want to focus on the solutions with better fitness values, as these are promising solutions found. However, we want to keep some 'worse' solutions as they could contain a few really good merges that can push the solution further when crossed over with a good solution. Tournament selection will select k random chromosomes from the population. The parameter k, is a user input parameter. Throughout the study, we have kept k constant at 5, as this was the value used in earlier work such as [12]. From these 5 random chromosomes, the best solution, according to fitness value, is selected as one of the parents

root	7	20	**32**	8	56
offset	99	60	**1**	5	42

root	7	20	**45**	8	56
offset	99	60	**6**	5	42

Figure 4.3: Single Point Mutation Example

for the reproduction operators. The same process is repeated for the second parent. Subsequently, the two parents are put through reproduction operators, until the next generation has reached the desired population size. The population size is also a user input parameter.

4.5 Mutation

Mutation is a slight random change. It adds diversity especially when convergence occurs. Mutation is very important in a GA as it is exploitative. It optimizes within a promising area, by just changing one merge rather than half the chromosome. The mutation rate is also a parameter meant to be optimized based on the GA and the problem to be solved. This GA uses a single point mutation. This is accomplished by choosing a random index in the chromosome and changing both the root and the offset value in the same fashion as the initialization process. Every other merge in the chromosome is left unchanged. An example of a mutation can be seen in Figure 4.3. Here, the third merge is mutated, while the rest of the chromosome is untouched.

4.6 Crossover

Crossover is when two chromosomes come together to make new chromosomes - these are usually called child chromosomes. Not every chromosome will go through crossover. The crossover rate is a parameter that is meant to be optimized based on the GA and the problem the GA is trying to solve. This GA uses a two point crossover. This works by choosing two random positions in the chromosome, copying the values between and including the two positions to the respective children, and then copying the other values of the chromosome to the opposite child. The crossover is applied to both the root array and the offset array. Figure 4.4 shows the two chromosomes picked to undergo the crossover operation. Figure 4.5 shows the resulting children chromosomes when the first index is 1 and the second index is 3. Crossover is

root A	**7**	**20**	**32**	**8**	**56**
offset A	**99**	**60**	**1**	**5**	**42**

root B	9	15	92	43	21
offset B	99	25	1	10	20

Figure 4.4: Parent Chromosomes for Two Point Crossover

root A	**7**	15	92	43	**56**
offset A	**99**	25	1	10	**42**

root B	9	**20**	**32**	**8**	21
offset B	99	**20**	**1**	**5**	20

Figure 4.5: Result of Two Point Crossover

important in GAs because they are exploratory. Crossover helps discover promising areas in the search space.

4.7 Elitism

The GA also includes elitism. This simply means the worst chromosome in the new generation (after crossover and mutation have taken place) is simply replaced with the best chromosome from the previous generation. This ensures that the GA does not lose track of the best found solution throughout the generations.

4.8 Fitness Function

All the chromosomes must go through the fitness function, giving each chromosome a fitness value. Recall this gives insight into how good the solution represented by the chromosome is. Two fitness functions were used. Both fitness functions measure the distortion between G and G''. Recall, that the distortion will be present in G'' in the form of a new edge, and/or as a change in any edge weights. Ideally, G'' is equivalent to G, therefore, both fitness functions should be minimized. Note that if an edge (u, v) does not exist, it is treated as an edge of weight zero. Let us look at both of the fitness functions in this study and relate it to an example merge in Figure 4.6.

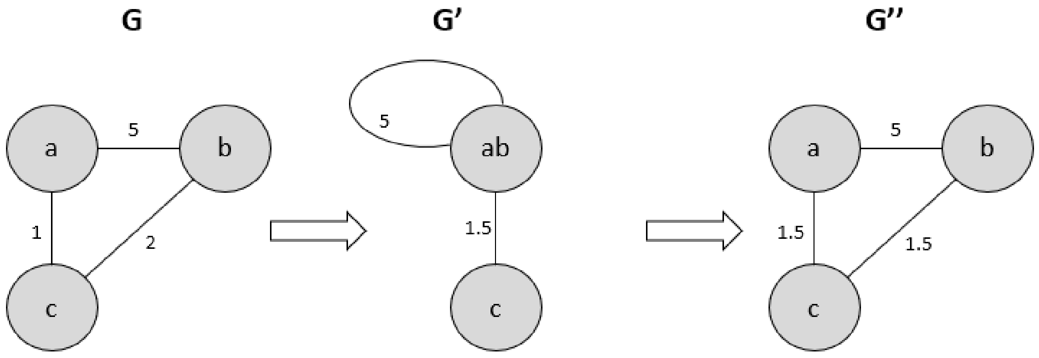


Figure 4.6: An example of a possible merge

4.8.1 First Fitness Function

The first fitness function is the sum of the absolute differences between the edge weights in G and G''. The fitness function is shown in Equation 4.1, where the weight of the edge (u, v) in G is denoted as $wt(u, v)$ and the weight of the edge (u, v) in G'' is denoted as $wt''(u, v)$.

$$fitness_1 = \sum_{u,v \in G} |wt(u, v) - wt''(u, v)| \tag{4.1}$$

The example merge would have the fitness value of 2, as shown below.

$$\begin{aligned}
fitness_1 &= |wt(A, B) - wt''(A, B)| + |wt(A, C) - wt''(A, C)| \\
&+ |wt(B, A) - wt''(B, A)| + |wt(B, C) - wt''(B, C)| \\
&+ |wt(C, A) - wt''(C, A)| + |wt(C, B) - wt''(C, B)| \\
&= |5 - 5| + |1 - 1.5| + |5 - 5| + |2 - 1.5| + |1 - 1.5| + |2 - 1.5| \\
&= 0 + 0.5 + 0 + 0.5 + 0.5 + 0.5 \\
&= 2
\end{aligned}$$

4.8.2 Second Fitness Function

The second fitness function was inspired by Toivonen *et al.* in [38]. It is based on the concept of distance, and resembles Euclidean Distance. It is calculated as

$$fitness_2 = \sqrt{\sum_{u,v \in G} (wt(u, v) - wt''(u, v))^2} \tag{4.2}$$

The example merge would have the fitness value of $\sqrt{1}$, as shown below.

$$\begin{aligned} fitness_2 &= \sqrt{(wt(A,B) - wt''(A,B))^2 + \cdots + (wt(C,B) - wt''(C,B))^2} \\ &= \sqrt{(5-5)^2 + (1-1.5)^2 + (5-5)^2 + (2-1.5)^2 + (1-1.5)^2 + (2-1.5)^2} \\ &= \sqrt{(0)^2 + (0.5)^2 + (0)^2 + (0.5)^2 + (0.5)^2 + (0.5)^2} \\ &= \sqrt{0 + 0.25 + 0 + 0.25 + 0.25 + 0.25} \\ &= \sqrt{1} \end{aligned}$$

Equation 4.2 is also referred to as ‘Root Mean Square Error’. It was used in [17] and [33] to compare compression techniques and capabilities.

4.9 Data Sets

The graphs used in the following experiments come from the “Infectious: Stay Away” event [35, 34, 14] held in 2009. During the event, all participants were given sensors. At each 20 second interval, the individual’s interactions were recorded via proximity of sensors. At the beginning of the day, one individual was labelled as ‘infectious’, and during each interval the epidemic spread to others that were in proximity. At the end of each day, a dynamic graph of all the interactions was created. The event continued for 69 days.

In this study, the data was taken from four different days, and static graphs were created. Each graph is identified by its date. Therefore, the graph 04-28 corresponds to the static graph taken from the 28th of April. Three version of the graph were created for each day. The *unweighted* graph shows an edge between nodes if the two corresponding individuals had any contact throughout the entire day. The *weighted* graph shows an edge between nodes if the two corresponding individuals had contact at any point throughout the day, and has edge weights that show how many interactions the two individuals had throughout the day. Note that if the two individuals spend an entire minute (60 seconds) together, that will be counted as 3 interactions due to the 20 second intervals. Finally, since some of the weights in the weighted graph are very high, an *adjusted* graph was created that split the edge weights into 10 different classes using Equation 4.3. A summary table of the graph data is given in Table 4.1. Note the details given in this table refer to the weighted graphs, not the adjusted weight graphs.

$$w_{adj} = \lceil 20 * e^{0.075*w}/(1.0 + e^{0.075*w}) - 10 \rceil \tag{4.3}$$

Table 4.1: Summary of graph data

Graph	Nodes			Edges		
(Day)	#	Max Deg	Avg Deg	#	Max Wt	Avg Wt
06 – 02	80	10	3.60	144	95	11.33
04 – 28	200	20	7.14	714	108	8.32
07 – 07	234	37	10.27	1202	167	8.20
07 – 15	410	50	13.49	2765	191	6.26

These graphs were picked due to their differing characteristics of order, degree and weight. Graph 06-02 has the least amount of participants - only 80 nodes in the graph. It also has a relatively small average degree, and high average edge weight. This gives insight to the interactions of the individuals that day. The average degree tells us how many people on average an individual had interactions with. The average weight tells us on average how many interactions did an individual have with another individual. Both are important from an epidemic point of view. Of course, at least one interaction is needed to pass on a disease, however, more interactions with the same person, will increase the chances of the disease spreading to that person.

Graph 07-15 has the highest order - a total of 410 nodes in the graph, and a size of 2765 edges. However, from the table, we also know that the individuals on this day interacted with more people on average, for a shorter amount of time. This is because the average degree is the highest of the chosen days, and the average weight of the edges is least of the chosen graphs. In the context of epidemics, this is important information. As stated previously, it only takes one interaction to infect another individual. If each person is interacting with more individuals during the day, the disease has a good chance to spread to more communities in the graph.

For a visualization of the aforementioned graphs, see Appendices A, B, C, and D for the weighted original graphs for each respective day. The GraphViz library [7] was used to render these and all graphs presented in the study.

4.10 Adjusted Rand Index

A measure is needed to compare the super-nodes that the different GAs create. Rand Index, measures the similarity between two sets, U and V, depending on how object pairs are classified. Object pairs can be classified in 4 main ways: (1) the pairs appear in the same class in both partitions, (2) the pairs appear in different classes in both partitions, (3) the pairs appear in the same class in U and different classes in V, and (4) the pairs appear in different classes in U and the same classes in V [13]. The Rand Index will produce a value between 0 and 1, where 0 means the two sets are completely disjoint, and 1 means the two sets are identical. The Adjusted Rand Index, takes this measure and adjusts it for chance by using Equation 4.4. The ARI is bounded by +1, however, may produce a negative number if the index is less than the expected index. The Adjusted Rand Index (ARI), is a widely known and used measure to compare partitions due to its superior performance [37] and is used in this study. The CDLIB Python library of Rossetti, Milli, and Cazabet [28] was used to calculate the Adjusted Rand Index values in this study.

$$ARI = \frac{RawIndex - ExpectedIndex}{MaximumIndex - ExpectedIndex} \tag{4.4}$$

Chapter 5

Unweighted Graphs vs Weighted Graphs

This experiment was originally published in the IEEE International Conference on Computational Intelligence in Bioinformatics and Computational Biology (CIBCB) 2020 conference proceedings [29]. Genetic algorithms are used to generate a sequence of merges that mitigates the amount of noise introduced to G'' on both unweighted and weighted graphs. The sequences of merges for the unweighted and weighted graphs are then compared to discover if the extra information given in terms of edge weights make a difference when compressing graphs. A small bug was found in the merging process of the genetic algorithm after the results were published. This bug was fixed, the experiments were re-run, and the new results are presented here.

5.1 Introduction

As we have seen, there have been many studies completed on unweighted graph compression, and not much research conducted on weighted graph compression. This begs the question: would taking weights into consideration make a difference when compressing a graph?

For this experiment, the compression between weighted and unweighted versions of the same graphs will be compared. To obtain the unweighted graphs, we simply delete the weights in the data file, so that there is no weight associated with the edges. Since there is no weight, the fitness functions introduced in Section 4.8 will not work because there are no edge weights to average. Subsequently, for the unweighted graphs, a different fitness function was used. This is the same fitness function that was used in [12]. It will be introduced in Section 5.1.1.

Note that preliminary tests have determined the crossover rate, mutation rate and the maximum distance used. Please see Appendix E to understand why a crossover rate of 0.90 and mutation rate of 0.15 is used, and Appendix F to discover why a maximum distance of one is used.

5.1.1 Unweighted Fitness Function

The fitness function used for the unweighted graphs counts the number of fake edges created from the compression [12], and is shown in Equation 5.1, where E_x represents the total number of edges in graph x. Recall that fake edges are edges that are present in graph G'' but are not present in G. They are formed due to the merger of two nodes not originally connected by an edge. Therefore, it is likely (for a small compression) to obtain a chromosome with a fitness value of 0. This fitness function, like the weighted fitness functions in Section 4.8, should also be minimized to reduce the noise introduced to the compressed graph.

$$fitness_{unweighted} = E_{G''} - E_G \tag{5.1}$$

5.2 Experiment Details

To test the difference between weighted and unweighted graphs, three sets of GAs were executed. The first GA was executed utilizing the unweighted graph data, and the unweighted fitness function mentioned in Section 5.1.1. The second GA was executed utilizing the weighted graph data, and the first fitness function mentioned in Section 4.8.1. Lastly, the third GA was executed utilizing the weighted graph data, and the second fitness function introduced in 4.8.2. The GA parameters stayed consisted throughout all three GAs, and are displayed in Table 5.1.

5.3 Results and Discussion

In this section, we will give the results of the experiments, and discuss and compare the first and second fitness functions to the non-weighted fitness function to see if edge weights make a difference when taken into consideration for compression.

Table 5.1: Summary of Experimental Parameters

Parameter	Value
Mutation Rate	15%
Crossover Rate	90%
Max Distance	1
Generations	200 (compr. ratio 10%)
	400 (compr. ratio 20%)
Population Size	250
Tournament Size	5
Number of Elites	1
Number of Runs	30

5.3.1 Unweighted vs First Fitness Function

We will first compare the unweighted fitness function to the first fitness function. In Table 5.2, we see the best fitness values produced by the GA within 30 runs. It is clear that the unweighted fitness function has a much lower fitness value than the weighted data. However, because the weighted data has more chances of introducing noise than the unweighted data, this makes sense. For instance, in the unweighted data, the only way to introduce noise is to create a fake edge. However, weighted graphs can introduce noise not only through a new edge, but also a difference in existing edge weights.

Table 5.2: Best fitness value obtained for each compression ratio for weighted graphs using the first fitness function (Section 4.8.1) and unweighted graphs using the unweighted fitness function (Section 5.1.1).

Compression	10		20	
Fitness	First	Unweighted	First	Unweighted
06 – 02	10.00	0.00	31.00	0.00
04 – 28	80.00	21.00	340.67	76.00
07 – 07	154.00	15.00	782.33	90.00
07 – 15	753.78	131.00	2520.23	408.00

Since, the fitness functions calculate noise differently, it is more insightful to look at the Adjusted Rand Index (ARI) between the two data. The ARI will tell us how similar the two chromosomes are. The best chromosomes produced (per compression rate and graph) from the unweighted fitness function and the first fitness function were taken, all the nodes that merged together were considered its own community or class (while the unmerged nodes were a community of one node), and the ARI was calculated. This is shown in Table 5.3. Recall that an ARI score of 1 means the two chromosomes are identical, and if the ARI score is around 0 it means the two chromosomes have nothing in common. Clearly, the communities that the two chromosomes create are very different, the most dissimilar being the communities found for the graph $06-02$ at 10% compression. The most similar communities were for graph $04-28$ at 10% compression. Let us look at the merges suggested by the genetic algorithms, and how they differ by digging into the chromosomes themselves.

Table 5.3: The Adjusted Rand Index between weighted graphs (using the First Fitness Function) and the non-weighted graphs

Compression	10	20
$06-02$	-0.003	0.125
$04-28$	0.585	0.443
$07-07$	0.444	0.301
$07-15$	0.224	0.309

Figure 5.1 shows the merges suggested by the weighted fitness function, and Figure 5.2 shows the suggested merges from the unweighted fitness function for the graph 06-02 at a compression rate of 20%. Nodes that are the same colour were suggested to be merged by the GA, except the nodes in grey. Those were left untouched, and are the same as in the original uncompressed graph. One major difference is the clique of nodes numbered 55, 56, 57, and 58. The unweighted version has 3 out of 4 of these nodes merged together, coloured in yellow, since they can merge without creating any noise (fake edges). However, in the weighted version, none of the nodes from this clique are merged. This is because the edge weight range is too dramatic (1 - 35) in the clique such that it would introduce a lot of noise into the compressed graph.

Another observation made when looking at Figure 5.1, is that the algorithm does an incredible job at merging nodes that are connected by a highly weighted edge, when they are surrounded by low weighted edges. For example, take nodes numbered 51 and 50, coloured in greyish-blue. The edge between them has a weight of 40, but the external edge connecting that pair to the rest of the graph has a low weight of 1. When compressed, that high weighted edge becomes the super-node's internal weight, which does not contribute to noise, and the low weighted edge of 1 is divided between

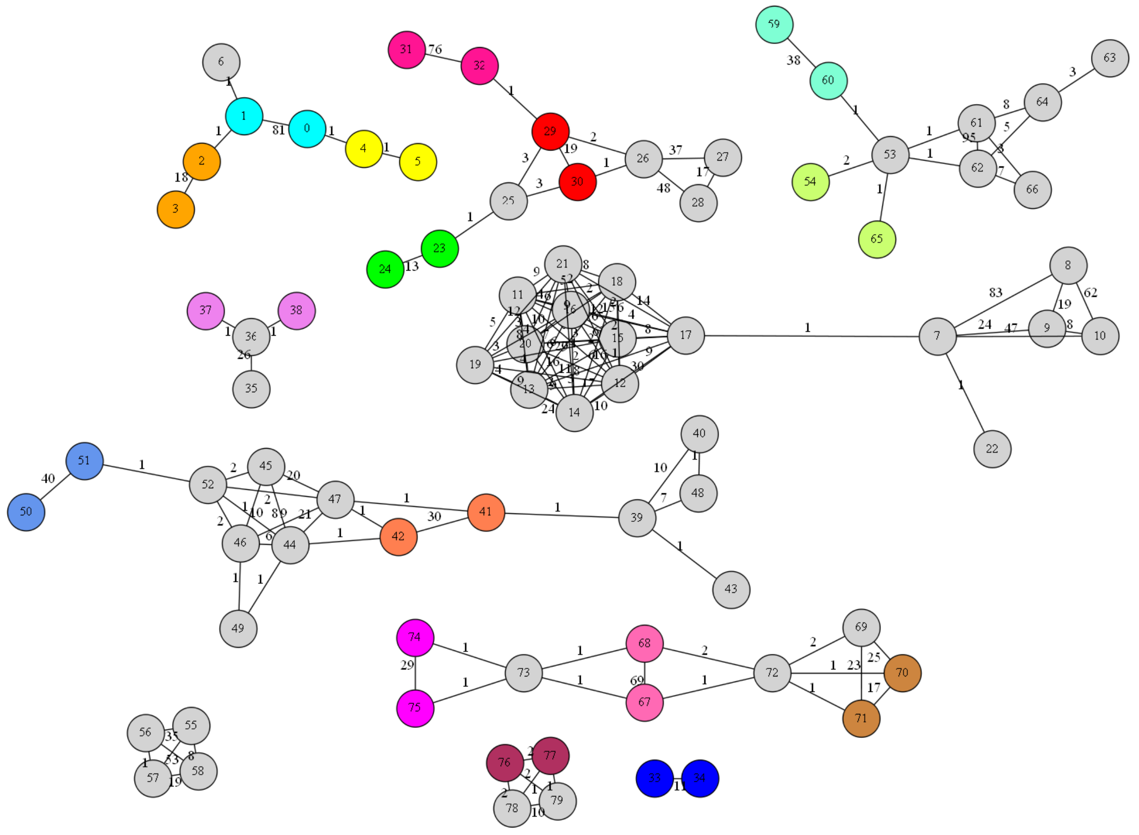


Figure 5.1: Weighted Graph 06-02, that was compressed 20% by the First Fitness Function

the two merged node, making it a super-edge of weight 0.5 instead of an edge of weight one - making a relative small amount of noise. This pattern is repeated numerous times throughout the graph. Including merging of nodes 41 and 42 (orange), 2 and 3 (gold), 31 and 32 (pink), 59 and 60 (pastel green), 1 and 0 (green-blue), etc. Where none of these nodes are merged by the unweighted fitness function, because a fake edge would be created in each case - creating maximal noise.

5.3.2 Unweighted vs Second Fitness Function

Similarly, we will now compare the unweighted fitness function and the second fitness function. Since the two fitness functions measure distortion in the graph in different ways, as mentioned in the previous section, it is more helpful to analyze the best merges suggested by the genetic algorithm. However, the best fitness values obtained by the GA within 30 runs are shown in Table 5.4 and the Adjusted Rand Index between the super-nodes created by the second fitness function and the unweighted fitness function, while keeping the compression, distance, crossover rate and mutation

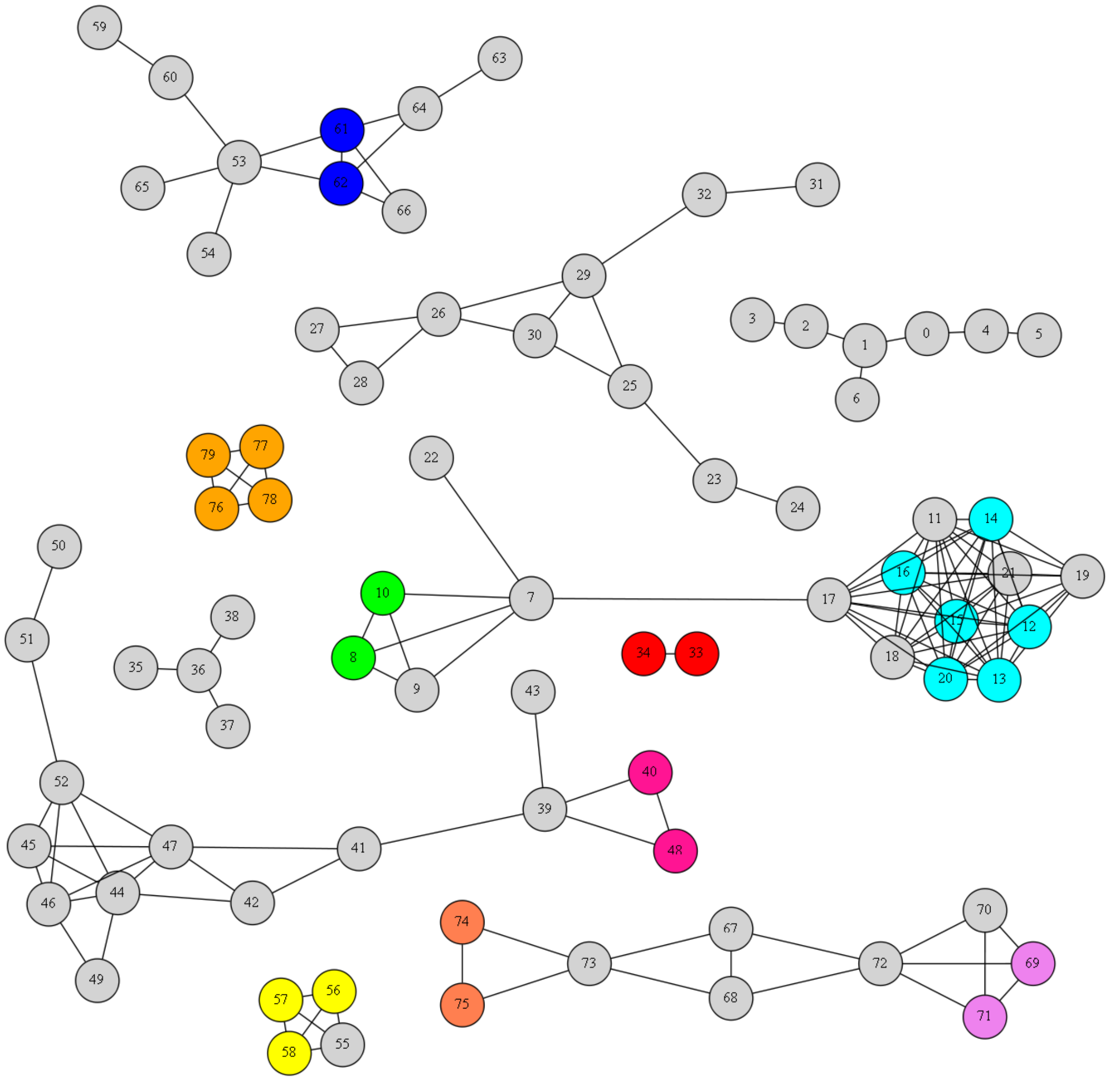


Figure 5.2: Unweighted Graph 06-02, that was compressed 20%

rate constant, are displayed in Table 5.5 for completeness. Although, it is important to note, that these ARI values are also low.

Table 5.4: Best fitness value obtained for each compression ratio for weighted graphs using the first fitness function (Section 4.8.2) and unweighted graphs using the unweighted fitness function (Section 5.1.1).

Compression	10		20	
Fitness	Second	Unweighted	Second	Unweighted
06 – 02	2.24	0.00	4.06	0.00
04 – 28	6.60	21.00	15.87	76.00
07 – 07	9.38	15.00	31.03	90.00
07 – 15	28.75	131.00	65.84	408.00

Table 5.5: The Adjusted Rand Index between weighted graphs (using the Second Fitness Function) and the non-weighted graphs

Compression	10	20
06 – 02	-0.003	0.168
04 – 28	0.585	0.403
07 – 07	0.444	0.217
07 – 15	0.215	0.212

The best found compression using the second fitness function for weighted graph 06-02 with a compression rate of 20% is shown in Figure 5.3. Similar observations can be made between this compression and the unweighted compression shown in Figure 5.2 when we compared the first fitness function. The clique of nodes 55, 56, 57, and 58 have not been merged due to the wide range of edge weights. This fitness function also does a tremendous job of merging nodes that are connected by an edge with a high weight and have external edges with low weight. This can be seen by the merging of nodes 59 and 60 (light green), 23 and 24 (dark green), 29 and 30 (orange-red), 31 and 32 (pink), 50 and 51 (pastel green), 41 and 42 (grey-blue), 74 and 75 (dark gold), 67 and 68 (light pink), 35 and 36 (light lavender), 0 and 1 (green-blue), and 2 and 3 (gold). It is interesting to note that all of these pairs of nodes, with the exception of 35 and 36, were also merged by the first fitness function.

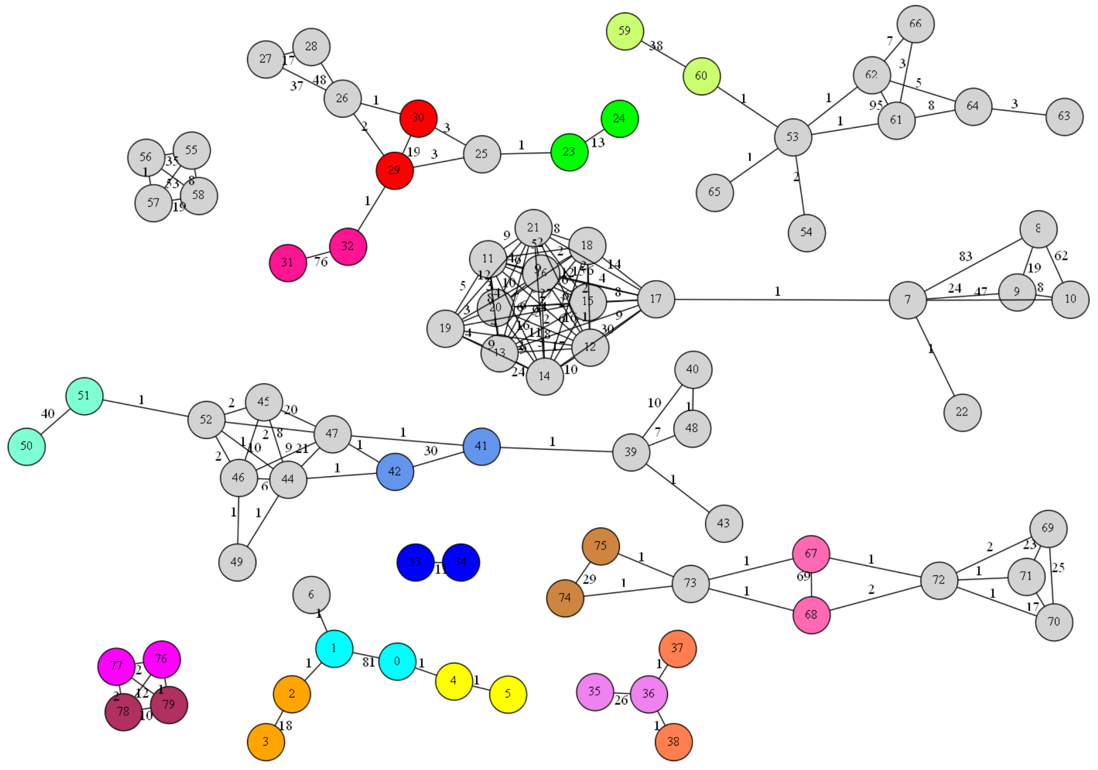


Figure 5.3: Weighted Graph 06-02, that was compressed 20% by the Second Fitness Function

5.4 Conclusion

In conclusion, it is unmistakable that the edge weights provide important information that can be useful when compressing graphs. The edge weights give us an entirely different representation/comprehension of the graph. Throughout the rest of this study, only weighted graphs and fitness functions will be considered.

Chapter 6

Fitness Functions

This experiment was originally published in the IEEE Congress on Evolutionary Computation (CEC) 2021 conference proceedings [30]. The sequence of merges provided by the GAs are translated into super-nodes. These sets of super-nodes are compared via the Adjusted Rand Index measure to verify if the two fitness functions result in different merges.

6.1 Introduction

In this section, we compare, for each graph and each compression ratio, the sets of super-nodes created by the compression algorithm using two fitness functions. Analyzing how the two fitness functions compress a graph will allow us to determine whether the patterns of the compression are similar. If the two functions produce different sets of super-nodes, the patterns will give us an insight into what situation the fitness function is best suited for.

6.2 Experiment Details

To analyze the differences between the two fitness functions, all the graphs, original and adjusted, were run in the GA with the parameters shown in Table 6.1 with both fitness functions. Recall that the crossover rate, mutation rate and the distance parameter were determined empirically. These preliminary tests are shown in Appendices E and F. These graphs were all compressed by 40% and 50%.

The resulting super-nodes were taken from the best found chromosome produced by each GA. For each graph and compression ratio the Adjusted Rand Index, introduced in Section 4.10, was calculated to discover the similarity between the two sets. We were also able to take the best chromosome from the first fitness function, and rerun it to get the corresponding fitness value using the second fitness function. This allows us to compare the two solutions on the same level to see which solution introduced less noise into the compression.

Table 6.1: Summary of Experiment Parameters

Parameter	Value
Mutation Rate	15%
Crossover Rate	90%
Max Distance	1
Generations	400
Population Size	250
Tournament Size	5
Number of Elites	1
Number of Runs	30

6.3 Results and Discussion

Before we look at any results, let us analyze the fitness functions in more detail. The first fitness function, introduced in Section 4.8.1, deals with the distortions of the graph by summing their absolute values. Whereas, the second fitness function, introduced in Section 4.8.2, deals with distortion by squaring it, and then taking the square root of the total difference. Below we will apply these fitness values to some common merges. All the merges discussed will have the two nodes connected by an edge, since we set the maximum distance parameter to 1 for this study.

1. The first merge to focus on will be the merging of a sub-graph with order 2. This is just two nodes that are only connected to each other, and no external edge exists, as shown in Figure 6.1 . This is straight forward, and will result in no distortion in graph G". Therefore, both fitness functions will prioritize this kind of merge.

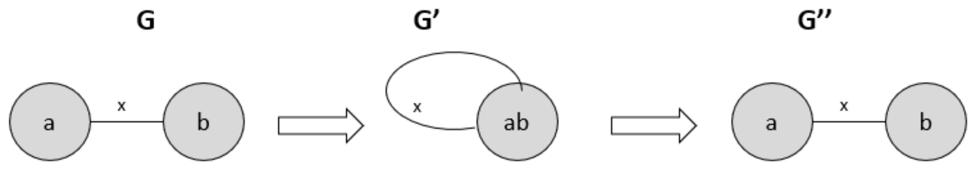


Figure 6.1: Two nodes merging without any external edges

2. The second merge, is the merging of two nodes with one external edge, as shown in Figure 6.2. This merge does produce distortion in G". Using the first fitness function, the fitness value would evaluate to $2w$, whereas the second fitness function would evaluate to w. In this case, this type of merge would be prioritized based on w. If w is high, that would contribute a lot of noise to G". However, if w is low, such as 1, this type of merge may be the best option for both fitness functions.

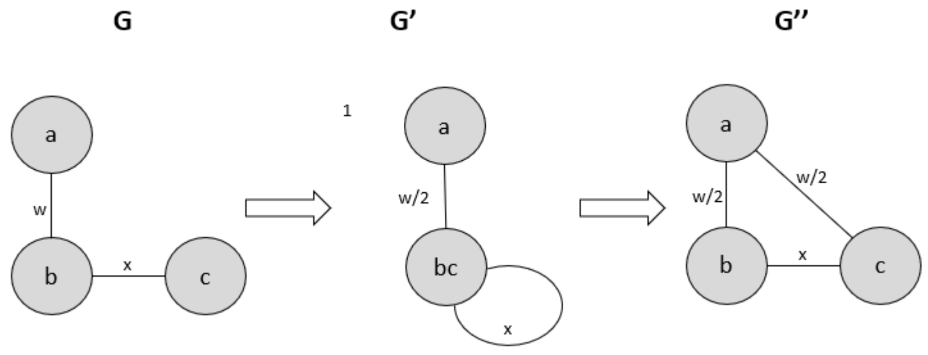


Figure 6.2: Two nodes merging with one external edge

3. Thirdly, a merge where the nodes have an edge to the same external node is also very common. There are two cases: when these edges have the same weight, shown in Figure 6.3, and when the edges have a different weight, shown in Figure 6.4.

   The case where the two edges have the same weight to the external edge will be prioritized by both fitness functions as no noise was introduced to G".

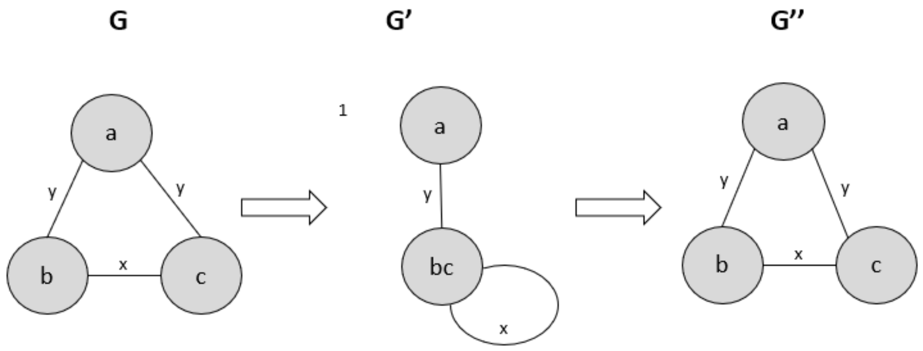


Figure 6.3: Two merging nodes have edges, of the same weight, to the same external node.

Subsequently, the case where the two edges to the external node vary in weight will add distortion to the graph. The amount of distortion added will depend on the difference between the weights. The higher the difference, the more noise is introduced. Note that this does depend on the difference between the two weights, not the value of the weights themselves. For example, if $y = 15$ and $x = 20$, a difference of 5, would produce less noise than if $y = 1$ and $x = 10$, since that would be a difference of 10. Once again, if the difference between the two weights is very low, then this may be the best merge option for both of the fitness functions.

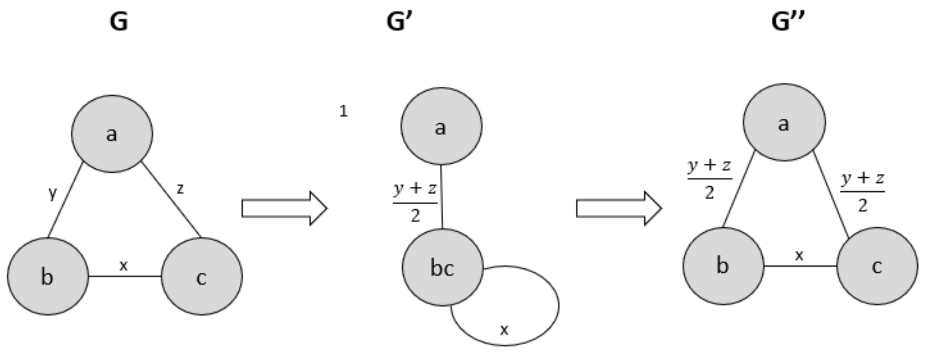


Figure 6.4: Two merging nodes have edges, of varying weight, to the same external node.

After this quick analysis, we have learned that both fitness functions aim to merge nodes that have commonly weighted edges with external nodes. If we continue to analyze more complex merges, such as when a super-node merges with another node (or another super-node), the internal weight is also averaged. Then, the same principle

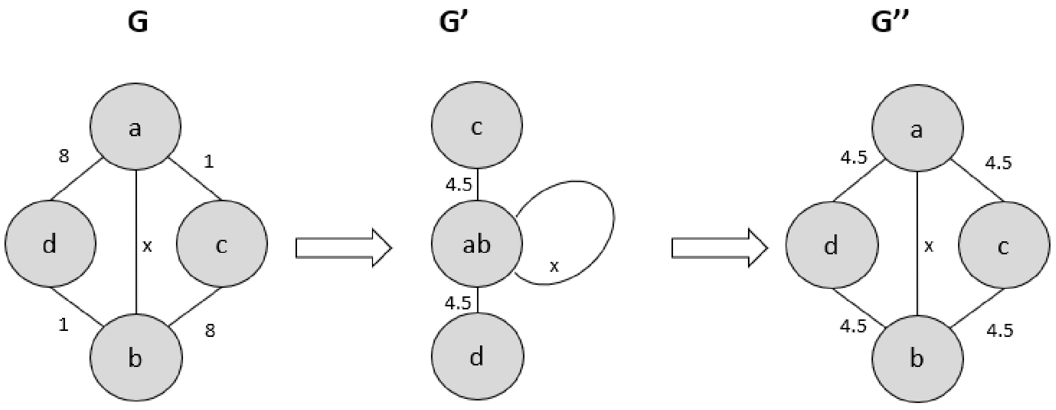


Figure 6.5: Example of merge with equal variation between external nodes

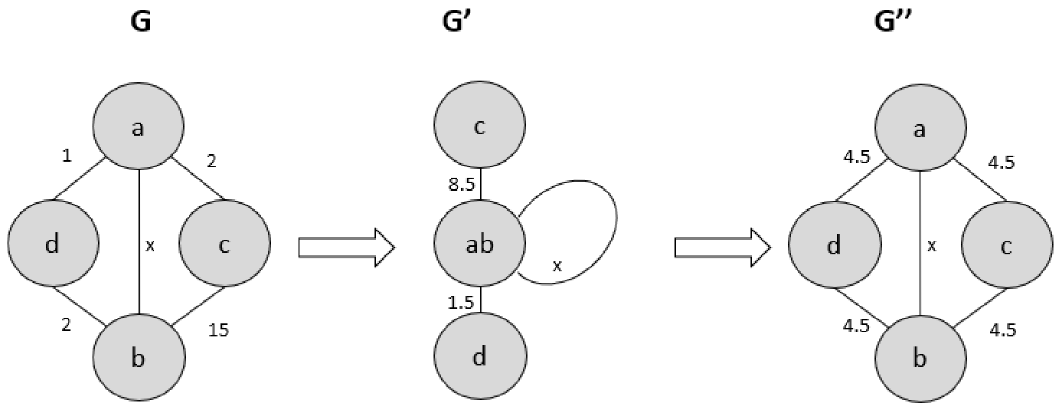


Figure 6.6: Example of merge with differing variation between external nodes

is seen. Both algorithms will prioritize merging nodes that have relatively similar internal weights in order to keep the distortion low.

There is one scenario where the two fitness functions differ in optimization. This is when it is presented with the choice of either merging two nodes with equal variation in the connections to external nodes, or merging two nodes with differing variation between external nodes. For example, in Figure 6.5 we see a graph where nodes A and B both have edges to nodes C and D. And between each of these nodes there is a difference of 7. Using the first fitness function, when nodes A and B are merged, the fitness value would equate to 28, and the second fitness function would equal to $\sqrt{98} =$ 9.899. In Figure 6.6 we see a similar merge. However, the difference between the edges from nodes A and B to C have a difference of 13 (15-2), and to node D have a difference of 1 (2-1). In this case, if nodes A and B merge, the first fitness function equates to 28, and the second fitness function equates to $\sqrt{170} =$ 13.038.

In both of these the total weight difference within the one merge equals 14 (7+7

Table 6.2: Similarity of sets of super nodes in best results from fitness function 4.1 vs those from fitness function 4.2.

Graph	Compression ratio 40%		Compression ratio 50%	
	Original	Adjusted	Original	Adjusted
06 – 02	0.82	0.83	0.82	0.80
04 – 28	0.66	0.59	0.53	0.43
07 – 07	0.63	0.61	0.61	0.52
07 – 15	0.52	0.44	0.54	0.45

or 1+13), but where the weight differences lay in the graph is distinct. However, this only affected the outcome of the second fitness function. It clearly optimized the merge with the equal variations between the external nodes. The first fitness function does not recognize this distinction and randomly chooses between the two options.

Due to this investigation of common merges, we expect the two fitness functions to result in very similar compressions. Table 6.2 shows the ARI between the two super-nodes created by the two fitness functions, per graph and compression ratio. From the table, we see that there is a general pattern where the larger the order of the graph the less similar the two resulting super-nodes are. This makes sense considering the fewer nodes in a graph, the fewer optimal choices there are between what nodes to merge, so the two fitness functions have a better chance to merge the same nodes. Whereas, the higher the order of a graph, the more optimal merges are possible. This can create a domino effect, where a different simple merge can create the possibility of a more optimal complex merge.

In order to compare the two fitness functions to each other, the best chromosome, per graph and compression ratio, produced by the second fitness function was applied to the original graph G, and the fitness value was calculated by the first fitness function. We can then compare these solutions to the solutions solely found by the first fitness function. This is shown in Figure 6.7. The same was done to the solutions produced by the first fitness function. The corresponding second fitness function value was calculated, so we can compare these solutions to the solutions found solely by the second fitness function. This comparison is found in Figure 6.8.

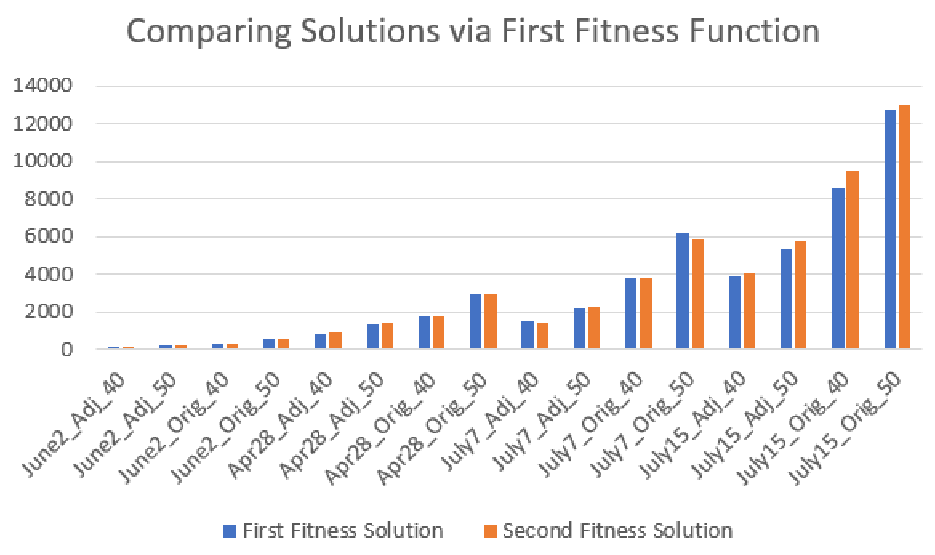


Figure 6.7: Translating the Second Fitness Function to the First Fitness Function

As can be observed by the two figures, for each graph and compression ratio, the fitness values are always around the same value. However, the solution found by the comparing fitness function is usually better (i.e. lower). This adheres to the main objective of a genetic algorithm - the solution is specifically designed to most optimally follow the restrictions set by the fitness function. Thus, the fact that the solutions that were found by the first fitness function had a better first fitness value than the solution found by the second fitness value is expected. And vice versa for the second fitness function.

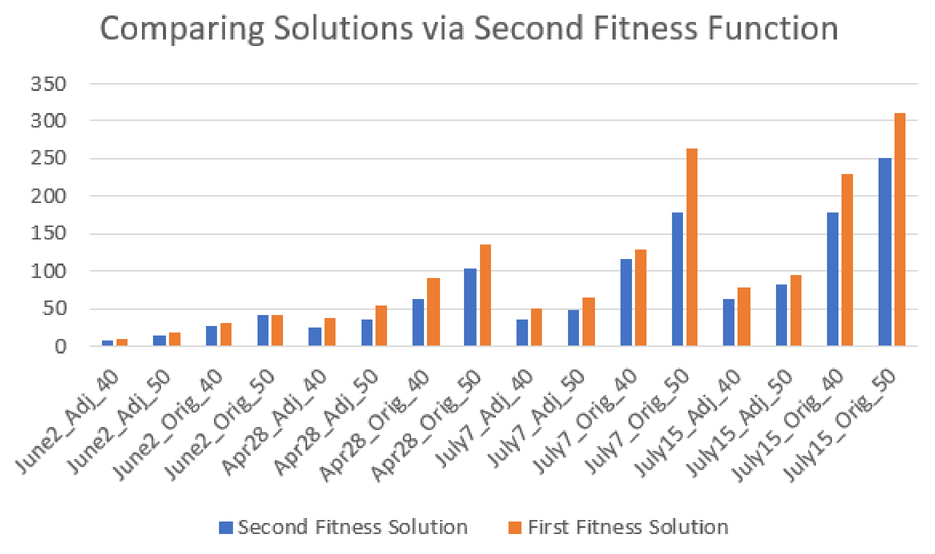


Figure 6.8: Translating the First Fitness Function to the Second Fitness Function

6.4 Conclusion

In conclusion, the two fitness function seem to be working towards merging the same type of nodes. The same patterns can be seen by both fitness functions, with the exception of the one difference mentioned. Both prioritize merging nodes that have similarly weighted edges to common external nodes, or completely merge nodes to form a clique with similar internal weight. Any differences that we see are primarily due to chance and past merges. Thus, we will continue to use both of the fitness functions throughout the rest of this study.

Chapter 7

Original Weights vs Adjusted Weights

This experiment was originally published in IEEE Congress on Evolutionary Computation (CEC) 2021 conference proceedings [30]. The extreme weights presented in the original graph have been modified to be in a range of [1,10]. The genetic algorithms are then run as normal to test how much the severity of weights actually affect the compression.

7.1 Introduction

It has clearly been proven in Chapter 5 that edge weights add vital information for compressing a graph. Edge weights give meaning to the strength of the connection between nodes, and can give an entirely different view of the network - especially in an epidemic point of view. However, the weights present in the original graph range a great deal. Table 4.1 shows the summary of the graphs used, including the maximum and average edge weight present. Edge weights in graph $06 - 02$ range from 1 to 95 with an average of 11.33, in $04 - 28$ range from 1 to 108 with an average of 8.32, in $07 - 07$ range from 1 to 167 with an average of 8.20, and in $07 - 15$ range from 1 to 191 with an average of 6.26. This is quite an extreme difference.

7.2 Experiment Details

In order to test how the magnitude of the weight affects the compression, each edge weight was run through equation 4.3, which adjusts the weight to be in one of 10

Table 7.1: How weights change when using Equation 4.3

Original Weight	Adjusted Weight
1	1
2	1
3	2
4	2
5	2
10	4
20	7
50	10
100	10

classes from [1, 10]. Table 7.1 shows what classes some different weights belong to. This will maintain the information that some connections are stronger than others but does not contain edges that have extreme weights.

Afterward, the GAs were run as before, with the parameters as outlined in Table 7.2. The mutation rate, crossover rate and max distance were set empirically by running preliminary tests. These tests are shown in Appendices E and F. The graphs were then compressed by 40% and 50%.

The super-nodes were then taken from the best found chromosome produced by each GA. For each compression ratio and fitness function, the Adjusted Rand Index, introduced in Section 4.10, was executed to find the similarity between the original graphs and the adjusted graphs.

Table 7.2: Summary of Experiment Parameters

Parameter	Value
Mutation Rate	15%
Crossover Rate	90%
Max Distance	1
Generations	400
Population Size	250
Tournament Size	5
Number of Elites	1
Number of Runs	30

7.3 Results and Discussion

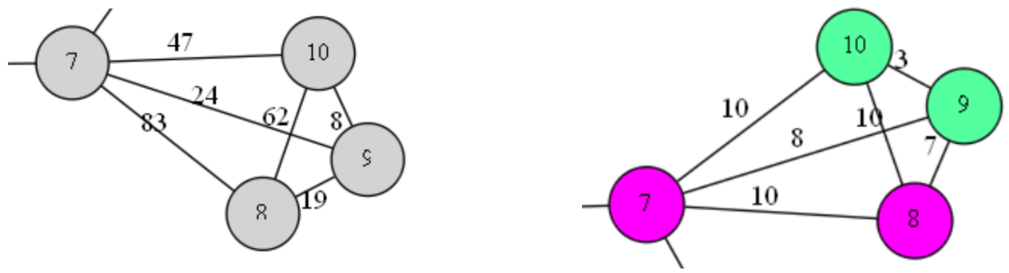
Table 7.3 shows the ARI values per compression ratio and fitness function. From this, we see that the magnitude of the weights contribute a lot to the solution of the compression. It can be seen that the smaller graph (06-02) has a higher similarity than the larger graphs, and that the similarity usually decreases when moving from 40% compression to 50% compression.

Table 7.3: Similarity of sets of super-nodes in the best results from using the original graphs vs those from using the adjusted graphs.

Graph	Compression ratio 40%		Compression ratio 50%	
	Fitness 1	Fitness 2	Fitness 1	Fitness 2
06 – 02	0.90	0.78	0.64	0.80
04 – 28	0.50	0.60	0.41	0.44
07 – 07	0.49	0.52	0.47	0.50
07 – 15	0.60	0.50	0.51	0.39

Figure 7.2 shows the best merges found on the 06-02 original graph using the first fitness function compressing by 50%, and Figure 7.3 shows the best merges found on the 06-02 adjusted graph using the first fitness function compressing by 50%. The two solutions have an ARI of 0.64. Consequently, we can see some differences with the chosen merges.

From Chapter 6 we know that both fitness functions prioritize merges that have commonly weighted edges to external nodes, and the smaller the difference between these two weights, the better the merge. When we adjusted the weight of all the edges, the number of edges with a similar weight increases, thus increasing the number of optimal merges in the adjusted graph. This leads to different merges such as with nodes 9 and 10, shown in Figure 7.1. Using the original weights, the two nodes were not merged due to the large difference in the external edge weights. The edges connecting node 7 to 9 and 7 to 10 have weights of 24 and 47 respectively. The edges connecting nodes 8 to 9 and 8 to 10 have weights of 19 and 62 respectively. These are large differences, and thus will lead to a lot of distortion in the compressed graph. However, once the weights were adjusted, these extreme values became more controlled. This lead to the edges connecting node 7 to 9 and 7 to 10 to have weights of 8 and 10 respectively, and the edges connecting nodes 8 to 9 and 8 to 10 have weights of 7 and 10 respectively. The differences between these weights are much smaller, and will lead to much less distortion in the compressed graph. The same



(a) Original Weights

(b) Adjusted Weights

Figure 7.1: Different merges with original weights vs adjusted weights using the First Fitness Function

example is seen when the second fitness function is used. More of these examples can be seen when comparing the best merges found for $06 - 02$ and $06 - 02_{adj}$ with the first fitness function shown in Figures 7.2 and 7.3 and the second fitness function in Figures 7.4 and 7.5.

7.4 Conclusion

In conclusion, adjusting the weights of the graph leads to different solutions, i.e. different compressions. This is not to say that adjusting the weights of a graph is wrong. It simply shows that having accurate and meaningful weights is important when compressing a graph. Therefore, we will continue to use both original and adjusted weights for the remainder of the study.

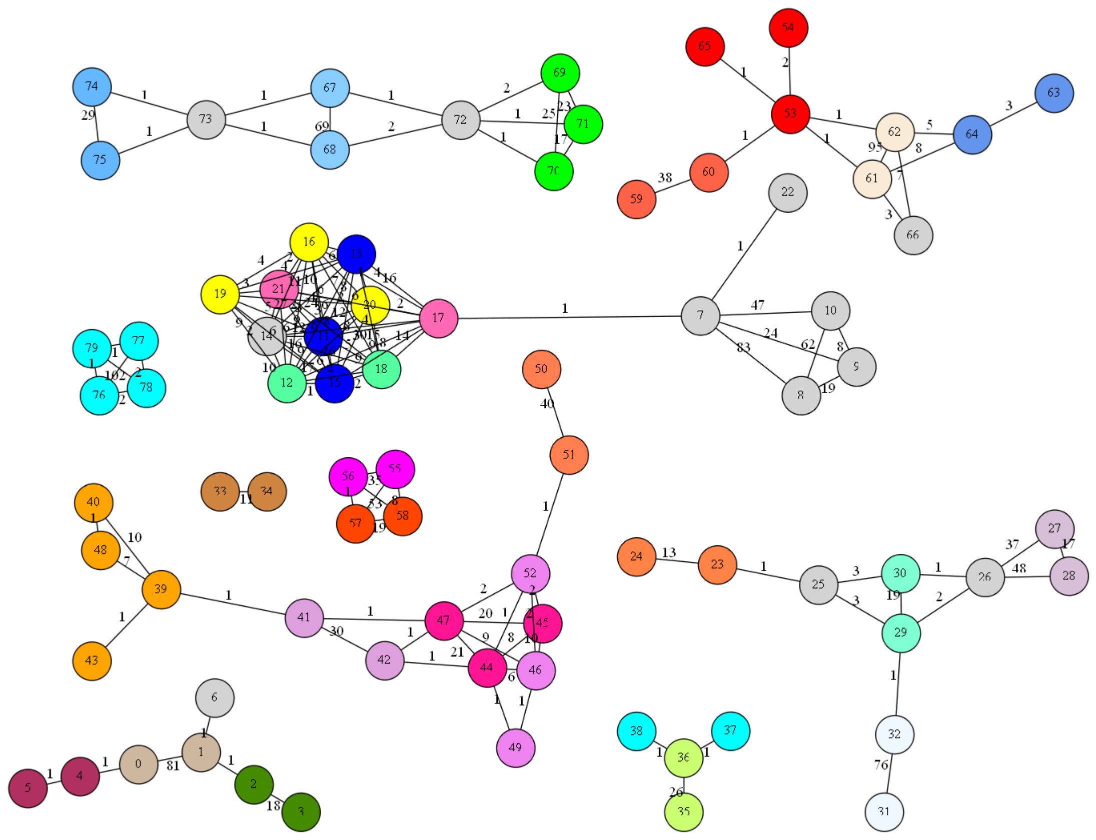


Figure 7.2: 06-02 with original weights, compressed 50% with First Fitness Function.

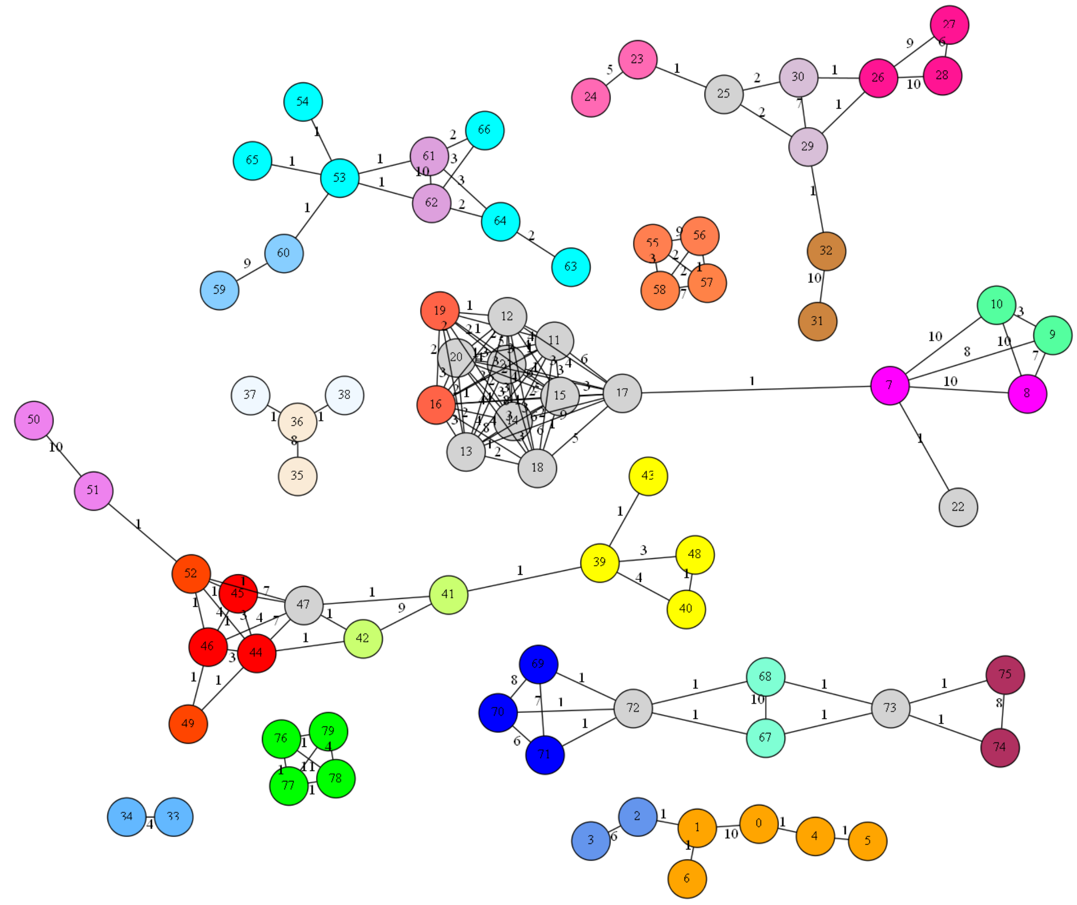


Figure 7.3: 06-02 with adjusted weights, compressed 50% with First Fitness Function.

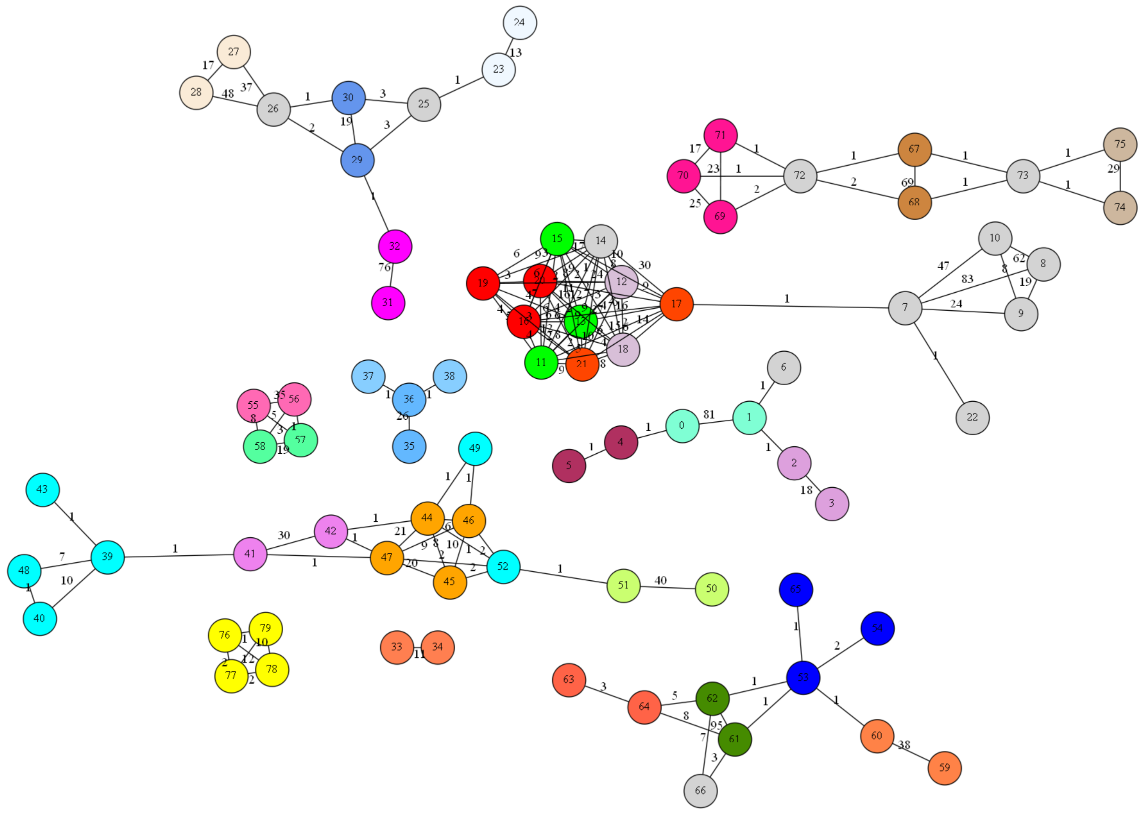


Figure 7.4: 06-02 with original weights, compressed 50% with Second Fitness Function.

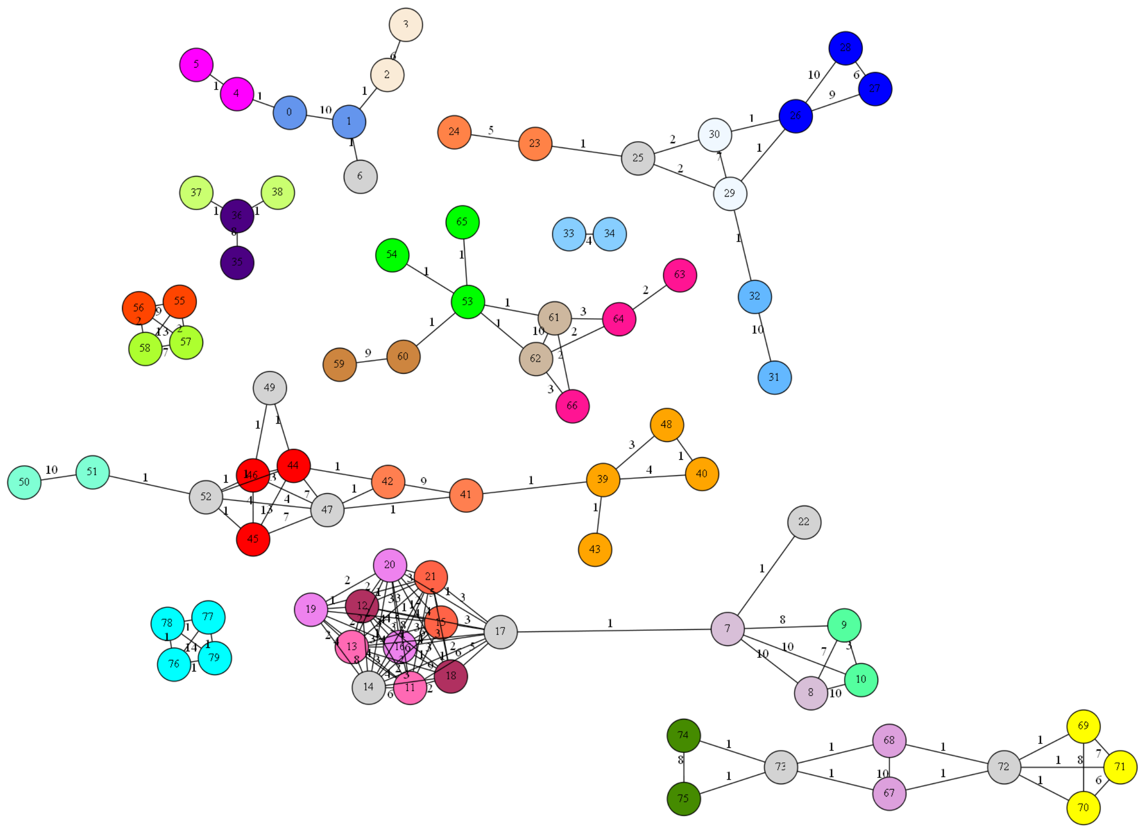


Figure 7.5: 06-02 with adjusted weights, compressed 50% with Second Fitness Function.

Chapter 8

Graph Compression vs Community Detection

This experiment was originally published in IEEE Congress on Evolutionary Computation (CEC) 2021 conference proceedings [30]. The super-nodes created from merges could be regarded as communities, if the compression is high enough. A *community* is a set of nodes that are relatively highly connected compared to the rest of the graph. The sets of super-nodes were taken from all the graphs that were compressed 40% and 50% and compared to the communities detected by a popular community detection algorithm: the Louvain Algorithm. For each graph, compression ratio, and fitness function, the two sets (of super-nodes and communities) were compared using the Adjusted Rand Index measure. Finally, we took the communities, created a chromosome solution that would result in the same super-nodes and calculated the fitness value via both fitness functions, and compared it to the best found solution from the GA.

8.1 Introduction

As the compression ratio increases, the amount of nodes in a single super-node also increases. Recall that when a super-node is decompressed, it forms a clique with each node which was merged into the super-node. This means that every node has an edge to every other node. When this occurs, the nodes that make up the super-node can be seen as a community. However, graph compression and community detection are two different methodologies altogether, which aim to resolve different goals. In graph compression, nodes are merged based on their common connections to other nodes, like we saw in Chapter 6. However, in community detection, nodes are grouped together due to their connections with each other [38].

8.2 Background

8.2.1 Community Detection

As mentioned, community detection aims to group nodes based on their connection to each other. Therefore, the result is a set of communities in which the nodes within each community are relatively highly connected, and have a minimum amount of connections to external nodes. Reviews were conducted on prevailing community detection algorithms and may be found in [15] and [20].

Modularity is often optimized in community detection algorithms. Newman-Girvan Modularity [24] can be defined as the fraction of edges that exists within a given community, minus the expected fraction of these edges if the distribution were random. Modularity can have a negative or positive value. If the value is positive, it indicates the presence of a community. Large positive values are also preferred.

8.2.2 Louvain Algorithm

When looking for a community detection algorithm for weighted graphs, there are significantly fewer options than for unweighted graphs. There are three algorithms that can be used: (1) Greedy Modularity Optimization, (2) the Leiden Algorithm, and (3) the Louvain Algorithm.

The Louvain algorithm was chosen for this study. This was due to its quick computation time, the quality of its results [2], and the feature of using a resolution specification [19], in order to choose how many communities the algorithm detects.

The Louvain Algorithm, presented in [2] is based on Newman-Girvan Modularity mentioned above. The algorithm is divided into two phases that are repeated iteratively. The first phase, "modularity optimization", applies a community to each node. Therefore, in the first iteration, each node is in a community by itself. Then, for each node i, the gain of modularity is evaluated if node i was placed in the community of its neighbor, j. The node is then placed in the community that yields the largest gain of modularity. This is repeated for all nodes until no further improvement to modularity can be made. The second phase, "community aggregation", essentially creates a new network by replacing the communities found in Phase One with nodes. Now, Phase One is able to be repeated. The Louvain Algorithm is performed by an external library. We use the CDLIB Python library of Rossetti, Milli, and Cazabet [28].

8.3 Experiment Details

In this chapter, we will be comparing the communities detected by the Louvain Algorithm, and the super-nodes created by the GA using both fitness functions. To do this, the best found solutions from the GAs run previously were taken. The GA parameter settings can be seen in Table 7.2.

The GAs were run with both fitness functions. From here, we calculate the number of super-nodes, x, created by multiplying the compression by the total number of nodes. For example, to get a corresponding solution from the Louvain Algorithm for a graph, G, of 200 nodes compressed by 50%, the estimated number of communities is set to $x = 0.5 * 200 = 100$. We then run the Louvain algorithm with the resolution specification set to x, so that we got the same number of communities detected by the Louvain algorithm. The nodes merged together were then obtained from the best solutions found by the GA, and compared to the communities detected by the Louvain Algorithm by using the Adjusted Rand Index.

We also translate the communities found by the Louvain Algorithm to a corresponding chromosome, such that all the nodes in a community are merged together. The chromosome is then evaluated by each fitness function to get the respective fitness values.

Note that we are not expecting that the Louvain Algorithm and the GA will create the same sets of communities. Recall that the two methodologies aim to resolve different goals. There will even be differences between different community detection algorithms. However, there may be some similarities in the results obtained by the Louvain Algorithm and the GA.

8.4 Results and Discussion

Table 8.1 shows the Adjusted Rand Index between the best found super-nodes created through the genetic algorithm and the communities found by the Louvain Algorithm per compression ratio, fitness function, and graph. At first glance, it's noticed that the smaller graphs have a higher ARI value, and the similarity between the two sets of results decrease as the order of the graph increases. Another pattern present is that in general, with the exception of graph 06-02, the ARI values are higher for the

Table 8.1: Adjusted Rand Index: Similarity of sets of super-nodes in best results from Compression vs those from Community Detection.

Graph	Compression ratio 40%				Compression ratio 50%			
	Fitness 1		Fitness 2		Fitness 1		Fitness 2	
	Orig	Adj	Orig	Adj	Orig	Adj	Orig	Adj
06 – 02	0.88	0.81	0.76	0.64	0.69	0.69	0.70	0.71
04 – 28	0.52	0.65	0.49	0.68	0.62	0.49	0.37	0.72
07 – 07	0.50	0.59	0.61	0.62	0.45	0.51	0.52	0.73
07 – 15	0.33	0.33	0.26	0.39	0.34	0.36	0.28	0.43

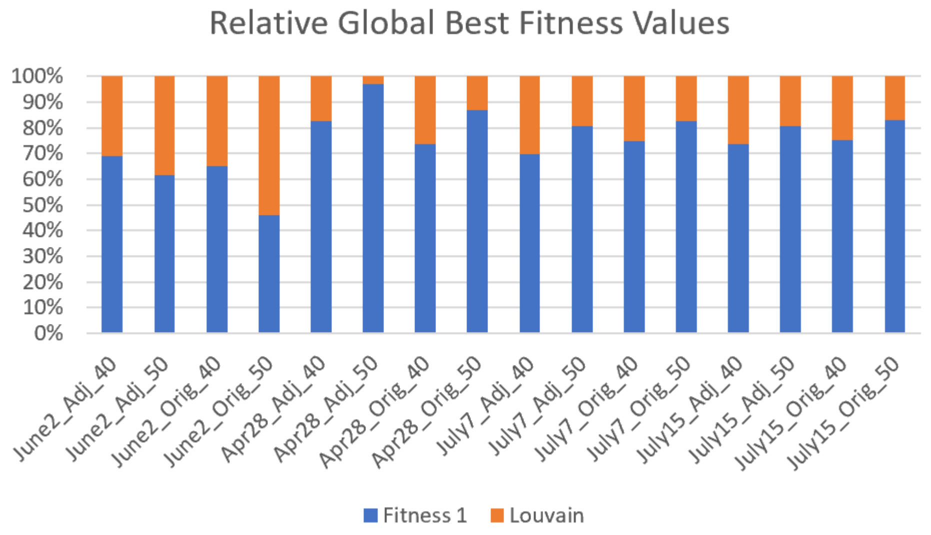


Figure 8.1: Louvain Communities compared to First Fitness Function

adjusted graphs than the original graphs. This could possibly demonstrate how the two algorithms handle a wide range of weights in the original graphs, compared to the fixed small range of weights in the adjusted graphs.

Figure 8.1 and 8.2 show the Louvain communities compared (relatively) to the best found super-nodes with the first and second fitness function respectively. Here the global best fitness from the genetic algorithm is shown as a percentage of the corresponding fitness value from the Louvain Algorithm. Recall that a lower fitness is better. It is clear that, in terms of fitness, the solution found by the GA is always better than the solution found by the Louvain Algorithm. Both fitness functions show a similar trends: the fitness of larger graphs are slightly closer than the smaller graphs, and the fitnesses are closer for 50% compression compared to 40% compression. However, the relative difference between the Louvain and GA solutions is larger when considering the second fitness function than the first fitness function.

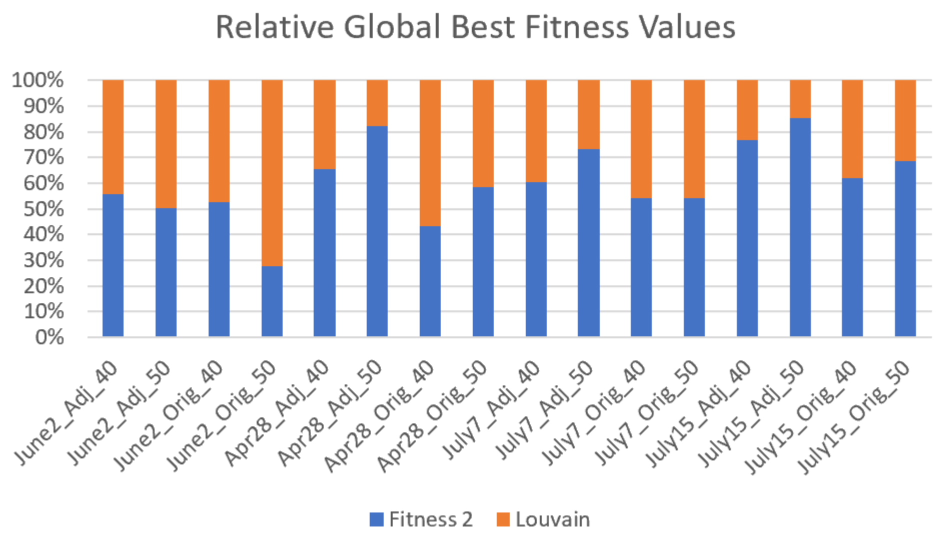


Figure 8.2: Louvain Communities compared to Second Fitness Function

One interesting case is the 50% compression of the adjusted graph for 04-28 using the first fitness function. In Figure 8.1 it can be seen that the two fitness values are very close to each other. However, the ARI value between the super-nodes created by the compression algorithm and the communities detected by the Louvain Algorithm is only 0.49. Subsequently, when the two graphs are compared, we expect to see a lot of differences. But these differences will still be optimal merges. The super-nodes created by the GA using the first fitness function are shown in Figure 8.3 and the communities detected by the Louvain Algorithm are shown in Figure 8.4. The nodes in grey represent nodes that have not been merged or put into a community, and each other colour represents nodes that have been merged together/put into the same community.

Even with a low ARI, many similarities between these super-nodes/communities can be seen. For example, the sub-graph of nodes 192-199 are separated the same way by both the compression algorithm and the community detection algorithm. Nodes 192, 193 and 194 are together, nodes 195, 196, and 197 are together, and nodes 198 and 199 are together. This is similarly the case for the super-node/community of nodes 17, 18, and 19; 20, 21, 22, and 23; 31, 32, 33, and 36. Many more can be seen in Figures 8.3 and 8.4.

Many differences can be seen too. For example, consider the huge super-node in Figure 8.3 consisting of nodes 110, 114, 115, 118, 119, 120, 121, 122, 123, 124, 125, and 126 highlighted in bright cyan. This community does not exist in Figure 8.4. Instead this one super-node is broken up into 5 communities: (1) 108, 109, 110; (2)

114, 115, 126; (3) 118, 119; (4) 120, 121, 122; and (5) 123, 124, 125. This example reinforces our findings that the communities differ, yet the merges/communities are still optimal.

Another interesting case is the 50% compression of the original graph 06-02 using the first and second fitness function. The fitness corresponding to the solution found by the Louvain Algorithm is more than double that of the compression algorithm when using the first fitness function and almost triple when using the second fitness function. However, the ARI between the communities found by the Louvain Algorithm and the super-nodes found by the genetic algorithm are 0.69 and 0.70 for the first and second fitness function respectively. Therefore, we expect to see a lot of similarities between these two sets of super-nodes/communities, but the few differences will create a major amount of distortion. Figure 8.5 shows the super-nodes created by the first fitness function for the original 06-02 graph when compressed 50%, Figure 8.6 shows the super-nodes created by the second fitness function for the original 06-02 graph when compressed 50%, and Figure 8.7 shows the communities identified by the Louvain Algorithm.

Many similarities can be seen between the solutions developed by the compression algorithm (using either fitness function) and the solution developed by the Louvain algorithm. All the small sub-graphs were merged the same way throughout all the algorithms. This includes super-nodes/communities of nodes 76-79, 33-34, 55-56, and 57-58.

Nevertheless, some differences are present as well. The genetic algorithm, using either fitness function, only merged nodes 74 and 75 together. As they are both connected to node 73 by an edge of weight 1, this merge is completed without introducing distortion to the graph. Recall common merges in Section 6.3. However, the community detection algorithm put all three nodes (73, 74, and 75) in one community, even though the edges in this graph represent interaction, and an edge of weight 1 is the lowest possible level of interaction.

8.5 Conclusion

In conclusion, some similarities are present between the solutions given by the compression algorithm and the community detection algorithm. However, there are more differences. Again this was expected due to the two methodologies resolving different goals. Nevertheless, the different sets of super-nodes and communities can help inform public health decision makers.

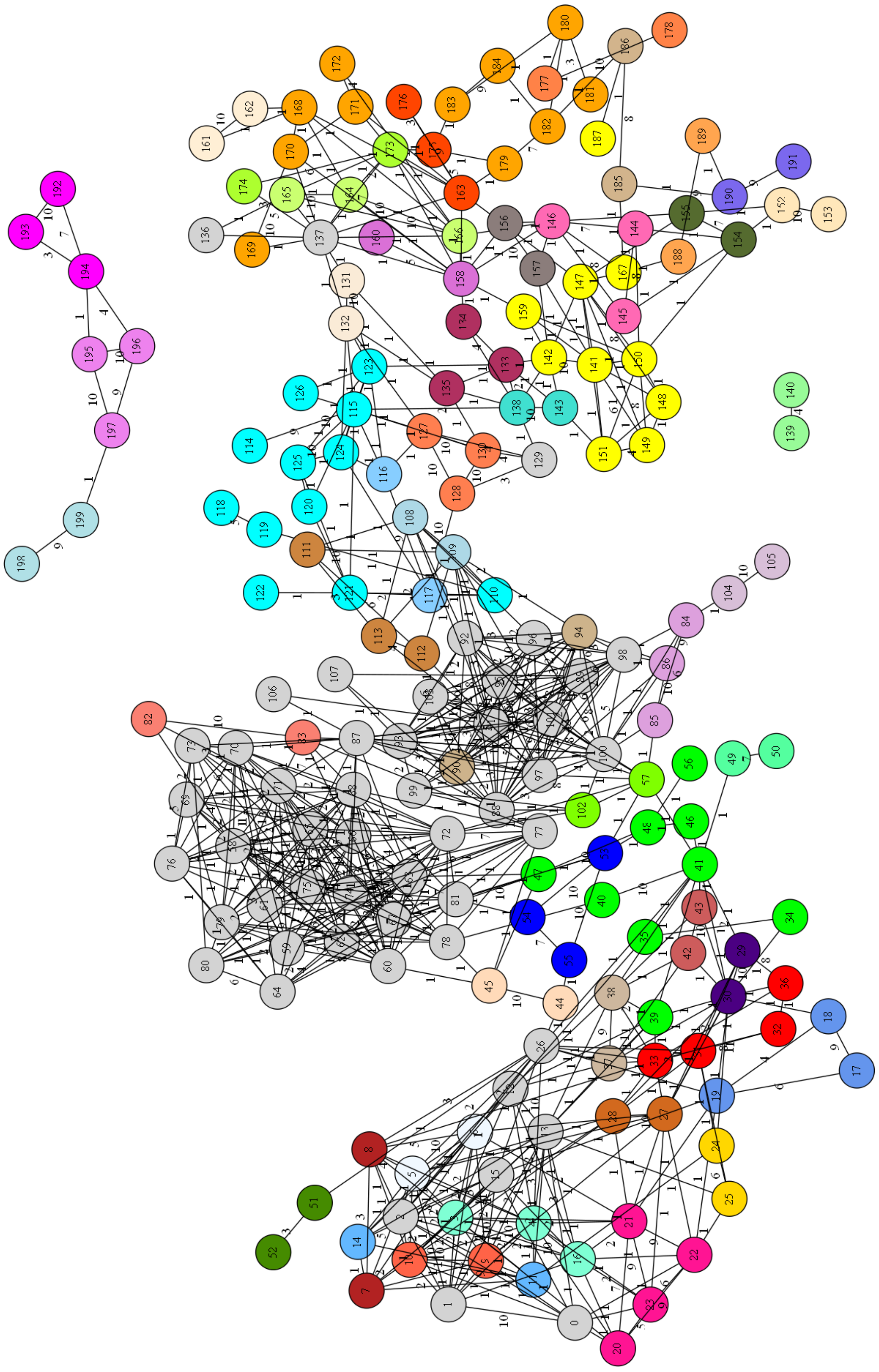


Figure 8.3: $04 - 28_{adj}$, 50% compression, First Fitness Function

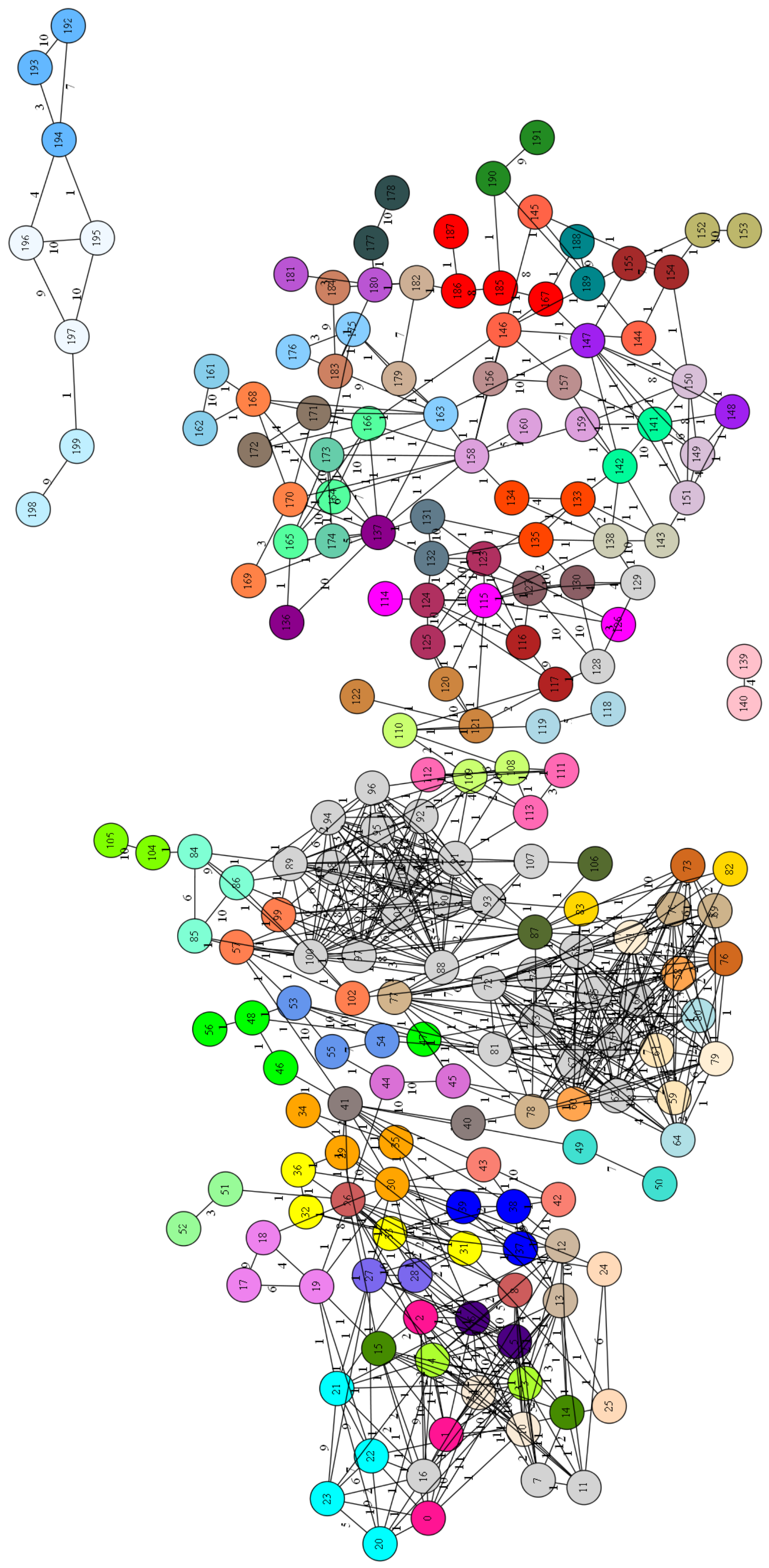


Figure 8.4: $04 - 28_{adj}$, 50% compression, Louvain Algorithm

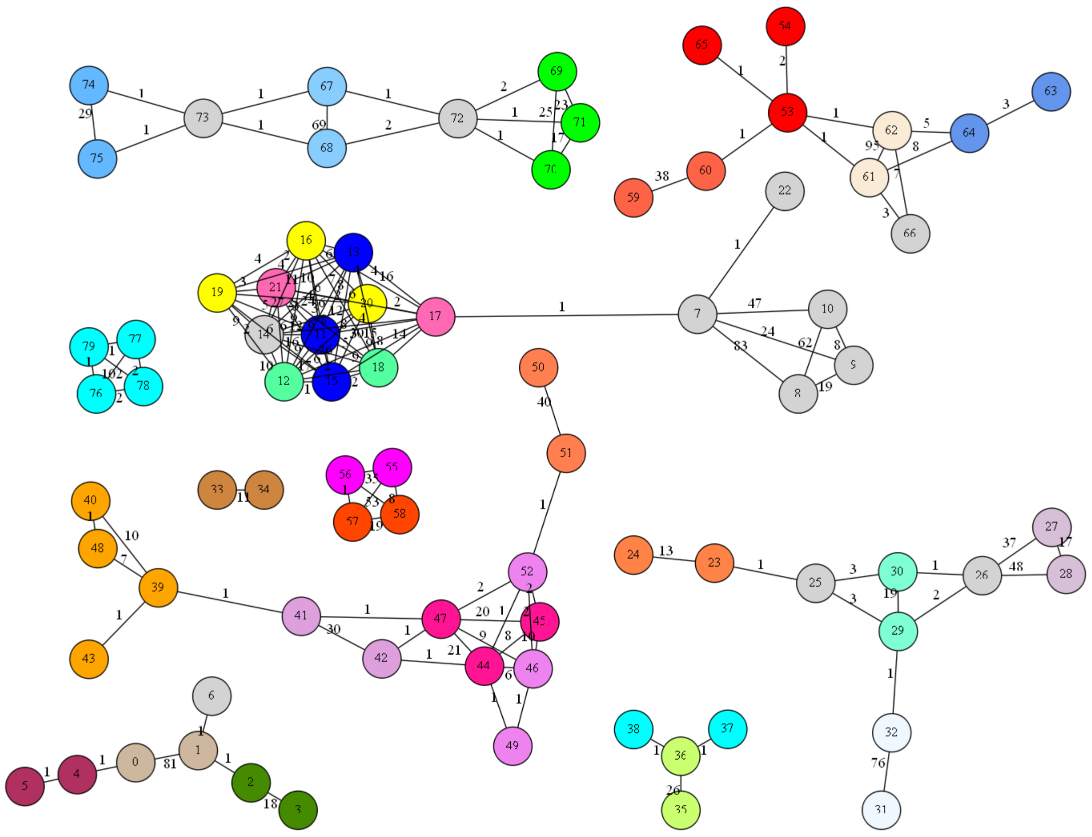


Figure 8.5: 06-02, 50% compression, First Fitness Function

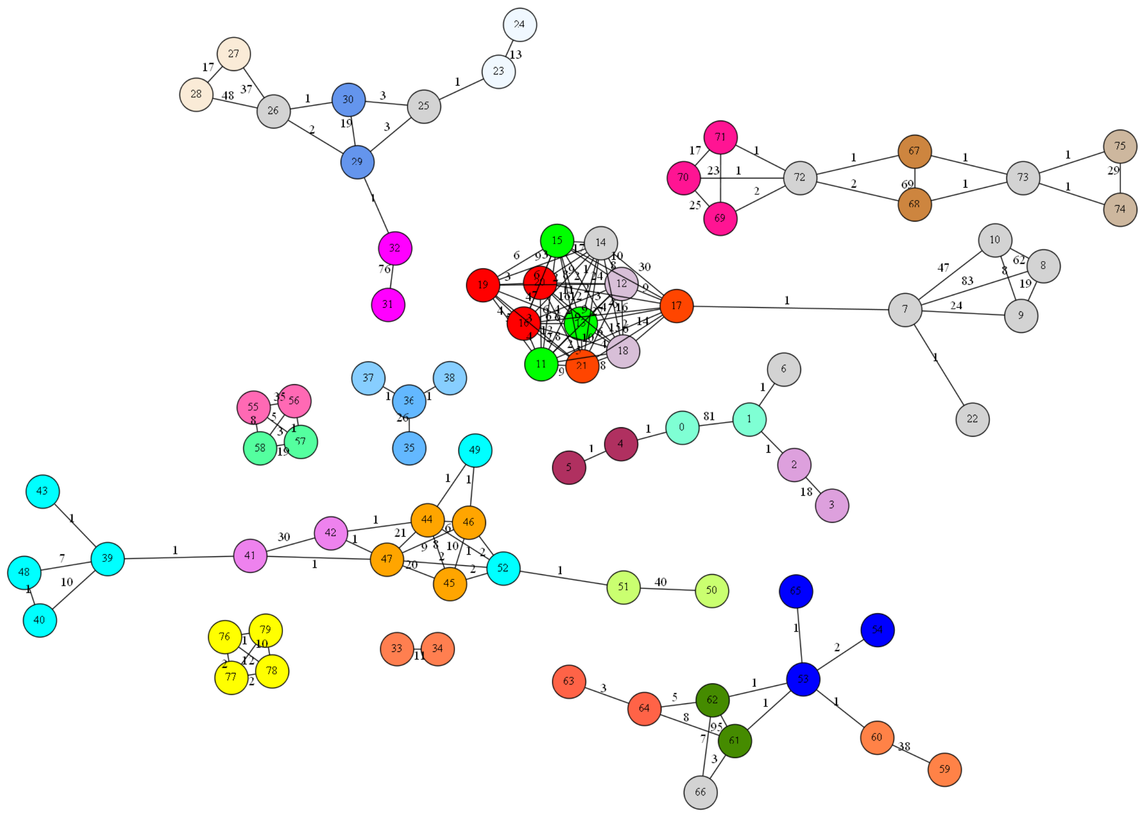


Figure 8.6: 06-02, 50% compression, Second Fitness Function

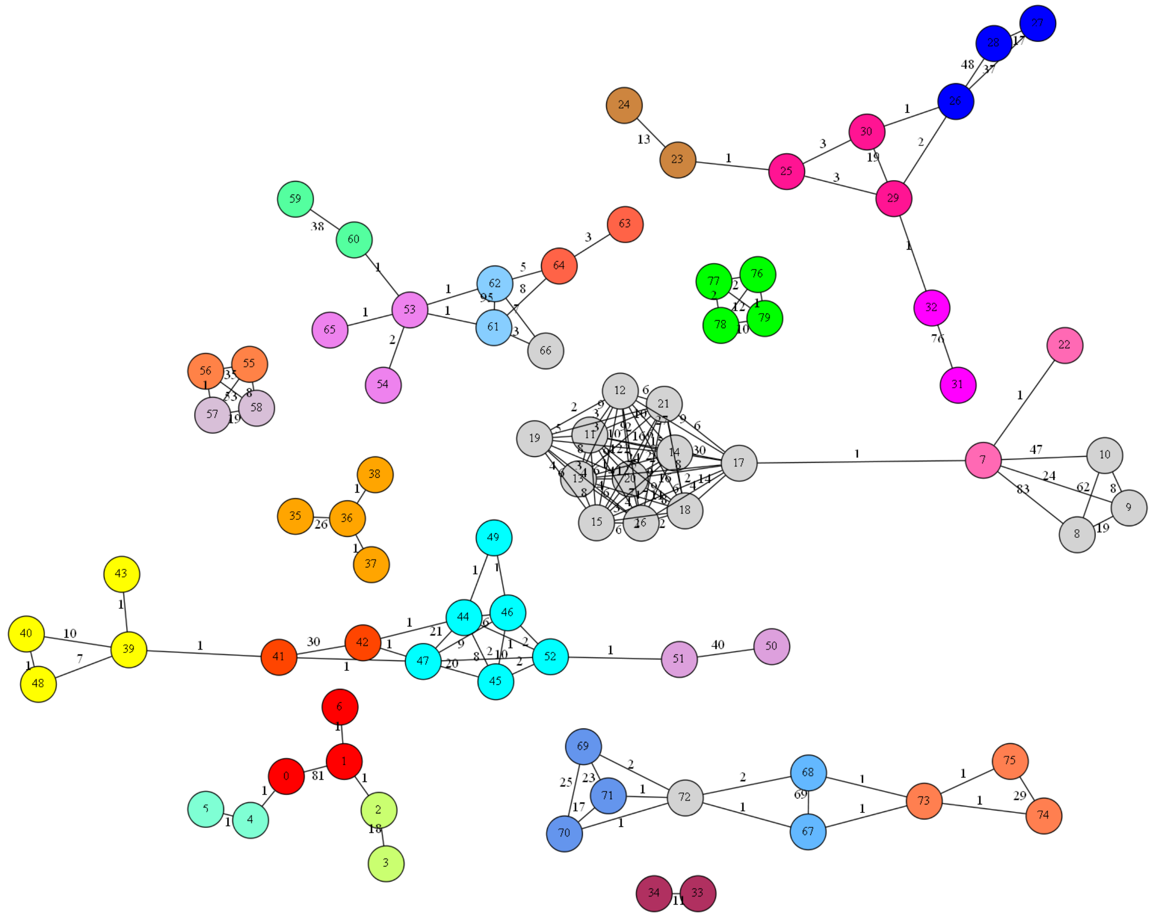


Figure 8.7: 06-02, 50% compression, Louvain Algorithm

Chapter 9

NSGA-II

NSGA-II was used to try to find a balance between the distortion of the graph and the compression ratio. NSGA-II, which stands for **N**on-Dominated **S**orting **G**enetic **A**lgorithm **II**, is a multi-objective evolutionary algorithm (MOEA) [6], meaning that it attempts to optimize multiple objectives with the same solution. We use this algorithm to balance the level of distortion introduced into the graph and the compression ratio.

9.1 Introduction

Using a single-objective genetic algorithm is great when the compression ratio needed is known. However, if the opposite is desired, for example, wanting to compress a graph as much as possible without going over a certain level of distortion, then another methodology is needed. This is where multi-objective algorithms are worthwhile.

NSGA-II was used as the multi-objective evolutionary algorithm due to the fact that it is an improved version of NSGA, proposed in [36]. NSGA-II addresses the three main concerns that NSGA provided: (1) the high computational complexity of $O(MN^3)$, where M is the number of objectives and N is the population size; (2) the lack of elitism; and (3) the need for a sharing parameter. NSGA-II was able to be improved to a computation complexity of $O(MN^2)$. To implement NSGA-II in our study, the DEAP Python library of De Rainville *et al.* [9] was used.

9.2 Background

9.2.1 NSGA-II

NSGA-II has the following steps [5, 21]:

1. A population, P, of size r is randomly generated.

2. The offspring population, Q, also of size r is generated from P using crossover and mutation reproduction functions.

3. Population P and Q are then combined into one population P' of size $2r$.

4. The Pareto rank for each solution in P' is found. In order to do this, n_p and S_p are calculated for each solution, where n_p is the number of solutions that dominate p, and S_p is the set of solutions that are dominated by p. If no solutions dominate p, then it is in the first Pareto front (the optimal Pareto front), and is brought over to a separate list from the rest of the population, F. Then, the n_p of every solution S_p in F is decremented by 1. At the end, any solutions in P' that now have an n_p of zero, are a part of the next Pareto front. This continues until no more solutions are left in P'. Pareto ranking will be explained further in the next subsection.

5. The crowding distance, described in Section 9.2.3, is also calculated for each solution originally in P'.

6. The next population P, of size r, for the next generation is selected based on a higher Pareto rank (where the first Pareto rank is the highest), and if their rank is equal, a larger crowding distance.

7. These steps are repeated until the termination criteria is reached.

9.2.2 Pareto Ranking

NSGA-II uses Pareto ranking to separate the solutions in P'. Solutions in Pareto rank x dominate those in Pareto rank $x + i$ for $i > 0$. The solutions in Pareto rank 1 are the best in at least one of the objectives from all other solutions. In order to execute Pareto ranking, Pareto dominance must be understood, and applied.

Firstly, all objectives must be measured with their own natural fitness. These objectives are kept in a vector of scores. Then, each objective of one solution is

Table 9.1: Example Solutions for Pareto Dominance

Solution	Objective One	Objective Two
A	0.85	345
B	0.30	120
C	0.99	230
D	0.85	267
E	0.74	496

Table 9.2: Pareto Ranks for Solutions in Table 9.1

Pareto Rank	Solution
1	A, C, E
2	D
3	B

compared to the corresponding objective of all the other solutions. Solution A is better than (dominates) solution B if it is as good in all objectives with at least one objective being better.

For example, let us take the solutions in Table 9.1. If both objectives are to be maximized, then solution A dominates B, since $0.30 < 0.85$ and $120 < 345$; and D, since $0.85 = 0.85$ and $267 < 345$. However, it does not dominate solution C since $0.99 > 0.85$; or solution E, since $496 > 345$. The Pareto ranks for the solutions are shown Table 9.2.

9.2.3 Crowding Distance

The crowding distance is present to ensure that the population stays full of diverse solutions. It is calculated for each chromosome, i, and is equal to the total normalized distance between the next best and next worst solutions relative to i for each objective. The maximum and minimum solutions, per objective, are selected as the bounds and are given an infinite value, Inf.

9.3 Methodology

The methodology is very similar to the methodology that was followed for the single objective genetic algorithms explained in Section 4, with a few minor changes.

9.3.1 Elitism

Firstly, there is no explicit elitism in the sense that the best solution from the previous generation replaces the worst chromosome from the new generation. However, elitism does exist within the process of NSGA-II. Recall that the next generation of chromosomes are the best from both the parent population and the offspring population.

9.3.2 Representation

The representation is similar to the representation used in Section 4.1. However, instead of having the first node showing as the node ID in the root array, and the second node showing as an offset from the first node, both nodes are now just shown by their ID's. For example, the chromosome in Figure 9.1 will merge nodes 23 with node 24, node 78 with node 81, and so forth.

node 1	23	78	45	0	23
node 2	24	81	48	5	25

Figure 9.1: NSGA-II Chromosome Example

9.3.3 Crossover

A different crossover method was used during the experiment. One point crossover was used instead of two point crossover. For one point crossover, only one random position from the chromosome is selected. Then, since the chromosomes are different lengths, the two parent chromosomes swap genes up to the random position selected. Figure 9.2 shows the chromosomes that are about to undergo the crossover process - also referred to as the parent chromosomes. Figure 9.3 shows the resulting chromosomes when the index of 3 is selected. In the case that the two chromosomes have different lengths, the random position selected must be lower than the length of the shortest chromosome.

node 1A	**23**	**78**	**45**	**0**	**23**
node 2A	**24**	**81**	**48**	**5**	**25**

node 1B	64	31	95	52	7
node 2B	66	28	92	55	8

Figure 9.2: Parent Chromosomes for One Point Crossover

node 1B	64	31	95	52	**23**
node 2B	66	28	92	55	**25**

node 1A	**23**	**78**	**45**	**0**	7
node 2A	**24**	**81**	**48**	**5**	8

Figure 9.3: Result of One Point Crossover with index 3

9.3.4 Fitness Functions

As mentioned, NSGA-II attempts to find a solution to optimize/balance multiple objectives. Each objective has its own fitness function. In our experiment, the first objective is the compression rate. The compression rate is calculated as the length of the chromosome, c, over the number of nodes in the original graph, as shown in Equation 9.1.

$$Compression = len(c)/N_G \tag{9.1}$$

The second objective is the amount of distortion introduced into the graph by the compression. We run the NSGA-II algorithm twice, once with each fitness function described in Section 4.8.

9.4 Experiment Details

The NSGA-II algorithm was only run with the original graphs, due to time constraint. Once initial runs were done, it was evident that this algorithm requires a lot of computation power. Running the algorithms on the full range of compression rate (0% compressed to 100% compressed) is too time consuming - especially if the population converges toward the 100% compressed solutions. Because of this, we decided to run

Table 9.3: Summary of Experiment Parameters

Parameter	Value
Mutation Rate	15%
Crossover Rate	90%
Max Distance	1
Generations	250 (40 for 07-15)
Population Size	1000

the NSGA-II algorithm between 30% compression and 70% compression. This, way the execution time would be fast, but we can still visualize and extrapolate the rest of the curve. The other genetic algorithm parameters were kept constant to the single objective genetic algorithms, and are displayed in Table 9.3. Note that the maximum generation was 250. However, graph 07-15 was so big, that we could only run it up to generation 40. Unfortunately, this is not enough generations to start to see a curve, or analyse the result, but it is included in the figure for completeness.

Once the algorithm finished executing, the fitness values were modified to be a similarity ratio, so that a curve, similar to the curve in [5], could be seen . This was completed by dividing each chromosome's second objective fitness value by the maximum amount of distortion possible to introduce (essentially the fitness function if the entire graph were compressed to one super-node), and then subtracted from 1. For the first and second fitness function the similarity equations are shown in Equation 9.2 and 9.3 respectively, where $wt^M(u, v)$ is the weight of edge (u, v) in the maximally compressed graph. Since the ratio is being subtracted from 1, both similarities would need to be minimized for optimization.

$$similarity_1 = 1 - \frac{\sum_{u,v \in G} |wt(u, v) - wt''(u, v)|}{\sum_{u,v \in G} |wt(u, v) - wt^M(u, v)|} \tag{9.2}$$

$$similarity_2 = 1 - \frac{\sqrt{\sum_{u,v \in G}(wt(u, v) - wt''(u, v))^2}}{\sqrt{\sum_{u,v \in G}(wt(u, v) - wt^M(u, v))^2}} \tag{9.3}$$

9.5 Results and Discussion

The final generation of chromosomes is shown on Figures 9.4 and 9.5. The x-axis shows the similarity value as explained in Section 9.4, where a similarity of 1.0 repre-

Table 9.4: Best Result Given by a Single Objective GA and a Multi Objective GA At 40% Compression Ratio using the First Fitness Function

Graph	GA		NSGA-II	
	Fitness	Similarity	Fitness	Similarity
06-02	256.17	0.9591	357.6667	0.9429
04-28	1744.21	0.9241	2576.5962	0.8879
07-07	3752.96	0.9007	4726.9317	0.8749

sents no distortion introduced to the graph from the compression, and 0.0 represents all the possible distortion was introduced to the graph. The y-axis shows the compression rate, where 0.0 is that no nodes were merged together, and 1.0 represents that all the nodes were merged into one super-node.

The main goal in this chapter, is to find a balance between the compression ratio and the amount of distortion introduced to the graph. The goal is to keep the distortion as low as possible (i.e. similarity as high as possible) while maximizing the compression. This can be accomplished by looking at the similarity/compression rates of the chromosomes in the last generation.

9.5.1 First Fitness Function

In Figure 9.4, we see the last generation of every graph that was run using the first fitness function. From the curve, and the fact that we know that a 0% compression will result in a 1.0 similarity, we can infer what the bottom of the curve will look like. A user can pick any compression rate that introduces the amount of distortion they are comfortable with. Let us say, for out purposes we only want to allow 10% of the possible distortion into the graph. This would be a similarity of 0.9. For graph 06-02, the compression rate would be around 50%. However, for graphs 04-28 and 07-07, a compression rate of 40% looks like a good balance.

Let us actually compare the best fitness values obtained by NSGA-II and the single objective GA in earlier experiments at 40% compression. Table 9.4 shows the best raw fitness value and the corresponding similarity. Note that if an exact compression rate of 0.4 wasn't available for NSGA-II, the closest compression rate and its result are shown. As can be seen, all graphs were able to obtain a better compression with the single objective GA focusing on one compression value.

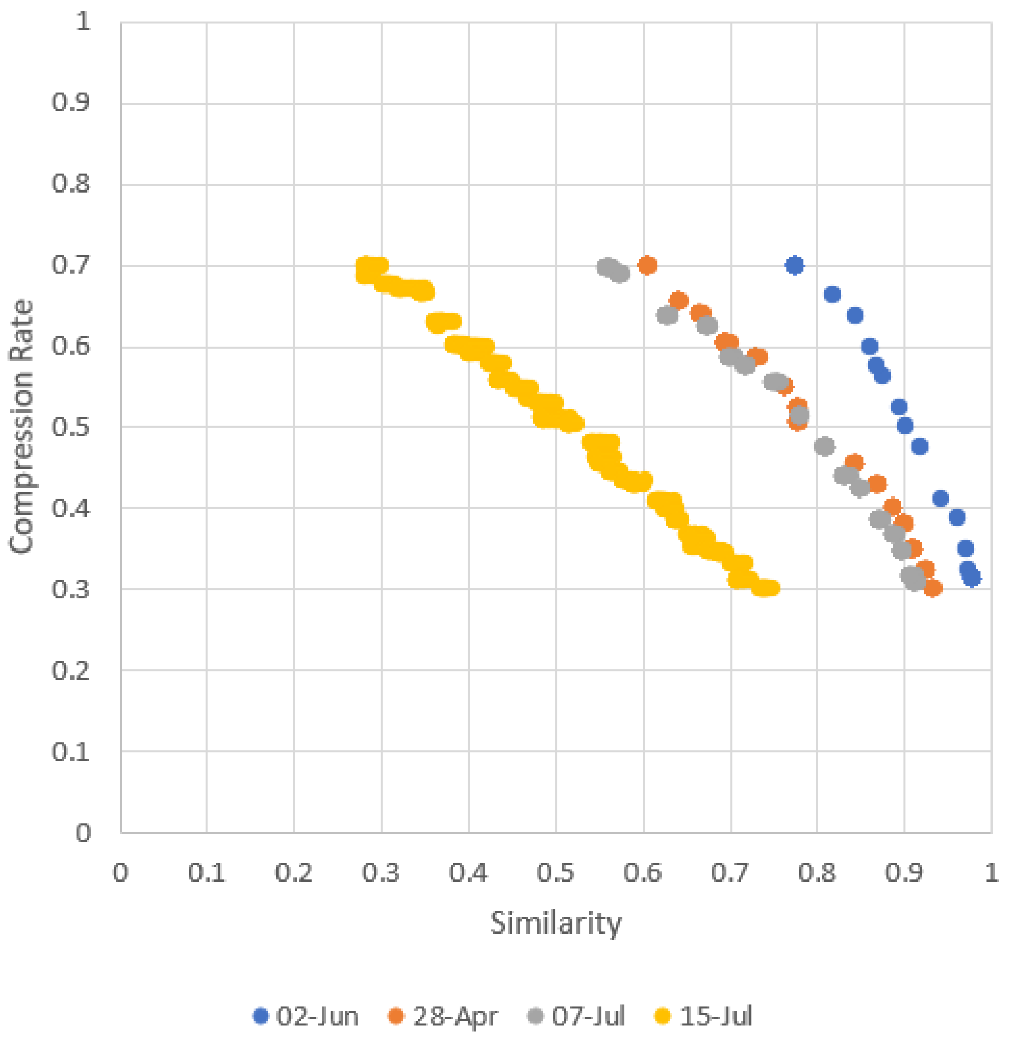


Figure 9.4: Similarity Curves for all Graphs using the first fitness function

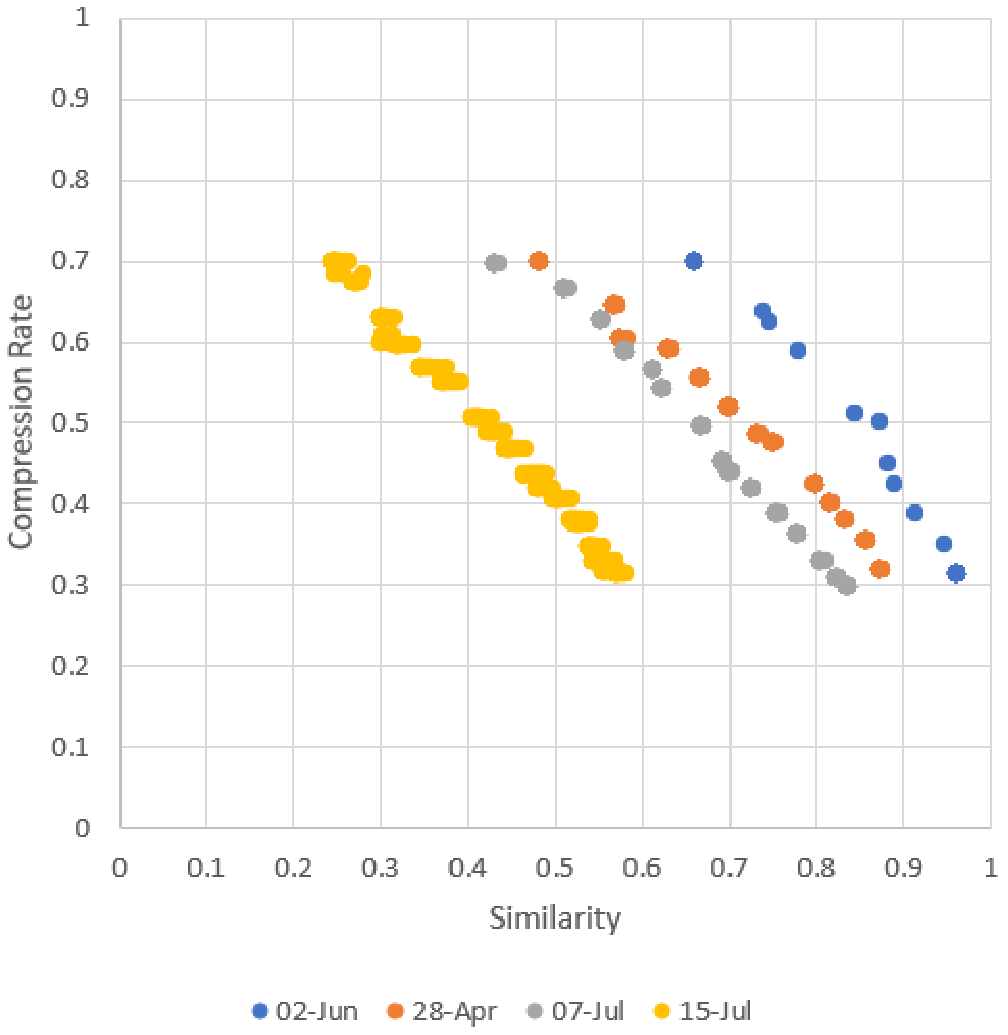


Figure 9.5: Similarity Curves for all Graphs using the second fitness function

Table 9.5: Best Result Given by a Single Objective GA and a Multi Objective GA At 40% Compression Ratio using the Second Fitness Function

Graph	GA		NSGA-II	
	Fitness	Similarity	Fitness	Similarity
06-02	29.84	0.9137	29.48	0.9147
04-28	62.77	0.9042	119.7823	0.8172
07-07	117.44	0.8633	234.0830	0.7807

9.5.2 Second Fitness Function

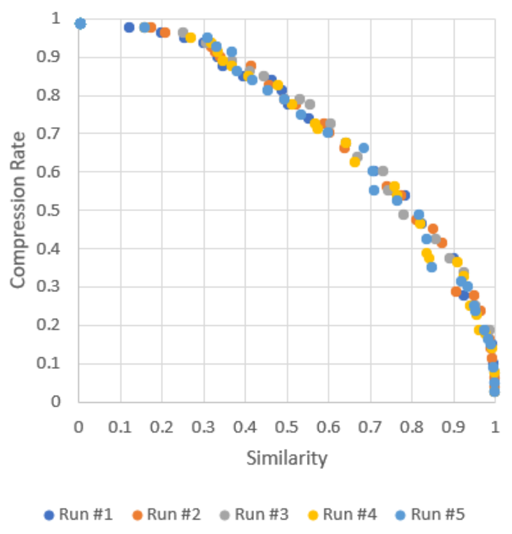
Similar observations are found for the second fitness function. Figure 9.5 shows the last generation of every graph that was run using the second fitness function. Again, we can infer what the rest of the graph would look like through extrapolation, and the compression ratio that provides the desired balance is up to the user. Lets say we again only want to introduce 10% of the possible distortion into the graphs. For graph 06-02, this would balance out to a compression rate of 40%. Graphs 04-28 and 07-07 introduce distortion to the graph more quickly. Choosing a compression rate of about 30% and 25% respectively seems to provide a good balance.

Let us compare the best results given by the single objective genetic algorithm from previous experiments to the results given by NSGA-II. This information is found in Table 9.5. Note that if an exact compression rate of 0.4 wasn't available for NSGA-II, the closest compression rate and its result are shown. For all but graph 06-02, the single objective genetic algorithm found a better solution (i.e. lower fitness, higher similarity). Thus, this reinforces our assumption that the single objective genetic algorithms provide better solutions.

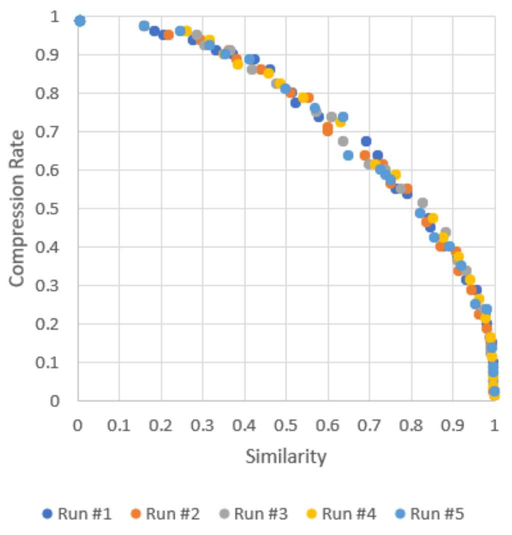
9.5.3 One Point vs Two Point Crossover

Since the single objective GA was run with two point crossover, and the NSGA-II was run with one point crossover, a mini sub-experiment was performed to see how much of a difference switching the two crossovers made. Figure 9.6 and 9.7 shows the final generation of 5 different runs of the 06-02 graph when run for 250 generations with one point crossover, and two point crossover using the first fitness function and second fitness function respectively.

From a quick glance at Figure 9.6 there doesn't seem to be a big difference between the results. However, the two point crossover is able to stay very close to a similarity of 1 up until about 20% compression. One point crossover starts to deviate from a



(a) 06-02 with One Point Crossover

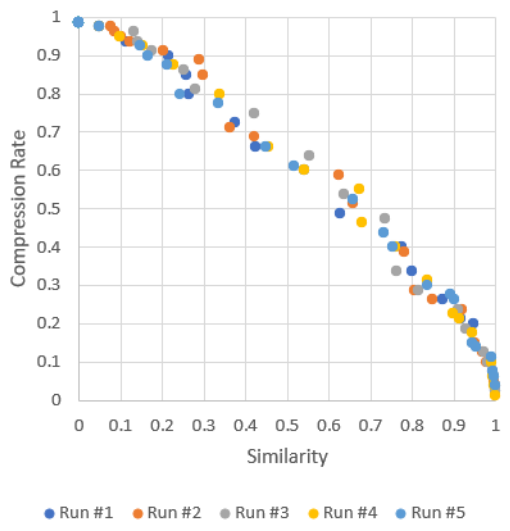


(b) 06-02 with Two Point Crossover

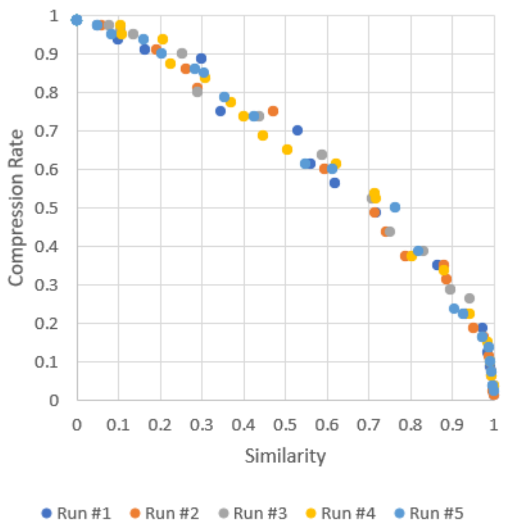
Figure 9.6: The last generations when 06-02 was run with One Point and Two Point Crossover for 250 generations using the First Fitness Function

similarity of 1 around a compression rate of 15%. Throughout the curve, the two point crossover curve provides a slightly greater or equal similarity when compared to the one point crossover curve. For example, at a compression rate of 40%, the one point crossover curve is between 0.8 and 0.9 similarity, but the two point crossover curve is between a similarity of 0.85 and 0.9. Therefore, even though two point crossover does make a slight improvement in these results, it is not enough to change the conclusion of this experiment.

Figure 9.7 shows a similar trend for the second fitness function. Again, the two point crossover stays close to a similarity of 1 for longer than one point crossover, 0.15 and 0.10 respectively. The one point crossover curve introduces about an average of 5% more distortion into the graph than the two point crossover counterpart. This can be seen at the 60% compression ratio, when the best found solution for the one point crossover GA is at 0.55 similarity, and the best found solution for the two point crossover GA is at 0.6 similarity. However, there are times when the difference between the two is smaller. Such as when the graph is compressed 40%. Again, this slight improvement by the two point crossover is not enough to change our conclusion of this experiment.



(a) 06-02 with One Point Crossover



(b) 06-02 with Two Point Crossover

Figure 9.7: The last generations when 06-02 was run with One Point and Two Point Crossover for 250 generations using the Second Fitness Function

9.6 Conclusion

In conclusion, NSGA-II is a powerful tool to get an idea of the landscape of solutions. It is possible to choose a compression rate depending on how much distortion is wanted to be introduced into the graph. However, in order to get the best possible solution at that compression, a single objective genetic algorithm is superior.

Chapter 10

Conclusion and Future Work

Throughout this thesis, we have completed many experiments, and learned a lot about weighted graph compression with genetic algorithms. In the next few sections, we will go over the results of all the experiments and outline some possible room for improvement.

10.1 Weighted vs Unweighted Graphs

We first saw how important edge weights are, and how much vital information edge weights provide for compression. From an epidemic point of view, edge weights show the level of interaction between nodes. The higher level of interaction, the greater the risk of transmitting a disease. Thus, it follows that those nodes with a high transmission risk should be treated similarly when it comes to quarantines and/or restrictions. When edge weights are not present, the genetic algorithm just merges random nodes as long as there is an edge connecting them - as this is done at no cost. When edge weights are provided, different and more meaningful nodes are merged together for all 16 experiments.

10.2 First vs Second Fitness Function

When analyzing the two fitness functions used in this thesis, it is found that both prioritize merging nodes that have commonly weighted edges to external nodes, as well as common internal weights. However, the second fitness function fares better in one situation: when the choice between merging two nodes with equal variation between two external nodes and merging two nodes with differing variation between two external nodes is present. Recall that the first fitness function evaluates both scenarios as equal distortion, and the second fitness function prioritizes the merge with equal variation. Even so, when the solutions obtained by two fitness functions were compared, they were very similar.

10.3 Original vs Adjusted Weights

The inclusion of edge weights provides important detail to the genetic algorithm, and gives a whole new understanding of the graph. When looking at the merges made by the compression algorithms, it is seen that it is merging nodes that are connected by a highly weighted edge, or with nodes that have similar connections to common nodes. In other words, the algorithms are recognizing sets of nodes that have important connections. This indicates that we may be able to use compression to reduce the complexity of large contact networks to identify individuals that should be given the same quarantine rules.

However, having meaningful edge weights in the graph is just as important, as different weights can lead to different results, and a different understanding of the graph. Providing classes for what kind of interaction the individuals are having or the length of the interaction may be beneficial. For example, the interaction of two individuals being together in the same room (i.e. a classroom or office) for hours, may not be equal to the interaction of saying, "Hello" to a stranger passing by on the street.

10.4 Community Detection

In our experiments, the genetic algorithm and the Louvain Algorithm resulted in different sets of super-nodes/communities. However, recall that there is no expectation that these results should be the same. Since the problem is essentially a clustering problem, there is no 'best' overall result without a clear definition of the application and graph in the real world. These different results may even be beneficial to public health officials. The different sets of super-nodes/communities contribute important pieces of information that can help when making decisions about restrictions, vaccinations, or isolation.

10.5 NSGA-II

NSGA-II is a powerful tool used to understand the landscape of different compressions on the graph. It is able to give a high level representation of the percentage of exposure that a certain compression ratio introduces into the graph. However, as was seen, single objective genetic algorithm produces a better solution when compared with NSGA-II results. Therefore, we recommend using NSGA-II to select a target ratio which represents the highest level of distortion that the user is willing to introduce to the graph, and then run a single objective GA with that specific compression ratio to obtain the final result.

10.6 Future Work

Many avenues are available for further research and possible improvement. First and foremost, future work should analyze larger networks, even if synthetic data would be used, in order to determine if similar trends are seen as in the current study. In the case of testing larger networks, it would be advantageous to implement the NSGA-II algorithm in a faster language than Python, such as Java, as the execution time for NSGA-II grows exponentially. It is also important to note that even though this study focused on social contact networks, this methodology can be applied to any weighted graphs. It may be beneficial to apply this methodology to large biological networks, due to its ability to identify closely related nodes. On this note, applying this methodology on graphs that contain negative weights, or where lower weights are best could be interesting. This may require the fitness function to be changed as well in order to incorporate the different meaning of the weights.

Secondly, it may be worthwhile to investigate an appropriate weight classification to represent the different levels of contact/risk in weighted contact networks.

Future work can also extend the analysis by focusing on the variance in the population, using different seeds in the Louvain Algorithm, and running more instances of NSGA-II. Furthermore, parameter testing with NSGA-II could perhaps aid in resulting in a better Pareto front.

Finally, many details in the genetic algorithms can be investigated to possibly improve results such as the fitness functions, reproductive operations, and selection method. An alternate selection method to be considered is the keep-best-reproduction, where the roulette wheel selection is used to select parents, and the best parent and best offspring are kept after reproduction [40].

Bibliography

[1] M. Adler and M. Mitzenmacher. Towards compressing web graphs. In *Proceedings DCC 2001. Data Compression Conference*, pages 203–212. IEEE, 2001.

[2] V. D. Blondel, J.L. Guillaume, R. Lambiotte, and Lefebvre E. Fast unfolding of communities in large networks. *Journal of Statistical Mechanics: Theory and Experiment*, 2008(10):P10008, 2008.

[3] J. A. Brown, S. Houghten, T. K. Collins, and Q. Qu. Evolving graph compression using similarity measures for bioinformatics applications. In *IEEE Conference on Computational Intelligence in Bioinformatics and Computational Biology*, pages 1–6, 2016.

[4] J.A. Brown, D.A. Ashlock, S. Houghten, and A. Romualdo. Evolutionary graph compression and diffusion methods for city discovery in role playing games. In *IEEE Congress on Evolutionary Computation*, 2020.

[5] T. K. Collins, A. Zakirov, J. A. Brown, and S. Houghten. Single-objective and multi-objective genetic algorithms for compression of biological networks. In *IEEE Conference on Computational Intelligence in Bioinformatics and Computational Biology*, pages 1–8, 2017.

[6] K. Deb, A. Pratap, S. Agarwal, and T. Meyarivan. A fast and elitist multiobjective genetic algorithm: Nsga-ii. *IEEE transactions on evolutionary computation*, 6(2):182–197, 2002.

[7] J. Ellson, E. R. Gansner, E. Koutsofios, S. C. North, and G. Woodhull. Graphviz and dynagraph—static and dynamic graph drawing tools. In *Graph drawing software*, pages 127–148. Springer, 2004.

[8] L. Ferretti, C. Wymant, M. Kendall, L. Zhao, A. Nurtay, L. Abeler-Dörner, M. Parker, D. Bonsall, and C. Fraser. Quantifying sars-cov-2 transmission suggests epidemic control with digital contact tracing. *Science*, 368(6491), 2020.

[9] F.A. Fortin, F.M. De Rainville, M.A. Gardner, M. Parizeau, and C. Gagné. DEAP: Evolutionary algorithms made easy. *Journal of Machine Learning Research*, 13:2171–2175, jul 2012.

[10] M. Girvan and M. E.J. Newman. Community structure in social and biological networks. *Proceedings of the national academy of sciences*, 99(12):7821–7826, 2002.

[11] D. T. Halperin. Coping with covid-19: learning from past pandemics to avoid pitfalls and panic. *Global Health: Science and Practice*, 8(2):155–165, 2020.

[12] S. Houghten, A. Romualdo, T. K. Collins, and J. A. Brown. Compression of biological networks using a genetic algorithm with localized merge. In *IEEE Conference on Computational Intelligence in Bioinformatics and Computational Biology*, pages 1–8, 2019.

[13] L. Hubert and P. Arabie. Comparing partitions. *J. Classification*, 2(1):193–218, 1985.

[14] L. Isella, J. Stehl´e, A. Barrat, C. Cattuto, J.F. Pinton, and W. Van den Broeck. What's in a crowd? analysis of face-to-face behavioral networks. *Journal of Theoretical Biology*, 271(1):166–180, 2011.

[15] M. A. Javed, M. S. Younis, S. Latif, J. Qadir, and A. Baig. Community detection in networks: A multidisciplinary review. *J. Network and Computer Applications*, 108:87–111, 2018.

[16] K. U. Khan, W. Nawaz, and Y. Lee. Set-based approximate approach for lossless graph summarization. *Computing*, 97(12):1185–1207, 2015.

[17] K. U. Khan, W. Nawaz, and Y. Lee. Scalable compression of a weighted graph. *arXiv preprint arXiv:1611.03159*, 2016.

[18] J. R. Koo, A. R. Cook, M. Park, Y. Sun, H. Sun, J. T. Lim, C. Tam, and B. L. Dickens. Interventions to mitigate early spread of sars-cov-2 in singapore: a modelling study. *The Lancet Infectious Diseases*, 20(6):678–688, 2020.

[19] R. Lambiotte et al. Laplacian Dynamics and Multiscale Modular Structure in Networks. *IEEE Transactions on Network Science and Engineering*, 1(2):76–90, 2014.

[20] A. Lancichinetti and S. Fortunato. Community detection algorithms: a comparative analysis. *Physical Review E*, 80(5), 2009.

[21] C. M. Lee. Solve multi-objective problem using nsga-ii and deap in python, available online at https://medium.com/@rossleecooloh/optimization-algorithm-nsga-ii-and-python-package-deap-fca0be6b2ffc (visited on 12-19-2021), 2019.

[22] W. Liu, A. Kan, J. Chan, J. Bailey, C. Leckie, J. Pei, and R. Kotagiri. On compressing weighted time-evolving graphs. In *Proceedings of the 21st ACM international conference on Information and knowledge management*, pages 2319–2322, 2012.

[23] P. A. Mackowiak and P. S. Sehdev. The origin of quarantine. *Clinical Infectious Diseases*, 35(9):1071–1072, 2002.

[24] M. E. J. Newman. Modularity and Community Structure in Networks. *Proceedings of the National Academy of Sciences*, 103(23):8577–8582, 2006.

[25] M.E.J. Newman. Analysis of weighted networks. *Physical review E*, 70(5):056131, 2004.

[26] M. S. Rahman. *Basic graph theory.* Springer, 2017.

[27] C. Reeves and J. E. Rowe. *Genetic algorithms: principles and perspectives: a guide to GA theory.* Springer Science & Business Media, 2002.

[28] G. Rossetti, L. Milli, and R. Cazabet. CDLIB: a python library to extract, compare and evaluate communities from complex networks. *Applied Network Science*, 4(1):52, 2019.

[29] E. Rutkowski, S. Houghten, and J.A. Brown. Extracting information from weighted contact networks via genetic algorithms. In *IEEE Conference on Computational Intelligence in Bioinformatics and Computational Biology*, pages 1–8, 2020.

[30] E. Rutkowski, J. Sargant, S. Houghten, and J.A. Brown. Evaluation of communities from exploratory evolutionary compression of weighted graphs. In *2021 IEEE Congress on Evolutionary Computation (CEC)*, pages 434–441. IEEE, 2021.

[31] J. Sadowski. When data is capital: Datafication, accumulation, and extraction. *Big Data & Society*, 6(1), 2019.

[32] M. Salathé, C. L. Althaus, R. Neher, S. Stringhini, E. Hodcroft, J. Fellay, M. Zwahlen, G. Senti, M. Battegay, A. Wilder-Smith, I. Eckerle, M. Egger, and N. Low. Covid-19 epidemic in switzerland: on the importance of testing, contact tracing and isolation. *Swiss medical weekly*, 150(1112), 2020.

[33] P. Serafino. Speeding up graph clustering via modular decomposition based compression. In *Proceedings of the 28th Annual ACM Symposium on Applied Computing*, pages 156–163, 2013.

[34] SocioPatterns. Deployment: Infectious SocioPatterns, available online at http://www.sociopatterns.org/deployments/infectious-sociopatterns/ (visited on 12/20/2021).

[35] SocioPatterns. Infectious contact networks, available online at http://www.sociopatterns.org/datasets/ (visited on 12/20/2021).

[36] N. Srinivas and K. Deb. Muiltiobjective optimization using nondominated sorting in genetic algorithms. *Evolutionary Computation*, 2(3):221–248, 1994.

[37] D. Steinley. Properties of the hubert-arable adjusted rand index. *Psychological Methods*, 9(3):386, 2004.

[38] H. Toivonen, A. Hartikainen, F. Zhou, and A. Hinkka. Compression of weighted graphs. In *ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 965–973, 2011.

[39] D. B. West. *Introduction to Graph Theory.* Prentice Hall, Upper Saddle River, NJ 07458, 1996.

[40] K. Wiese and S. D. Goodwin. Keep-best reproduction: a selection strategy for genetic algorithms. In *Proceedings of the 1998 ACM Symposium on Applied Computing*, pages 343–348, 1998.

[41] A. Wilder-Smith, C. J. Chiew, and V. J. Lee. Can we contain the covid-19 outbreak with the same measures as for sars? *The Lancet Infectious Diseases*, 20(5):e102–e107, 2020.

[42] A. Wilder-Smith and D. O. Freedman. Isolation, quarantine, social distancing and community containment: pivotal role for old-style public health measures in the novel coronavirus (2019-ncov) outbreak. *Journal of Travel Medicine*, pages 1–4, 2020.

[43] A.N. Zakirov and J.A. Brown. NSGA-II for biological graph compression. *Advanced Studies in Biology*, 9(1):1–7, 2017.

Appendix A

Original Graph 06-02

The below Figure is the visualization of the uncompressed 06-02 graph introduced in Section 4.9. This is the smallest graph that we use throughout the study.

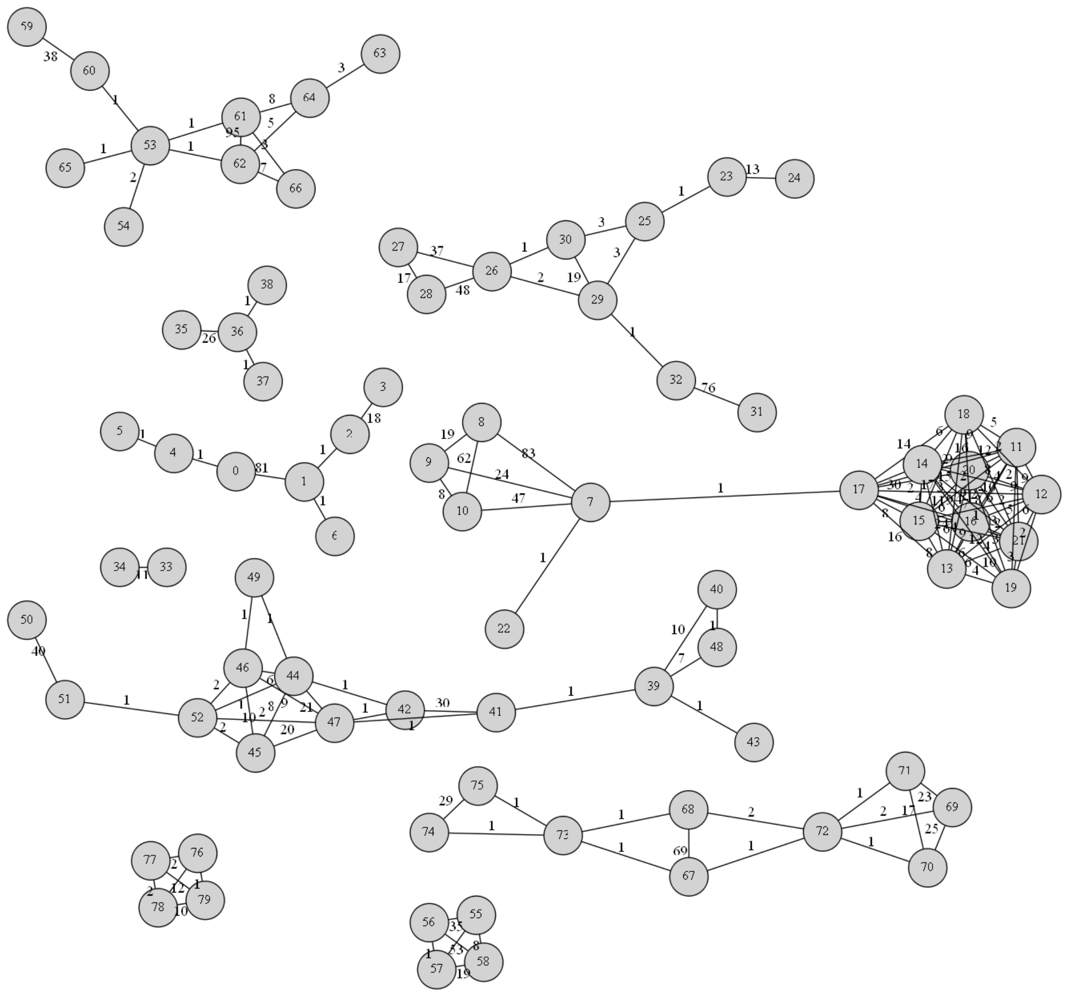


Figure A.1: Data set from “Infectious: Stay Away Event” Day 06-02

Appendix B

Original Graph 04-28

The below Figure is the visualization of the uncompressed 04-28 graph introduced in Section 4.9 with the original weights.

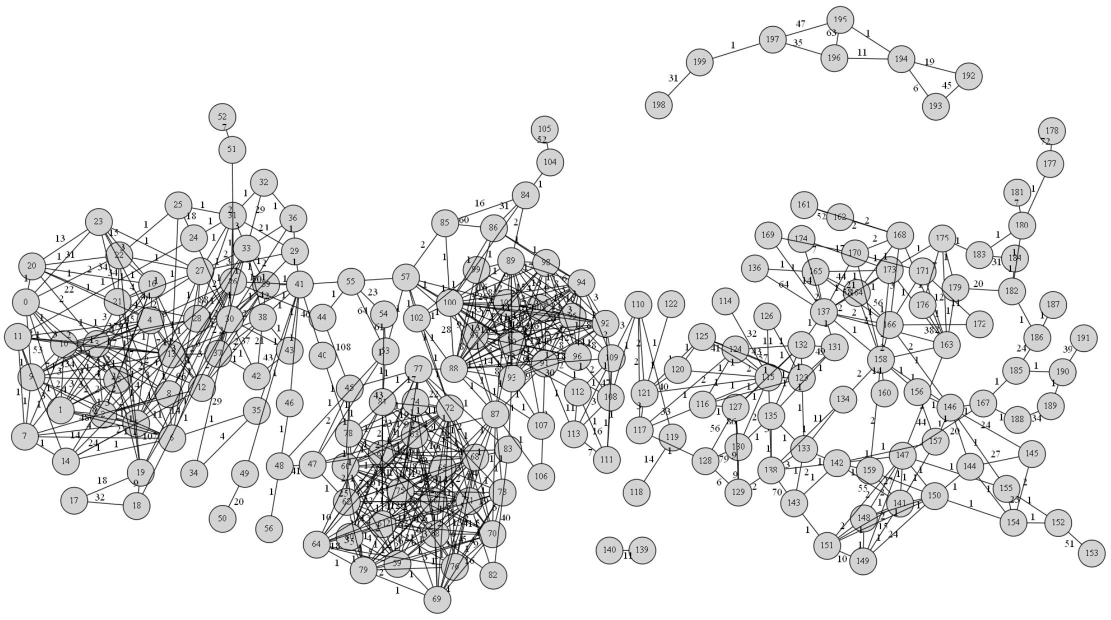


Figure B.1: Data set from “Infectious: Stay Away Event” Day 04-28

Appendix C

Original Graph 07-07

The below Figure is the visualization of the uncompressed 07-07 graph introduced in Section 4.9 with the original weights.

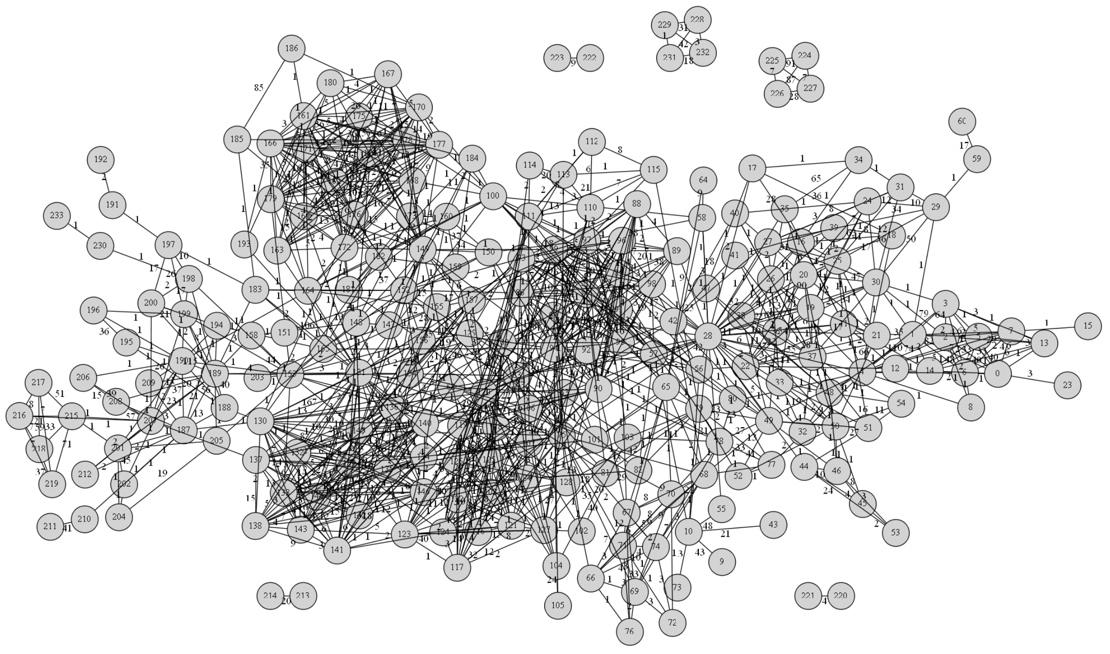


Figure C.1: Data set from “Infectious: Stay Away Event” Day 07-07

Appendix D

Original Graph 07-15

The below Figure is the visualization of the uncompressed 07-15 graph introduced in Section 4.9 with the original weights. This is the largest graph we use throughout the study.

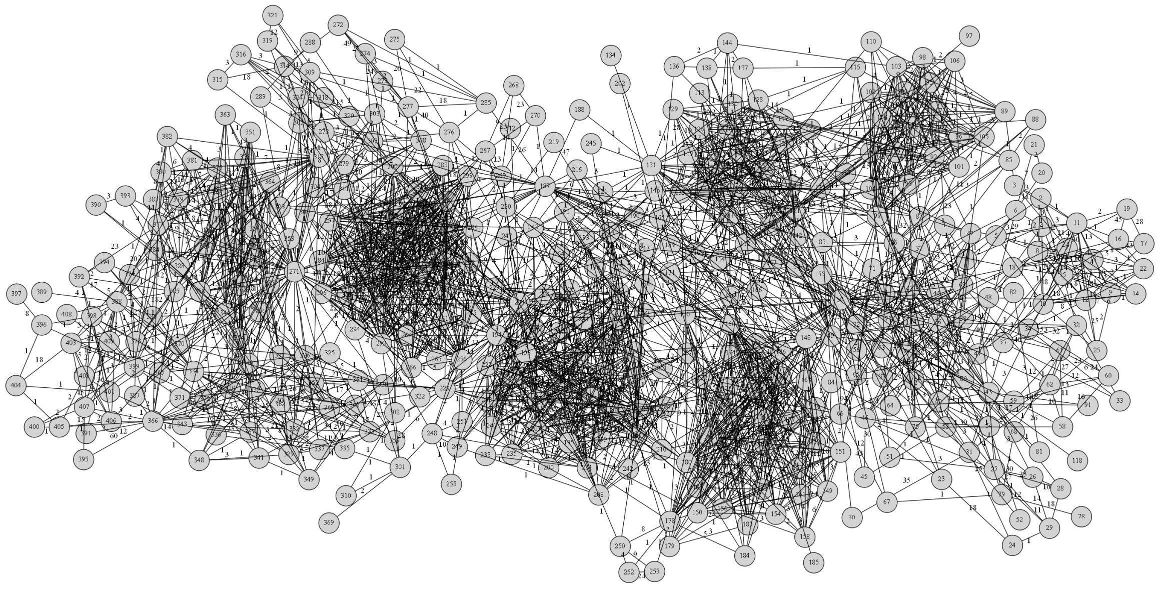


Figure D.1: Data set from “Infectious: Stay Away Event” Day 07-15

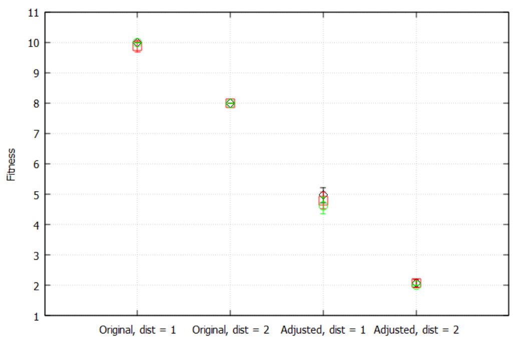
Appendix E

Mutation And Crossover Rates

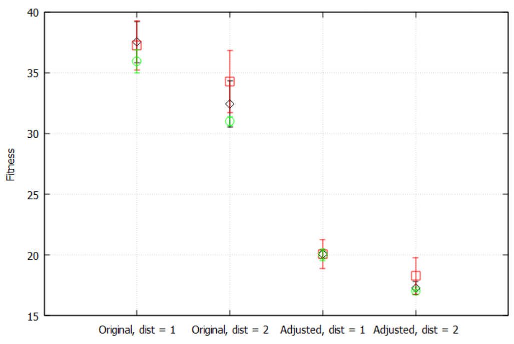
Preliminary runs were conducted in order to choose an adequate crossover and mutation rate. Three different crossover and mutation rate combinations were tested: (1) 80% crossover and 10% mutation, (2) 90% crossover and 10% mutation, and (3) 90% crossover and 15% mutation. We ran these different crossover and mutation rate combinations with all the weighted data (original weights and adjusted weights) and two different distance parameters (1 and 2).

In Figure E.1, we see the mean best fitness values obtained with the first fitness function with the 95% confidence interval broken up by date. Recall that all the fitness functions should be minimized. Many of the results obtained were not statistically significant. The only graph that produced statistically significant results was graph 07-15. The graph produced statistically better results when using a crossover rate of 90% and a mutation rate of 15% for 3/4 experiments when compressed 10% (original weights & distance = 1, original weights & distance = 2, and adjusted weights & distance = 2), and 2/4 experiments when compressed 20% (original weights & distance = 2, and adjusted weights & distance = 2). The symbol representations are described in Table E.1.

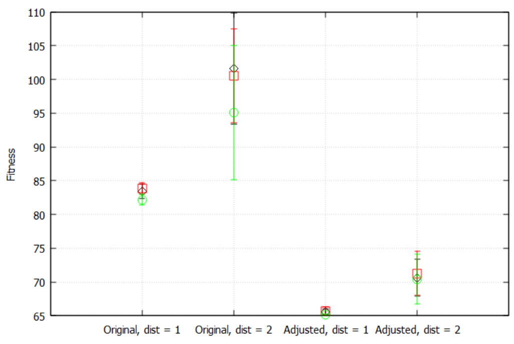
Let us now consider the second fitness function shown in Figure E.2. Here, again, the smallest graph, 06-02, did not return any statistically significant result. A crossover rate of 90% and a mutation rate of 15% are statistically better for 3/4 experiments when graph 04-28 was compressed 10% and 2/4 experiments when compressed 20%.The same parameter combination remained statistically significant for the original weighted graph 07-07 with a distance of 2 when compressed 10%, and both original weighted graphs 07-07 when compressed 20%. Finally, the original weighted graph 07-15, with distance 1 and 2, also performed better with a crossover and mutation rate set to 90% and 15% respectively when compressed 10% and 20%.



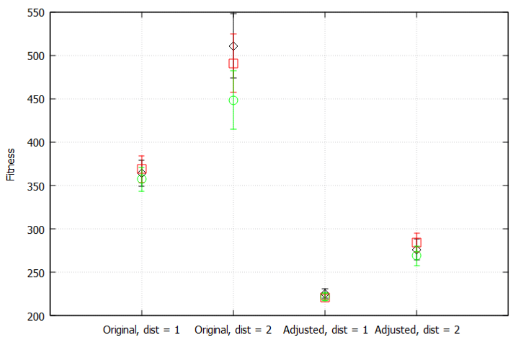
(a) 06-02, compr = 10%



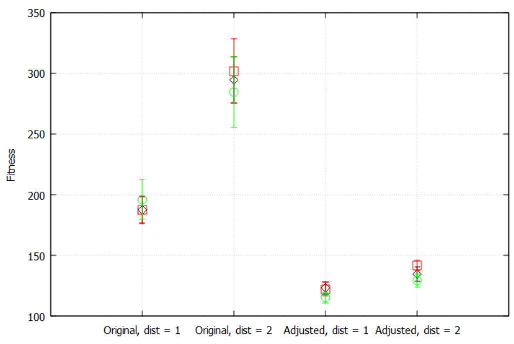
(b) 06-02, compr = 20%



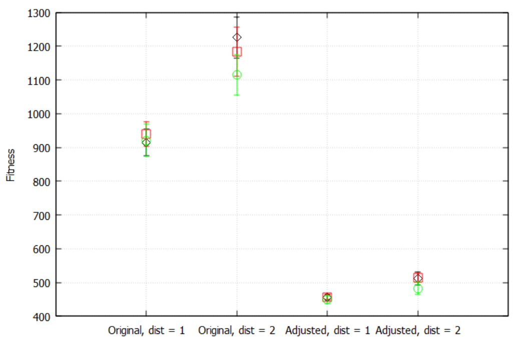
(c) 04-28, compr = 10%



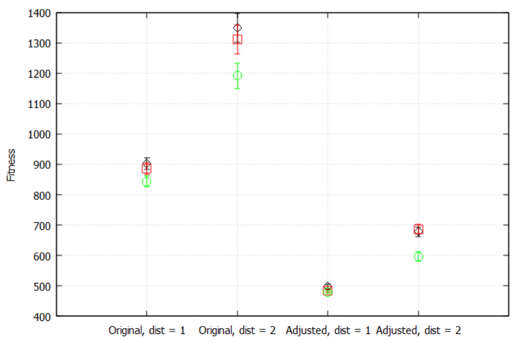
(d) 04-28, compr = 20%



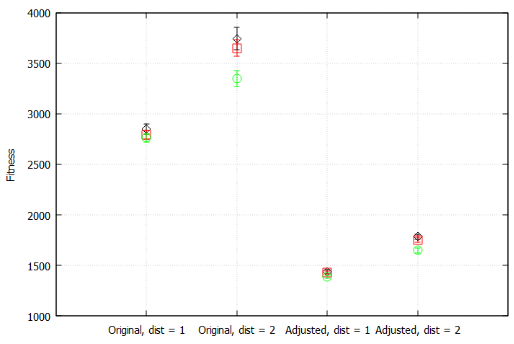
(e) 07-07, compr = 10%



(f) 07-07, compr = 20%



(g) 07-15, compr = 10%



(h) 07-15, compr = 20%

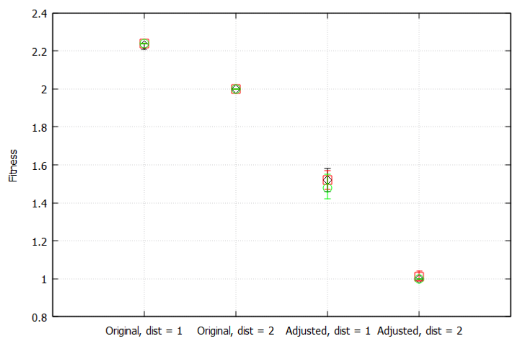
Figure E.1: The mean best fitness value with 95% confidence intervals for the First Fitness Function 4.8.1. See Table E.1 for symbol representation.

Table E.1: Symbol Representation

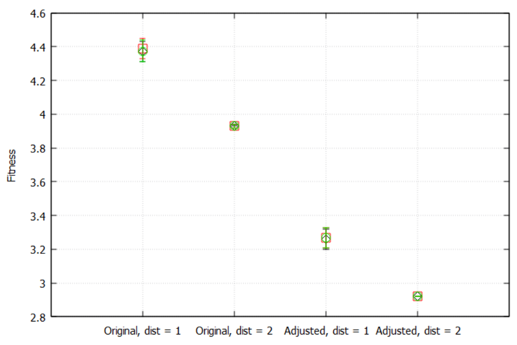
Symbol	Crossover Rate	Mutation Rate
Back Diamond	80	10
Red Square	90	10
Green Circle	90	15

The adjusted 07-15 graph with a distance of 2 is also statistically significant when compressed 20%.

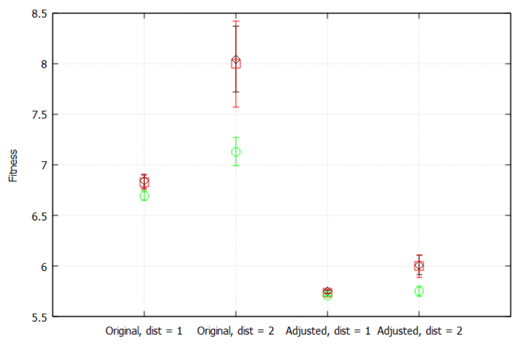
From these experiments, it is clear that a 90% crossover and 15% mutation rate is best for the GAs in general. Even though, these experiment parameters are not always statistically significant, they are the only parameters that resulted in a statistically better result for some of the data. Therefore, these parameters will be used for the rest of the study. There is evidence that the larger the graph is and/or the higher the compression, the larger an impact the parameters will have on the results as well. These results were published in the CIBCB 2020 conference proceedings [29].



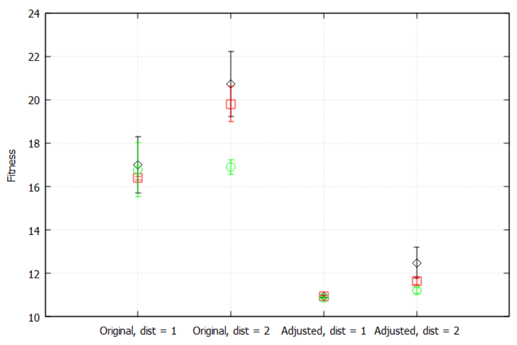
(a) 06-02, compr = 10%



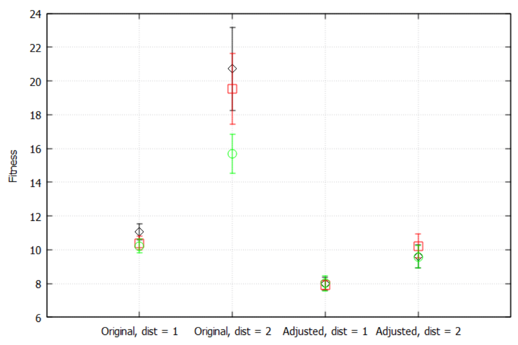
(b) 06-02, compr = 20%



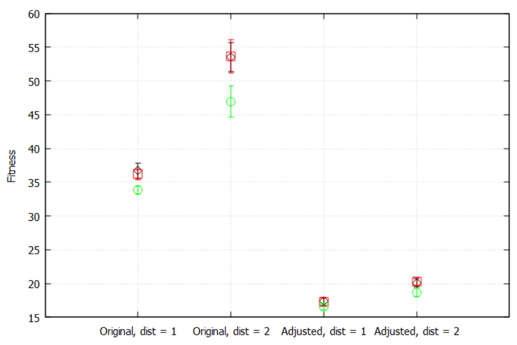
(c) 04-28, compr = 10%



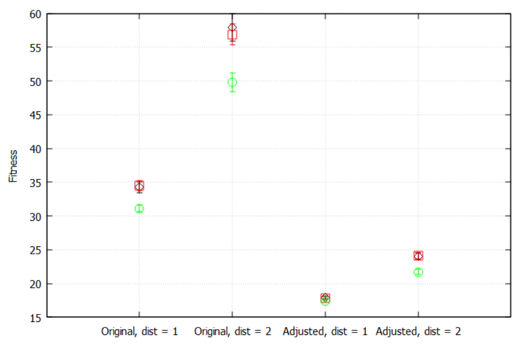
(d) 04-28, compr = 20%



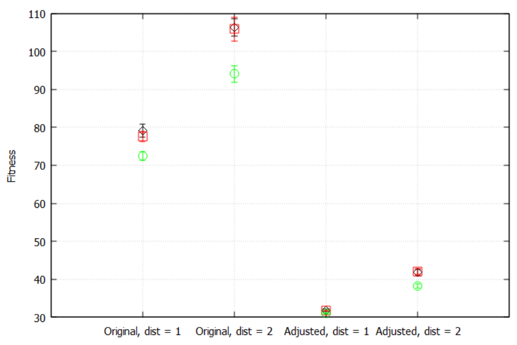
(e) 07-07, compr = 10%



(f) 07-07, compr = 20%



(g) 07-15, compr = 10%



(h) 07-15, compr = 20%

Figure E.2: The mean best fitness values with 95% confidence intervals for the Second Fitness Function 4.8.2. See Table E.1 for symbol representation.

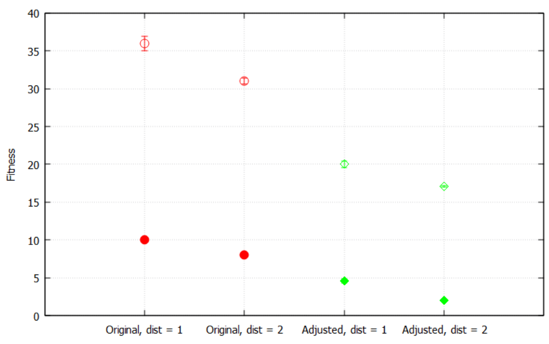
Appendix F

Distance

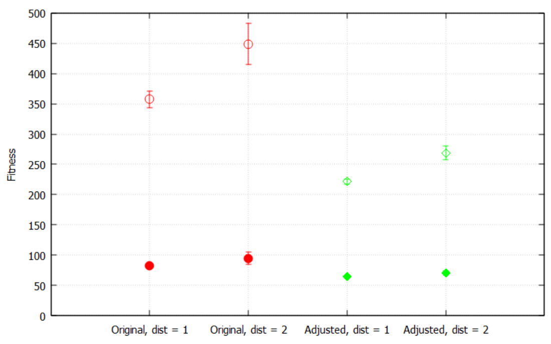
Preliminary runs were conducted to select the best performing distance parameter. The experiments conducted in [12] proved that a local merge (i.e. a smaller maximum distance) leads to lower fitness, and thus, a better compression. The paper tests distances of 10, 5, and 3. The smallest distance reduced the overall distortion of the graph for 2/3 of the graphs tested. This hypothesis test was taken further, and distances of 2 and 1 were examined with four experiments: (1) original graphs when compressed 10%, (2) original graphs compressed 20%, (3) adjusted graphs when compressed 10%, and (4) adjusted graphs when compressed 20%. These experiments were run with both fitness functions.

Figure F.1 shows the mean fitness value of the original and adjusted graphs using the first fitness function with the 95% confidence interval. The only graph where a distance of 2 is better than a distance of 1 is graph 06-02. This may be due to the structure of the graph. Graph 06-02 is comprised of many disconnected sub-graphs, as seen in Appendix A. Graphs 04-28, 07-07, and 07-15 show that a distance of 1 is statistically significantly better than a distance of 2 for all four experiments. As the graphs get larger in order, the greater the differences get between the confidence intervals.

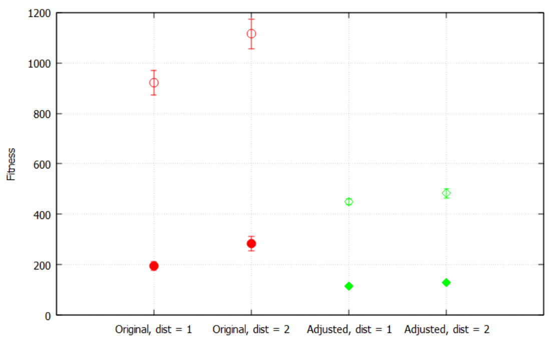
Very similar results are seen when the second fitness function is used. These results are shown in Figure F.2. The only difference seen is in graph 04-28. Using the second fitness function, the distance of 1 was only statistically significant for 2/4 of the experiments, namely when the original graph was compressed 10%, and when the adjusted graph was compressed 20%. In the other two experiments, neither of the values resulted in being statistically significant.



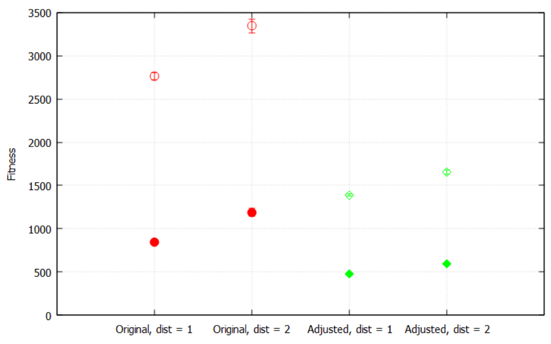
(a) 06-02



(b) 04-28

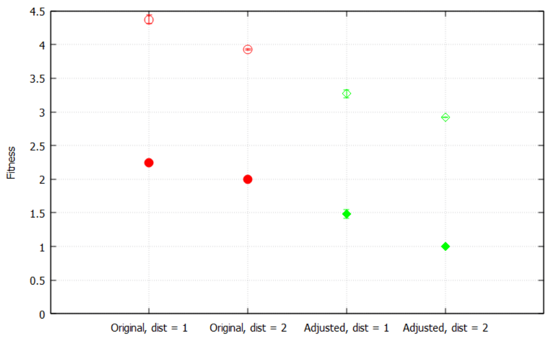


(c) 07-07

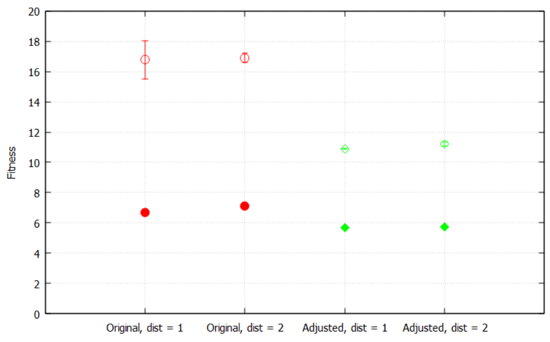


(d) 07-15

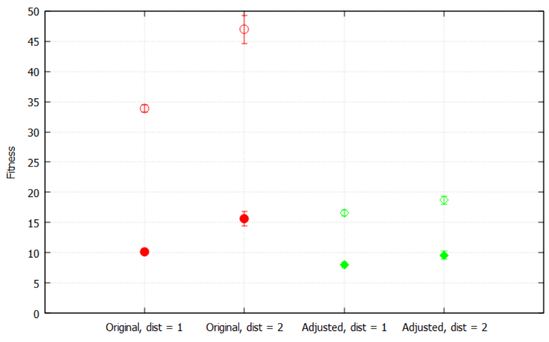
Figure F.1: The mean best fitness values with 95% confidence intervals for the First Fitness Function 4.8.1. Where solid symbols show the results for 10% compression, and the hallow symbols show the results for 20% compression.



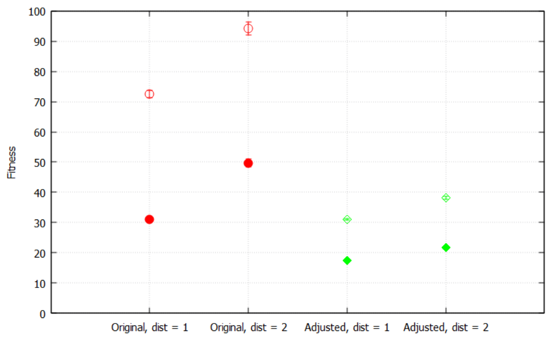
(a) 06-02



(b) 04-28



(c) 07-07



(d) 07-15

Figure F.2: The mean best fitness values with 95% confidence intervals for the Second Fitness Function 4.8.2.Where solid symbols show the results for 10% compression, and the hallow symbols show the results for 20% compression.

From these results, it is clear that for most of the experiments ran, a distance of 1 is better. Therefore, for the rest of the study, we will only use a distance of 1 in the parameters. There is also evidence to show that as the graph is more connected, and has a higher order, the value entered for the distance parameter will make a larger difference. These results were published in the CIBCB 2020 conference proceedings [29].